

Lisa I

Tehisintellekti kasutati tulemuste logimise parandamiseks, treenitud mudelite mõõdikute arvutamiseks, mudelite ennustuste visualiseerimiseks ja märgendite konverteerimiseks.

ChatGPT abil Faster R-CNN programmi arendamine

I Sisend (prompt):

Let's say I have trained 20 epochs. How to measure each model F1 score, accuracy, precision and recall?

I have this code:

```
import torch
import torchvision
from torch.utils.data import DataLoader
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
from torchvision.datasets import CocoDetection
from torchvision import transforms
from torchvision.transforms import functional as F
import matplotlib.pyplot as plt
from PIL import Image

#Define transformations
class CocoTransform:
    def __call__(self, image, target):
        image = F.to_tensor(image) # Convert PIL image to tensor
        return image, target

#dokumenteerimiseks
f=open("katsetus_3veebr2025.txt","a")

# Dataset class
def get_coco_dataset(img_dir, ann_file):
    return CocoDetection(
        root=img_dir,
        annFile=ann_file,
        transforms=CocoTransform()
    )

# Load datasets
train_dataset = get_coco_dataset(
    #img_dir="dataset/train",
    #ann_file="dataset/train/annotations/benthos_train_coco.json" #NB! siin on polügonid.
    img_dir="bbox vs polygon/polygon/dataset/train",
    ann_file="bbox vs
polygon/polygon/dataset/train/annotations/polygon_train_test_coco.json"
```

```

)
f.write("---")
f.write("""katsetus 40 pildiga, ainult nematoodid, keriloomad, polygons annoteeringud.
img_dir="bbox vs polygon/polygon/dataset/train",
    ann_file="bbox vs
polygon/polygon/dataset/train/annotations/polygon_train_test_coco.json" """)

val_dataset = get_coco_dataset(
    #img_dir="dataset/val",
    #ann_file="dataset/val/annotations/benthos_validate_coco.json" #NB! siin on polügonid.
    img_dir="bbox vs polygon/polygon/dataset/val",
    ann_file="bbox vs polygon/polygon/dataset/val/annotations/polygon_val_test_coco.json"
)

# DataLoader
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True, collate_fn=lambda x:
tuple(zip(*x)))
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False, collate_fn=lambda x:
tuple(zip(*x)))

# Load Faster R-CNN with ResNet-50 backbone
def get_model(num_classes):
    # Load pre-trained Faster R-CNN
    #model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
#esiagline rida
    weights = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=weights)

    # Get the number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features # we use this
    pretrained model to extract features

    # Replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes) #
    necessary to use with custom dataset
    return model

# Initialize the model
num_classes = 6 # Background + rotifer + testate amoeba + nematode + tardigrade +
unknown_animal
model = get_model(num_classes)

# Move model to GPU if available
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model.to(device)

# Define optimizer and learning rate scheduler
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)

```

```

def train_one_epoch(model, optimizer, data_loader, device, epoch):
    model.train() #set the model to train model
    for images, targets in data_loader: #target means bounding boxes and class labels
        (annotations)
        # Move images to the device, images and model need to be on the same device
        images = [img.to(device) for img in images]

        # Validate and process targets
        processed_targets = []
        valid_images = []
        for i, target in enumerate(targets):
            boxes = []
            labels = []
            for obj in target:
                # Extract bbox
                bbox = obj["bbox"] # Format: [x, y, width, height]
                x, y, w, h = bbox

                # Ensure the width and height are positive
                if w > 0 and h > 0:
                    boxes.append([x, y, x + w, y + h]) # Convert to [x_min, y_min, x_max, y_max]
                    labels.append(obj["category_id"])

            # Only process if there are valid boxes
            if boxes:
                processed_target = {
                    "boxes": torch.tensor(boxes, dtype=torch.float32).to(device),
                    "labels": torch.tensor(labels, dtype=torch.int64).to(device),
                }
                processed_targets.append(processed_target)
                valid_images.append(images[i]) # Add only valid images

        # Skip iteration if no valid targets
        if not processed_targets:
            continue

        # Ensure images and targets are aligned
        images = valid_images

        # Forward pass
        loss_dict = model(images, processed_targets)
        losses = sum(loss for loss in loss_dict.values())

        # Backpropagation
        optimizer.zero_grad()
        losses.backward()
        optimizer.step()

    print(f"Epoch [{epoch}] Loss: {losses.item():.4f}")

```

```

f.write(f'Epoch [{epoch}] Loss: {losses.item():.4f}; ')

# Training loop
num_epochs = 20 # could be much larger number
for epoch in range(num_epochs):
    train_one_epoch(model, optimizer, train_loader, device, epoch)
    lr_scheduler.step()

    # Save the model's state dictionary after every epoch
    model_path = f'fasterrcnn_resnet50_epoch_{epoch + 1}.pth; '
    torch.save(model.state_dict(), model_path)
    print(f'Model saved: {model_path}')
    f.write(f'Model saved: {model_path}')

f.close()

```

I Vastus:

To evaluate each trained Faster R-CNN model's performance (F1-score, precision, recall, and accuracy), you need to compare the predicted bounding boxes with the ground-truth boxes in your validation dataset.

Steps to Evaluate the Model

Run Inference on the Validation Set

Use your trained model to predict bounding boxes and class labels.

Compute Precision, Recall, and F1-Score

Use Intersection over Union (IoU) to determine if a prediction is a true positive (TP) or false positive (FP).

Calculate precision, recall, and F1-score using TP, FP, and false negatives (FN).

Compute mAP (mean Average Precision)

You can use the COCO evaluation metrics to compute mAP, which is commonly used in object detection.

1. Load a Trained Model

Load a specific model checkpoint for evaluation:

```

python
import torch
import torchvision
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
from torchmetrics.detection.mean_ap import MeanAveragePrecision
from torchvision.transforms import functional as F

# Load trained model
def load_model(model_path, num_classes, device):
    model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=FasterRCNN_Res
Net50_FPN_Weights.DEFAULT)
    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)

```

```

        model.load_state_dict(torch.load(model_path, map_location=device))
        model.to(device)
        model.eval()
        return model

# Specify the checkpoint and device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_classes = 6 # Adjust based on your dataset
model_path = "fasterrcnn_resnet50_epoch_20.pth" # Change this to evaluate
different epochs
model = load_model(model_path, num_classes, device)

```

2. Evaluate the Model on Validation Data

Create a function to evaluate the model:

```

python
Copy
Edit
from torchmetrics.detection import MeanAveragePrecision
import numpy as np

# Initialize evaluation metric
metric = MeanAveragePrecision(iou_thresholds=[0.5], class_metrics=True)

# Function to evaluate model
def evaluate_model(model, data_loader, device):
    model.eval()
    metric.reset()

    with torch.no_grad():
        for images, targets in data_loader:
            images = [img.to(device) for img in images]

            # Get predictions
            predictions = model(images)

            # Convert predictions to expected format
            pred_boxes = [pred["boxes"].cpu() for pred in predictions]
            pred_scores = [pred["scores"].cpu() for pred in predictions]
            pred_labels = [pred["labels"].cpu() for pred in predictions]

            target_boxes = [t["boxes"].cpu() for t in targets]
            target_labels = [t["labels"].cpu() for t in targets]

            preds = [
                {"boxes": pred_boxes[i], "scores": pred_scores[i],
"labels": pred_labels[i]}
                for i in range(len(predictions))
            ]
            targets = [
                {"boxes": target_boxes[i], "labels": target_labels[i]}

```

```

        for i in range(len(targets))
    ]

    # Update metric
    metric.update(preds, targets)

# Compute final scores
result = metric.compute()
return result

```

3. Run Evaluation

Run evaluation for a specific trained model:

python

Copy

Edit

```

results = evaluate_model(model, val_loader, device)

# Extract metrics
mAP_50 = results["map_50"].item()
precision = results["map"].item() # Average precision across IoU
thresholds
recall = results["mar_100"].item() # Maximum recall with 100 detections
per image
f1_score = 2 * (precision * recall) / (precision + recall + 1e-7) #
F1-score

print(f"mAP@0.5: {mAP_50:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1_score:.4f}")

```

4. Save and Compare Across Epochs

To compare across multiple epochs, modify your script to iterate through all saved models:

python

Copy

Edit

```

epoch_results = []

for epoch in range(1, 21): # Iterate through 20 epochs
    model_path = f"fasterrcnn_resnet50_epoch_{epoch}.pth"
    model = load_model(model_path, num_classes, device)

    results = evaluate_model(model, val_loader, device)
    precision = results["map"].item()
    recall = results["mar_100"].item()
    f1_score = 2 * (precision * recall) / (precision + recall + 1e-7)

    epoch_results.append((epoch, precision, recall, f1_score))

```

```
# Print results for all epochs
for epoch, prec, rec, f1 in epoch_results:
    print(f"Epoch {epoch}: Precision={prec:.4f}, Recall={rec:.4f},
F1-Score={f1:.4f}")
```

Summary of Metrics


mAP@0.5: Measures model performance at IoU=0.5 (higher is better).

Precision: Measures how many detected objects were correct.

Recall: Measures how many ground-truth objects were correctly detected.

F1-score: Harmonic mean of precision and recall.

This method will let you compare models across different epochs and pick the best one based on F1-score or mAP.

Would you like help visualizing results (e.g., precision-recall curves)? 

II Sisend (prompt): Please provide data_loader code as well.

II Vastus:

Sure! Below is the complete data_loader code that you can use for loading your validation dataset. This includes dataset transformation, annotation parsing, and DataLoader setup.

1. Define COCO Dataset with Transformations

Since COCO format contains annotations, we need to ensure that images and targets are preprocessed correctly.

```
python
Copy
Edit
import torch
from torch.utils.data import DataLoader
from torchvision.datasets import CocoDetection
from torchvision import transforms
from torchvision.transforms import functional as F

# Custom transformation class
class CocoTransform:
    def __call__(self, image, target):
        image = F.to_tensor(image) # Convert PIL image to tensor
        return image, target

# Function to create COCO dataset
def get_coco_dataset(img_dir, ann_file):
    return CocoDetection(
        root=img_dir,
        annFile=ann_file,
        transforms=CocoTransform())
```

2. Initialize Training and Validation DataLoader

Now, use the dataset to create DataLoaders for training and validation.

```
python
```

Copy

Edit

```
# Define paths to datasets
train_img_dir = "bbox vs polygon/polygon/dataset/train"
train_ann_file = "bbox vs
polygon/polygon/dataset/train/annotations/polygon_train_test_coco.json"

val_img_dir = "bbox vs polygon/polygon/dataset/val"
val_ann_file = "bbox vs
polygon/polygon/dataset/val/annotations/polygon_val_test_coco.json"

# Load datasets
train_dataset = get_coco_dataset(train_img_dir, train_ann_file)
val_dataset = get_coco_dataset(val_img_dir, val_ann_file)

# Custom collate function to handle variable-length annotations
def collate_fn(batch):
    return tuple(zip(*batch))

# Create DataLoaders
train_loader = DataLoader(
    train_dataset,
    batch_size=4, # Adjust based on GPU memory
    shuffle=True,
    num_workers=4, # Use multiple workers to speed up loading
    collate_fn=collate_fn)

val_loader = DataLoader(
    val_dataset,
    batch_size=4,
    shuffle=False, # No need to shuffle validation data
    num_workers=4,
    collate_fn=collate_fn)
```

3. Explanation

CocoDetection: Loads images and annotations in COCO format.

CocoTransform: Converts PIL images into PyTorch tensors.

collate_fn: Ensures that images and targets are packed correctly in the batch.

Training DataLoader (train_loader)

Uses shuffle=True to randomize batches.

Validation DataLoader (val_loader)

Uses shuffle=False to maintain consistency.

Now, you can pass val_loader into the evaluation function to compute precision, recall, and F1-score:

python

Copy

Edit

```
results = evaluate_model(model, val_loader, device)
print(results)
```

Would you like to visualize predictions on sample images as well? 🎨📊

III Sisend (prompt): Yes

III Vastus:

Great! Below is the code to visualize model predictions on sample images from your validation set. This will help you inspect how well the model detects objects.

1. Load Required Libraries

python

Copy

Edit

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import random
import torchvision.transforms.functional as F
```

2. Define Visualization Function

This function will:

Load an image from val_loader

Run inference using the trained model

Overlay bounding boxes and labels on the image

Display ground-truth vs. predicted bounding boxes

python

Copy

Edit

```
# Define class names (adjust according to your dataset)
class_names = {0: "background", 1: "rotifer", 2: "testate_amoeba", 3:
"nematode", 4: "tardigrade", 5: "unknown_animal"}

def visualize_predictions(model, data_loader, device, num_images=3,
score_threshold=0.5):
    model.eval() # Set model to evaluation mode

    # Get a batch of images and targets from DataLoader
    images, targets = next(iter(data_loader))
    images = list(img.to(device) for img in images)

    # Run inference
    with torch.no_grad():
        predictions = model(images)

    # Plot results
    for i in range(min(num_images, len(images))):
        img = images[i].cpu().permute(1, 2, 0).numpy() # Convert tensor
image to numpy
        fig, ax = plt.subplots(1, figsize=(8, 8))
        ax.imshow(img)

        # Plot ground-truth bounding boxes (blue)
        for j, box in enumerate(targets[i]["boxes"]):
            x_min, y_min, x_max, y_max = box.tolist()
```

```

        rect = patches.Rectangle(
            (x_min, y_min), x_max - x_min, y_max - y_min,
            linewidth=2, edgecolor='blue', facecolor='none'
        )
        ax.add_patch(rect)
        ax.text(x_min, y_min - 5, f"GT:
{class_names[target[i]['labels'][j].item()})", color='blue', fontsize=10)

    # Plot predicted bounding boxes (red) with score filter
    for j, (box, score, label) in
enumerate(zip(predictions[i]["boxes"], predictions[i]["scores"],
predictions[i]["labels"])):
        if score > score_threshold: # Only display confident
detections
            x_min, y_min, x_max, y_max = box.tolist()
            rect = patches.Rectangle(
                (x_min, y_min), x_max - x_min, y_max - y_min,
                linewidth=2, edgecolor='red', facecolor='none'
            )
            ax.add_patch(rect)
            ax.text(x_min, y_min - 5, f"{class_names[label.item()]}:
{score:.2f}", color='red', fontsize=10)

    ax.set_title("Ground Truth (Blue) vs Predictions (Red)")
    plt.show()

```

Run visualization on validation set
visualize_predictions(model, val_loader, device)

3. Explanation

Ground-truth boxes (from dataset) are drawn in blue.

Predicted boxes (from model) are drawn in red.

Only predictions with a confidence above 0.5 are displayed.

Labels and confidence scores are overlaid on predictions.

4. Sample Output

This will generate images like:

 Image Example

markdown

Copy

Edit

```

-----
| GT: nematode (Blue)          |
| Model: nematode (Red, 0.92 confidence) |
-----

```

You will see the original image with bounding boxes, making it easy to compare ground truth vs. model output.

This visualization helps in debugging and understanding how well the model is detecting objects.

IV Sisend (prompt): Now let's continue with visualizing predictions. I have a number of test images in test_image folder. I would like to test all these and save predictions with bounding boxes to sub-folder test_image_results.

IV Vastus:

Great! Let's create a script that will:

Load your trained Faster R-CNN model
Apply it to all images in the test_image folder
Draw bounding boxes with class labels on detected objects
Save the visualized predictions in test_image_results



Steps to Implement

Load the trained model
Preprocess test images (convert to tensor, normalize)
Run inference using Faster R-CNN
Draw bounding boxes with labels on the images
Save the images in test_image_results



Python Script: visualize_predictions.py

python

Copy

Edit

```
import torch
import torchvision.transforms as transforms
import torchvision
import os
import cv2
import numpy as np
from PIL import Image, ImageDraw, ImageFont
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights

# Define paths
TEST_IMAGE_FOLDER = "test_image"
RESULTS_FOLDER = "test_image_results"
MODEL_PATH = "fasterrcnn_resnet50_epoch_20.pth" # Update this with your
trained model path

# Ensure output folder exists
os.makedirs(RESULTS_FOLDER, exist_ok=True)

# Load trained model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
weights = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=weights)
num_classes = 6 # Update based on your dataset
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 trained weights
model.load_state_dict(torch.load(MODEL_PATH, map_location=device))
model.to(device)
model.eval()

# Define class labels (update according to your dataset)
CLASS_NAMES = ["background", "rotifer", "testate_amoeba", "nematode",
"tardigrade", "unknown_animal"]

# Define image transformation
transform = transforms.Compose([transforms.ToTensor()])

# Function to draw bounding boxes
def draw_boxes(image, boxes, labels, scores, threshold=0.5):
    draw = ImageDraw.Draw(image)
    for box, label, score in zip(boxes, labels, scores):
        if score > threshold:
            x1, y1, x2, y2 = box
            draw.rectangle([x1, y1, x2, y2], outline="red", width=3)
            text = f"{CLASS_NAMES[label]}: {score:.2f}"
            draw.text((x1, y1), text, fill="yellow")
    return image

# Process all test images
for image_name in os.listdir(TEST_IMAGE_FOLDER):
    image_path = os.path.join(TEST_IMAGE_FOLDER, image_name)
    image = Image.open(image_path).convert("RGB")

    # Preprocess image
    image_tensor = transform(image).unsqueeze(0).to(device)

    # Perform inference
    with torch.no_grad():
        prediction = model(image_tensor)

    # Extract results
    boxes = prediction[0]["boxes"].cpu().numpy()
    labels = prediction[0]["labels"].cpu().numpy()
    scores = prediction[0]["scores"].cpu().numpy()

    # Draw bounding boxes
    image_with_boxes = draw_boxes(image, boxes, labels, scores,
threshold=0.5)

    # Save output
    result_path = os.path.join(RESULTS_FOLDER, image_name)
    image_with_boxes.save(result_path)
    print(f"Saved: {result_path}")

```

```
print("✅ All predictions saved in 'test_image_results'!")
```

📁 Expected Folder Structure

bash

Copy

Edit

/your_project

```
├── dataset/
├── test_image/ ✅ (Your test images go here)
├── test_image_results/ ✅ (Predictions will be saved here)
├── models/
├── train_model.py
├── evaluate_model.py
└── visualize_predictions.py ✅ (New script!)
```

🚀 Running the Script

Simply execute:

bash

Copy

Edit

```
python visualize_predictions.py
```

And all test images in test_image will be processed and saved in test_image_results.

✅ Features of This Script

- ✓ Loads trained Faster R-CNN
- ✓ Reads all test images
- ✓ Runs inference and detects objects
- ✓ Draws bounding boxes & labels
- ✓ Saves results in test_image_results

Would you like any enhancements, such as color-coding different classes? 😊

V Sisend (prompt): I would like to evaluate multiple models. All models that needs to be evaluated are in folder Models_for_eval.

V Vastus:

Got it! You want to evaluate multiple Faster R-CNN models stored in the "Models_for_eval" folder. I'll modify the code to:

- 1) Automatically detect all model files in the folder (.pth files).
- 2) Load and evaluate each model on the validation dataset.
- 3) Log results for each model separately in a structured format.

Would you like to save results in:

Separate log files for each model (e.g., evaluation_log_model1.txt, evaluation_log_model2.txt)

A single consolidated CSV/JSON file for easy comparison?

Let me know your preference!

VI Sisend (prompt): Let's save results in a single CSV file.

VI Vastus:

Great choice! I'll modify the code to:

- ✓ Iterate through all .pth model files in the "Models_for_eval" folder.
- ✓ Evaluate each model using the same dataset.
- ✓ Save results in a single CSV file (evaluation_results.csv) with columns:

Model Name

Precision

Recall

F1-score

Accuracy

Evaluation Date

I'll generate the updated code now. 🚀

```
import torch
import torchvision
import os
import csv
import datetime
import numpy as np
from torch.utils.data import DataLoader
from torchvision.transforms import functional as F
from torchvision.datasets import CocoDetection
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score

# Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Function to load Faster R-CNN model
def get_model(num_classes):
    weights =
torchvision.models.detection.FasterRCNN_ResNet50_FPN_Weights.DEFAULT
    model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=weights)
    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)
    return model

# Load validation dataset
class CocoTransform:
    def __call__(self, image, target):
        return F.to_tensor(image), target

def collate_fn(batch):
    images, targets = zip(*batch)
```

```

        return list(images), list(targets)

val_dataset = CocoDetection(
    root="dataset/val",
    annFile="dataset/val/annotations/benthos_validate_coco.json",
    transforms=CocoTransform()
)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False,
collate_fn=collate_fn)

# Function to compute IoU
def compute_iou(box1, boxes2):
    x1, y1, x2, y2 = box1
    ious = []
    for box in boxes2:
        x1g, y1g, w, h = box
        x2g, y2g = x1g + w, y1g + h
        xi1, yi1 = max(x1, x1g), max(y1, y1g)
        xi2, yi2 = min(x2, x2g), min(y2, y2g)
        inter_area = max(0, xi2 - xi1) * max(0, yi2 - yi1)
        box1_area = (x2 - x1) * (y2 - y1)
        box2_area = (x2g - x1g) * (y2g - y1g)
        union_area = box1_area + box2_area - inter_area
        iou = inter_area / union_area if union_area > 0 else 0
        ious.append(iou)
    return ious

# Evaluation function
def evaluate_model(model, data_loader, device, iou_threshold=0.5):
    all_preds, all_labels = [], []
    model.eval()
    with torch.no_grad():
        for images, targets in data_loader:
            images = [img.to(device) for img in images]
            outputs = model(images)
            for i in range(len(images)):
                gt_labels = [obj["category_id"] for obj in targets[i]]
                gt_boxes = torch.tensor([obj["bbox"] for obj in
targets[i]], dtype=torch.float32)
                pred_labels = outputs[i]["labels"].cpu().numpy()
                pred_scores = outputs[i]["scores"].cpu().numpy()
                pred_boxes = outputs[i]["boxes"].cpu().numpy()
                keep = pred_scores > 0.5
                pred_labels, pred_boxes = pred_labels[keep],
pred_boxes[keep]
                matched_preds, matched_labels = [], []
                for pred_box, pred_label in zip(pred_boxes, pred_labels):
                    ious = compute_iou(pred_box, gt_boxes.numpy())
                    best_iou_idx = np.argmax(ious) if len(ious) > 0 else -1
                    best_iou = ious[best_iou_idx] if best_iou_idx >= 0 else
0

                    if best_iou >= iou_threshold:

```

```

        matched_preds.append(pred_label)
        matched_labels.append(gt_labels[best_iou_idx])
    else:
        matched_preds.append(pred_label)
        matched_labels.append(0)
    all_preds.extend(matched_preds)
    all_labels.extend(matched_labels)
    precision = precision_score(all_labels, all_preds, average="macro",
zero_division=1)
    recall = recall_score(all_labels, all_preds, average="macro",
zero_division=1)
    f1 = f1_score(all_labels, all_preds, average="macro", zero_division=1)
    accuracy = accuracy_score(all_labels, all_preds)
    return precision, recall, f1, accuracy

# Get list of models from folder
models_dir = "Models_for_eval"
model_files = [f for f in os.listdir(models_dir) if f.endswith(".pth")]

# CSV file setup
csv_filename = "evaluation_results.csv"
with open(csv_filename, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Model Name", "Precision", "Recall", "F1-score",
"Accuracy", "Evaluation Date"])

    # Evaluate each model
    for model_file in model_files:
        model_path = os.path.join(models_dir, model_file)
        print(f"Evaluating {model_file}...")
        model = get_model(num_classes=11)
        model.load_state_dict(torch.load(model_path, map_location=device))
        model.to(device)
        precision, recall, f1, accuracy = evaluate_model(model, val_loader,
device)
        eval_date = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        writer.writerow([model_file, precision, recall, f1, accuracy,
eval_date])
        print(f"{model_file}: Precision={precision:.4f},
Recall={recall:.4f}, F1={f1:.4f}, Accuracy={accuracy:.4f}")


print(f"All results saved to {csv_filename}")

```

What This Code Does:

- ✓ Finds all .pth model files in "Models_for_eval".
- ✓ Loads and evaluates each model using the validation dataset.
- ✓ Stores results in evaluation_results.csv for easy comparison.
- ✓ Prints real-time progress to keep track of model evaluations.

Next Steps

Would you like additional sorting, filtering, or visualization of the results (e.g., plotting precision vs. recall)? 

VII Sisend (prompt): I got this error: line 54, in <module> image =
Image.open(image_path).convert("RGB") ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File
"C:\\ProgramData\\anaconda3\\Lib\\site-packages\\PIL\\Image.py", line 3431, in open fp =
builtins.open(filename, "rb") ^^^ PermissionError: [Errno 13]
Permission denied

VII Vastus:

The error occurs because your code is trying to open a directory (in this case, "test_image_results") as if it were an image file. This can happen if your test image folder contains subdirectories or files that are not images.

To fix this, you can filter out directories by checking whether each item is a file before trying to open it. For example:

python
Copy
Edit

```
for image_name in os.listdir(TEST_IMAGE_FOLDER):
    image_path = os.path.join(TEST_IMAGE_FOLDER, image_name)
    # Skip if the path is not a file (i.e., it's a directory)
    if not os.path.isfile(image_path):
        continue

    image = Image.open(image_path).convert("RGB")
    # ... rest of your code ...
```

This ensures that only files (and not directories) are processed. Make sure your test image folder only contains valid image files (or filter them based on typical image file extensions) to avoid similar issues.

By applying this filter, your script should no longer try to open directories as images, and the error should be resolved.

VIII Sisend (prompt): Please fix this error: `AttributeError: 'ImageDraw' object has no attribute 'textsize'`

VIII Vastus:

You can replace the call to `draw.textsize` with `font.getsize` which returns the width and height of the text. Here's the updated function:

python
CopyEdit

```
def draw_boxes(image: Image.Image, boxes, labels, scores, threshold: float
= 0.5) -> Image.Image:
    draw = ImageDraw.Draw(image)
```

```

try:
    # Increase the font size for better visibility
    font = ImageFont.truetype("arial.ttf", size=25)
except IOError:
    font = ImageFont.load_default()

for box, label, score in zip(boxes, labels, scores):
    if score >= threshold:
        x1, y1, x2, y2 = box
        draw.rectangle([x1, y1, x2, y2], outline="red", width=3)
        text = f"{CLASS_NAMES[label]}: {score:.2f}"
        text_width, text_height = font.getsize(text)
        # Draw text background for better visibility
        text_background = [x1, y1 - text_height, x1 + text_width, y1]
        draw.rectangle(text_background, fill="red")
        draw.text((x1, y1 - text_height), text, fill="yellow",
font=font)
    return image

```

This change uses `font.getsize(text)` to determine the dimensions of the text, ensuring compatibility and resolving the `AttributeError`.

IX Sisend (prompt): Now I get this error: `AttributeError: 'NoneType' object has no attribute 'save'`

IX Vastus:

This error usually means that the function meant to return an image is instead returning `None`. In your case, it suggests that your `draw_boxes` function isn't returning the modified image.

Make sure that your `draw_boxes` function ends with a return statement so that it always returns the image (even if no boxes were drawn). For example:

```

python
CopyEdit
def draw_boxes(image: Image.Image, boxes, labels, scores, threshold: float
= 0.5) -> Image.Image:
    draw = ImageDraw.Draw(image)
    try:
        font = ImageFont.truetype("arial.ttf", size=25)
    except IOError:
        font = ImageFont.load_default()

    for box, label, score in zip(boxes, labels, scores):
        if score >= threshold:
            x1, y1, x2, y2 = box
            draw.rectangle([x1, y1, x2, y2], outline="red", width=3)
            text = f"{CLASS_NAMES[label]}: {score:.2f}"
            text_width, text_height = font.getsize(text)
            text_background = [x1, y1 - text_height, x1 + text_width, y1]
            draw.rectangle(text_background, fill="red")

```

```

        draw.text((x1, y1 - text_height), text, fill="yellow",
font=font)
    # Ensure the modified image is returned
    return image

```

Double-check that this return statement is present and that you're not accidentally overriding the function or missing it in one of your branches. This should resolve the `AttributeError` when you try to call the save method.

X Sisend (prompt): This function now gives error: `AttributeError: 'FreeTypeFont' object has no attribute 'getsize'`

X Vastus:

You can use the `textbbox` method from the `ImageDraw` object to get the bounding box of your text. This method returns a tuple with the coordinates (left, top, right, bottom), from which you can compute the width and height. Here's an updated version of your function:

python

CopyEdit

```

def draw_boxes(image: Image.Image, boxes, labels, scores, threshold: float
= 0.5) -> Image.Image:
    """

```

```

        Draws bounding boxes and labels on the provided image.

```

Args:

```

    image (Image.Image): The PIL image to annotate.
    boxes (list): List of bounding boxes [x1, y1, x2, y2].
    labels (list): List of label indices corresponding to CLASS_NAMES.
    scores (list): List of confidence scores.
    threshold (float): Confidence threshold for drawing boxes.

```

Returns:

```

    Image.Image: Annotated image.
    """

```

```

draw = ImageDraw.Draw(image)

```

```

try:

```

```

    # Increase the font size for better visibility
    font = ImageFont.truetype("arial.ttf", size=25)

```

```

except IOError:

```

```

    font = ImageFont.load_default()

```

```

for box, label, score in zip(boxes, labels, scores):

```

```

    if score >= threshold:

```

```

        x1, y1, x2, y2 = box

```

```

        draw.rectangle([x1, y1, x2, y2], outline="red", width=3)

```

```

        text = f"{CLASS_NAMES[label]}: {score:.2f}"

```

```

        # Use textbbox to compute the size of the text

```

```

        text_bbox = draw.textbbox((x1, y1), text, font=font)

```

```

        text_width = text_bbox[2] - text_bbox[0]

```

```

        text_height = text_bbox[3] - text_bbox[1]

```

```

        # Draw text background for better visibility

```

```

        text_background = [x1, y1 - text_height, x1 + text_width, y1]

```

```

        draw.rectangle(text_background, fill="red")

```

```

        draw.text((x1, y1 - text_height), text, fill="yellow",
font=font)
    return image

```

This modification replaces the use of `font.getsize` (and `draw.textsize`) with `draw.textbbox`, which should work reliably with your PIL version.

XI Sisend (prompt): I have 10 classes (+ background). I would like to additionally evaluate model accuracy for each class too.

XI Vastus:

```

import torch
import torchvision
import os
import csv
import datetime
import numpy as np
from torch.utils.data import DataLoader
from torchvision.transforms import functional as F
from torchvision.datasets import CocoDetection
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score, classification_report

# Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define number of classes
num_classes = 11 # Background + 10 classes

# Function to load Faster R-CNN model
def get_model(num_classes):
    weights =
torchvision.models.detection.FasterRCNN_ResNet50_FPN_Weights.DEFAULT
    model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=weights)
    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)
    return model

# Load validation dataset
class CocoTransform:
    def __call__(self, image, target):
        return F.to_tensor(image), target

def collate_fn(batch):
    images, targets = zip(*batch)
    return list(images), list(targets)

```

```

val_dataset = CocoDetection(
    root="dataset/val",

    annFile="dataset/val/annotations/meiobenthos_validate_10groups_coco.json",
    transforms=CocoTransform()
)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False,
collate_fn=collate_fn)

# Function to compute IoU
def compute_iou(box1, boxes2):
    x1, y1, x2, y2 = box1
    ious = []
    for box in boxes2:
        x1g, y1g, w, h = box
        x2g, y2g = x1g + w, y1g + h
        xi1, yi1 = max(x1, x1g), max(y1, y1g)
        xi2, yi2 = min(x2, x2g), min(y2, y2g)
        inter_area = max(0, xi2 - xi1) * max(0, yi2 - yi1)
        box1_area = (x2 - x1) * (y2 - y1)
        box2_area = (x2g - x1g) * (y2g - y1g)
        union_area = box1_area + box2_area - inter_area
        iou = inter_area / union_area if union_area > 0 else 0
        ious.append(iou)
    return ious

# Evaluation function
def evaluate_model(model, data_loader, device, iou_threshold=0.5):
    all_preds, all_labels = [], []
    model.eval()
    with torch.no_grad():
        for images, targets in data_loader:
            images = [img.to(device) for img in images]
            outputs = model(images)
            for i in range(len(images)):
                gt_labels = [obj["category_id"] for obj in targets[i]]
                gt_boxes = torch.tensor([obj["bbox"] for obj in
targets[i]], dtype=torch.float32)
                pred_labels = outputs[i]["labels"].cpu().numpy()
                pred_scores = outputs[i]["scores"].cpu().numpy()
                pred_boxes = outputs[i]["boxes"].cpu().numpy()
                keep = pred_scores > 0.5
                pred_labels, pred_boxes = pred_labels[keep],
pred_boxes[keep]
                matched_preds, matched_labels = [], []
                for pred_box, pred_label in zip(pred_boxes, pred_labels):
                    ious = compute_iou(pred_box, gt_boxes.numpy())
                    best_iou_idx = np.argmax(ious) if len(ious) > 0 else -1

```

```

        best_iou = ious[best_iou_idx] if best_iou_idx >= 0 else
0
        if best_iou >= iou_threshold:
            matched_preds.append(pred_label)
            matched_labels.append(gt_labels[best_iou_idx])
        else:
            matched_preds.append(pred_label)
            matched_labels.append(0)
        all_preds.extend(matched_preds)
        all_labels.extend(matched_labels)
    precision = precision_score(all_labels, all_preds, average="macro",
zero_division=1)
    recall = recall_score(all_labels, all_preds, average="macro",
zero_division=1)
    f1 = f1_score(all_labels, all_preds, average="macro", zero_division=1)
    accuracy = accuracy_score(all_labels, all_preds)
    class_report = classification_report(all_labels, all_preds,
target_names=[str(i) for i in range(num_classes)], zero_division=1)
    return precision, recall, f1, accuracy, class_report

# Get list of models from folder
models_dir = "Models_for_eval"
model_files = [f for f in os.listdir(models_dir) if f.endswith(".pth")]

# CSV file setup
csv_filename = "evaluation_results.csv"
with open(csv_filename, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Model Name", "Precision", "Recall", "F1-score",
"Accuracy", "Evaluation Date"])

    # Evaluate each model
    for model_file in model_files:
        model_path = os.path.join(models_dir, model_file)
        print(f"Evaluating {model_file}...")
        model = get_model(num_classes=11)
        model.load_state_dict(torch.load(model_path, map_location=device,
weights_only=True))
        model.to(device)
        precision, recall, f1, accuracy, class_report =
evaluate_model(model, val_loader, device)
        eval_date = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        writer.writerow([model_file, precision, recall, f1, accuracy,
eval_date])
        print(f"{model_file}: Precision={precision:.4f},
Recall={recall:.4f}, F1={f1:.4f}, Accuracy={accuracy:.4f}")
        print(f"Classification Report:\n{class_report}")

print(f"All results saved to {csv_filename}")

```

I've modified your script to include a classification report for per-class evaluation. It now computes precision, recall, F1-score, and accuracy for each class in addition to the overall metrics. Let me know if you need further refinements! 🚀

XII Sisend (prompt): I would like to add mAP50 and mAP50-95 calculation.

XII Vastus:

Great idea! Adding mAP@50 and mAP@[50:95] will give you a much better picture of your model's performance, especially if you're doing object detection tasks like with Faster R-CNN.

✅ What Are mAP@50 and mAP@[50:95]?

mAP@50: mean Average Precision at IoU threshold 0.5 (aka PASCAL VOC style).

mAP@[50:95]: the COCO-style metric, averaging AP over IoU thresholds from 0.50 to 0.95 in steps of 0.05.

These are standard for COCO-style datasets and are way more informative than just precision/recall.

📦 Option 1: Use Torchvision's Built-in COCO Evaluator

Torchvision offers an API compatible with the COCOEvaluator via pycocotools. Since you're already using CocoDetection, this is a natural fit.

Step 1: Install pycocotools (if not installed yet)

bash

CopyEdit

```
pip install pycocotools
```

Step 2: Add the Torchvision COCO Evaluator to Your Pipeline

You can use torchvision.datasets.CocoDetection with torchvision.models.detection BUT you'll need COCO format predictions and ground truths for evaluation.

However, there's a much simpler way with torchmetrics or pycocotools. Below, I'll show both a manual COCOEvaluator and a torchmetrics version.

🔨 Option 2: Use Torchmetrics mAP

This is probably easier to integrate if you're doing your own loop.

Step 1: Install torchmetrics

bash

CopyEdit

```
pip install torchmetrics
```

Step 2: Add torchmetrics mAP Evaluation

python

CopyEdit

```
from torchmetrics.detection.mean_ap import MeanAveragePrecision
```

```
def evaluate_with_map(model, data_loader, device):
```

```
    model.eval()
```

```
    metric = MeanAveragePrecision(iou_type="bbox") # mAP@[50:95] by
```

```
default
```

```
    with torch.no_grad():
```

```
        for images, targets in data_loader:
```

```

images = [img.to(device) for img in images]

processed_targets = []
for target in targets:
    boxes = []
    labels = []
    for obj in target:
        if "bbox" in obj:
            x, y, w, h = obj["bbox"]
            if w > 0 and h > 0:
                boxes.append([x, y, x + w, y + h])
                labels.append(obj["category_id"])

    if boxes:
        processed_targets.append({
            "boxes": torch.tensor(boxes,
dtype=torch.float32).to(device),
            "labels": torch.tensor(labels,
dtype=torch.int64).to(device)
        })
    else:
        processed_targets.append({
            "boxes": torch.zeros((0, 4),
dtype=torch.float32).to(device),
            "labels": torch.zeros((0, ),
dtype=torch.int64).to(device)
        })

# Perform prediction
outputs = model(images)

# Convert predictions and targets to CPU numpy
preds = []
targs = []

for output in outputs:
    preds.append({
        "boxes": output["boxes"].cpu(),
        "scores": output["scores"].cpu(),
        "labels": output["labels"].cpu()
    })

for target in processed_targets:
    targs.append({
        "boxes": target["boxes"].cpu(),
        "labels": target["labels"].cpu()
    })

# Update the metric
metric.update(preds, targs)

# Compute the results

```



```

    results = metric.compute()

    print(f"✅ Validation Results -> mAP@50: {results['map_50']:.4f},
mAP@[50:95]: {results['map']:.4f}")
    return results

```

Step 3: Call it inside your training loop

Replace your evaluate() call with:

python

CopyEdit

```

results = evaluate_with_map(model, val_loader, device)
f1 = results["map_50"] # Use mAP@50 as your threshold for saving best
model

```



What does torchmetrics mAP compute?

MeanAveragePrecision() computes:

map: mAP averaged over IoUs [0.50:0.95].

map_50: mAP@50.

map_75: mAP@75.

Per-class APs.

Recall metrics too (optional to display).



Option 3: Use the PyTorch coco_eval.py (Advanced)

Torchvision offers an official example using coco_eval.py. You can port their code into your repo and evaluate using true COCO metrics.

But if you're cool with torchmetrics, it's easier, cleaner, and avoids extra JSON file generation or messing with COCO API loaders.



Advantages of torchmetrics:

Runs inside your PyTorch loop.

No need to write files or handle external tools.

Clean integration with your batch loop.



What Should You Do?

I'd recommend:

Use torchmetrics for quick and clean mAP@50/mAP@[50:95] integration.

Keep your current evaluate() function if you still want precision/recall/F1.

Save the model based on map_50 or map rather than F1 (optional but common).

TL;DR:

Install torchmetrics and add the MeanAveragePrecision evaluator.

It will give you mAP@50 and mAP@[50:95].

Call it during validation, and you'll get richer metrics than F1 alone.

If you want, I can help you:

Merge torchmetrics into your training loop.

Add per-class mAP reporting.

Use pycocotools for official COCO-style evaluation.

Which way do you want to go? 😊

XIII Sisend (prompt): My data is annotated in COCO annotation format.

XIII Vastus:

Perfect! Since your data is already in COCO annotation format, you can take full advantage of pycocotools and torchvision's COCOEvaluator to compute mAP@50 and mAP@[50:95], just like in COCO official evaluations.

This is the industry-standard approach, and it integrates well with Faster R-CNN and CocoDetection.



Plan to Add mAP Evaluation Using COCO API



What You'll Get:

Official COCO mAP metrics:

AP@[50:95] (mean average precision averaged over IoU thresholds from 0.50 to 0.95)

AP@50 (aka PASCAL VOC style)

AP@75

Per-size APs: small, medium, large objects

No manual IoU calculations

Cleaner evaluation with pycocotools



Step 1: Install pycocotools

If you haven't already:

bash

CopyEdit

```
pip install pycocotools
```



Step 2: Use Torchvision's COCO Evaluator Utilities

Torchvision provides tools for COCO evaluation, and the easiest way is by using their coco_eval and coco_utils scripts.

These files live in their official detection references.



I can share the minimal version here or help you integrate their full pipeline. Below is a simplified version you can add right away.



Step 3: Add COCO Evaluator Helpers



Download These Two Files:

coco_eval.py: [LINK](#)

coco_utils.py: [LINK](#)

Place them in your project directory (e.g., utils/ folder).



Step 4: Create an Evaluation Function Using COCO Evaluator

Here's an example using the CocoEvaluator from coco_eval.py and coco_utils.py:

python

CopyEdit

```
import utils.coco_eval as coco_eval
```

```
import utils.coco_utils as coco_utils
```

```
def evaluate_coco(model, data_loader, device):  
    n_threads = torch.get_num_threads()  
    torch.set_num_threads(1) # For evaluation stability
```

```

model.eval()

# Load the COCO API ground truth from dataset
coco = coco_utils.get_coco_api_from_dataset(data_loader.dataset)

iou_types = ["bbox"] # You can add "segm" if you're doing masks too
coco_evaluator = coco_eval.CocoEvaluator(coco, iou_types)

with torch.no_grad():
    for images, targets in data_loader:
        images = [img.to(device) for img in images]
        outputs = model(images)

        # Move outputs to CPU (coco_evaluator expects CPU tensors)
        outputs = [{k: v.cpu() for k, v in t.items()} for t in outputs]

        # Format as COCO expects
        res = {target["image_id"].item(): output for target, output in
zip(targets, outputs)}

        coco_evaluator.update(res)

# Gather all predictions and compute metrics
coco_evaluator.synchronize_between_processes()
coco_evaluator.accumulate()
coco_evaluator.summarize()

torch.set_num_threads(n_threads) # Restore thread count

# Return the summary stats as a dict (optional)
stats = coco_evaluator.coco_eval["bbox"].stats
return {
    "mAP@[50:95]": stats[0], # mAP averaged over IoU=0.50:0.95
    "mAP@50": stats[1],      # mAP at IoU=0.50
    "mAP@75": stats[2],      # mAP at IoU=0.75
    "AR@1": stats[6],        # Recall
    "AR@10": stats[7]
}

```

Step 5: Call This in Your Training Loop

Replace your previous evaluate() call with:

python

CopyEdit

```

results = evaluate_coco(model, val_loader, device)

# Log or print the mAP results
print(f"✅ COCO Evaluation -> mAP@[50:95]: {results['mAP@[50:95]']:.4f},
mAP@50: {results['mAP@50']:.4f}")

# Use mAP@50 as your model checkpoint criteria (or mAP@[50:95])
f1 = results['mAP@50'] # Or 'mAP@[50:95]'

```

🔥 Step 6: Make Sure You Have image_id in Your Dataset Targets!

For CocoEvaluator to work, your dataset targets must include:

python

CopyEdit

```
{
    'boxes': Tensor[N, 4],
    'labels': Tensor[N],
    'image_id': Tensor[1]
}
```

If your targets don't have image_id, modify your dataset class:

python

CopyEdit

```
class AugmentedCocoDataset(CocoDetection):
    def __getitem__(self, idx):
        img, target = super().__getitem__(idx)

        # Convert COCO annotation to required format
        boxes = []
        labels = []
        for obj in target:
            x, y, w, h = obj['bbox']
            boxes.append([x, y, x + w, y + h])
            labels.append(obj['category_id'])

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.as_tensor(labels, dtype=torch.int64)
        image_id = torch.tensor([idx]) # This is critical for
CocoEvaluator

        target_dict = {
            'boxes': boxes,
            'labels': labels,
            'image_id': image_id
        }

        if self._transforms is not None:
            img = self._transforms(img)

        return img, target_dict
```



What the Evaluation Prints

sql

CopyEdit

```
Average Precision (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] =
0.492
Average Precision (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] =
0.745
Average Precision (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] =
0.582
```

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.678

✅ Why Use the COCO Evaluator?

Gives you official COCO metrics, including:

mAP@[.5:.95]

mAP@50

Recall metrics

Per-object-size AP (small/medium/large)

No need to manually compute IoU and F1.

It's standard practice for COCO-formatted datasets.

TL;DR

You're already in COCO format, so leverage pycocotools + Torchvision COCOEvaluator.

Add image_id in your dataset targets.

Use evaluate_coco() to compute mAP metrics.

Optionally, switch your model-saving trigger from F1 to mAP@[50:95] or mAP@50.

🚀 Want me to paste a ready-to-run evaluate_coco() block and help refactor your dataset class for image_id support?

XIV Sisend (prompt): Which lr_scheduler should I use in Faster R-CNN?

lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)

or

lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=15)

XIV Vastus:

Both StepLR and CosineAnnealingLR can work with Faster R-CNN, but the best choice depends on your training setup and goals. Here's a comparison:

1] Option 1: StepLR

python

Copy

Edit

```
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
```

✅ How it works:

Reduces the learning rate by a factor of gamma=0.1 every step_size=3 epochs.

Example: If your initial LR is 0.005, the decay looks like this:

yaml

Copy

Edit

Epoch 0: LR = 0.005

Epoch 3: LR = 0.0005 (10x smaller)

Epoch 6: LR = 0.00005 (another 10x smaller)

...

✅ Why use it?

Works well when training for longer epochs (30+).

Common for object detection models like Faster R-CNN.

Helps maintain stable training while still reducing LR to refine learning.

✗ Potential issue

The sudden drop in LR might not be optimal for fine-tuning.

📌 Best for:

✓ General Faster R-CNN fine-tuning

✓ Training for more than 15-20 epochs

✓ Stable and widely used in object detection

② Option 2: CosineAnnealingLR

python

Copy

Edit

```
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
T_max=15)
```

✓ How it works:

Decreases the learning rate smoothly following a cosine curve over T_max epochs.

Example: If T_max=15, the LR follows a gradual decay:

yaml

Copy

Edit

```
Epoch 0: LR = 0.005
```

```
Epoch 7: LR ≈ 0.0025
```

```
Epoch 15: LR ≈ 0.0000 (almost zero)
```

✓ Why use it?

Works well for shorter training schedules (≤ 20 epochs).

Reduces LR more smoothly, which may help prevent instability.

Encourages better fine-tuning at later stages.

✗ Potential issue

If T_max is too short, the LR decays too fast, leading to underfitting.

Once it reaches zero, training stalls unless restarted.

📌 Best for:

✓ Training for a short number of epochs (≤ 20)

✓ Fine-tuning on smaller datasets

✓ Avoiding sudden learning rate jumps

🔥 Which One is Best for Your Faster R-CNN?

If training for 30+ epochs → Use StepLR

✓ More stable learning, commonly used in object detection.

If training for ≤ 20 epochs → Use CosineAnnealingLR

✓ Smooth decay, better for fine-tuning.

🚀 Bonus: Best of Both Worlds?

If you want cosine decay but with warm restarts, try:

python

Copy

Edit

```
lr_scheduler =  
torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0=10,  
T_mult=2)
```

This restarts the LR every 10 epochs, preventing it from going to zero.

♦ Final Answer?

For Faster R-CNN, StepLR (step_size=3, gamma=0.1) is the safer, widely-used choice. But if training for ≤ 20 epochs, CosineAnnealingLR can work well.

Want me to help tune the hyperparameters further? 🚀

XV Sisend (prompt): I'm training for 50 epochs. 2250 training images and 540 validation images. 1 class + background.

XV Vastus:

Thanks for the details! Let's break this down to figure out the best learning rate scheduler and settings for your Faster R-CNN training.



Your Training Setup Recap

Training images: 2250

Validation images: 540

Epochs: 50

Classes: 1 class + background (so num_classes = 2)

Model: Faster R-CNN with ResNet-50 backbone

Batch size: 4 (as per your earlier code)



Learning Rate Scheduler Recommendations

Since you're training for 50 epochs, here's how I'd approach the learning rate and scheduler:

① StepLR Scheduler (Common + Reliable for Object Detection)

python

Copy

Edit

```
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=15,  
gamma=0.1)
```

♦ Why this works:

You reduce the learning rate by 10x every 15 epochs.

At 50 epochs, it decays twice:

Epoch 15 → LR drops

Epoch 30 → LR drops again

Common strategy for object detection models, especially Faster R-CNN.

Helps the model make big adjustments early and fine-tune later.

② CosineAnnealingLR (Smooth Decay, Fine-tuning Friendly)

python

Copy

Edit

```
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,  
T_max=50)
```

- ♦ Why this works:

Gradually decays the learning rate from your starting LR (0.005 in your case) down to nearly zero over 50 epochs.

Encourages better convergence during fine-tuning.

Great if you want the learning rate to decrease smoothly without sudden jumps.

3 CosineAnnealingWarmRestarts (Optional Advanced)

If you want to restart the learning rate to avoid the model getting stuck:

python

Copy

Edit

```
lr_scheduler =  
torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0=10,  
T_mult=2)
```

- ♦ Why this works:

Restarts the learning rate every 10 epochs, then doubles the cycle length.

Helps explore new solutions by restarting LR (similar to simulated annealing).

Can be beneficial for complex datasets but requires some tuning.

✓ Recommended Learning Rate and Scheduler for You

If you're training for 50 epochs with 2250 images, I'd recommend:

Start with StepLR:

python

Copy

Edit

```
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9,  
weight_decay=0.0005)  
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=15,  
gamma=0.1)
```

✓ Stable, predictable, well-tested for detection tasks.

If you want to experiment with CosineAnnealingLR, use:

python

Copy

Edit

```
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,  
T_max=50)
```

✓ Smooth decay, might help with fine-tuning performance, especially for smaller datasets (2250 is not huge).

✓ Other Suggestions for Your Setup

Batch Size: 4 is fine if you're limited by GPU memory. If you can go higher, it can stabilize gradients.

Learning Rate: Stick with 0.005 to start (standard for Faster R-CNN), but you can experiment with learning rate range tests if training isn't improving.

TL;DR

- Training for 50 epochs on 2250 images
- StepLR is a safe, effective choice → `step_size=15, gamma=0.1`
- CosineAnnealingLR is great if you want smooth LR decay and are fine-tuning
- Start with SGD + StepLR for baseline results, then explore CosineAnnealingLR

Want me to help you plot learning rate schedules or set up early stopping or reduce on plateau schedulers? 😊

XVI Sisend (prompt): Let's add saving checkpoints to my script:

```
import torch
import torchvision
from torch.utils.data import DataLoader
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
from torchvision.datasets import CocoDetection
from torchvision.transforms import functional as F

import os
import json
import datetime
from PIL import Image
import time

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Get current date for logging
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

# Define paths
base_path = "/content/drive/MyDrive/meiobenthos"
dataset_path = os.path.join(base_path, "dataset")
model_save_path = os.path.join(base_path, f"models_trained_19mar2025")
log_path = os.path.join(model_save_path, f"training_log_{current_date}.txt")

# Define transformations
class CocoTransform:
    def __call__(self, image, target):
        image = F.to_tensor(image) # Convert PIL image to tensor
        return image, target

# Dataset function
def get_coco_dataset(img_dir: str, ann_file: str):
    """Loads COCO dataset with custom transforms."""
```

```

return CocoDetection(root=img_dir, annFile=ann_file, transforms=CocoTransform())

# Load datasets (Fixed paths)
train_dataset = get_coco_dataset(
    img_dir=os.path.join(dataset_path, "train"),
    ann_file=os.path.join(dataset_path,
"train/annotations/meiobenthos_train_1class_250background_images_added_coco.json")
)

val_dataset = get_coco_dataset(
    img_dir=os.path.join(dataset_path, "val"),
    ann_file=os.path.join(dataset_path, "val/annotations/meiobenthos_val_1class_coco.json")
)

# DataLoader
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True, collate_fn=lambda x:
tuple(zip(*x)))
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False, collate_fn=lambda x:
tuple(zip(*x)))

# Model function. Load Faster R-CNN with ResNet-50 backbone
def get_model(num_classes):
    # Load pre-trained Faster R-CNN
    weights = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=weights)

    # Replace the classifier head
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
    return model

# Initialize the model
num_classes = 2 # Background + meiofauna
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = get_model(num_classes).to(device)

# Define optimizer and learning rate scheduler
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=15, gamma=0.1)
#controls how much the model's weights change in response to the loss gradient, reduces the
learning rate by 10x every 15 epochs (step_size=15)

# Training Function
def train_one_epoch(model, optimizer, data_loader, device, epoch, log_file):
    model.train() # Set the model to train mode
    total_loss = 0.0

    for images, targets in data_loader:

```

```

# Move images to the device
images = [img.to(device) for img in images]

# Validate and process targets
processed_targets = []
valid_images = []
for i, target in enumerate(targets):
    boxes = []
    labels = []
    for obj in target:
        if "bbox" in obj:
            x, y, w, h = obj["bbox"]
            if w > 0 and h > 0:
                boxes.append([x, y, x + w, y + h])
                labels.append(obj["category_id"])

    if boxes:
        processed_targets.append({
            "boxes": torch.tensor(boxes, dtype=torch.float32).to(device),
            "labels": torch.tensor(labels, dtype=torch.int64).to(device),
        })
        valid_images.append(images[i])

if not processed_targets:
    continue

# Forward pass
loss_dict = model(valid_images, processed_targets)
losses = sum(loss for loss in loss_dict.values())

# Backpropagation
optimizer.zero_grad()
losses.backward()
optimizer.step()

total_loss += losses.item()

avg_loss = total_loss / len(data_loader)
print(f"Epoch [{epoch}] Loss: {avg_loss:.4f}")

return avg_loss # Return loss for logging

# Training loop with loss logging
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()} ---\n")

for epoch in range(50):
    start_time = time.time()
    avg_loss = train_one_epoch(model, optimizer, train_loader, device, epoch, log_file)
    lr_scheduler.step()

```

```

# Save Model Checkpoint (Fixed location of saving)
model_path = os.path.join(model_save_path, f'fasterrcnn_resnet50_epoch_{epoch +
1}.pth")
torch.save(model.state_dict(), model_path)
print(f"Model saved: {model_path}")

# Log loss to file
log_file.write(f"Epoch {epoch + 1}, Loss: {avg_loss:.4f}\n")
log_file.write(f"{datetime.datetime.now()} - Model saved: {model_path}\n")

end_time = time.time()
print(f"Time per epoch: {(end_time - start_time) / 60:.2f} minutes")

log_file.write(f"--- Training Completed at {datetime.datetime.now()} ---\n").

```

XVI Vastus: You're super close to having an awesome setup! You've already got the training loop and model saving going, but you're currently saving only the `model.state_dict()`, which is fine but not enough if you want to resume training seamlessly after a disconnection.

✓ What to Add

You'll want to:

Save a full checkpoint:

```

epoch
model.state_dict()
optimizer.state_dict()
lr_scheduler.state_dict() (optional but recommended)

```

Add resume capability, so if you restart Colab, you can load the latest checkpoint and continue.

🔑 Updated Code with Checkpointing and Resume Support

python

Copy

Edit

```

# ---- Additional function to save checkpoints ----
def save_checkpoint(model, optimizer, scheduler, epoch, path):
    checkpoint = {
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'scheduler_state_dict': scheduler.state_dict() if scheduler else
None
    }
    torch.save(checkpoint, path)
    print(f"Checkpoint saved: {path}")

# ---- Optional function to load checkpoint ----
def load_checkpoint(model, optimizer, scheduler, path, device):
    checkpoint = torch.load(path, map_location=device)
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    if scheduler and checkpoint.get('scheduler_state_dict'):

```

```

        scheduler.load_state_dict(checkpoint['scheduler_state_dict'])
    epoch = checkpoint['epoch']
    print(f"Checkpoint loaded: {path} (Epoch {epoch})")
    return epoch

```

Modify Your Training Loop to Include Resume Capability

At the start of your training loop, before `for epoch in range(50):`, add this to check for an existing checkpoint:

python

Copy

Edit

```

# Directory for saving models
os.makedirs(model_save_path, exist_ok=True)

# Check for existing checkpoint to resume
latest_checkpoint = None
start_epoch = 0

# Example: Automatically find the latest checkpoint (if you want
automation)
checkpoints = [f for f in os.listdir(model_save_path) if
f.endswith('.pth')]
if checkpoints:
    # Sort by epoch number (assuming filename format:
    fasterrcnn_resnet50_epoch_{epoch}.pth)
    checkpoints.sort(key=lambda x: int(x.split('_')[-1].split('.')[0]))
    latest_checkpoint = os.path.join(model_save_path, checkpoints[-1])

if latest_checkpoint:
    start_epoch = load_checkpoint(model, optimizer, lr_scheduler,
latest_checkpoint, device) + 1
    print(f"Resuming training from epoch {start_epoch}")
else:
    print("Starting training from scratch")

```

Modify Your Training Loop for epoch in range(50):

Change this line:

python

Copy

Edit

`for epoch in range(50):`

To this:

python

Copy

Edit

`num_epochs = 50`

`for epoch in range(start_epoch, num_epochs):`



Inside the Training Loop — Replace the Current Saving Block

Replace:

python

Copy

Edit

```
model_path = os.path.join(model_save_path,
f"fasterrcnn_resnet50_epoch_{epoch + 1}.pth")
torch.save(model.state_dict(), model_path)
print(f"Model saved: {model_path}")
```

With:

python

Copy

Edit

```
checkpoint_path = os.path.join(model_save_path,
f"fasterrcnn_resnet50_epoch_{epoch + 1}.pth")
save_checkpoint(model, optimizer, lr_scheduler, epoch, checkpoint_path)
```

✓ Now Your Full Loop Looks Like This:

python

Copy

Edit

```
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()}
---\n")

    num_epochs = 50
    for epoch in range(start_epoch, num_epochs):
        start_time = time.time()
        avg_loss = train_one_epoch(model, optimizer, train_loader, device,
epoch, log_file)
        lr_scheduler.step()

        # Save full checkpoint (model + optimizer + scheduler + epoch)
        checkpoint_path = os.path.join(model_save_path,
f"fasterrcnn_resnet50_epoch_{epoch + 1}.pth")
        save_checkpoint(model, optimizer, lr_scheduler, epoch,
checkpoint_path)

        # Log loss to file
        log_file.write(f"Epoch {epoch + 1}, Loss: {avg_loss:.4f}\n")
        log_file.write(f"{datetime.datetime.now()} - Checkpoint saved:
{checkpoint_path}\n")

        end_time = time.time()
        print(f"Time per epoch: {(end_time - start_time) / 60:.2f}
minutes")

    log_file.write(f"--- Training Completed at {datetime.datetime.now()}
---\n")
```

✓ Summary of What's Improved:

↺ Resumes from the latest checkpoint if the session disconnects



Saves not only model weights but also optimizer & scheduler states + epoch



You no longer lose training progress beyond the last saved checkpoint

Let me know if you want to add automatic backup to Drive or early stopping next!

XVII Sisend (prompt): Let's add a predictions report as well (to folder DEFAULT_RESULTS_FOLDER).

The report should be an Excel spreadsheet. First column would be image file name. Second column would contain a list of detected organisms in image. If there are more than one organism of the same class, then I need all of them in the list.

XVII Vastus:

Absolutely! Let's generate an Excel prediction report and save it in your DEFAULT_RESULTS_FOLDER. The report will:

Be a .xlsx file.

Contain two columns:

Image File Name

List of Detected Organisms (all occurrences, including duplicates).

We'll use pandas and openpyxl for this. If they aren't installed yet:

bash

Copy

Edit

pip install pandas openpyxl



Here's your updated code with report generation:

1. Add pandas import

At the top with your other imports:

python

Copy

Edit

```
import pandas as pd
```

2. Modify process_images() to collect prediction data and save a report

python

Copy

Edit

```
def process_images(test_folder: str, results_folder: str, model, transform,
device: torch.device,
                    threshold: float = 0.5):
    """
    Processes images from the test folder, performs inference, draws
    bounding boxes,
    saves the annotated images, cropped organisms, and generates a
    prediction report.
    """
    os.makedirs(results_folder, exist_ok=True)

    # Create folder for cropped results
    crops_folder = "Detected_organisms_crop_results"
    os.makedirs(crops_folder, exist_ok=True)
```

```

# Store report data in a list of dictionaries
report_data = []

# Filter for typical image file extensions
image_files = [f for f in os.listdir(test_folder) if
f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp'))]

for image_name in tqdm(image_files, desc="Processing images"):
    image_path = os.path.join(test_folder, image_name)

    if not os.path.isfile(image_path):
        continue

    image = Image.open(image_path).convert("RGB")

    # Preprocess image
    image_tensor = transform(image).unsqueeze(0).to(device)

    # Perform inference
    with torch.no_grad():
        prediction = model(image_tensor)

    # Extract results
    boxes = prediction[0]["boxes"].cpu().numpy()
    labels = prediction[0]["labels"].cpu().numpy()
    scores = prediction[0]["scores"].cpu().numpy()

    # Draw bounding boxes and labels on the image
    image_with_boxes = draw_boxes(image.copy(), boxes, labels, scores,
threshold)

    # Save the annotated image
    result_path = os.path.join(results_folder, image_name)
    image_with_boxes.save(result_path)

    # Crop and save detected organisms
    crop_and_save_organisms(image, boxes, scores, threshold,
crops_folder, image_name, margin=20)

    # Prepare the detected organisms list for the report
    indices = np.where(scores >= threshold)[0]

    detected_organisms = []
    for idx in indices:
        label_index = labels[idx]
        detected_organisms.append(CLASS_NAMES[label_index])

    report_data.append({
        "Image File Name": image_name,
        "Detected Organisms": ", ".join(detected_organisms) if
detected_organisms else "None"
    })

```



```
# Save the report to an Excel file
report_df = pd.DataFrame(report_data)
report_path = os.path.join(results_folder, "Prediction_Report.xlsx")
report_df.to_excel(report_path, index=False)

print(f"✅ All annotated images saved in '{results_folder}'!")
print(f"✅ All cropped organisms saved in '{crops_folder}'!")
print(f"✅ Prediction report saved as '{report_path}'!")
```

✅ Explanation:

We collect data in `report_data`, a list of dictionaries where:

"Image File Name" is the image name.

"Detected Organisms" is a comma-separated string listing all detected classes (including duplicates).

We convert `report_data` into a `pandas.DataFrame` and save it as `Prediction_Report.xlsx` inside your `results_folder`.

📝 Example Report Output:

Image File Name	Detected Organisms
img_01.jpg	meiofauna, meiofauna
img_02.jpg	meiofauna
img_03.jpg	None

✅ Next Steps

You can run the script the same way:

bash

Copy

Edit

```
python your_script_name.py --test_folder test_image --results_folder
test_image_results_detect_animal_only
```

Want to customize?

Save report per image folder?

Add coordinates or scores to the report?

Different Excel sheet structure?

Let me know!

XVIII Sisend (prompt): Let's add one more column to the report: alert if any of the prediction score is under 0.75. The cells should contain the lowest prediction score in image.

XVIII Vastus: Perfect! Let's extend the Excel report to include an "Alert" column.

✅ New Column Details:

Column Name: Alert (Score < 0.75)

If any prediction score is below 0.75, we show the lowest score (up to 2 decimal places).

If no scores are below 0.75, we leave it blank (or you can change the message).



Here's the updated `process_images()` function with the new alert column:

python

Copy

Edit

```
def process_images(test_folder: str, results_folder: str, model, transform,
device: torch.device,
```

```

        threshold: float = 0.5):
    """
    Processes images from the test folder, performs inference, draws
    bounding boxes,
    saves the annotated images, cropped organisms, and generates a
    prediction report with alerts.
    """
    os.makedirs(results_folder, exist_ok=True)

    # Create folder for cropped results
    crops_folder = "Detected_organisms_crop_results"
    os.makedirs(crops_folder, exist_ok=True)

    # Store report data in a list of dictionaries
    report_data = []

    # Filter for typical image file extensions
    image_files = [f for f in os.listdir(test_folder) if
f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp'))]

    for image_name in tqdm(image_files, desc="Processing images"):
        image_path = os.path.join(test_folder, image_name)

        if not os.path.isfile(image_path):
            continue

        image = Image.open(image_path).convert("RGB")

        # Preprocess image
        image_tensor = transform(image).unsqueeze(0).to(device)

        # Perform inference
        with torch.no_grad():
            prediction = model(image_tensor)

        # Extract results
        boxes = prediction[0]["boxes"].cpu().numpy()
        labels = prediction[0]["labels"].cpu().numpy()
        scores = prediction[0]["scores"].cpu().numpy()

        # Draw bounding boxes and labels on the image
        image_with_boxes = draw_boxes(image.copy(), boxes, labels, scores,
threshold)

        # Save the annotated image
        result_path = os.path.join(results_folder, image_name)
        image_with_boxes.save(result_path)

        # Crop and save detected organisms
        crop_and_save_organisms(image, boxes, scores, threshold,
crops_folder, image_name, margin=20)

```

```

# Prepare the detected organisms list for the report
indices = np.where(scores >= threshold)[0]

detected_organisms = []
scores_above_threshold = []

for idx in indices:
    label_index = labels[idx]
    detected_organisms.append(CLASS_NAMES[label_index])
    scores_above_threshold.append(scores[idx])

# Default alert values
alert_value = ""

# If there are detections above threshold, check for low scores
if scores_above_threshold:
    min_score = min(scores_above_threshold)
    if min_score < 0.75:
        alert_value = f"{min_score:.2f}"

report_data.append({
    "Image File Name": image_name,
    "Detected Organisms": ", ".join(detected_organisms) if
detected_organisms else "None",
    "Alert (Score < 0.75)": alert_value
})

# Save the report to an Excel file
report_df = pd.DataFrame(report_data)
report_path = os.path.join(results_folder, "Prediction_Report.xlsx")
report_df.to_excel(report_path, index=False)

print(f"✅ All annotated images saved in '{results_folder}'!")
print(f"✅ All cropped organisms saved in '{crops_folder}'!")
print(f"✅ Prediction report saved as '{report_path}'!")

```

✅ What's new:

We check `scores_above_threshold` (filtered by your confidence threshold).

If any score is under 0.75, we display the lowest score in the Alert column.

If there are no detections above threshold, or if the lowest score is ≥ 0.75 , we leave the column empty.

📝 Example Output:

Image File Name	Detected Organisms	Alert (Score < 0.75)
img_01.jpg	meiofauna, meiofauna	0.60
img_02.jpg	meiofauna	
img_03.jpg	None	

✅ Ready to run!

bash

Copy

Edit

```
python your_script_name.py --test_folder test_image --results_folder
test_image_results_detect_animal_only
```



Ideas for next steps (optional):

Customize the alert threshold via an argument (`--alert_threshold 0.75`).

Color-code or highlight cells in Excel.

Include number of detections per class in the report.

Let me know if you want to explore any of those!

XIX Sisend (prompt): Maybe it would be better to save the results as a single report. So, let's append model names in column headers: there will be 6 columns, 3 for each model.

XIX Vastus: Yes — combining everything into a single report with clearly labeled columns is a great idea! That gives you a cleaner comparison across models per image.



Here's how we'll structure the final Excel report:

Image File Name Model 1: Detected Organisms Model 1: Alert (Score < 0.75)

Model 2: Detected Organisms Model 2: Alert (Score < 0.75)



Updated Plan:

We will:

Modify `process_images()` to return the report data as a dictionary.

Call it for both models.

Merge both model results into one final DataFrame.

Save that as a single Excel file.



Modified code:

python

Copy

Edit

```
def process_images(folder, model, class_names, device, threshold=0.5):
    transform = transforms.Compose([transforms.ToTensor()])
    files = [f for f in os.listdir(folder) if f.lower().endswith((''.png',
'.jpg', '.jpeg', '.bmp'))]
```

```
    report_data = {}
    for file in tqdm(files, desc="Processing images"):
        image_path = os.path.join(folder, file)
        image = Image.open(image_path).convert("RGB")
        image_tensor = transform(image).unsqueeze(0).to(device)
```

```
    with torch.no_grad():
        prediction = model(image_tensor)
```

```
    boxes = prediction[0]["boxes"].cpu().numpy()
    labels = prediction[0]["labels"].cpu().numpy()
    scores = prediction[0]["scores"].cpu().numpy()
```

```
    indices = np.where(scores >= threshold)[0]
    detected_organisms = []
```

```

scores_above_threshold = []

for idx in indices:
    label_index = labels[idx]
    if label_index < len(class_names):
        detected_organisms.append(class_names[label_index])
    else:
        detected_organisms.append(f"Unknown({label_index})")
    scores_above_threshold.append(scores[idx])

alert_value = ""
if scores_above_threshold:
    min_score = min(scores_above_threshold)
    if min_score < 0.75:
        alert_value = f"{min_score:.2f}"

report_data[file] = {
    "Detected Organisms": ", ".join(detected_organisms) if
detected_organisms else "None",
    "Alert (Score < 0.75)": alert_value
}

return report_data

```

✓ Combine results and save final Excel report:

python

Copy

Edit

```

def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model1 = load_model(DEFAULT_MODEL1_PATH, len(CLASS_NAMES), device)
    model2 = load_model(DEFAULT_MODEL2_PATH, len(CLASS_NAMES_SECOND),
device)

    results_model1 = process_images(TEST_IMAGE_FOLDER, model1, CLASS_NAMES,
device)
    results_model2 = process_images(TEST_IMAGE_FOLDER, model2,
CLASS_NAMES_SECOND, device)

    # Merge results into a unified dataframe
    all_files = sorted(set(results_model1.keys()) |
set(results_model2.keys()))
    combined_rows = []

    for file in all_files:
        row = {
            "Image File Name": file,
            "Model 1: Detected Organisms": results_model1.get(file,
{}).get("Detected Organisms", "None"),
            "Model 1: Alert (Score < 0.75)": results_model1.get(file,
{}).get("Alert (Score < 0.75)", ""),

```

```

        "Model 2: Detected Organisms": results_model2.get(file,
{}).get("Detected Organisms", "None"),
        "Model 2: Alert (Score < 0.75)": results_model2.get(file,
{}).get("Alert (Score < 0.75)", "")
    }
    combined_rows.append(row)

    final_report_df = pd.DataFrame(combined_rows)
    os.makedirs("final_results", exist_ok=True)
    report_path = os.path.join("final_results",
"final_combined_report.xlsx")
    final_report_df.to_excel(report_path, index=False)
    print(f"✅ Final combined report saved to {report_path}")

```

Let me know if you'd like to also include visual outputs (e.g., annotated image saving) or keep this just as a clean reporting script.

XX Sisend (prompt): I still need the images with predicted bounding boxes and class names.

XX Vastus: Perfect — we'll keep saving the annotated images alongside generating the final unified report. Here's how we'll handle everything cleanly:

✅ Updated Plan
Process each image:

Predict using the model.

Save annotated image with bounding boxes and class labels.

Collect prediction results for the report.

Save images to separate folders:

results_model1/annotated_<image>

results_model2/annotated_<image>

Build a single Excel report with:

Image File Name

Model 1: Detected Organisms, Model 1: Alert

Model 2: Detected Organisms, Model 2: Alert

✅ Updated process_images():

python

Copy

Edit

```

def process_images(folder, results_folder, model, class_names, device,
threshold=0.5):

```

```

os.makedirs(results_folder, exist_ok=True)
transform = transforms.Compose([transforms.ToTensor()])
files = [f for f in os.listdir(folder) if f.lower().endswith((''.png',
'.jpg', '.jpeg', '.bmp'))]

report_data = {}

for file in tqdm(files, desc=f"Processing {results_folder}"):
    image_path = os.path.join(folder, file)
    image = Image.open(image_path).convert("RGB")
    image_tensor = transform(image).unsqueeze(0).to(device)

    with torch.no_grad():
        prediction = model(image_tensor)

    boxes = prediction[0]["boxes"].cpu().numpy()
    labels = prediction[0]["labels"].cpu().numpy()
    scores = prediction[0]["scores"].cpu().numpy()

    # Save annotated image
    annotated_image = draw_boxes(image.copy(), boxes, labels, scores,
threshold, class_names)
    annotated_path = os.path.join(results_folder, f"annotated_{file}")
    annotated_image.save(annotated_path)

    # Report details
    indices = np.where(scores >= threshold)[0]
    detected_organisms = []
    scores_above_threshold = []

    for idx in indices:
        label_index = labels[idx]
        if label_index < len(class_names):
            detected_organisms.append(class_names[label_index])
        else:
            detected_organisms.append(f"Unknown({label_index})")
        scores_above_threshold.append(scores[idx])

    alert_value = ""
    if scores_above_threshold:
        min_score = min(scores_above_threshold)
        if min_score < 0.75:
            alert_value = f"{min_score:.2f}"

    report_data[file] = {
        "Detected Organisms": ", ".join(detected_organisms) if
detected_organisms else "None",
        "Alert (Score < 0.75)": alert_value
    }

return report_data

```

✅ Updated main():

python

Copy

Edit

```
def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model1 = load_model(DEFAULT_MODEL1_PATH, len(CLASS_NAMES), device)
    model2 = load_model(DEFAULT_MODEL2_PATH, len(CLASS_NAMES_SECOND),
device)

    results_model1 = process_images(TEST_IMAGE_FOLDER, "results_model1",
model1, CLASS_NAMES, device)
    results_model2 = process_images(TEST_IMAGE_FOLDER, "results_model2",
model2, CLASS_NAMES_SECOND, device)

    all_files = sorted(set(results_model1.keys()) |
set(results_model2.keys()))
    combined_rows = []

    for file in all_files:
        row = {
            "Image File Name": file,
            "Model 1: Detected Organisms": results_model1.get(file,
{}).get("Detected Organisms", "None"),
            "Model 1: Alert (Score < 0.75)": results_model1.get(file,
{}).get("Alert (Score < 0.75)", ""),
            "Model 2: Detected Organisms": results_model2.get(file,
{}).get("Detected Organisms", "None"),
            "Model 2: Alert (Score < 0.75)": results_model2.get(file,
{}).get("Alert (Score < 0.75)", "")
        }
        combined_rows.append(row)

    final_report_df = pd.DataFrame(combined_rows)
    os.makedirs("final_results", exist_ok=True)
    report_path = os.path.join("final_results",
"final_combined_report.xlsx")
    final_report_df.to_excel(report_path, index=False)
    print(f"✅ Final combined report saved to {report_path}")
```

Let me know if you'd like the bounding boxes to be color-coded per class, or if you want to include confidence scores in the report too.

XXI Sisend (prompt): Let's log evaluation results per object class as well. 9 classes: 1 -

nematode

2 - rotifer

3 - testacea

4 - ciliate

5 - turbellarians

6 - annelida

7 - arthropoda

8 - gastrotricha

9 - tardigrada

XXI Vastus:

To log evaluation results per object class, we need to compute precision, recall, and F1-score separately for each of the 9 classes. I'll modify the script to store per-class metrics in both the console output and the CSV file.

Changes to the Script

Modify `evaluate_model` to compute per-class precision, recall, and F1-score.

Append per-class results to the CSV file.

Updated CSV Format

Now, the CSV file will include per-class precision, recall, and F1-score:

Model Name	Overall Precision	Overall Recall	Overall F1-score	Accuracy	
mAP@50	mAP@[50:95]	Class 1 Precision	Class 1 Recall	Class 1 F1-score	...
	Class 9 Precision	Class 9 Recall	Class 9 F1-score	Evaluation Date	
model1.pth	0.7642	0.8125	0.7877	0.8921	0.6534
	0.4312	0.80	0.85	0.82	...
	0.75	0.78	0.76	2025-03-21	14:35:12

Code Updates

Here's how the modified evaluation function should look:

python

Copy

Edit

```
# Evaluate model with per-class metrics
def evaluate_model(model, data_loader, device, num_classes,
iou_threshold=0.5):
    all_preds = []
    all_labels = []

    model.eval()
    with torch.no_grad():
        for images, targets in data_loader:
            images = [img.to(device) for img in images]
            outputs = model(images)

            for i in range(len(images)):
                gt_labels = [obj["category_id"] for obj in targets[i]]
                gt_boxes = torch.tensor([obj["bbox"] for obj in
targets[i]], dtype=torch.float32)
                pred_labels = outputs[i]["labels"].cpu().numpy()
                pred_scores = outputs[i]["scores"].cpu().numpy()
                pred_boxes = outputs[i]["boxes"].cpu().numpy()

                # Filter predictions (confidence > 0.5)
                keep = pred_scores > 0.5
                pred_labels, pred_boxes = pred_labels[keep],
pred_boxes[keep]
```

```

        matched_preds, matched_labels = [], []
        assigned_gt = set()

        for pred_box, pred_label in zip(pred_boxes, pred_labels):
            ious = compute_iou(pred_box, gt_boxes.numpy())
            best_iou_idx = np.argmax(ious) if len(ious) > 0 else -1
            best_iou = ious[best_iou_idx] if best_iou_idx >= 0 else
0
            if best_iou >= iou_threshold and best_iou_idx not in
assigned_gt:
                matched_preds.append(pred_label)
                matched_labels.append(gt_labels[best_iou_idx])
                assigned_gt.add(best_iou_idx)
            else:
                matched_preds.append(pred_label)
                matched_labels.append(0) # False positive

        for idx, label in enumerate(gt_labels):
            if idx not in assigned_gt:
                matched_preds.append(0) # Missed detection
                matched_labels.append(label)

        all_preds.extend(matched_preds)
        all_labels.extend(matched_labels)

    # Compute overall metrics
    precision = precision_score(all_labels, all_preds, average="macro",
zero_division=1)
    recall = recall_score(all_labels, all_preds, average="macro",
zero_division=1)
    f1 = f1_score(all_labels, all_preds, average="macro", zero_division=1)
    accuracy = accuracy_score(all_labels, all_preds)

    # Compute per-class metrics
    per_class_metrics = {}
    for class_id in range(1, num_classes):
        class_precision = precision_score(all_labels, all_preds,
labels=[class_id], average="binary", zero_division=1)
        class_recall = recall_score(all_labels, all_preds,
labels=[class_id], average="binary", zero_division=1)
        class_f1 = f1_score(all_labels, all_preds, labels=[class_id],
average="binary", zero_division=1)
        per_class_metrics[class_id] = (class_precision, class_recall,
class_f1)

    return precision, recall, f1, accuracy, per_class_metrics

```

CSV Logging Updates

Modify the CSV writing part to log per-class results:

python

Copy

Edit

```
# ➡ CSV file setup (including per-class metrics)
csv_filename = os.path.join(models_dir, "evaluation_results.csv")
with open(csv_filename, mode='w', newline='') as file:
    writer = csv.writer(file)

    # CSV Header
    header = ["Model Name", "Precision", "Recall", "F1-score", "Accuracy",
    "mAP@50", "mAP@[50:95]"]
    for class_id in range(1, num_classes): # Class 1 to 9
        header.extend([f"Class {class_id} Precision", f"Class {class_id}
Recall", f"Class {class_id} F1-score"])
    header.append("Evaluation Date")

    writer.writerow(header)

# ➡ Evaluate each model
for model_file in model_files:
    model_path = os.path.join(models_dir, model_file)
    print(f"\n🔍 Evaluating {model_file}...")

    model = get_model(num_classes=num_classes)
    checkpoint = torch.load(model_path, map_location=device)
    model.load_state_dict(checkpoint['model_state_dict'])
    model.to(device)

    # ➡ Precision / Recall / F1 Evaluation (including per-class)
    precision, recall, f1, accuracy, per_class_metrics =
evaluate_model(model, val_loader, device, num_classes)

    # ➡ COCO mAP Evaluation
    mAP_50_95, mAP_50 = evaluate_coco_map(model, val_loader, device,
annFile)

    eval_date = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    # Build row data
    row = [model_file, precision, recall, f1, accuracy, mAP_50,
mAP_50_95]
    for class_id in range(1, num_classes):
        row.extend(per_class_metrics[class_id])
    row.append(eval_date)

    writer.writerow(row)

# Print summary
print(f"✅ Results for {model_file}:")
print(f"    Precision = {precision:.4f}")
print(f"    Recall    = {recall:.4f}")
print(f"    F1-Score  = {f1:.4f}")
print(f"    Accuracy  = {accuracy:.4f}")
```

```

print(f"    mAP@50      = {mAP_50:.4f}")
print(f"    mAP@50-95 = {mAP_50_95:.4f}")

# Print per-class metrics
print("\nPer-Class Metrics:")
for class_id in range(1, num_classes):
    class_precision, class_recall, class_f1 =
per_class_metrics[class_id]
    print(f"    Class {class_id}: Precision={class_precision:.4f},
Recall={class_recall:.4f}, F1={class_f1:.4f}")

print(f"\n🎉 All results saved to {csv_filename}")

```

Summary of Improvements

- ✅ Logs per-class precision, recall, and F1-score
- ✅ Updated CSV file to store per-class metrics
- ✅ Prints detailed per-class results in the console

This will help analyze how well the model performs on each class (e.g., if certain classes are underperforming). 🚀 Let me know if you need any tweaks!

ChatGPT abil YOLO programmi arendamine

I Sisend (prompt): Let's try YOLOv8. Can you transform this code for YOLOv8:

```
import torch
import torchvision
from torch.utils.data import DataLoader
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
from torchvision.datasets import CocoDetection
from torchvision.transforms import functional as F
import os
import json
import datetime
from PIL import Image
import time

#Get current date for logging
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
log_path = f"training_log_{current_date}.txt"

#Define transformations
class CocoTransform:
    def __call__(self, image, target):
        image = F.to_tensor(image) # Convert PIL image to tensor
        return image, target

# Dataset function
def get_coco_dataset(img_dir: str, ann_file: str):
    """Loads COCO dataset with custom transforms."""
    return CocoDetection(root=img_dir, annFile=ann_file, transforms=CocoTransform())

# Load datasets
train_dataset = get_coco_dataset(
    img_dir="dataset/train",
    ann_file="dataset/train/annotations/meiobenthos_train_10groups_coco.json")

val_dataset = get_coco_dataset(
    img_dir="dataset/val",
    ann_file="dataset/val/annotations/meiobenthos_validate_10groups_coco.json")

# DataLoader
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True, collate_fn=lambda x:
tuple(zip(*x)))
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False, collate_fn=lambda x:
tuple(zip(*x)))

# Model function. Load Faster R-CNN with ResNet-50 backbone
def get_model(num_classes):
    # Load pre-trained Faster R-CNN
```

```

weights = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=weights)

# Replace the classifier head
in_features = model.roi_heads.box_predictor.cls_score.in_features # pretrained model to
extract features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
return model

# Initialize the model
num_classes = 11 # Background + nematode + rotifer + Testacea + ciliate + turbellarians +
annelida + arthropoda + gastrotricha + tardigrada + mollusca
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = get_model(num_classes).to(device)

# Define optimizer and learning rate scheduler
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)

# Training Function
def train_one_epoch(model, optimizer, data_loader, device, epoch, log_file):
    model.train() #set the model to train model
    total_loss = 0.0

    for images, targets in data_loader: #target means bounding boxes and class labels
(annotations)
        # Move images to the device, images and model need to be on the same device
        images = [img.to(device) for img in images]

        # Validate and process targets
        processed_targets = []
        valid_images = []
        for i, target in enumerate(targets):
            boxes = []
            labels = []
            for obj in target:
                # Extract bbox
                if "bbox" in obj:
                    x, y, w, h = obj["bbox"]
                    if w > 0 and h > 0:
                        boxes.append([x, y, x + w, y + h])
                        labels.append(obj["category_id"])

            # Only process if there are valid boxes
            if boxes:
                processed_targets.append({
                    "boxes": torch.tensor(boxes, dtype=torch.float32).to(device),
                    "labels": torch.tensor(labels, dtype=torch.int64).to(device), })

```

```

        valid_images.append(images[i]) # Add only valid images

# Skip iteration if no valid targets
if not processed_targets:
    continue

# Forward pass
loss_dict = model(valid_images, processed_targets)
losses = sum(loss for loss in loss_dict.values())

# Backpropagation
optimizer.zero_grad()
losses.backward()
optimizer.step()

total_loss += losses.item()

avg_loss = total_loss / len(data_loader)
print(f"Epoch [{epoch}] Loss: {avg_loss:.4f}")

return avg_loss

# Training loop
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()} ---\n")

    for epoch in range(8):
        start_time = time.time()
        avg_loss = train_one_epoch(model, optimizer, train_loader, device, epoch, log_file)
        lr_scheduler.step()

        # Save Model Checkpoint
        model_path = f"fasterrcnn_resnet50_epoch_{epoch + 1}.pth"
        torch.save(model.state_dict(), model_path)
        print(f"Model saved: {model_path}")

        # Log loss to file
        log_file.write(f"Epoch {epoch + 1}, Loss: {avg_loss:.4f}\n")
        log_file.write(f"{datetime.datetime.now()} - Model saved: {model_path}\n")
        end_time = time.time()
        print(f"Time per epoch: {(end_time - start_time) / 60:.2f} minutes")

    log_file.write(f"--- Training Completed at {datetime.datetime.now()} ---\n")

```

I Vastus:

Here's the equivalent code for training YOLOv8 instead of Faster R-CNN. YOLOv8 is implemented using Ultralytics' YOLO class, which simplifies training and evaluation.

This script:

Converts COCO format to YOLO format.
Loads the YOLOv8 model (you can change between yolov8n, yolov8s, etc.).
Trains the model on your dataset.
Saves model checkpoints and logs training progress.

Converted Code (YOLOv8)

python

Copy

Edit

```
from ultralytics import YOLO
import datetime
import time

# Get current date for logging
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
log_path = f"training_log_{current_date}.txt"

# Convert COCO dataset to YOLO format (one-time conversion)
# Assuming dataset structure:
# dataset/train/images/ (contains images)
# dataset/train/labels/ (contains YOLO txt labels)
# dataset/val/images/
# dataset/val/labels/

# Load YOLOv8 model (choose model size: n, s, m, l, x)
model = YOLO("yolov8m.pt") # Medium model, balanced for accuracy & speed

# Define training parameters
train_params = {
    "data": "dataset/data.yaml", # YAML config pointing to train & val
    "epochs": 8,
    "batch": 4,
    "imgsz": 640,
    "device": "cuda", # Use GPU if available
    "save": True, # Save best model
    "project": "YOLOv8_training",
    "name": f"yolov8_experiment_{current_date}",
    "cache": False, # Set to True if dataset fits in RAM
}

# Start training
start_time = time.time()
results = model.train(**train_params)

# Save training log
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()}\n")
    log_file.write(f"Training results: {results}\n")
    log_file.write(f"--- Training Completed at {datetime.datetime.now()}\n")
```



```

end_time = time.time()
print(f"Total Training Time: {(end_time - start_time) / 60:.2f} minutes")

# Validate the model after training
metrics = model.val()
print(metrics)

```

How to Set Up Your Dataset for YOLOv8

Convert COCO annotations to YOLO format:

Each image needs a .txt file in dataset/train/labels/ and dataset/val/labels/.

Format:

php-template

Copy

Edit

```
<class_id> <x_center> <y_center> <width> <height>
```

Values are normalized (0 to 1) relative to image size.

Can be automated using a COCO → YOLO conversion script (I can provide one).

Create a data.yaml file for YOLO:

yaml

Copy

Edit

```

train: dataset/train/images # Path to train images
val: dataset/val/images     # Path to validation images

```

```

nc: 10 # Number of classes (excluding background)
names: ["nematode", "rotifer", "Testacea", "ciliate", "turbellarians",
"annelida", "arthropoda", "gastrotricha", "tardigrada", "mollusca"]

```

Run the training script! 🚀

Why YOLOv8?

- ✓ Faster than Faster R-CNN
- ✓ Works well for objects of different sizes
- ✓ Easier to train & deploy
- ✓ Automatic anchor-free detection (handles varying object sizes better)

Would you like a COCO-to-YOLO conversion script to help set up your dataset? 😊

II Sisend (prompt): Yes, please do that.

II Vastus:

Here's a COCO to YOLO format conversion script. This script:

Reads COCO JSON annotation files.

Converts bounding boxes to YOLO format (normalized).

Saves the YOLO annotations in dataset/train/labels/ and dataset/val/labels/.

COCO to YOLO Conversion Script

python

Copy

Edit

```
import os
import json
import shutil
from PIL import Image

# Paths (Modify these as needed)
coco_annotations = {
    "train":
"dataset/train/annotations/meiobenthos_train_10groups_coco.json",
    "val":
"dataset/val/annotations/meiobenthos_validate_10groups_coco.json"
}
image_dirs = {
    "train": "dataset/train/images",
    "val": "dataset/val/images"
}
output_dirs = {
    "train": "dataset/train/labels",
    "val": "dataset/val/labels"
}

# Ensure output label directories exist
for split in ["train", "val"]:
    os.makedirs(output_dirs[split], exist_ok=True)

# Function to convert COCO bbox to YOLO format
def coco_to_yolo_bbox(image_w, image_h, bbox):
    x_min, y_min, width, height = bbox
    x_center = (x_min + width / 2) / image_w
    y_center = (y_min + height / 2) / image_h
    width /= image_w
    height /= image_h
    return x_center, y_center, width, height

# Function to process COCO JSON and create YOLO labels
def convert_coco_to_yolo(split):
    with open(coco_annotations[split], "r") as f:
        coco_data = json.load(f)

    # Map category IDs to zero-based YOLO class IDs
    category_mapping = {cat["id"]: idx for idx, cat in
enumerate(coco_data["categories"])}

    # Process each image
    image_id_to_filename = {img["id"]: img["file_name"] for img in
coco_data["images"]}
    annotations_by_image = {}
```

```

# Group annotations by image
for ann in coco_data["annotations"]:
    image_id = ann["image_id"]
    if image_id not in annotations_by_image:
        annotations_by_image[image_id] = []
    annotations_by_image[image_id].append(ann)

# Convert annotations
for image_id, filename in image_id_to_filename.items():
    image_path = os.path.join(image_dirs[split], filename)
    label_path = os.path.join(output_dirs[split],
filename.replace(".jpg", ".txt").replace(".png", ".txt"))

    # Get image size
    with Image.open(image_path) as img:
        image_w, image_h = img.size

    # Write YOLO annotations
    with open(label_path, "w") as label_file:
        if image_id in annotations_by_image:
            for ann in annotations_by_image[image_id]:
                bbox = coco_to_yolo_bbox(image_w, image_h, ann["bbox"])
                class_id = category_mapping[ann["category_id"]]
                label_file.write(f"{class_id} {' '.join(map(str,
bbox)))}\n")

    print(f"Converted {split} set: {len(image_id_to_filename)} images
processed.")

# Run conversion for both train and val sets
convert_coco_to_yolo("train")
convert_coco_to_yolo("val")

print("COCO to YOLO conversion complete!")

```

How to Use

Ensure your dataset is structured as follows:

pgsql
Copy
Edit

```

dataset/
├── train/
│   ├── images/ (JPEG/PNG images)
│   ├── annotations/
│   │   └── meiobenthos_train_10groups_coco.json
├── val/
│   ├── images/
│   └── annotations/

```

```
| | | — meiobenthos_validate_10groups_coco.json
```

Run the script.

It will create labels/ folders inside dataset/train/ and dataset/val/.

Each image will have a corresponding .txt file with YOLO-format annotations.

Modify your data.yaml file to:

yaml

Copy

Edit

```
train: dataset/train/images
```

```
val: dataset/val/images
```

```
nc: 10 # Number of classes
```

```
names: ["nematode", "rotifer", "Testacea", "ciliate", "turbellarians",  
"annelida", "arthropoda", "gastrotricha", "tardigrada", "mollusca"]
```

Train YOLOv8 using the converted dataset!

Now you're ready to train YOLOv8! 🚀

Would you like help with visualizing YOLO labels to verify the conversion? 😊

III Sisend (prompt): Yes, this can be useful as well.

III Vastus:

Here's a YOLO label visualization script to verify that the COCO-to-YOLO conversion worked correctly. It overlays bounding boxes on images to check if annotations are aligned properly.

YOLO Label Visualization Script

python

Copy

Edit

```
import os
```

```
import cv2
```

```
import matplotlib.pyplot as plt
```

```
# Define dataset paths
```

```
image_dir = "dataset/train/images" # Change to "dataset/val/images" for  
validation set
```

```
label_dir = "dataset/train/labels" # Change to "dataset/val/labels" for  
validation set
```

```
# Class names (ensure they match `data.yaml`)
```

```
class_names = ["nematode", "rotifer", "Testacea", "ciliate",  
"turbellarians",
```

```
                "annelida", "arthropoda", "gastrotricha", "tardigrada",  
"mollusca"]
```

```

# Function to read YOLO labels and draw them on images
def visualize_yolo_labels(image_path, label_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    height, width, _ = image.shape

    # Read YOLO labels
    with open(label_path, "r") as file:
        lines = file.readlines()

    # Draw bounding boxes
    for line in lines:
        parts = line.strip().split()
        class_id = int(parts[0])
        x_center, y_center, box_width, box_height = map(float, parts[1:])

        # Convert YOLO format (normalized) to pixel values
        x_min = int((x_center - box_width / 2) * width)
        y_min = int((y_center - box_height / 2) * height)
        x_max = int((x_center + box_width / 2) * width)
        y_max = int((y_center + box_height / 2) * height)

        # Draw rectangle and label
        color = (255, 0, 0) # Red color for bounding boxes
        cv2.rectangle(image, (x_min, y_min), (x_max, y_max), color, 2)
        label_text = f"{class_names[class_id]} ({class_id})"
        cv2.putText(image, label_text, (x_min, y_min - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    # Show image
    plt.figure(figsize=(8, 6))
    plt.imshow(image)
    plt.axis("off")
    plt.show()

# Select a random image for visualization
import random
image_files = [f for f in os.listdir(image_dir) if f.endswith((' .jpg',
'.png'))]
random_image = random.choice(image_files)
image_path = os.path.join(image_dir, random_image)
label_path = os.path.join(label_dir, random_image.replace('.jpg',
'.txt').replace('.png', '.txt'))

# Visualize the chosen image
visualize_yolo_labels(image_path, label_path)

```

How It Works

Reads an image from dataset/train/images/

Loads its corresponding YOLO .txt file from dataset/train/labels/

Draws bounding boxes on the image using OpenCV


Displays the annotated image with class labels using Matplotlib

How to Use

Run the script to randomly select an image and display its YOLO annotations.

If needed, change the image_dir and label_dir to check the validation dataset.

Now You Can Verify Annotations! 




Let me know if you need additional adjustments or multiple images visualized in a loop! 

IV Sisend (prompt): Please modify YOLOv8 code so that the metrics would be also written in the log file.

IV Vastus:

Here's the modified YOLOv8 training script that logs metrics (mAP, precision, recall, loss) into a log file.

Modifications:

-  Logs training loss, mAP, precision, recall per epoch
-  Saves logs in training_log_<timestamp>.txt
-  Ensures logs are readable and formatted

Updated YOLOv8 Training Script

python

Copy

Edit

```
from ultralytics import YOLO
import datetime
import time

# Get current timestamp for logging
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
log_path = f"training_log_{current_date}.txt"

# Define YOLO model and dataset
model = YOLO("yolov8s.pt") # Load pre-trained YOLOv8 model (change "s" to
                             "m/l/x" if needed)

# Training parameters
epochs = 8
batch_size = 4
img_size = 640
data_yaml = "dataset/data.yaml" # Make sure this file exists and is
correctly configured

# Open log file for writing
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()}
    ---\n")

    for epoch in range(epochs):
        start_time = time.time()
```

```

# Train for one epoch
results = model.train(
    data=data_yaml,
    epochs=1, # Train one epoch at a time to log results
    batch=batch_size,
    imgsz=img_size,
    device="cuda" if model.device.type == "cuda" else "cpu",
    verbose=True
)

# Extract metrics from results
metrics = results[-1] # Get last training run results
map50 = metrics.box.map50
map50_95 = metrics.box.map
precision = metrics.box.precision
recall = metrics.box.recall
loss_cls = metrics.loss.cls
loss_box = metrics.loss.box
loss_obj = metrics.loss.obj

# Save model checkpoint
model_path = f"yolov8_epoch_{epoch + 1}.pt"
model.save(model_path)

# Log results
log_file.write(f"\nEpoch {epoch + 1} Metrics:\n")
log_file.write(f"mAP@50: {map50:.4f}\n")
log_file.write(f"mAP@50-95: {map50_95:.4f}\n")
log_file.write(f"Precision: {precision:.4f}\n")
log_file.write(f"Recall: {recall:.4f}\n")
log_file.write(f"Loss (Classification): {loss_cls:.4f}\n")
log_file.write(f"Loss (BBBox): {loss_box:.4f}\n")
log_file.write(f"Loss (Objectness): {loss_obj:.4f}\n")
log_file.write(f"Model saved: {model_path}\n")

end_time = time.time()
log_file.write(f"Time per epoch: {(end_time - start_time) / 60:.2f}
minutes\n")

log_file.write(f"\n--- Training Completed at {datetime.datetime.now()}
---\n")

print(f"Training completed. Logs saved to {log_path}")

```


How It Works:

Trains YOLOv8 one epoch at a time to extract per-epoch metrics.

Logs precision, recall, mAP, and losses into a structured log file.

Saves model checkpoints per epoch for later use.

Handles CUDA automatically (runs on GPU if available).

Now You Get Full Logging! 

Would you like to add validation logging after training?

V Sisend (prompt): Ok. I need all these four metrics: precision, recall, f1 and accuracy. And I would like to save these also in log_file.

Current code is this:

```
from ultralytics import YOLO
import datetime
import time
```

```
# Get current timestamp for logging
```

```
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
```

```
log_path = f"training_log_{current_date}.txt"
```

```
# Define YOLO model and dataset
```

```
model = YOLO("yolov8s.pt") # Load pre-trained YOLOv8 model (change "s" to "m/l/x" if needed)
```

```
# Training parameters
```

```
epochs = 1
```

```
batch_size = 4
```

```
img_size = 640
```

```
data_yaml = "dataset/data.yaml" # Make sure this file exists and is correctly configured
```

```
# Open log file for writing
```

```
with open(log_path, "a") as log_file:
```

```
    log_file.write(f"--- Training Started at {datetime.datetime.now()} ---\n")
```

```
for epoch in range(epochs):
```

```
    start_time = time.time()
```

```
    # Train for one epoch
```

```
    results = model.train(
```

```
        data=data_yaml,
```

```
        epochs=1, # Train one epoch at a time to log results
```

```
        batch=batch_size,
```

```
        imgsz=img_size,
```

```
        rect=True,
```

```
        device="cuda" if model.device.type == "cuda" else "cpu",
```

```
        verbose=True
```

```
)
```

```
# Extract metrics from results
```

```
metrics = results[-1] # Get last training run results
```

```
map50 = metrics.box.map50
```

```
map50_95 = metrics.box.map
```

```
precision = metrics.box.precision
```

```
recall = metrics.box.recall
```

```
loss_cls = metrics.loss.cls
```

```
loss_box = metrics.loss.box
```

```
loss_obj = metrics.loss.obj
```



```

# Save model checkpoint
model_path = f'yolov8_epoch_{epoch + 1}.pt"
model.save(model_path)

# Log results
log_file.write(f"\nEpoch {epoch + 1} Metrics:\n")
log_file.write(f"mAP@50: {map50:.4f}\n")
log_file.write(f"mAP@50-95: {map50_95:.4f}\n")
log_file.write(f"Precision: {precision:.4f}\n")
log_file.write(f"Recall: {recall:.4f}\n")
log_file.write(f"Loss (Classification): {loss_cls:.4f}\n")
log_file.write(f"Loss (BBox): {loss_box:.4f}\n")
log_file.write(f"Loss (Objectness): {loss_obj:.4f}\n")
log_file.write(f"Model saved: {model_path}\n")

end_time = time.time()
log_file.write(f"Time per epoch: {(end_time - start_time) / 60:.2f} minutes\n")

log_file.write(f"\n--- Training Completed at {datetime.datetime.now()} ---\n")

print(f"Training completed. Logs saved to {log_path}")

```

V Vastus:

Your current code logs precision and recall, but it's missing F1-score and accuracy.

In Ultralytics YOLO, F1-score can be computed from precision and recall:

$F1 = 2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$

However, accuracy isn't directly provided in YOLO metrics. You might approximate it based on predictions and ground truth, but for object detection, accuracy is not a common metric.

Fix: Adding F1-score and logging everything

Modify the logging part inside your loop like this:


python

Copy

Edit

```

# Compute F1-score
f1_score = (2 * precision * recall) / (precision + recall) if
(precision + recall) > 0 else 0

# Log results
log_file.write(f"\nEpoch {epoch + 1} Metrics:\n")
log_file.write(f"mAP@50: {map50:.4f}\n")
log_file.write(f"mAP@50-95: {map50_95:.4f}\n")
log_file.write(f"Precision: {precision:.4f}\n")
log_file.write(f"Recall: {recall:.4f}\n")
log_file.write(f"F1-score: {f1_score:.4f}\n") #  Adding F1-score
log_file.write(f"Loss (Classification): {loss_cls:.4f}\n")
log_file.write(f"Loss (BBox): {loss_box:.4f}\n")
log_file.write(f"Loss (Objectness): {loss_obj:.4f}\n")
log_file.write(f"Model saved: {model_path}\n")

```

Notes:

F1-score is now calculated and logged.

Accuracy is tricky for object detection (not directly meaningful), but if you have another way to compute it (like per-class accuracy from detections), you can add that manually.

Let me know if you need further clarification!

VI Sisend (prompt): Can you provide YOLO analogue also for this code:

```
import os
import torch
import torchvision
import torchvision.transforms as transforms
from PIL import Image, ImageDraw, ImageFont
import argparse
import numpy as np
from tqdm import tqdm

# Global defaults
TEST_IMAGE_FOLDER = "test_image"
DEFAULT_RESULTS_FOLDER = "test_image_results"
DEFAULT_MODEL_PATH =
"Models_for_eval/fasterrcnn_resnet50_epoch_6_26Feb2025.pth"

# Define class labels (update according to your dataset)
CLASS_NAMES = [
    "background", "nematode", "rotifer", "Testacea", "ciliate",
    "turbellarians", "annelida", "arthropoda", "gastrotricha",
    "tardigrada", "mollusca"
]

def load_model(model_path: str, num_classes: int, device: torch.device):
    """
    Loads the Faster R-CNN model with a custom predictor and trained weights.
    """
    weights = torchvision.models.detection.FasterRCNN_ResNet50_FPN_Weights.DEFAULT
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=weights)
    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)

    state_dict = torch.load(model_path, map_location=device)
    model.load_state_dict(state_dict)
    model.to(device)
    model.eval()
    return model
```

```
def draw_boxes(image: Image.Image, boxes, labels, scores, threshold: float = 0.5) -> Image.Image:
```

```
    """
    Draws only the best (highest scoring) bounding box and label on the provided image,
    if its confidence score is above the threshold.
```

```
    Args:
```

```
        image (Image.Image): The PIL image to annotate.
        boxes (np.array): Array of bounding boxes [x1, y1, x2, y2].
        labels (np.array): Array of label indices corresponding to CLASS_NAMES.
        scores (np.array): Array of confidence scores.
        threshold (float): Confidence threshold for drawing boxes.
```

```
    Returns:
```

```
        Image.Image: Annotated image.
```

```
    """
```

```
    draw = ImageDraw.Draw(image)
```

```
    try:
```

```
        # Increase the font size for better visibility
        font = ImageFont.truetype("arial.ttf", size=25)
```

```
    except IOError:
```

```
        font = ImageFont.load_default()
```

```
    # Filter indices with scores above threshold
```

```
    indices = np.where(scores >= threshold)[0]
```

```
    if len(indices) == 0:
```

```
        # No predictions over threshold; return the original image
        return image
```

```
    # Select the best prediction (highest score among the filtered ones)
```

```
    best_idx = indices[np.argmax(scores[indices])]
```

```
    box = boxes[best_idx]
```

```
    label = labels[best_idx]
```

```
    score = scores[best_idx]
```

```
    x1, y1, x2, y2 = box
```

```
    draw.rectangle([x1, y1, x2, y2], outline="red", width=3)
```

```
    text = f"{CLASS_NAMES[label]}: {score:.2f}"
```

```
    # Use textbbox to calculate text dimensions
```

```
    text_bbox = draw.textbbox((x1, y1), text, font=font)
```

```
    text_width = text_bbox[2] - text_bbox[0]
```

```
    text_height = text_bbox[3] - text_bbox[1]
```

```
    # Draw a background for text for better visibility
```

```
    text_background = [x1, y1 - text_height, x1 + text_width, y1]
```

```
    draw.rectangle(text_background, fill="red")
```

```
    draw.text((x1, y1 - text_height), text, fill="yellow", font=font)
```

```
    return image
```

```

def process_images(test_folder: str, results_folder: str, model, transform, device:
torch.device,
                    threshold: float = 0.5):
    """
    Processes images from the test folder, performs inference, draws bounding boxes,
    and saves the annotated images to the results folder.
    """
    os.makedirs(results_folder, exist_ok=True)
    # Filter for typical image file extensions
    image_files = [f for f in os.listdir(test_folder) if f.lower().endswith(('.png', '.jpg', '.jpeg',
    '.bmp'))]

    for image_name in os.listdir(TEST_IMAGE_FOLDER):
        image_path = os.path.join(TEST_IMAGE_FOLDER, image_name)
        # Skip if the path is not a file (i.e., it's a directory)
        if not os.path.isfile(image_path):
            continue

        image = Image.open(image_path).convert("RGB")

        # Preprocess image
        image_tensor = transform(image).unsqueeze(0).to(device)

        # Perform inference
        with torch.no_grad():
            prediction = model(image_tensor)

        # Extract results
        boxes = prediction[0]["boxes"].cpu().numpy()
        labels = prediction[0]["labels"].cpu().numpy()
        scores = prediction[0]["scores"].cpu().numpy()

        # Draw bounding boxes and labels on the image
        image_with_boxes = draw_boxes(image, boxes, labels, scores, threshold)

        # Save the annotated image
        result_path = os.path.join(results_folder, image_name)
        image_with_boxes.save(result_path)

    print(f"✅ All predictions saved in '{results_folder}'!")

def parse_args():
    parser = argparse.ArgumentParser(description="Object detection on test images using
Faster R-CNN")
    parser.add_argument("--test_folder", type=str, default=TEST_IMAGE_FOLDER,
                        help="Folder containing test images")

```

```

    parser.add_argument("--results_folder", type=str,
                        default=DEFAULT_RESULTS_FOLDER,
                        help="Folder to save result images")
    parser.add_argument("--model_path", type=str, default=DEFAULT_MODEL_PATH,
                        help="Path to trained model weights")
    parser.add_argument("--threshold", type=float, default=0.5, help="Confidence threshold
for detections")
    parser.add_argument("--num_classes", type=int, default=11, help="Number of classes
including background")
    return parser.parse_args()

```

```

def main():
    args = parse_args()
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    transform = transforms.Compose([transforms.ToTensor()])

    # Load the trained model
    model = load_model(args.model_path, args.num_classes, device)

    # Process the images and save predictions
    process_images(args.test_folder, args.results_folder, model, transform, device,
args.threshold)

if __name__ == "__main__":
    main()

```

VI Vastus:

Here's the YOLOv8 analogue of your Faster R-CNN script. This version:

- ✓ Uses Ultralytics YOLOv8 for object detection
- ✓ Loads a trained YOLO model instead of Faster R-CNN
- ✓ Runs inference on test images and saves annotated results
- ✓ Filters detections based on confidence threshold
- ✓ Uses the same class names and structure for compatibility

YOLOv8 Version

python
Copy
Edit

```

import os
import argparse
import torch
import numpy as np
from PIL import Image, ImageDraw, ImageFont
from ultralytics import YOLO
from tqdm import tqdm

```

```

# Global defaults
TEST_IMAGE_FOLDER = "test_image"
DEFAULT_RESULTS_FOLDER = "test_image_results"
DEFAULT_MODEL_PATH = "Models_for_eval/yolov8s_custom.pt" # Change to your
trained YOLOv8 model

# Define class labels (same as in Faster R-CNN version)
CLASS_NAMES = [
    "background", "nematode", "rotifer", "Testacea", "ciliate",
    "turbellarians", "annelida", "arthropoda", "gastrotricha",
    "tardigrada", "mollusca"]

def load_model(model_path: str, device: torch.device):
    """
    Loads the YOLOv8 model from the given path.
    """
    model = YOLO(model_path)
    model.to(device)
    return model

def draw_boxes(image: Image.Image, detections, threshold: float = 0.5) ->
Image.Image:
    """
    Draws bounding boxes and labels on the image based on YOLOv8
    detections.

    Args:
        image (PIL.Image): The image to annotate.
        detections: YOLOv8 inference results.
        threshold (float): Confidence threshold for drawing boxes.

    Returns:
        PIL.Image: The annotated image.
    """
    draw = ImageDraw.Draw(image)
    try:
        font = ImageFont.truetype("arial.ttf", size=25)
    except IOError:
        font = ImageFont.load_default()

    for detection in detections:
        x1, y1, x2, y2, confidence, class_id = detection
        if confidence < threshold:
            continue # Skip low-confidence detections

        label = CLASS_NAMES[int(class_id)] if int(class_id) <
len(CLASS_NAMES) else f"Class {int(class_id)}"
        text = f"{label}: {confidence:.2f}"

        draw.rectangle([x1, y1, x2, y2], outline="red", width=3)

```

```

        text_bbox = draw.textbbox((x1, y1), text, font=font)
        text_width = text_bbox[2] - text_bbox[0]
        text_height = text_bbox[3] - text_bbox[1]

        draw.rectangle([x1, y1 - text_height, x1 + text_width, y1],
            fill="red")
        draw.text((x1, y1 - text_height), text, fill="yellow", font=font)

    return image

def process_images(test_folder: str, results_folder: str, model, device:
torch.device, threshold: float = 0.5):
    """
    Runs YOLOv8 inference on images in the test folder and saves results.

    Args:
        test_folder (str): Path to the folder containing test images.
        results_folder (str): Path to save the annotated images.
        model: YOLO model.
        device: Torch device (CPU or GPU).
        threshold (float): Confidence threshold for detections.
    """
    os.makedirs(results_folder, exist_ok=True)
    image_files = [f for f in os.listdir(test_folder) if
f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp'))]

    for image_name in tqdm(image_files, desc="Processing images"):
        image_path = os.path.join(test_folder, image_name)
        image = Image.open(image_path).convert("RGB")

        # Perform inference with YOLO
        results = model(image)

        # Extract detections (boxes, confidence, class IDs)
        detections = results[0].boxes.data.cpu().numpy() # Shape:
(num_detections, 6)

        # Draw bounding boxes and labels
        image_with_boxes = draw_boxes(image, detections, threshold)

        # Save the annotated image
        result_path = os.path.join(results_folder, image_name)
        image_with_boxes.save(result_path)

    print(f"✅ All predictions saved in '{results_folder}'!")

def parse_args():
    parser = argparse.ArgumentParser(description="Object detection using
YOLOv8")
    parser.add_argument("--test_folder", type=str,
default=TEST_IMAGE_FOLDER, help="Folder containing test images")

```

```

    parser.add_argument("--results_folder", type=str,
default=DEFAULT_RESULTS_FOLDER, help="Folder for output images")
    parser.add_argument("--model_path", type=str,
default=DEFAULT_MODEL_PATH, help="Path to trained YOLOv8 model")
    parser.add_argument("--threshold", type=float, default=0.5,
help="Confidence threshold for detections")
    return parser.parse_args()

def main():
    args = parse_args()
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Load trained YOLO model
    model = load_model(args.model_path, device)

    # Process images and save predictions
    process_images(args.test_folder, args.results_folder, model, device,
args.threshold)

if __name__ == "__main__":
    main()

```

💡 Key Differences from Faster R-CNN Version

Model	Faster R-CNN (Torchvision)	YOLOv8 (Ultralytics)
Bounding Box Extraction	prediction[0]["boxes"]	results[0].boxes.data
Feature	Faster R-CNN	YOLOv8
Processing Speed	Slower	Faster 🚀
Inference Call	model(image_tensor)	model(image)

- ✓ What This YOLO Version Does
- ✓ Loads a trained YOLOv8 model
- ✓ Runs inference on test images
- ✓ Filters detections based on confidence threshold
- ✓ Draws bounding boxes with class names & confidence scores
- ✓ Saves annotated images

Now you can run YOLOv8 inference on your dataset with the same workflow as your Faster R-CNN version, but faster and more efficient. Let me know if you need further modifications! 🚀

VII Sisend (prompt): I still get this error: line 38, in <module> metrics = results if isinstance(results, dict) else results[-1] TypeError: 'DetMetrics' object is not subscriptable

VII Vastus:

The error occurs because Ultralytics YOLOv8 returns a DetMetrics object instead of a list. Unlike lists or dictionaries, a DetMetrics object cannot be accessed using results[-1].



Change:

python
Copy
Edit

```
metrics = results if isinstance(results, dict) else results[-1]
```

To:

python
Copy
Edit

```
metrics = results # Directly use the DetMetrics object
```

VIII Sisend (prompt): It seems, I'm not calculating validation loss (or I'm not printing the result after each epoch). Since I'm saving only the best model, I would like to see what the validation loss of each epoch is. Let's add this also to log.

```
from ultralytics import YOLO
import datetime
import time
import shutil
import os
import pandas as pd # Import for analyzing results.csv
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```

# 🕒 Get current timestamp for logging
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
log_path = f"training_log_{current_date}.txt"

# 🎯 Load YOLO model
model = YOLO("yolo11s.pt")

# 🚀 Training parameters
epochs = 2 # Adjust as needed
batch_size = 4
img_size = 1024
data_yaml = "/content/drive/MyDrive/meiobenthos/YOLO11/dataset/data.yaml" # Ensure
this file exists and is correctly configured

# YOLO save directory
project_dir = "/content/drive/MyDrive/meiobenthos/YOLO11/runs/detect"
train_folder_name = f"train_1class_{current_date}"

# 📝 Open log file for writing
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()} ---\n")

start_time = time.time() # Start timer

# 🔥 Train YOLO model
results = model.train(
    data=data_yaml,
    epochs=epochs, # Train for all epochs at once
    batch=batch_size,
    imgsz=img_size,
    device="cuda" if model.device.type == "cuda" else "cpu",
    verbose=True,
    name = train_folder_name
)

# ⌚ Compute training time
end_time = time.time()
log_file.write(f"Total Training Time: {(end_time - start_time) / 60:.2f} minutes\n")

# ✅ Save best model (from YOLO's default save path)
best_model_path = f"{project_dir}/{train_folder_name}/weights/best.pt"
if os.path.exists(best_model_path):
    best_model_save_path = f"best_model_{current_date}.pt"
    shutil.copy(best_model_path, best_model_save_path)
    log_file.write(f"Best model saved as: {best_model_save_path}\n")
    print(f"✅ Best model copied to: {best_model_save_path}")
else:
    print("⚠️ Best model not found in runs/detect/train/weights/.")

# 🏆 Identify Best Model Epoch from results.csv

```

```

results_csv_path = f"{project_dir}/{train_folder_name}/results.csv"

if os.path.exists(results_csv_path):
    df = pd.read_csv(results_csv_path)

    # Find the epoch with the highest mAP@50 (best model criteria)
    best_epoch_index = df["metrics/mAP50(B)"].idxmax()
    best_epoch = df.loc[best_epoch_index]

    best_epoch_num = int(best_epoch["epoch"]) # Get epoch number
    best_map50 = best_epoch["metrics/mAP50(B)"]
    best_map50_95 = best_epoch["metrics/mAP50-95(B)"]
    best_precision = best_epoch["metrics/precision(B)"]
    best_recall = best_epoch["metrics/recall(B)"]

    # Compute F1-score for the best model
    best_f1_score = (2 * best_precision * best_recall) / (best_precision + best_recall) if
(best_precision + best_recall) > 0 else 0

    # 📝 Log Best Model Metrics
    with open(log_path, "a") as log_file:
        log_file.write("\n--- Best Model Identified ---\n")
        log_file.write(f"Best Model Epoch: {best_epoch_num}\n")
        log_file.write(f"mAP@50: {best_map50:.4f}\n")
        log_file.write(f"mAP@50-95: {best_map50_95:.4f}\n")
        log_file.write(f"Precision: {best_precision:.4f}\n")
        log_file.write(f"Recall: {best_recall:.4f}\n")
        log_file.write(f"F1-score: {best_f1_score:.4f}\n")

    print(f"✅ Best model was from epoch {best_epoch_num} with mAP@50:
{best_map50:.4f}")


else:
    print("⚠️ Results CSV not found! Could not determine best epoch.")

print(f"✅ Training completed! Logs saved to {log_path}")

```

VIII Vastus: You're almost there! To log validation loss (or other metrics like classification loss, box loss, etc.) after each epoch, we can extract it directly from the results.csv file that YOLO creates after training. You already have the logic to read that CSV for the best model, so you can extend it to log the validation losses for each epoch too.

✅ Here's an Updated Version of Your Code
I added:

Extraction and logging of validation losses (box loss, objectness loss, class loss).
A per-epoch loop that writes the validation loss to the log file.
You'll find comments marked with #  on the new bits!

python

Copy

Edit

```
from ultralytics import YOLO
import datetime
import time
import shutil
import os
import pandas as pd # Import for analyzing results.csv

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# 🕒 Get current timestamp for logging
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
log_path = f"training_log_{current_date}.txt"

# 🎯 Load YOLO model
model = YOLO("yolo11s.pt")

# 🚀 Training parameters
epochs = 2 # Adjust as needed
batch_size = 4
img_size = 1024
data_yaml = "/content/drive/MyDrive/meiobenthos/YOLO11/dataset/data.yaml"
# Ensure this file exists and is correctly configured

# YOLO save directory
project_dir = "/content/drive/MyDrive/meiobenthos/YOLO11/runs/detect"
train_folder_name = f"train_1class_{current_date}"

# 📝 Open log file for writing
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()}
    ---\n")

    start_time = time.time() # Start timer

# 🔥 Train YOLO model
results = model.train(
    data=data_yaml,
    epochs=epochs,
    batch=batch_size,
    imgsz=img_size,
    device="cuda" if model.device.type == "cuda" else "cpu",
    verbose=True,
    name=train_folder_name
)

# ⌚ Compute training time
end_time = time.time()
```

```

log_file.write(f"Total Training Time: {(end_time - start_time) /
60:.2f} minutes\n")

# ✅ Save best model (from YOLO's default save path)
best_model_path = f"{project_dir}/{train_folder_name}/weights/best.pt"
if os.path.exists(best_model_path):
    best_model_save_path = f"best_model_{current_date}.pt"
    shutil.copy(best_model_path, best_model_save_path)
    log_file.write(f"Best model saved as: {best_model_save_path}\n")
    print(f"✅ Best model copied to: {best_model_save_path}")
else:
    print("⚠️ Best model not found in runs/detect/train/weights/.")

# 🏆 Analyze Results CSV
results_csv_path = f"{project_dir}/{train_folder_name}/results.csv"

if os.path.exists(results_csv_path):
    df = pd.read_csv(results_csv_path)

    # 🆕 Log validation loss per epoch
    with open(log_path, "a") as log_file:
        log_file.write("\n--- Per-Epoch Validation Losses ---\n")

        for idx, row in df.iterrows():
            epoch_num = int(row["epoch"])
            val_box_loss = row.get("val/box_loss", None)
            val_cls_loss = row.get("val/cls_loss", None)
            val_dfl_loss = row.get("val/dfl_loss", None) # DFL:
Distribution Focal Loss (YOLOv8+)

            # Optional: objectness loss if available
            val_obj_loss = row.get("val/obj_loss", None) # May not exist
in newer YOLO versions

            log_file.write(
                f"Epoch {epoch_num}: "
                f"val/box_loss={val_box_loss:.4f}, "
                f"val/cls_loss={val_cls_loss:.4f}, "
                f"val/dfl_loss={val_dfl_loss:.4f}"
            )
            if val_obj_loss is not None:
                log_file.write(f", val/obj_loss={val_obj_loss:.4f}")

            log_file.write("\n")

# ✅ Find the best model based on mAP@50
best_epoch_index = df["metrics/mAP50(B)"].idxmax()
best_epoch = df.loc[best_epoch_index]

best_epoch_num = int(best_epoch["epoch"])
best_map50 = best_epoch["metrics/mAP50(B)"]
best_map50_95 = best_epoch["metrics/mAP50-95(B)"]

```

```

best_precision = best_epoch["metrics/precision(B)"]
best_recall = best_epoch["metrics/recall(B)"]

# Compute F1-score for the best model
best_f1_score = (
    (2 * best_precision * best_recall) / (best_precision + best_recall)
    if (best_precision + best_recall) > 0
    else 0
)

# 📝 Log Best Model Metrics
with open(log_path, "a") as log_file:
    log_file.write("\n--- Best Model Identified ---\n")
    log_file.write(f"Best Model Epoch: {best_epoch_num}\n")
    log_file.write(f"mAP@50: {best_map50:.4f}\n")
    log_file.write(f"mAP@50-95: {best_map50_95:.4f}\n")
    log_file.write(f"Precision: {best_precision:.4f}\n")
    log_file.write(f"Recall: {best_recall:.4f}\n")
    log_file.write(f"F1-score: {best_f1_score:.4f}\n")

    print(f"✅ Best model was from epoch {best_epoch_num} with mAP@50: {best_map50:.4f}")

else:
    print("⚠️ Results CSV not found! Could not determine best epoch or validation losses.")

print(f"✅ Training completed! Logs saved to {log_path}")

```

✅ Key Additions:

1. Per-Epoch Validation Loss Logging

python

Copy

Edit

```
log_file.write("\n--- Per-Epoch Validation Losses ---\n")
```

```

for idx, row in df.iterrows():
    epoch_num = int(row["epoch"])
    val_box_loss = row.get("val/box_loss", None)
    val_cls_loss = row.get("val/cls_loss", None)
    val_dfl_loss = row.get("val/dfl_loss", None)

    # Optional: objectness loss
    val_obj_loss = row.get("val/obj_loss", None)

    log_file.write(
        f"Epoch {epoch_num}: "
        f"val/box_loss={val_box_loss:.4f}, "
        f"val/cls_loss={val_cls_loss:.4f}, "
        f"val/dfl_loss={val_dfl_loss:.4f}"
    )
    if val_obj_loss is not None:

```

```

        log_file.write(f", val/obj_loss={val_obj_loss:.4f}")

    log_file.write("\n")
2. Where to Find These Losses
YOLOv11 and YOLOv8 log these in results.csv by default.
The columns include:
val/box_loss: Bounding box regression loss.
val/cls_loss: Classification loss.
val/df1_loss: Distribution Focal Loss (for precise box regression).
val/obj_loss: Objectness loss (in some YOLO versions).

```

✓ Optional: Print to Console Too

If you want to print per-epoch validation loss while training, you could add:

```

python
Copy
Edit
print(
    f"Epoch {epoch_num} - "
    f"Box Loss: {val_box_loss:.4f}, "
    f"Cls Loss: {val_cls_loss:.4f}, "
    f"DFL Loss: {val_dfl_loss:.4f}"
)

```

Right inside the loop!

✓ Next Steps?

Plot Loss Curves in Colab for easier visual insights.
 Early Stopping or LR schedulers to avoid overfitting.
 Export models to ONNX or TensorRT for deployment.
 Let me know if you want code for plotting, early stopping, or model exporting!

IX Sisend (prompt): I would like to save trained models and all the files the script generates during training and training log in content/drive/MyDrive/meiobenthos/YOLO11.

```

from ultralytics import YOLO
import datetime
import time
import shutil
import os
import pandas as pd # Import for analyzing results.csv

```

```

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

```

```

# 🕒 Get current timestamp for logging
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
log_path = f"training_log_{current_date}.txt"

```

```

# 🎯 Load YOLO model
model = YOLO("yolo11m.pt")

```

```

# 🚀 Training parameters
epochs = 30 # Adjust as needed
batch_size = 4
img_size = 1024
data_yaml = "/content/drive/MyDrive/meiobenthos/YOLO11/dataset/data.yaml" # Ensure
this file exists and is correctly configured

# YOLO save directory
project_dir = "/content/drive/MyDrive/meiobenthos/YOLO11/runs/detect"
train_folder_name = f"train_1class{current_date}"

# 📝 Open log file for writing
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()} ---\n")

start_time = time.time() # Start timer

# 🔥 Train YOLO model
results = model.train(
    data=data_yaml,
    epochs=epochs, # Train for all epochs at once
    batch=batch_size,
    imgsz=img_size,
    device=0, #"cuda" if model.device.type == "cuda" else "cpu",
    verbose=True,
    name = train_folder_name
)

# ⌚ Compute training time
end_time = time.time()
log_file.write(f"Total Training Time: {(end_time - start_time) / 60:.2f} minutes\n")

# ✅ Save best model (from YOLO's default save path)
best_model_path = f"{project_dir}/{train_folder_name}/weights/best.pt"
if os.path.exists(best_model_path):
    best_model_save_path = f"best_model_{current_date}.pt"
    shutil.copy(best_model_path, best_model_save_path)
    log_file.write(f"Best model saved as: {best_model_save_path}\n")
    print(f"✅ Best model copied to: {best_model_save_path}")
else:
    print("⚠️ Best model not found in runs/detect/train/weights/.")

# 🏆 Identify Best Model Epoch from results.csv
results_csv_path = f"{project_dir}/{train_folder_name}/results.csv"

if os.path.exists(results_csv_path):
    df = pd.read_csv(results_csv_path)

    # Find the epoch with the highest mAP@50 (best model criteria)
    best_epoch_index = df["metrics/mAP50(B)"].idxmax()

```



```

best_epoch = df.loc[best_epoch_index]

best_epoch_num = int(best_epoch["epoch"]) # Get epoch number
best_map50 = best_epoch["metrics/mAP50(B)"]
best_map50_95 = best_epoch["metrics/mAP50-95(B)"]
best_precision = best_epoch["metrics/precision(B)"]
best_recall = best_epoch["metrics/recall(B)"]

# Compute F1-score for the best model
best_f1_score = (2 * best_precision * best_recall) / (best_precision + best_recall) if
(best_precision + best_recall) > 0 else 0

# 📝 Log Best Model Metrics
with open(log_path, "a") as log_file:
    log_file.write("\n--- Best Model Identified ---\n")
    log_file.write(f"Best Model Epoch: {best_epoch_num}\n")
    log_file.write(f"mAP@50: {best_map50:.4f}\n")
    log_file.write(f"mAP@50-95: {best_map50_95:.4f}\n")
    log_file.write(f"Precision: {best_precision:.4f}\n")
    log_file.write(f"Recall: {best_recall:.4f}\n")
    log_file.write(f"F1-score: {best_f1_score:.4f}\n")

print(f"✅ Best model was from epoch {best_epoch_num} with mAP@50:
{best_map50:.4f}")

else:
    print("⚠️ Results CSV not found! Could not determine best epoch.")

print(f"✅ Training completed! Logs saved to {log_path}")

```

IX Vastus:

Great job laying out your script! Let's tweak it so everything—trained models, logs, and all output files—are saved in your /content/drive/MyDrive/meiobenthos/YOLO11 folder on Google Drive.

Here's what we'll address:

Training logs: Your log_path should be in the target directory.

YOLO training outputs (runs/detect/...): We'll force YOLO to save inside your Drive folder.

Best model save: Already good, but we can clean it up.

Optional cleanup: Avoid leaving stray files in /content if you want.

✅ Modified Code with Full Saving to Google Drive

python

Copy

Edit

```

from ultralytics import YOLO
import datetime
import time
import shutil
import os

```

```

import pandas as pd # For analyzing results.csv

# 🚗 Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# 🕒 Timestamp for unique run folder/logs
current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

# 🌟 Base directory where everything will be saved
base_dir = "/content/drive/MyDrive/meiobenthos/YOLO11"

# 📁 Log file path (save logs inside the base directory)
log_path = os.path.join(base_dir, f"training_log_{current_date}.txt")

# 🎯 Load YOLO model (change path to your checkpoint if resuming)
# model = YOLO(f"{base_dir}/best_model_2025-03-12_13-19-23.pt")
model = YOLO("yolo11m.pt") # or your pretrained checkpoint

# 🚀 Training hyperparameters
epochs = 30
batch_size = 4
img_size = 1024
data_yaml = f"{base_dir}/dataset/data.yaml" # Make sure this exists and is
configured

# 📁 YOLO save directory (force all runs here)
project_dir = os.path.join(base_dir, "runs", "detect")
train_folder_name = f"train_1class_{current_date}"

# 📝 Open log file for writing
with open(log_path, "a") as log_file:
    log_file.write(f"--- Training Started at {datetime.datetime.now()}
    ---\n")

    start_time = time.time() # Start timer

# 🔥 Train YOLO model (save outputs to Google Drive)
results = model.train(
    data=data_yaml,
    epochs=epochs,
    batch=batch_size,
    imgsiz=img_size,
    device=0, # Force GPU (T4) usage. 0 = first CUDA device.
    verbose=True,
    project=project_dir, # 💣 Save everything under Drive folder!
    name=train_folder_name, # Folder name inside project_dir
)

# ⌚ Compute and log total training time
end_time = time.time()
elapsed_time = (end_time - start_time) / 60

```

```

log_file.write(f"Total Training Time: {elapsed_time:.2f} minutes\n")

# ✅ Save best model (copy to root of base_dir)
best_model_src = f"{project_dir}/{train_folder_name}/weights/best.pt"
best_model_dest = os.path.join(base_dir,
f"best_model_{current_date}.pt")

if os.path.exists(best_model_src):
    shutil.copy(best_model_src, best_model_dest)
    log_file.write(f"✅ Best model copied to: {best_model_dest}\n")
    print(f"✅ Best model copied to: {best_model_dest}")
else:
    log_file.write("⚠️ Best model not found! Check training folder.\n")
    print("⚠️ Best model not found!")

# 🏆 Analyze training results (from results.csv)
results_csv_path = f"{project_dir}/{train_folder_name}/results.csv"

if os.path.exists(results_csv_path):
    df = pd.read_csv(results_csv_path)

    # Find the epoch with the highest mAP@50 (best model criteria)
    best_epoch_index = df["metrics/mAP50(B)"].idxmax()
    best_epoch = df.loc[best_epoch_index]

    best_epoch_num = int(best_epoch["epoch"])
    best_map50 = best_epoch["metrics/mAP50(B)"]
    best_map50_95 = best_epoch["metrics/mAP50-95(B)"]
    best_precision = best_epoch["metrics/precision(B)"]
    best_recall = best_epoch["metrics/recall(B)"]

    # Compute F1-score for the best model
    best_f1_score = (
        (2 * best_precision * best_recall) / (best_precision + best_recall)
        if (best_precision + best_recall) > 0
        else 0
    )

    # 📝 Log Best Model Metrics
    with open(log_path, "a") as log_file:
        log_file.write("\n--- Best Model Identified ---\n")
        log_file.write(f"Best Model Epoch: {best_epoch_num}\n")
        log_file.write(f"mAP@50: {best_map50:.4f}\n")
        log_file.write(f"mAP@50-95: {best_map50_95:.4f}\n")
        log_file.write(f"Precision: {best_precision:.4f}\n")
        log_file.write(f"Recall: {best_recall:.4f}\n")
        log_file.write(f"F1-score: {best_f1_score:.4f}\n")

    print(f"✅ Best model was from epoch {best_epoch_num} with mAP@50: {best_map50:.4f}")
else:
    print("⚠️ Results CSV not found! Could not determine best epoch.")

```

```
print(f"✅ Training completed! Logs saved to: {log_path}")
```

🔧 Changes Made for You

What	Old Code	New Code
Log file location	"training_log_..."	os.path.join(base_dir, "training_log_...")
YOLO output folder	/content/runs/detect (default)	project=project_dir under your Drive folder
Copy best model	to local Colab /content	to your Drive best_model_{date}.pt
Training folder name	train_1class{date}	train_1class_{date} (for readability)

✅ Where Files Will Be Saved

What	Where
------	-------

YOLO runs

/content/drive/MyDrive/meiobenthos/YOLO11/runs/detect/train_1class_{timestamp}

Best model /content/drive/MyDrive/meiobenthos/YOLO11/best_model_{timestamp}.pt

Log file /content/drive/MyDrive/meiobenthos/YOLO11/training_log_{timestamp}.txt

✅ Optional: Clean Up /content

If you're running a long session and want to clear /content:

python

Copy

Edit

```
!rm -rf /content/runs
```

Bonus Tip!

If you resume training, make sure to:

python

Copy

Edit

```
model.train(resume=True)
```

Or, if starting from a checkpoint, lower the learning rate:

python

Copy

Edit

```
model = YOLO('path_to_best_model.pt')
```

```
model.train(lr0=0.0001, ...)
```

Let me know if you want to automatically sync folders or handle older runs cleanup!

Happy to help optimize further! 😊

X Sisend (prompt): I would like to have a combined report also for YOLO models results.

This is my current script:

```
import os
import argparse
import torch
import numpy as np
import pandas as pd
from PIL import Image, ImageDraw, ImageFont
from ultralytics import YOLO
from tqdm import tqdm
import datetime

current_date = datetime.datetime.now().strftime("%Y-%m-%d_%H")

# Global defaults
TEST_IMAGE_FOLDER = "test_image"
CROPPED_FOLDER = "results_cropped_images"
RESULTS_FOLDER_1 = f"results_model1_{current_date}"
RESULTS_FOLDER_2 = f"results_model2_{current_date}"
REPORT_FILE_1 = f"report_model1_{current_date}.xlsx"
REPORT_FILE_2 = f"report_model2_{current_date}.xlsx"

CLASS_NAMES_1 = ["meiofauna"]
CLASS_NAMES_2 = [
    "nematode", "rotifer", "Testacea", "ciliate",
    "turbellarians", "annelida", "arthropoda", "gastrotricha", "tardigrada"
]

# Load YOLO model
def load_model(model_path: str, device: torch.device):
    model = YOLO(model_path)
    model.to(device)
    return model

# Draw bounding boxes
def draw_boxes(image: Image.Image, boxes, labels, scores, threshold: float, class_names) -> Image.Image:
    draw = ImageDraw.Draw(image)
    try:
        font = ImageFont.truetype("arial.ttf", size=35)
    except IOError:
        font = ImageFont.load_default()

    for box, label, score in zip(boxes, labels, scores):
        if score >= threshold:
            x1, y1, x2, y2 = box
            draw.rectangle([x1, y1, x2, y2], outline="red", width=3)
            text = f"{class_names[label]}: {score:.2f}"
```

```

text_bbox = draw.textbbox((x1, y1), text, font=font)
text_width = text_bbox[2] - text_bbox[0]
text_height = text_bbox[3] - text_bbox[1]
text_background = [x1, y1, x1 + text_width, y1 + text_height]
draw.rectangle(text_background, fill="red")
draw.text((x1, y1), text, fill="yellow", font=font)

```

```

return image

```

```

# Process images with Model 1

```

```

def process_model1(images, model, device):
    os.makedirs(CROPPED_FOLDER, exist_ok=True)
    os.makedirs(RESULTS_FOLDER_1, exist_ok=True)
    results = []
    for img_name in tqdm(images, desc="Model 1 Processing"):
        image = Image.open(os.path.join(TEST_IMAGE_FOLDER,
img_name)).convert("RGB")
        preds = model(image)[0].boxes.data.cpu().numpy()
        boxes = preds[:, :4]
        scores = preds[:, 4]
        labels = preds[:, 5].astype(int)
        detected_classes = []
        image_with_boxes = draw_boxes(image.copy(), boxes, labels, scores, 0.5,
CLASS_NAMES_1)
        image_with_boxes.save(os.path.join(RESULTS_FOLDER_1,
f"annotated_{img_name}"))
        for x1, y1, x2, y2, conf, cls in preds:
            alert = f"{conf:.2f}" if conf < 0.75 else ""
            detected_classes.append(CLASS_NAMES_1[int(cls)])
            results.append([img_name, ", ".join(detected_classes), alert])
    pd.DataFrame(results, columns=["Image Name", "Class",
"Alert"]).to_excel(REPORT_FILE_1, index=False)

```

```

# Process images with Model 2

```

```

def process_model2(images, model, device):
    os.makedirs(RESULTS_FOLDER_2, exist_ok=True)
    results = []
    for img_name in tqdm(images, desc="Model 2 Processing"):
        image = Image.open(os.path.join(TEST_IMAGE_FOLDER,
img_name)).convert("RGB")
        preds = model(image)[0].boxes.data.cpu().numpy()
        boxes = preds[:, :4]
        scores = preds[:, 4]
        labels = preds[:, 5].astype(int)
        detected_classes = []
        image_with_boxes = draw_boxes(image.copy(), boxes, labels, scores, 0.5,
CLASS_NAMES_2)
        image_with_boxes.save(os.path.join(RESULTS_FOLDER_2,
f"annotated_{img_name}"))

```

```

all_classes = []
alert_values = []
for score, label in zip(scores, labels):
    all_classes.append(CLASS_NAMES_2[label])
    if score < 0.75:
        alert_values.append(f"{score:.2f}")
    alert = ", ".join(alert_values) if alert_values else ""
    results.append([img_name, ", ".join(all_classes), alert])
pd.DataFrame(results, columns=["Image Name", "Classes",
"Alert"]).to_excel(REPORT_FILE_2, index=False)

# Main function
def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model1 = load_model("Models_for_test/best_model1_1class.pt", device)
    model2 = load_model("Models_for_test/best_model2_9classes.pt", device)
    images = [f for f in os.listdir(TEST_IMAGE_FOLDER) if f.endswith(('.jpg', '.JPG',
'.png'))]
    process_model1(images, model1, device)
    process_model2(images, model2, device)

if __name__ == "__main__":
    main()

```

X Vastus:

Great! Let's enhance your YOLO script to:

- ✓ Generate a combined report similar to the one for the Faster R-CNN.
- ✓ Keep saving annotated images for both models, as before.
- ✓ Unify columns in the final report:

Image Name

Model 1: Classes, Model 1: Alert

Model 2: Classes, Model 2: Alert

- ✓ Changes Summary
- We'll:

Refactor both `process_model1()` and `process_model2()` to return dictionaries like:
`{ "img1.jpg": { "classes": "...", "alert": "..."}, ... }`

Build a unified DataFrame in `main()` and save as `final_combined_yolo_report.xlsx`

- ✓ Updated Script Snippet
- Here's the adjusted part of your code:

python

Copy

Edit

```
# Unified processing and return report data
def process_yolo_model(images, model, device, class_names, result_folder,
threshold=0.5):
    os.makedirs(result_folder, exist_ok=True)
    report_data = {}
    for img_name in tqdm(images, desc=f"Processing {result_folder}"):
        image = Image.open(os.path.join(TEST_IMAGE_FOLDER,
img_name)).convert("RGB")
        preds = model(image)[0].boxes.data.cpu().numpy()
        boxes = preds[:, :4]
        scores = preds[:, 4]
        labels = preds[:, 5].astype(int)
        image_with_boxes = draw_boxes(image.copy(), boxes, labels, scores,
threshold, class_names)
        image_with_boxes.save(os.path.join(result_folder,
f"annotated_{img_name}"))

        all_classes = []
        alert_values = []
        for score, label in zip(scores, labels):
            if label < len(class_names):
                all_classes.append(class_names[label])
            else:
                all_classes.append(f"Unknown({label})")
            if score < 0.75:
                alert_values.append(f"{score:.2f}")

        report_data[img_name] = {
            "Classes": ", ".join(all_classes) if all_classes else "None",
            "Alert": ", ".join(alert_values) if alert_values else ""
        }
    return report_data
```

✓ Update to main():

python

Copy

Edit

```
def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model1 = load_model("Models_for_test/best_model1_1class.pt", device)
    model2 = load_model("Models_for_test/best_model2_9classes.pt", device)

    images = [f for f in os.listdir(TEST_IMAGE_FOLDER) if
f.endswith(('.jpg', '.JPG', '.png'))]

    data1 = process_yolo_model(images, model1, device, CLASS_NAMES_1,
RESULTS_FOLDER_1)
```



```

data2 = process_yolo_model(images, model2, device, CLASS_NAMES_2,
RESULTS_FOLDER_2)

all_files = sorted(set(data1.keys()) | set(data2.keys()))
combined = []
for file in all_files:
    row = {
        "Image Name": file,
        "Model 1: Classes": data1.get(file, {}).get("Classes", "None"),
        "Model 1: Alert": data1.get(file, {}).get("Alert", ""),
        "Model 2: Classes": data2.get(file, {}).get("Classes", "None"),
        "Model 2: Alert": data2.get(file, {}).get("Alert", "")
    }
    combined.append(row)

df_combined = pd.DataFrame(combined)
os.makedirs("final_results", exist_ok=True)
report_path = os.path.join("final_results",
f"final_combined_yolo_report_{current_date}.xlsx")
df_combined.to_excel(report_path, index=False)
print(f"✅ Final YOLO combined report saved to {report_path}")

```

Would you like:

- to also include confidence scores per class in the report?
- a side-by-side comparison image between model 1 and model 2 predictions for each image?

Let me know!