DAFTAR PUSTAKA

Ahmed, A., Imran, A. S., Manaf, A., Kastrati, Z., & Daudpota, S. M. (2024). Enhancing wrist abnormality detection with YOLO: Analysis of state-of-the-art single-stage detection models. *Biomedical Signal Processing and Control*, *93*, 106144. https://doi.org/10.1016/j.bspc.2024.106144

Anderson, W. D. (1967). *Field guide to the snappers (Lutjanidae) of the western Atlantic* (Vol. 252). US Department of the Interior, Fish and Wildlife Service, Bureau of~….

Bashir, A., Ishak, Z., Asngari, I., Mukhlis, M., Atiyatna, D. P., & Hamidi, I. (2019). The Performance and Strategy of Indonesian's Fisheries: A Descriptive Review. *International Journal of Economics and Financial Issues*, *9*(1), 31–36. https://ideas.repec.org/a/eco/journ1/2019-01-5.html

Chatla, D., Padmavathi, P., & Srinu, G. (2019). DNA divergence and genetic relatedness of Epinephelus species (Perciformes: Serranidae) of Indian waters inferred from COI sequence data. *Conference Paper: Recent Trends in Advance Biology, Adikavi Nannaya University, Rajamahendrvaram, AP, India. Editor: P. Vijaya Nirmala, NSRTAB-2018*.

Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, *21*(1), 6. https://doi.org/10.1186/s12864-019-6413-7

Craig, M. T., & Hastings, P. A. (2007). A molecular phylogeny of the groupers of the subfamily Epinephelinae (Serranidae) with a revised classification of the Epinephelini. *Ichthyological Research*, *54*, 1–17.

Darwin, C., Pamulapati, P., & Srinu, G. (2020). Taxonomic validation of Areolate grouper, Epinephelus areolatus (Perciformes: Serranidae) along the Nizampatnam coast, India. *Journal of Applied Biology and Biotechnology*, *8*(4), 7–15.

de Myttenaere, A., Golden, B., Le Grand, B., & Rossi, F. (2016). Mean Absolute entage Error for regression models. *Neurocomputing*, *192*, 38–48. s://doi.org/10.1016/j.neucom.2015.12.114

Zhuang, X., Guo, F., Wang, J., Su, Y., Zhang, Q., & Li, Q. (2006). ecular phylogenetic relationships of China Seas groupers based on

cytochrome b gene fragment sequences. *Science in China, Series C: Life Sciences*, *49*(3), 235–242. https://doi.org/10.1007/S11427-006-0235-Y/METRICS

Fernandes, A. F. A., Turra, E. M., de Alvarenga, É. R., Passafaro, T. L., Lopes, F. B., Alves, G. F. O., Singh, V., & Rosa, G. J. M. (2020). Deep Learning image segmentation for extraction of fish body measurements and prediction of body weight and carcass traits in Nile tilapia. *Computers and Electronics in Agriculture*, *170*, 105274. https://doi.org/10.1016/j.compag.2020.105274

Froese, R., & Pauly, D. (2019). *FishBase*. http://www.fishbase.org

Froese, R., Tsikliras, A. C., & Stergiou, K. I. (2011). Editorial note on weight–length relations of fishes. *Acta Ichthyologica et Piscatoria*, *41*(4), 261–263.

Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing, Global Edition*. Pearson Education. https://books.google.co.id/books?id=p74oEAAAQBAJ

He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, *21*(9), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

Herath, D., Perera, C., Hettiarachchi, C., & Murphy, B. (2019). Length-weight and length-length relationships of three neritic tuna species of Sri Lankan coastal waters. *International Journal of Fisheries and Aquatic Studies*, *7*(2), 129–133.

Jähne, B. (2005). *Digital Image Processing*. Springer Berlin Heidelberg. https://books.google.co.id/books?id=GVWYBrDXMNkC

Jiang, P., Ergu, D., Liu, F., Cai, Y., & Ma, B. (2022). A Review of Yolo algorithm developments. *Procedia Computer Science*, *199*, 1066–1073.

Karim, S., Zhang, Y., Yin, S., & Bibi, I. (2021). Auxiliary Bounding Box Regression for Object Detection in Optical Remote Sensing Imagery. *Sensing and Imaging*, *22*(1), 5. https://doi.org/10.1007/s11220-020-00319-x

Le Cren, E. D. (1951). The length-weight relationship and seasonal cycle in gonad weight and condition in the perch (Perca fluviatilis). *The Journal of Animal Ecology*, 201–219.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), -444. https://doi.org/10.1038/nature14539

, Fu, Wang, Zhang, & Kong. (2019). Dual-NMS: A Method for onomously Removing False Detection Boxes from Aerial Image Object ction Results. *Sensors*, *19*(21), 4691. https://doi.org/10.3390/s19214691

Maricar, M. A. (2019). Analisa perbandingan nilai akurasi moving average dan exponential smoothing untuk sistem peramalan pendapatan pada perusahaan xyz. *Jurnal Sistem dan Informatika (JSI)*, *13*(2), 36–45.

Miesle, P. (2023). Understanding the Real-World Applications of Cosine Similarity. Dalam *DataStax*. https://www.datastax.com/guides/real-world-applications-of-cosine-similarity

Nabillah, I., & Ranggadara, I. (2020). Mean Absolute Percentage Error untuk Evaluasi Hasil Prediksi Komoditas Laut. *JOINS (Journal of Information System)*, *5*(2), 250–255. https://doi.org/10.33633/joins.v5i2.3900

Ningsih, A. A., & others. (2015). *Identifikasi Parasit Ikan Kerapu (Ephinephelus sp.) Pasca Terjadinya Harmfull Algal Blooms (HABs) di Pantai Ringgung Kabupaten Pesawaran*. Fakultas Pertanian.

Nugroho, K. S. (2020, Juni). *Confusion Matrix untuk Evaluasi Model pada Supervised Learning*. Medium. https://ksnugroho.medium.com/confusion-matrix-untuk-evaluasi-model-pada-unsupervised-machine-learning-bc4b1ae9ae3f

Oktaviyani, S. (2018). Mengenal marga Lutjanus, salah satu komoditas unggulan dalam perikanan tangkap. *Oseana*, *43*(3), 29–39.

Park, E., & Berg, A. C. (2016). *Learning to decompose for object detection and instance segmentation*.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 779–788. https://doi.org/10.1109/CVPR.2016.91

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *39*(06), 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

Saha, S. (2018). *A Guide to Convolutional Neural Networks — the ELI5 way*. https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

Chakraborty, S., Sehrin, S., Habib, K., & Baki, M. (2018). *NEW RECORDS OF TWO LUTJANUS SPECIES (TELEOSTEI: PERCIFORMES: LUTJANIDAE) WITH RE-DESCRIPTION OF SIX LUTJANIDS FROM*

SAINT MARTIN'S ISLAND OF THE BAY OF BENGAL, BANGLADESH. 239–253. https://doi.org/10.3329/bjz.v46i.39056

Shianto, K. A., Gunadi, K., & Setyati, E. (2019). Deteksi jenis mobil menggunakan metode yolo dan faster r-cnn. *Jurnal Infra*, *7*(1), 157–163.

Sultana, F., Sufian, A., & Dutta, P. (2020). A review of object detection models based on convolutional neural network. *Intelligent computing: image processing based applications*, 1–16.

Ting, K. M. (2017). Confusion Matrix. Dalam G. I. Sammut Claude and Webb (Ed.), *Encyclopedia of Machine Learning and Data Mining* (hlm. 260). Springer US. https://doi.org/10.1007/978-1-4899-7687-1_50

White, D. J., Svellingen, C., & Strachan, N. J. C. (2006). Automated measurement of species and length of fish by computer vision. *Fisheries Research*, *80*(2), 203–210. https://doi.org/https://doi.org/10.1016/j.fishres.2006.04.009

Wootton, R. J. (1990). *Ecology of Teleost Fishes* (Second Edition). Chapman and Hall. https://openlibrary.org/books/OL2196217M/Ecology_of_teleost_fishes

Zhang, C.-J., Liu, T., Wang, J., Zhai, D., Zhang, Y., Gao, Y., Wu, H.-Z., Yu, J., & Chen, M. (2024). Evaluation of the YOLO models for discrimination of the alfalfa pollinating bee species. *Journal of Asia-Pacific Entomology*, *27*(1), 102195. https://doi.org/10.1016/j.aspen.2023.102195

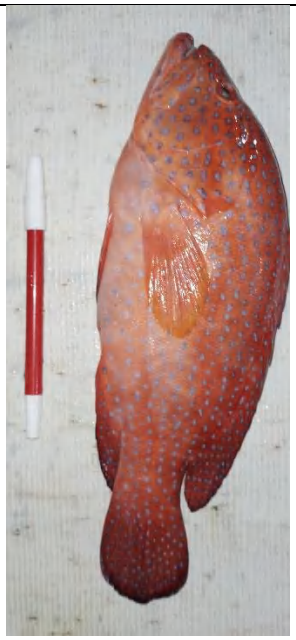
Zhong, Z., Zheng, L., Kang, G., Li, S., & Yang, Y. (2017). Random Erasing Data Augmentation. *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, 13001–13008. https://doi.org/10.1609/aaai.v34i07.7000

Zion, B. (2012). The use of computer vision technologies in aquaculture – A review. *Computers and Electronics in Agriculture*, *88*, 125–132. https://doi.org/https://doi.org/10.1016/j.compag.2012.07.010

Zou, X. (2019). A review of object detection techniques. *2019 International conference on smart grid and electrical automation (ICSGEA)*, 251–254.

Lampiran 1. Tabel jenis ikan pada *dataset*

No	Nama latin	Foto
1	*Anyperodon leucogrammicus*	
2	*Cephalopholis argus*	
3	*Cephalopholis boenak*	
4	*Cephalopholis cyanostigma*	

No	Nama latin	Foto
5	*Cephalopholis miniata*	
6	*Epinephelus areolatus*	
7	*Epinephelus fasciatus*	

No	Nama latin	Foto
8	*Epinephelus faveatus*	
9	*Epinephelus ongus*	
10	*Epinephelus quoyanus*	

No	Nama latin	Foto
11	*Lutjanus bengalensis*	
12	*Lutjanus carponotatus*	
13	*Lutjanus decussatus*	

No	Nama latin	Foto
14	*Lutjanus fulviflamma*	
15	*Lutjanus gibbus*	
16	*Lutjanus lutjanus*	

No	Nama latin	Foto
17	*Lutjanus vittae*	
18	*Variola albimarginata*	

Lampiran 2. Nilai pendekatan konstanta jenis dan pertumbuhan pada ikan

Lampiran 2.1. Tabel nilai konstanta ‘a’ dan ‘b’ ikan kerapu

NO	SPESIES	a	b
1	*Anyperodon leucogrammicus*	0,0069	3,16
2	*Cephalopholis argus*	0,0126	3,1
3	*Cephalopholis boenak*	0,0186	3,01
4	*Cephalopholis cyanostigma*	0,0257	2,92
5	*Cephalopholis miniata*	0,0112	3,1
6	*Epinephelus areolatus*	0,0129	3,02
7	*Epinephelus fasciatus*	0,0186	2,94
8	*Epinephelus faveatus*	0,01175	3,04
9	*Epinephelus ongus*	0,0178	3,01
10	*Epinephelus quoyanus*	0,0219	2,99
11	*Variola albimarginata*	0,0186	3,07

Lampiran 2.2. Tabel nilai konstanta ‘a’ dan ‘b’ ikan kakap

NO	SPESIES	a	b
1	*Lutjanus vittae*	0,0145	3,07
2	*Lutjanus lutjanus*	0,0229	2,91
3	*Lutjanus gibbus*	0,0204	2,98
4	*Lutjanus fulviflamma*	0,0245	2,94
5	*Lutjanus decussatus*	0,0245	2,96
6	*Lutjanus carponotatus*	0,0234	2,88
7	*Lutjanus bengalensis*	0,01445	2,97

Lampiran 3. *Source code*

Lampiran 3.1 *Training* model YOLO

```
!nvidia-smi

import os
HOME = os.getcwd()
print(HOME)

# Pip install method (recommended)

!pip install ultralytics==8.0.20

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()

from ultralytics import YOLO
from IPython.display import display, Image

!mkdir {HOME}/datasets
%cd {HOME}/datasets

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="rlqdT5OCox5dHxJas5IZ")
project = rf.workspace("tes2").project("model4scriptsweet")
dataset = project.version(3).download("yolov8")

%cd {HOME}

# dataset.location = '/kaggle/input/datayolopt3'
!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml
epochs=45 imgsz=640 batch=32 lr0=0.0001 plots=True

!ls {HOME}/runs/detect/train/
%cd {HOME}
Image(filename=f'{HOME}/runs/detect/train/confusion_matrix.png', width=600)

%cd {HOME}
Image(filename=f'{HOME}/runs/detect/train/results.png', width=600)

%cd {HOME}

!yolo task=detect mode=val conf=0.85 iou=0.7
model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml

%cd {HOME}
!yolo task=detect mode=predict model={HOME}/runs/detect/train/weights/best.pt
conf=0.25 source="/content/runs/detect/predict" save=True
```

Lampiran 3.2 *Training* model Faster R-CNN

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import torch
import torchvision
from torchvision import datasets, models
from torchvision.transforms import functional as FT
from torchvision import transforms as T
from torch import nn, optim
from torch.nn import functional as F
from torch.utils.data import DataLoader, sampler, random_split, Dataset
import copy
import math
from PIL import Image
import cv2
import albumentations as A  # our data augmentation library
import matplotlib.pyplot as plt
%matplotlib inline
# remove warnings (optional)
import warnings
warnings.filterwarnings("ignore")
from collections import defaultdict, deque
import datetime
import time
from tqdm import tqdm # progress bar
from torchvision.utils import draw_bounding_boxes
!pip install pycocotools
from pycocotools.coco import COCO

# Now, we will define our transforms
from albumentations.pytorch import ToTensorV2

def get_transforms(train=False):
    if train:
        transform = A.Compose([
            A.Resize(640, 640), # our input size can be 600px
            A.HorizontalFlip(p=0.3),
            A.VerticalFlip(p=0.3),
            A.RandomBrightnessContrast(p=0.1),
            A.ColorJitter(p=0.1),
            ToTensorV2()
        ], bbox_params=A.BboxParams(format='coco'))
    else:
        transform = A.Compose([
            A.Resize(600, 600), # our input size can be 600px
            ToTensorV2()
        ], bbox_params=A.BboxParams(format='coco'))
    return transform
class AquariumDetection(datasets.VisionDataset):
    def __init__(self, root, split='train', transform=None,
target_transform=None, transforms=None):
        # the 3 transform parameters are reuqired for datasets.VisionDataset
        super().__init__(root, transforms, transform, target_transform)
        self.split = split #train, valid, test
        self.coco = COCO(os.path.join(root, split, "_annotations.coco.json")) #
annotatiosn stored here
        self.ids = list(sorted(self.coco.imgs.keys()))
        self.ids = [id for id in self.ids if (len(self._load_target(id)) > 0)]

    def _load_image(self, id: int):
        path = self.coco.loadImgs(id)[0]['file_name']
        image = cv2.imread(os.path.join(self.root, self.split, path))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        return image
    def _load_target(self, id):
        return self.coco.loadAnns(self.coco.getAnnIds(id))

    def __getitem__(self, index):
        id = self.ids[index]
```

```
        image = self._load_image(id)
        target = self._load_target(id)
        target = copy.deepcopy(self._load_target(id))

        boxes = [t['bbox'] + [t['category_id']] for t in target] # required
annotation format for albumentations
        if self.transforms is not None:
            transformed = self.transforms(image=image, bboxes=boxes)

        image = transformed['image']
        boxes = transformed['bboxes']

        new_boxes = [] # convert from xywh to xyxy
        for box in boxes:
            xmin = box[0]
            xmax = xmin + box[2]
            ymin = box[1]
            ymax = ymin + box[3]
            new_boxes.append([xmin, ymin, xmax, ymax])

        boxes = torch.tensor(new_boxes, dtype=torch.float32)

        targ = {} # here is our transformed target
        targ['boxes'] = boxes
        targ['labels'] = torch.tensor([t['category_id'] for t in target],
dtype=torch.int64)
        targ['image_id'] = torch.tensor([t['image_id'] for t in target])
        targ['area'] = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:,
0]) # we have a different area
        targ['iscrowd'] = torch.tensor([t['iscrowd'] for t in target],
dtype=torch.int64)
        return image.div(255), targ # scale images
    def __len__(self):
        return len(self.ids)


#load classes
coco = COCO(os.path.join(dataset_path, "train", "_annotations.coco.json"))
categories = coco.cats
n_classes = len(categories.keys())
categories

classes = [i[1]['name'] for i in categories.items()]
classes

train_dataset = AquariumDetection(root=dataset_path,
transforms=get_transforms(True))

# Lets view a sample
sample = train_dataset[2]
img_int = torch.tensor(sample[0] * 255, dtype=torch.uint8)
plt.imshow(draw_bounding_boxes(
    img_int, sample[1]['boxes'], [classes[i] for i in sample[1]['labels']],
width=4
).permute(1, 2, 0))

# lets load the faster rcnn model
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.transforms import functional as F
model = fasterrcnn_resnet50_fpn(pretrained=True)
in_features = model.roi_heads.box_predictor.cls_score.in_features # we need to
    e head
    l.roi_heads.box_predictor =
    tection.faster_rcnn.FastRCNNPredictor(in_features, n_classes)

    collate_fn(batch):
    return tuple(zip(*batch))

    n_loader = DataLoader(train_dataset, batch_size=8, shuffle=True,
    rs=4, collate_fn=collate_fn)
```

```
130.
131. images,targets = next(iter(train_loader))
132. images = list(image for image in images)
133. targets = [{k:v for k, v in t.items()} for t in targets]
134. output = model(images, targets) # just make sure this runs without error
135.
136. device = torch.device("cuda") # use GPU to train
137. model = model.to(device)
138.
139. # Now, and optimizer
140. params = [p for p in model.parameters() if p.requires_grad]
141. optimizer = torch.optim.SGD(params, lr=0.0001, momentum=0.9, nesterov=True,
weight_decay=1e-4)
142. # lr_scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer,
milestones=[16, 22], gamma=0.1) # lr scheduler
143.
144. import sys
145. import math
146. import numpy as np
147. import pandas as pd
148. import matplotlib.pyplot as plt
149. from tqdm import tqdm
150. from pycocotools.coco import COCO
151. from pycocotools.cocoeval import COCOeval
152.
153. def train_one_epoch(model, optimizer, loader, device, epoch):
154.     model.to(device)
155.     model.train()
156.     all_losses = []
157.     all_losses_dict = []
158.
159.     for images, targets in tqdm(loader):
160.         images = list(image.to(device) for image in images)
161.         targets = [{k: torch.tensor(v).to(device) for k, v in t.items()} for t
in targets]
162.
163.         loss_dict = model(images, targets)
164.         losses = sum(loss for loss in loss_dict.values())
165.         loss_dict_append = {k: v.item() for k, v in loss_dict.items()}
166.         loss_value = losses.item()
167.
168.         all_losses.append(loss_value)
169.         all_losses_dict.append(loss_dict_append)
170.
171.         if not math.isfinite(loss_value):
172.             print(f"Loss is {loss_value}, stopping training")
173.             print(loss_dict)
174.             sys.exit(1)
175.
176.         optimizer.zero_grad()
177.         losses.backward()
178.         optimizer.step()
179.
180.     all_losses_dict = pd.DataFrame(all_losses_dict)
181.
182.     # Append average losses for the current epoch
183.     epoch_losses.append(np.mean(all_losses))
184.     epoch_loss_classifier.append(all_losses_dict['loss_classifier'].mean())
185.     epoch_loss_box.append(all_losses_dict['loss_box_reg'].mean())
186.     epoch_loss_rpn_box.append(all_losses_dict['loss_rpn_box_reg'].mean())
187.     epoch_loss_object.append(all_losses_dict['loss_objectness'].mean())
188.
     print("Epoch {}, lr: {:.6f}, loss: {:.6f}, loss_classifier: {:.6f},
     {:.6f}, loss_rpn_box: {:.6f}, loss_object: {:.6f}".format(
         epoch, optimizer.param_groups[0]['lr'], np.mean(all_losses),
         all_losses_dict['loss_classifier'].mean(),
         all_losses_dict['loss_box_reg'].mean(),
         all_losses_dict['loss_rpn_box_reg'].mean(),
         all_losses_dict['loss_objectness'].mean()
     )
```

```
# Lists to store average losses for each epoch
epoch_losses = []
epoch_loss_classifier = []
epoch_loss_box = []
epoch_loss_rpn_box = []
epoch_loss_object = []

num_epochs = 45

for epoch in range(num_epochs):
    train_one_epoch(model, optimizer, train_loader, device, epoch)

# Plotting the losses after training
plt.figure(figsize=(12, 8))
plt.plot(epoch_losses, label='Total Loss')
plt.plot(epoch_loss_classifier, label='Classifier Loss')
plt.plot(epoch_loss_box, label='Box Regression Loss')
plt.plot(epoch_loss_rpn_box, label='RPN Box Regression Loss')
plt.plot(epoch_loss_object, label='Objectness Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Losses')
plt.legend()
plt.grid(True)
plt.show()

# we will watch first epoich to ensure no errrors
# while it is training, lets write code to see the models predictions. lets try again
model.eval()
torch.cuda.empty_cache()
test_dataset = AquariumDetection(root=dataset_path, split="test", transforms=get_transforms(False))
img, _ = test_dataset[5]
img_int = torch.tensor(img*255, dtype=torch.uint8)
with torch.no_grad():
    prediction = model([img.to(device)])
    pred = prediction[0]

fig = plt.figure(figsize=(14, 10))
plt.imshow(draw_bounding_boxes(img_int,
    pred['boxes'][pred['scores'] > 0.8],
    [classes[i] for i in pred['labels'][pred['scores'] > 0.8].tolist()], width=4
).permute(1, 2, 0))

# Assuming 'model' is your pretrained model
# Save the entire model
torch.save(model.state_dict(), 'pretrainedfastervep45_resnet50_case1.pth')
```

Lampiran 3.3. *Testing* model YOLO

```
import os
from ultralytics import YOLO
import torch
import matplotlib.pyplot as plt
import numpy as np

# Function to calculate IoU
def calculate_iou(boxA, boxB):
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])
    intersection_area = max(0, xB - xA) * max(0, yB - yA)
    boxA_area = (boxA[2] - boxA[0]) * (boxA[3] - boxA[1])
    boxB_area = (boxB[2] - boxB[0]) * (boxB[3] - boxB[1])
    iou = intersection_area / float(boxA_area + boxB_area - intersection_area)
    return iou

# Function to parse ground truth files
def parse_ground_truth(txt_file):
    ground_truths = []
    with open(txt_file, 'r') as f:
        for line in f:
            parts = line.strip().split()
            class_id = int(parts[0])
            bbox = [float(part) for part in parts[1:]]
            ground_truths.append({'class_id': class_id, 'bbox': bbox})
    return ground_truths

# Function to generate confusion matrix
def generate_confusion_matrix(ground_truths, predictions, num_classes):
    confusion_matrix = np.zeros((num_classes, num_classes), dtype=int)
    for gt, pred in zip(ground_truths, predictions):
        confusion_matrix[int(gt), int(pred)] += 1  # Ensure indices are integers
    return confusion_matrix

# Load the YOLO model
model = YOLO('/content/drive/MyDrive/OpenDrive/best45epochyolo.pt')

# Retrieve class names from the model
class_names = model.names

# Create a dictionary to map class IDs to class names
class_id_to_name = {i: class_names[i] for i in range(len(class_names))}

# Paths to your images and ground truths
image_dir = '/content/drive/MyDrive/OpenDrive/testyolo/images'
gt_dir = '/content/drive/MyDrive/OpenDrive/testyolo/labels'

# Initialize counters
num_classes = len(class_names)
confusion_matrix = np.zeros((num_classes, num_classes), dtype=int)

# Process each image and its corresponding ground truth
for image_path in os.listdir(image_dir):
    full_image_path = os.path.join(image_dir, image_path)
    gt_path = os.path.join(gt_dir, os.path.splitext(image_path)[0] + '.txt')

    if not os.path.exists(gt_path):
        continue  # Skip if no corresponding ground truth file

    ound_truths = parse_ground_truth(gt_path)
    sults_list = model(full_image_path)  # Run model prediction

    Get predicted class IDs for each bounding box
    edictions = []
    r results in results_list:
```

```
68.         for _, pred_class, _ in zip(results.boxes.xyxy.cpu(),
results.boxes.cls.cpu(), results.boxes.conf.cpu()):
69.             predictions.append(pred_class.item())  # Convert tensor to scalar
70.
71.     # Map ground truth class IDs to model class IDs
72.     ground_truth_ids = [gt['class_id'] for gt in ground_truths]
73.
74.     # Update confusion matrix
75.     confusion_matrix += generate_confusion_matrix(ground_truth_ids, predictions,
num_classes)
76.
77. # Normalize confusion matrix
78. confusion_matrix_normalized = confusion_matrix.astype('float') /
confusion_matrix.sum(axis=1)[:, np.newaxis]
79.
80. plt.figure(figsize=(12, 10))  # Set the figure size to 12x10 inches
81. plt.imshow(confusion_matrix, interpolation='nearest', cmap=plt.cm.Blues)
82.
83. # Add normalized values to the confusion matrix plot
84. for i in range(num_classes):
85.     for j in range(num_classes):
86.         text_color = 'white' if confusion_matrix[i, j] > confusion_matrix.max()
/ 2 else 'black'
87.         plt.text(j, i, f'{confusion_matrix_normalized[i, j]:.2f}',
color=text_color, horizontalalignment='center', verticalalignment='center')
88.
89. plt.title('Confusion Matrix')
90. plt.colorbar()
91. plt.xticks(np.arange(num_classes), [class_id_to_name[i] for i in
range(num_classes)], rotation=90)
92. plt.yticks(np.arange(num_classes), [class_id_to_name[i] for i in
range(num_classes)])
93. plt.xlabel('Predicted Class')
94. plt.ylabel('Actual Class')
95. plt.tight_layout()
96. plt.show()
97.
```

Lampiran 3.4. *Testing* model Faster R-CNN

```
# Required libraries
import torch
import torchvision
from torchvision.models.detection import fasterrcnn_resnet50_fpn
import torchvision.transforms as T
from sklearn.metrics import precision_recall_fscore_support, accuracy_score
from PIL import Image
import json
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import os

# Setup device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Initialize the model with the architecture used during training
model = fasterrcnn_resnet50_fpn(pretrained=False)  # No pre-trained weights
model.to(device)

# Load the model directly from the checkpoint
checkpoint_path =
"/content/drive/MyDrive/OpenDrive/pretrained_modelfasterv2_45epoch_resnet50.pth"
model = torch.load(checkpoint_path, map_location=device)
model.eval()

# Define the transformation to be applied to images
def transform_image(image):
    transform = T.Compose([T.ToTensor()])
    return transform(image).unsqueeze(0)

# Function to perform prediction using the model
def get_predictions(image_path, model):
    image = Image.open(image_path).convert("RGB")
    image = transform_image(image).to(device)
    with torch.no_grad():
        preds = model(image)
    return preds

# Load annotations function
def load_annotations(json_path):
    with open(json_path, 'r') as f:
        annotations = json.load(f)
    return annotations

# Load annotations from the COCO format JSON file
annotations_path =
'/content/drive/MyDrive/OpenDrive/test/_annotations.coco.json'
with open(annotations_path, 'r') as f:
    coco_annotations = json.load(f)

# Extract categories mapping from category ID to category name
categories_mapping = {category['id']: category['name'] for category in
coco_annotations['categories']}

# Prepare for predictions and true labels
predicted_labels = []
true_labels = []
image_files = {image['file_name']: image['id'] for image in
coco_annotations['images']}

    lder path where images are stored
    er_path = '/content/drive/MyDrive/OpenDrive/test'

    erate over image files and match annotations
    image_file, image_id in image_files.items():
    image_path = os.path.join(folder_path, image_file)
    preds = get_predictions(image_path, model)
```

```
    # Assuming single highest-scored prediction
    predicted_label = preds[0]['labels'][0].item() if len(preds[0]['labels']) > 0 else None
    predicted_labels.append(predicted_label)

    # Find corresponding annotation using image_id
    annotation = next((ann for ann in coco_annotations['annotations'] if ann['image_id'] == image_id), None)
    if annotation:
        category_name = categories_mapping.get(annotation['category_id'])
        true_labels.append(category_name)  # Use the category name as the true label

# Compute and plot normalized confusion matrix
plt.figure(figsize=(13,13))
cm_normalized = confusion_matrix(true_labels, [categories_mapping.get(label) for label in predicted_labels if label in categories_mapping], labels=list(categories_mapping.values()), normalize='true')
sns.heatmap(cm_normalized, annot=True, fmt=".2f", cmap='Blues', xticklabels=categories_mapping.values(), yticklabels=categories_mapping.values())
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Convert predicted labels from IDs to category names, handling None values
predicted_label_names = [categories_mapping[label] if label in categories_mapping else None for label in predicted_labels]

# Filter None values from predictions and corresponding true labels for metrics calculation
filtered_predicted_labels = [pred for pred in predicted_label_names if pred is not None]
filtered_true_labels = [true for true, pred in zip(true_labels, predicted_label_names) if pred is not None]

# Calculate precision, recall, and F1-score
precision, recall, f1_score, _ = precision_recall_fscore_support(filtered_true_labels, filtered_predicted_labels, average='macro', zero_division=0)

# Calculate accuracy
accuracy = accuracy_score(filtered_true_labels, filtered_predicted_labels)

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1_score:.4f}")
print(f"Accuracy: {accuracy:.4f}")

```

Lampiran 3.5 Estimasi berat pada YOLO menggunakan data citra

```
import cv2
from ultralytics import YOLO
from google.colab.patches import cv2_imshow ## only for colab patches since if using imshow from cv2 wont work well. PLEASE REMOVE THIS IF U WANT TO RUN THIS OUTSIDE GOOGLE COLAB!
import os

# Load the YOLOv8 model
model = YOLO('/content/drive/MyDrive/OpenDrive/best45epochyolo.pt')

# Path to the folder containing images
folder_path = '/content/drive/MyDrive/OpenDrive/datatestweight/'

# List all files in the folder
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.jpeg', '.png'))]

# Process each image file
for image_file in image_files:
    # Load an image
    image_path = os.path.join(folder_path, image_file)
    image = cv2.imread(image_path)
    # Skala konversi dari piksel ke cm
    scale = 0.0075 # cm per piksel, sesuaikan dengan resolusi kamera atau citra yang digunakan.

    # The rest of your weight estimation functions and estimate_weight function remain the same
    weight_estimation_functions = {
    'anyperodon_leucogrammicus': lambda length_cm: 0.0069 * length_cm**3.16,
    'cephalopholis_argus': lambda length_cm: 0.0126 * length_cm**3.1,
    'cephalopholis_boenak': lambda length_cm: 0.0186 * length_cm**3.01,
    'cephalopholis_cyanostigma': lambda length_cm: 0.0257 * length_cm**2.92,
    'cephalopholis_miniata': lambda length_cm: 0.0112 * length_cm**3.10,
    'epinephelus_areolatus': lambda length_cm: 0.0129 * length_cm**3.02,
    'epinephelus_fasciatus': lambda length_cm: 0.0186 * length_cm**2.94,
    'epinephelus_faveatus': lambda length_cm: 0.0117 * length_cm**3.04,
    'epinephelus_ongus': lambda length_cm: 0.0178 * length_cm**3.01,
    'epinephelus_quoyanus': lambda length_cm: 0.0219 * length_cm**2.99,
    'lutjanus_bengalensis': lambda length_cm: 0.0027 * length_cm**3.22,
    'lutjanus_carponotatus': lambda length_cm: 0.0234 * length_cm**2.88,
    'lutjanus_decusstus': lambda length_cm: 0.0245 * length_cm**2.96,
    'lutjanus_fulviflamma': lambda length_cm: 0.0245 * length_cm**2.94,
    'lutjanus_gibbus': lambda length_cm: 0.0204 * length_cm**2.98,
    'lutjanus_lutjanus': lambda length_cm: 0.0229 * length_cm**2.91,
    'lutjanus_vittae': lambda length_cm: 0.0145 * length_cm**3.07,
    'variola_albimarginata': lambda length_cm: 0.0186 * length_cm**3.07
    }

    # Include the estimate_weight function
    def estimate_weight(class_name, length_cm):
        if class_name in weight_estimation_functions:
            weight_function = weight_estimation_functions[class_name]
            weight = weight_function(length_cm)
            print(f"Estimasi berat: {weight:.2f} gram")
            return weight
        else:
            print(f"Kelas tidak ditemukan!")
            return None

    # Set the confidence threshold
    onfidence_threshold = 0.5  # Set your desired confidence threshold here

    # Process the loaded image with YOLOv8 model
    esults = model(image, conf=confidence_threshold)

    # Process each detection
    or result in results:
```

```
64.         # Get bounding boxes
65.         boxes = result.boxes
66.         classes = result.names
67.         for i, box in enumerate(boxes):
68.             x1, y1, x2, y2 = map(int, box.xyxy[0])
69.             width_px = abs(x2 - x1)  # Calculate width in pixels
70.             height_px = abs(y2 - y1)  # Calculate height in pixels
71.
72.             # Convert dimensions from pixels to centimeters
73.             # Use the longer dimension for length estimation
74.             length_cm = max(width_px, height_px) * scale
75.
76.             # Get class name
77.             class_index = int(result.boxes.cls[i])
78.             class_name = classes[class_index]
79.
80.             # Estimate weight using the provided function
81.             estimated_weight = estimate_weight(class_name, length_cm=length_cm)
82.             print(f"panjang terdeteksi : {length_cm} cm")
83.
84.             # Draw bounding box and display class name and estimated weight
85.             cv2.rectangle(image, (x1,y1), (x2, y2), (0, 255, 0), 2)
86.             label = f"{class_name}: {estimated_weight:.2f}g" if estimated_weight
is not None else class_name
87.             cv2.putText(image, label, (x1, y2 + 20), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (255, 255, 255), 2)
88.
89.         plottedframe = results[0].plot()
90.         resizedplot = cv2.resize(plottedframe, (360, 640))
91.
92.     # Display the image with annotations
93.     cv2_imshow(resizedplot)
94.
95. # # Close the display window
96. # cv2.destroyAllWindows()
97.
```

Lampiran 3.6. Estimasi berat pada Faster R-CNN menggunakan data citra

```
import cv2
import torch
import torchvision
import torchvision.transforms as T
import numpy as np
from google.colab.patches import cv2_imshow
import os

# Load the custom pre-trained model
custom_model_weights =
"/content/drive/MyDrive/OpenDrive/pretrained_modelfastervep45statedict_resnet50.pth"
# Create the Faster R-CNN model with a ResNet-50 FPN backbone
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False)
# Modify the number of classes in the output layer
num_classes = 19
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor =
torchvision.models.detection.faster_rcnn.FastRCNNPredictor(in_features, num_classes)

# Load the model weights
model.load_state_dict(
    torch.load(custom_model_weights, map_location=torch.device('cuda' if
torch.cuda.is_available() else 'cpu')))
model.to('cuda' if torch.cuda.is_available() else 'cpu')
model.eval()

# Skala konversi dari piksel ke cm
scale = 0.0075 # cm per piksel, sesuaikan dengan resolusi kamera atau citra
yang digunakan.

# Path to the folder containing images
image_folder_path = "/content/drive/MyDrive/OpenDrive/datatestweight"

weight_estimation_functions = {
    'anyperodon_leucogrammicus': lambda length_cm: 0.0069 * length_cm ** 3.16,
    'cephalopholis_argus': lambda length_cm: 0.0126 * length_cm ** 3.1,
    'cephalopholis_boenak': lambda length_cm: 0.0186 * length_cm ** 3.01,
    'cephalopholis_cyanostigma': lambda length_cm: 0.0257 * length_cm ** 2.92,
    'cephalopholis_miniata': lambda length_cm: 0.0112 * length_cm ** 3.10,
    'epinephelus_areolatus': lambda length_cm: 0.0129 * length_cm ** 3.02,
    'epinephelus_fasciatus': lambda length_cm: 0.0186 * length_cm ** 2.94,
    'epinephelus_faveatus': lambda length_cm: 0.0117 * length_cm ** 3.04,
    'epinephelus_ongus': lambda length_cm: 0.0178 * length_cm ** 3.01,
    'epinephelus_quoyanus': lambda length_cm: 0.0219 * length_cm ** 2.99,
    'lutjanus_bengalensis': lambda length_cm: 0.0027 * length_cm ** 3.22,
    'lutjanus_carponotatus': lambda length_cm: 0.0234 * length_cm ** 2.88,
    'lutjanus_decusstus': lambda length_cm: 0.0245 * length_cm ** 2.96,
    'lutjanus_fulviflamma': lambda length_cm: 0.0245 * length_cm ** 2.94,
    'lutjanus_gibbus': lambda length_cm: 0.0204 * length_cm ** 2.98,
    'lutjanus_lutjanus': lambda length_cm: 0.0229 * length_cm ** 2.91,
    'lutjanus_vittae': lambda length_cm: 0.0145 * length_cm ** 3.07,
    'variola_albimarginata': lambda length_cm: 0.0186 * length_cm ** 3.07
}

def estimate_weight(class_name, length_cm):
    if class_name in weight_estimation_functions:
        weight_function = weight_estimation_functions[class_name]
        weight = weight_function(length_cm)
        print(f"Estimated weight: {weight:.2f} grams")
        return weight
    else:
        print("Class not found!")
        return None

ıfidence threshold for bounding boxes
ıdence_threshold = 0.5

ping from class IDs to class names
```

```
class_names = [
    'fish',
    'anyperodon_leucogrammicus',
    'cephalopholis_argus',
    'cephalopholis_boenak',
    'cephalopholis_cyanostigma',
    'cephalopholis_miniata',
    'epinephelus_areolatus',
    'epinephelus_fasciatus',
    'epinephelus_faveatus',
    'epinephelus_ongus',
    'epinephelus_quoyanus',
    'lutjanus_bengalensis',
    'lutjanus_carponotatus',
    'lutjanus_decusstus',
    'lutjanus_fulviflamma',
    'lutjanus_gibbus',
    'lutjanus_lutjanus',
    'lutjanus_vittae',
    'variola_albimarginata'
]

# Define the transformation to convert images to tensors
transform = T.Compose([T.ToTensor()])

# Loop through the images in the folder
for filename in os.listdir(image_folder_path):
    if filename.endswith(".jpg") or filename.endswith(".png"):
        # Read the image
        image_path = os.path.join(image_folder_path, filename)
        frame = cv2.imread(image_path)

        if frame is None:
            print(f"Error: Unable to read image '{image_path}'. Skipping...")
            continue

    # Convert the frame to a PyTorch tensor
    img_tensor = transform(frame).unsqueeze(0).to('cuda')

    # Run inference on the frame
    with torch.no_grad():
        outputs = model(img_tensor)

    # Process each detection
    for i, detection in enumerate(outputs[0]['boxes']):
        score = outputs[0]['scores'][i].cpu().numpy()

        if score < confidence_threshold:
            continue  # Skip detections below confidence threshold

        x1, y1, x2, y2 = map(int, detection.cpu().numpy())
        width_px = abs(x2 - x1)  # Calculate width in pixels
        height_px = abs(y2 - y1)  # Calculate height in pixels

        # Convert dimensions from pixels to centimeters
        width_cm = width_px * scale
        height_cm = height_px * scale
        if width_px > height_px:
            dimension = width_px * scale
        else:
            dimension = height_px * scale  # length checker which one is longer

        # Get predicted class and its confidence
        label = int(outputs[0]['labels'][i])
        class_name = class_names[label]
        confidence = f"Confidence: {score:.2f}"

        print(f"Detected object width: {width_cm:.2f} cm, height:
1:.2f} cm, Class: {class_name}, {confidence}")
        # Display estimated weight from the above function
        estimated_weight = estimate_weight(class_name, length_cm=dimension)
```

```
135.
136.         # Draw bounding box and label on the frame
137.         color = (0, 255, 0)  # Green color for the bounding box
138.         # Increase thickness for better visibility
139.         thickness = 3
140.         cv2.rectangle(frame, (x1, y1), (x2, y2), color, thickness)
141.         # Adjust font scale for larger text
142.         font_scale = 2
143.         cv2.putText(frame, f"{class_name} {confidence} Est. weight:
{estimated_weight:.2f} grams", (x1, y1 - 25),
144.                     cv2.FONT_HERSHEY_SIMPLEX, font_scale, color, thickness)
145.
146.         frame_resized = cv2.resize(frame, (360, 640))
147.
148.         # Display the annotated frame
149.         cv2_imshow(frame_resized)
150.         cv2.waitKey(0)
151.
152. # Close all OpenCV windows
153. cv2.destroyAllWindows()
154.
```

Lampiran 3.7 Estimasi berat pada model YOLO menggunakan data video

```
import cv2
from ultralytics import YOLO

# Load the YOLOv8 model
model = YOLO('E:/Extractivity/barang2jupyter/skripsiku/best45epochyolo.pt')

# Open the video file
cap = cv2.VideoCapture('VID_20240118_073121.mp4')
cap.set(3, 1920)
cap.set(4, 1080)

# Skala konversi dari piksel ke cm
scale = 0.01551  # cm per piksel, sesuaikan dengan resolusi kamera atau citra yang digunakan. ingat: panjang objek asli/panjang dalam piksel

weight_estimation_functions = {
 'anyperodon_leucogrammicus': lambda length_cm: 0.0069 * length_cm**3.16,
 'cephalopholis_argus': lambda length_cm: 0.0126 * length_cm**3.1,
 'cephalopholis_boenak': lambda length_cm: 0.0186 * length_cm**3.01,
 'cephalopholis_cyanostigma': lambda length_cm: 0.0257 * length_cm**2.92,
 'cephalopholis_miniata': lambda length_cm: 0.0112 * length_cm**3.10,
 'epinephelus_areolatus': lambda length_cm: 0.0129 * length_cm**3.02,
 'epinephelus_fasciatus': lambda length_cm: 0.0186 * length_cm**2.94,
 'epinephelus_faveatus': lambda length_cm: 0.0117 * length_cm**3.04,
 'epinephelus_ongus': lambda length_cm: 0.0178 * length_cm**3.01,
 'epinephelus_quoyanus': lambda length_cm: 0.0219 * length_cm**2.99,
 'lutjanus_bengalensis': lambda length_cm: 0.0027 * length_cm**3.22,
 'lutjanus_carponotatus': lambda length_cm: 0.0234 * length_cm**2.88,
 'lutjanus_decusstus': lambda length_cm: 0.0245 * length_cm**2.96,
 'lutjanus_fulviflamma': lambda length_cm: 0.0245 * length_cm**2.94,
 'lutjanus_gibbus': lambda length_cm: 0.0204 * length_cm**2.98,
 'lutjanus_lutjanus': lambda length_cm: 0.0229 * length_cm**2.91,
 'lutjanus_vittae': lambda length_cm: 0.0145 * length_cm**3.07,
 'variola_albimarginata': lambda length_cm: 0.0186 * length_cm**3.07
}

# function for weight estimation based on class name and length
def estimate_weight(class_name, length_cm):
    if class_name in weight_estimation_functions:
        weight_function = weight_estimation_functions[class_name]
        weight = weight_function(length_cm)
        print(f"estimasi berat : {weight:.2f} gram")
        return weight
    else:
        print(f"Kelas tidak ditemukan!")
        return None

# Set the confidence threshold
confidence_threshold = 0.5  # Set your desired confidence threshold here

# Loop through the video frames
while cap.isOpened():
    # Read a frame from the video
    success, frame = cap.read()

    if success:
        # Run YOLOv8 inference on the frame
        # frame=cv2.resize(frame, (1920, 1080)
        results = model(frame, conf=confidence_threshold)

        # Process each detection
        for result in results:
            # Get bounding boxes
            boxes = result.boxes
            classes = result.names
            for i, box in enumerate(boxes):
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                width_px = abs(x2 - x1)  # Calculate width in pixels
```

```
                height_px = abs(y2 - y1)  # Calculate height in pixels

                # # Convert dimensions from pixels to centimeters
                # width_cm = width_px * scale
                # height_cm = height_px * scale
                # Check if width is longer than height
                if width_px > height_px:
                    dimension = width_px * scale
                else:
                    dimension = height_px * scale

                # print(f"Detected object width: {width_px:.2f} cm, height: {height_px:.2f} cm")
                # print(f"Detected object width: {width_cm:.2f} cm, height: {height_cm:.2f} cm")

                #printing class names
                class_index = int(result.boxes.cls[i])
                class_name = classes[class_index]
                # print(f"{class_name}")

                #tampilkan estimasi berat from the above function
                estimated_weight = estimate_weight(class_name, length_cm=dimension)
                print(f"panjang terdeteksi : {dimension} cm")

                #gambar bbox utk berat
                cv2.rectangle(frame, (x1,y1), (x2, y2), (0, 255, 0), 2)

                #display class name and  the weight, well see
                label = f"{class_name}: {estimated_weight:.2f}g" if estimated_weight is not None else class_name
                full_label = cv2.putText(frame, label, (x1, y2 + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

            # Visualize the results on the frame
            annotated_frame = results[0].plot()
            annotated_frame = cv2.resize(annotated_frame, (360, 640))


            # Display the annotated frame
            cv2.imshow('YOLO detector', annotated_frame) # PLEASE CHANGE THIS TO cv2.imshow('Detection', annotated_frame) if u want to run this notebook in another platform!

            # Break the loop if 'q' is pressed
            if cv2.waitKey(1) & 0xFF == ord("q"):
                break
        else:
            # Break the loop if the end of the video is reached
            break

# Release the video capture object and close the display window
cap.release()
cv2.destroyAllWindows()

```

Lampiran 3.8. Estimasi berat pada model Faster R-CNN menggunakan data video

```
import cv2
import torch
import torchvision
import torchvision.transforms as T
import numpy as np

# Load the custom pre-trained model
custom_model_weights = "pretrained_modelfastervep45statedict_resnet50.pth "
# Create the Faster R-CNN model with a ResNet-50 FPN backbone
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False)
# Modify the number of classes in the output layer
num_classes = 19
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor =
torchvision.models.detection.faster_rcnn.FastRCNNPredictor(in_features, num_classes)

# Load the model weights
model.load_state_dict(
    torch.load(custom_model_weights, map_location=torch.device('cuda' if
torch.cuda.is_available() else 'cpu')))
model.to('cuda' if torch.cuda.is_available() else 'cpu')
model.eval()

# Open the video file
cap = cv2.VideoCapture('VID_20231203_102209.mp4')
# Scale conversion from pixels to cm
scale = 0.01551  # cm per pixel, adjust according to the camera resolution or
image used.

weight_estimation_functions = {
 'anyperodon_leucogrammicus': lambda length_cm: 0.0069 * length_cm**3.16,
 'cephalopholis_argus': lambda length_cm: 0.0126 * length_cm**3.1,
 'cephalopholis_boenak': lambda length_cm: 0.0186 * length_cm**3.01,
 'cephalopholis_cyanostigma': lambda length_cm: 0.0257 * length_cm**2.92,
 'cephalopholis_miniata': lambda length_cm: 0.0112 * length_cm**3.10,
 'epinephelus_areolatus': lambda length_cm: 0.0129 * length_cm**3.02,
 'epinephelus_fasciatus': lambda length_cm: 0.0186 * length_cm**2.94,
 'epinephelus_faveatus': lambda length_cm: 0.0117 * length_cm**3.04,
 'epinephelus_ongus': lambda length_cm: 0.0178 * length_cm**3.01,
 'epinephelus_quoyanus': lambda length_cm: 0.0219 * length_cm**2.99,
 'lutjanus_bengalensis': lambda length_cm: 0.0027 * length_cm**3.22,
 'lutjanus_carponotatus': lambda length_cm: 0.0234 * length_cm**2.88,
 'lutjanus_decusstus': lambda length_cm: 0.0245 * length_cm**2.96,
 'lutjanus_fulviflamma': lambda length_cm: 0.0245 * length_cm**2.94,
 'lutjanus_gibbus': lambda length_cm: 0.0204 * length_cm**2.98,
 'lutjanus_lutjanus': lambda length_cm: 0.0229 * length_cm**2.91,
 'lutjanus_vittae': lambda length_cm: 0.0145 * length_cm**3.07,
 'variola_albimarginata': lambda length_cm: 0.0186 * length_cm**3.07
}
def estimate_weight(class_name, length_cm):
    if class_name in weight_estimation_functions:
        weight_function = weight_estimation_functions[class_name]
        weight = weight_function(length_cm)
        print(f"estimasi berat : {weight:.2f} gram")
        return weight
    else:
        print(f"Kelas tidak ditemukan!")
        return None

# Set GPU memory limit (adjust as needed)
    memory_limit = 2048  # in megabytes

    fine the transformation to convert images to tensors
    sform = T.Compose([T.ToTensor()])

    nfidence threshold for bounding boxes
    idence_threshold = 0.5
```

```
# Mapping from class IDs to class names
class_names = [
    'fish',
 'anyperodon_leucogrammicus',
 'cephalopholis_argus',
 'cephalopholis_boenak',
 'cephalopholis_cyanostigma',
 'cephalopholis_miniata',
 'epinephelus_areolatus',
 'epinephelus_fasciatus',
 'epinephelus_faveatus',
 'epinephelus_ongus',
 'epinephelus_quoyanus',
 'lutjanus_bengalensis',
 'lutjanus_carponotatus',
 'lutjanus_decusstus',
 'lutjanus_fulviflamma',
 'lutjanus_gibbus',
 'lutjanus_lutjanus',
 'lutjanus_vittae',
 'variola_albimarginata'
]

# Loop through the video frames
while cap.isOpened():
    # Read a frame from the video
    success, frame = cap.read()

    if success:
        # Check GPU memory usage
        gpu_memory_used = torch.cuda.memory_allocated(0) / (1024 ** 2)  # in
megabytes

        # Reduce batch size or take other actions if GPU memory usage is high
        if gpu_memory_used > gpu_memory_limit:
            # You may adjust the GPU memory limit handling logic here
            print(f"High GPU memory usage: {gpu_memory_used:.2f} MB")

        # Convert the frame to a PyTorch tensor
        img_tensor = transform(frame).unsqueeze(0).to('cuda')

        # Run inference on the frame
        with torch.no_grad():
            outputs = model(img_tensor)

        # Process each detection
        for i, detection in enumerate(outputs[0]['boxes']):
            score = outputs[0]['scores'][i].cpu().numpy()

            if score < confidence_threshold:
                continue  # Skip detections below confidence threshold

            x1, y1, x2, y2 = map(int, detection.cpu().numpy())
            width_px = abs(x2 - x1)  # Calculate width in pixels
            height_px = abs(y2 - y1)  # Calculate height in pixels

            # Convert dimensions from pixels to centimeters
            width_cm = width_px * scale
            height_cm = height_px * scale
            if width_px > height_px:
                dimension = width_px * scale
            else:
                dimension = height_px * scale # length checker which one is


            # Get predicted class and its confidence
            label = int(outputs[0]['labels'][i])
            class_name = class_names[label]
            confidence = f"Confidence: {score:.2f}"
```

```
134.             print(f"Detected object width: {width_cm:.2f} cm, height:
{height_cm:.2f} cm, {class_name}, {confidence}")
135.             # tampilkan estimasi berat from the above function
136.             estimated_weight = estimate_weight(class_name, length_cm=dimension)
137.
138.             # Draw bounding box and label on the frame
139.             color = (0, 255, 0)  # Green color for the bounding box
140.             cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
141.             cv2.putText(frame, f"{class_name} {confidence} es.berat
{estimated_weight:.2f} gram", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color,
2)
142.             frameresize= cv2.resize(frame, (360, 640))
143.
144.         # Display the annotated frame
145.             cv2.imshow('Detection in Faster R-CNN', frameresize)
146.
147.         # Break the loop if 'q' is pressed
148.         if cv2.waitKey(1) & 0xFF == ord("q"):
149.             break
150.     else:
151.         # Break the loop if the end of the video is reached
152.         break
153.
154. # Release the video capture object and close the display window
155. cap.release()
156. cv2.destroyAllWindows()
```

Lampiran 4. Tabel *Box Loss* dan *Classification Loss* pada Model YOLO dan Faster R-CNN

Epoch	***Loss***			
	Box Loss		***Classification Loss***	
	YOLO	**Faster R-CNN**	**YOLO**	**Faster R-CNN**
1	1,10	0,139013	4,436	0,205246
2	0,9382	0,131558	3,081	0,134915
3	0,8515	0,130052	2,501	0,130785
4	0,7859	0,127797	1,95	0,129190
5	0,7586	0,123144	1,602	0,130356
6	0,7317	0,111171	1,366	0,127249
7	0,7231	0,096346	1,203	0,117696
8	0,7052	0,082856	1,071	0,102768
9	0,7064	0,073428	0,9949	0,088792
10	0,6857	0,067168	0,9284	0,078540
11	0,681	0,062067	0,8704	0,069796
12	0,6688	0,058647	0,8229	0,064626
13	0,6742	0,055398	0,7953	0,059548
14	0,6662	0,052574	0,7586	0,055357
15	0,6555	0,050633	0,7388	0,051950
16	0,6583	0,049069	0,72	0,049294
17	0,6534	0,047431	0,6951	0,046961
18	0,6469	0,045680	0,6786	0,044895
19	0,6452	0,044187	0,6593	0,043047
20	0,6423	0,043251	0,6598	0,041480
21	0,6378	0,041959	0,636	0,039579
22	0,6393	0,041244	0,6346	0,038721
23	0,6425	0,040237	0,6243	0,037331
24	0,6383	0,039671	0,6157	0,037357
[illegible]	0,6369	0,038934	0,6102	0,035578
	0,6287	0,038074	0,5887	0,034927
	0,6346	0,037444	0,5935	0,033563
	0,63	0,036823	0,5861	0,032576

Epoch	*Loss*			
	Box Loss		*Classification Loss*	
	YOLO	**Faster R-CNN**	**YOLO**	**Faster R-CNN**
29	0,6242	0,036139	0,57	0,031799
30	0,6295	0,035812	0,5735	0,031489
31	0,6227	0,035009	0,5734	0,030757
32	0,6317	0,034576	0,5677	0,030165
33	0,6302	0,034223	0,5664	0,029154
34	0,6258	0,033664	0,5525	0,028514
35	0,6182	0,033060	0,5503	0,028647
36	0,4812	0,032803	0,4739	0,028116
37	0,4711	0,032188	0,4416	0,027299
38	0,4691	0,031809	0,4216	0,027079
39	0,4659	0,031675	0,4182	0,026634
40	0,4656	0,031135	0,4088	0,025951
41	0,4595	0,030904	0,4048	0,025899
42	0,4595	0,030387	0,3973	0,025351
43	0,4597	0,030144	0,3977	0,025206
44	0,4556	0,029852	0,4004	0,024277
45	0,4571	0,029612	0,3922	0,024550

Lampiran 5. Lembar perbaikan skripsi

LEMBAR PERBAIKAN SKRIPSI

"ANALISIS PERBANDINGAN KINERJA YOLO DAN FASTER R-CNN PADA DETEKSI JENIS DAN ESTIMASI BERAT IKAN KERAPU DAN IKAN KAKAP MENGGUNAKAN CITRA DIGITAL"

OLEH:

SABDA ANSARI BAKE
D121191037

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 14 Juni 2024.

Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Prof. Dr. Ir. Indrabayu, S.T., M.T., M.Bus.Sys., IPM,ASEAN. Eng.	
Sekretaris	Elly Warni, S.T., M.T.	
Anggota	Prof. Dr. Eng. Ir. Intan Sari Areni, S.T., M.T., IPU	
	Dr. Eng. Zulkifli Tahir, S.T., M.Sc.	

Persetujuan Perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Prof. Dr. Ir. Indrabayu, S.T., M.T., M.Bus.Sys., IPM,ASEAN. Eng.	
II	Elly Warni, S.T., M.T.	