YOLO Accuracy Calculator

Complete Line-by-Line Code Explanation

Using Ultralytics Mathematical Formulas

File:	test_accuracy_with_labels.py
Purpose:	Calculate model accuracy using exact YOLO formulas
Source:	ultralytics/utils/metrics.py
Author:	Generated for IEEE Project

Table of Contents

1. Import Statements

These are the libraries required to run the accuracy calculator.

```
from ultralytics import YOLO
```

■ Imports the YOLO class from Ultralytics library. This is the main class used to load and run YOLO models for object detection.

```
import numpy as np
```

■ NumPy is used for numerical computations - arrays, mathematical operations like mean, sum, cumsum, etc. Essential for metric calculations.

```
import os
```

■ Provides functions for interacting with the operating system - checking if files exist, joining paths, creating directories.

```
import glob
```

■ Used to find all files matching a pattern (e.g., all .jpg files in a folder).

```
import yaml
```

■ YAML parser to read the data.yaml configuration file that contains dataset paths and class names.

```
from pathlib import Path
```

■ Modern way to handle file paths. Used to extract filename without extension.

```
from collections import defaultdict
```

■ Dictionary that provides default values. Useful for counting and grouping.

```
import cv2
```

■ OpenCV library for reading images, getting dimensions, and saving annotated results.

2. Configuration Section

User-editable settings at the top of the file.

```
MODEL_PATH = "best.pt"
```

- ■ Path to your trained YOLO model file. "best.pt" is the model with best fitness during training.

```
DATA_YAML = "data.yaml"
```

- ■ Path to your dataset configuration file. Contains paths to images/labels and class names.

```
CONF_THRESHOLD = 0.001
```

- ■ Confidence threshold for detections. Set very low (0.001) to get ALL possible detections during validation. This matches YOLO's default validation behavior.

```
IOU_THRESHOLD = 0.5
```

- ■ IoU threshold for matching predictions to ground truth. A prediction is "correct" if IoU $\geq$ 0.5 with a ground truth box.

```
SAVE_RESULTS = True
```

- ■ Whether to save annotated images showing detections. Useful for visual verification.

```
OUTPUT_FOLDER = "accuracy_results"
```

- ■ Folder where results (annotated images, report) will be saved.

3. IoU (Intersection over Union) Function

This function calculates how much two bounding boxes overlap. It's the fundamental metric for object detection.

Mathematical Formula:

IoU = Area of Intersection / Area of Union

Source: ultralytics/utils/metrics.py, lines 56-76

```
def box_iou(box1, box2, eps=1e-7):
```

■ Function definition. Takes two boxes and epsilon (small number to prevent division by zero).

```
# Intersection area
```

■ Comment indicating we're calculating where the boxes overlap.

```
inter_x1 = max(box1[0], box2[0])
```

■ Left edge of intersection = rightmost of the two left edges.

```
inter_y1 = max(box1[1], box2[1])
```

■ Top edge of intersection = bottommost of the two top edges.

```
inter_x2 = min(box1[2], box2[2])
```

■ Right edge of intersection = leftmost of the two right edges.

```
inter_y2 = min(box1[3], box2[3])
```

■ Bottom edge of intersection = topmost of the two bottom edges.

```
inter_area = max(0, inter_x2 - inter_x1) * max(0, inter_y2 - inter_y1)
```

■ Intersection area = width × height. max(0, ...) ensures no negative values if boxes don't overlap.

```
# Union area
```

■ Comment indicating we're calculating total area covered by both boxes.

```
box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
```

■ Area of first box = width × height.

```
box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])
```

■ Area of second box = width × height.

```
union_area = box1_area + box2_area - inter_area
```

■ Union = sum of areas minus intersection (to avoid counting overlap twice).

```
return inter_area / (union_area + eps)
```

■ Final IoU = intersection / union. Add eps to prevent division by zero.

Visual Representation:

```
■■■■■■■■■■■■■■■■■■■
■     Box 1       ■
■   ■■■■■■■■■■■■■■■■■■
■   ■ INTER-  ■   ■  ■
■   ■ SECTION ■   ■  ■
■■■■■■■■■■■■■■■   ■  ■
    ■     Box 2   ■  ■
    ■■■■■■■■■■■■■■■

IoU = Intersection Area / (Box1 + Box2 - Intersection)
```

4. Average Precision (AP) Calculation

Calculates the Area Under the Precision-Recall Curve using 101-point interpolation (COCO method).

Source: ultralytics/utils/metrics.py, lines 711-737

```
def compute_ap(recall, precision):
```

- Function takes arrays of recall and precision values at different confidence thresholds.

```
mrec = np.concatenate(([0.0], recall, [1.0]))
```

- Add sentinel values: recall starts at 0 and ends at 1. This ensures the curve covers full range.

```
mpre = np.concatenate(([1.0], precision, [0.0]))
```

- Add sentinel values: precision starts at 1 (perfect at no detections) and ends at 0.

```
mpre = np.flip(np.maximum.accumulate(np.flip(mpre)))
```

- Make precision monotonically decreasing from right to left. This is the "envelope" of the PR curve.

```
x = np.linspace(0, 1, 101)
```

- 101 evenly spaced points from 0 to 1. This is the COCO-style interpolation (at recall points 0, 0.01, 0.02, ..., 1.0).

```
ap = np.trapezoid(np.interp(x, mrec, mpre), x)
```

- Interpolate precision at 101 recall points, then calculate area using trapezoidal rule. This IS the AP.

Why 101-Point Interpolation?

COCO benchmark uses 101 points for smoother, more accurate AP calculation. Points are at recall = 0.00, 0.01, 0.02, ..., 0.99, 1.00.

5. Metrics Calculation Function

This is the core function that calculates Precision, Recall, F1, and mAP.

Source: ultralytics/utils/metrics.py, ap_per_class function, lines 743-823

5.1 Data Concatenation

```
tp = np.concatenate(tp_list, axis=0)
```

- Combine True Positive arrays from all images into one array.

```
conf = np.concatenate(conf_list)
```

- Combine confidence scores from all predictions.

```
pred_cls = np.concatenate(pred_cls_list)
```

- Combine predicted class IDs from all predictions.

```
target_cls = np.concatenate(target_cls_list)
```

- Combine ground truth class IDs from all labels.

5.2 Sorting by Confidence

```
i = np.argsort(-conf)
```

- Sort all predictions by confidence (highest first). The negative sign makes it descending order.

```
tp, conf, pred_cls = tp[i], conf[i], pred_cls[i]
```

- Reorder all arrays according to sorted confidence. High-confidence predictions are processed first.

5.3 Per-Class Calculation Loop

```
for ci, c in enumerate(unique_classes):
```

- Loop through each unique class found in ground truth.

```
i = pred_cls == c
```

- Boolean mask: True where prediction class matches current class.

```
n_l = nt[ci] # number of labels
```

- Count of ground truth objects for this class.

```
n_p = i.sum() # number of predictions
```

- Count of predictions for this class.

```
tpc = tp[i].cumsum(axis=0)
```

- Cumulative sum of True Positives. At each position, how many TPs so far.

```
fpc = (1 - tp[i]).cumsum(axis=0)
```

- Cumulative sum of False Positives. (1 - TP) = FP at each position.

5.4 Precision and Recall Formulas

```
recall = tpc / (n_l + eps)
```

- **Recall = TP / Total Ground Truth**. How many actual objects did we find?

```
precision = tpc / (tpc + fpc + eps)
```

- **Precision = TP / (TP + FP)**. Of all our predictions, how many were correct?

5.5 AP Calculation for Each IoU Threshold

YOLO calculates AP at 10 different IoU thresholds: 0.50, 0.55, 0.60, ..., 0.95

```
for j in range(tp.shape[1]): ap[ci, j] = compute_ap(rec, prec)
```

■ For each IoU threshold, calculate AP using the 101-point interpolation method.

5.6 Final Metrics

```
mp = p_values.mean()
```

■ Mean Precision across all classes.

```
mr = r_values.mean()
```

■ Mean Recall across all classes.

```
map50 = ap[:, 0].mean()
```

■ mAP at IoU=0.5 (first threshold). Average AP across all classes at IoU 0.5.

```
map_val = ap.mean()
```

■ mAP at IoU=0.5:0.95. Average of all APs across all classes and all 10 IoU thresholds.

```
f1 = 2 * mp * mr / (mp + mr + eps)
```

■ **F1 Score = 2 × P × R / (P + R)**. Harmonic mean of precision and recall.

6. Fitness Score Function

The FITNESS score is THE final accuracy metric used by YOLO to select the best model.

Source: ultralytics/utils/metrics.py, lines 955-957

```
def fitness(metrics):
```

■ Function to calculate the overall model fitness/accuracy.

```
w = np.array([0.0, 0.0, 0.0, 1.0])
```

■ Weights for [Precision, Recall, mAP@0.5, mAP@0.5:0.95]. Only mAP@0.5:0.95 has weight 1.0!

```
values = np.array([metrics["precision"], metrics["recall"], metrics["map50"], metrics["map"]])
```

■ Array of the four main metrics.

```
return (values * w).sum()
```

■ Weighted sum. With these weights: Fitness = 0×P + 0×R + 0×mAP@0.5 + 1×mAP@0.5:0.95 = mAP@0.5:0.95

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  KEY INSIGHT: FITNESS = mAP@0.5:0.95                                       ■
■                                                                             ■
■  YOLO uses ONLY mAP@0.5:0.95 to determine the "best" model.                 ■
■  Precision, Recall, and mAP@0.5 have ZERO weight in this calculation.       ■
■                                                                             ■
■  This is because mAP@0.5:0.95 is the strictest metric, requiring            ■
■  accurate detection across ALL IoU thresholds from 0.5 to 0.95.             ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

7. Helper Functions

7.1 load_yaml()

```
def load_yaml(yaml_path):
```

■ Function to load a YAML configuration file.

```
with open(yaml_path, "r") as f:
```

■ Open the file in read mode.

```
return yaml.safe_load(f)
```

■ Parse YAML content into a Python dictionary. safe_load prevents code execution.

7.2 load_labels()

Converts YOLO format labels to absolute coordinates.

```
parts = line.strip().split()
```

■ Split label line into parts: [class_id, x_center, y_center, width, height]

```
cls_id = int(parts[0])
```

■ First value is the class ID (0, 1, 2, etc.)

```
x_center = float(parts[1]) * img_width
```

■ Convert normalized x_center (0-1) to pixel coordinates.

```
x1 = x_center - width / 2
```

■ Calculate left edge: center minus half width.

```
x2 = x_center + width / 2
```

■ Calculate right edge: center plus half width.

YOLO Label Format:

```
Each line in a .txt label file:
<class_id> <x_center> <y_center> <width> <height>

Example: 0 0.5 0.5 0.2 0.3
- Class 0 (e.g., "helmet")
- Center at 50% width, 50% height
- Box is 20% of image width, 30% of image height

All values are NORMALIZED (0 to 1), not pixels!
```

7.3 match_predictions()

Matches each prediction to ground truth boxes using IoU.

```
sorted_indices = sorted(range(num_preds), key=lambda i: predictions[i]["confidence"], reverse=True)
```

■ Sort predictions by confidence (highest first). High-confidence predictions get first chance to match.

```
for pred_idx in sorted_indices:
```

■ Process each prediction in order of confidence.

```
iou = box_iou(pred["bbox"], gt["bbox"])
```

■ Calculate IoU between prediction and each ground truth box.

```
if iou > best_iou:
```

■ Keep track of the best matching ground truth (highest IoU).

```
for t_idx, threshold in enumerate(iou_thresholds):
```

■ Check if match is valid at each IoU threshold (0.5, 0.55, ..., 0.95).

```
if best_iou >= threshold: tp[pred_idx, t_idx] = True
```

- If IoU meets threshold, mark as True Positive for that threshold.

```
matched_gt.add(best_gt_idx)
```

- Mark ground truth as matched so it can't be matched again (prevents double counting).

8. Main Accuracy Test Function

The test_accuracy() function orchestrates the entire accuracy calculation.

8.1 Model Loading

```
model = YOLO(MODEL_PATH)
```

■ Load your trained model from the .pt file.

```
class_names = model.names
```

■ Get class names dictionary {0: 'helmet', 1: 'no_helmet', ...}

8.2 IoU Thresholds Setup

```
iou_thresholds = np.linspace(0.5, 0.95, 10)
```

■ Creates array [0.5, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95]. These are the 10 COCO IoU thresholds.

8.3 Image Processing Loop

```
for img_path in image_paths:
```

■ Loop through all test images.

```
img = cv2.imread(img_path)
```

■ Read image to get dimensions.

```
ground_truths = load_labels(label_path, img_width, img_height)
```

■ Load ground truth boxes from corresponding .txt file.

```
results = model.predict(img_path, conf=CONF_THRESHOLD, verbose=False)
```

■ Run model inference. conf=0.001 gets all possible detections.

```
tp = match_predictions(predictions, ground_truths, iou_thresholds)
```

■ Match predictions to ground truth, get True Positive array.

8.4 Final Calculation

```
metrics = calculate_metrics(all_tp, all_conf, all_pred_cls, all_target_cls, num_classes)
```

■ Calculate all metrics from accumulated results.

```
fitness_score = fitness(metrics)
```

■ Calculate final fitness score (= mAP@0.5:0.95).

9. YOLO Built-in Validation

The simplest and most accurate method - uses YOLO's own validation code.

```
model = YOLO(MODEL_PATH)
```

■ Load your trained model.

```
results = model.val(data=DATA_YAML, conf=CONF_THRESHOLD, iou=IOU_THRESHOLD)
```

■ Run validation using YOLO's built-in validator. This is IDENTICAL to what runs during training.

```
results.box.mp
```

■ Mean Precision across all classes.

```
results.box.mr
```

■ Mean Recall across all classes.

```
results.box.map50
```

■ mAP at IoU=0.5.

```
results.box.map
```

■ mAP at IoU=0.5:0.95 (the FITNESS score).

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  RECOMMENDATION: Use model.val() for production                      ■
■                                                                        ■
■  The custom implementation is for LEARNING how metrics are calculated. ■
■  For actual accuracy testing, model.val() is:                          ■
■  • More accurate (handles edge cases)                                  ■
■  • Faster (optimized code)                                             ■
■  • Same results as training validation                                 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

10. Main Entry Point

```
if __name__ == "__main__":
```

- This code only runs when the script is executed directly (not imported as a module).

```
choice = input("Enter 1 or 2: ")
```

- Ask user to choose between custom calculation or YOLO validation.

```
if choice == "1": test_accuracy()
```

- Run custom calculation that shows all formulas.

```
else: test_with_yolo_val()
```

- Run YOLO's built-in validation (recommended).

Summary: All Formulas at a Glance

Metric	Formula	Code Location
IoU	Intersection / Union	box_iou()
Precision	TP / (TP + FP)	calculate_metrics()
Recall	TP / (TP + FN)	calculate_metrics()
F1 Score	2 × P × R / (P + R)	calculate_metrics()
AP	Area under PR curve (101-point)	compute_ap()
mAP@0.5	Mean AP at IoU=0.5	calculate_metrics()
mAP@0.5:0.95	Mean AP at IoU=0.5 to 0.95	calculate_metrics()
Fitness	1.0 × mAP@0.5:0.95	fitness()

Quick Usage Guide

```
1. Edit MODEL_PATH and DATA_YAML at top of script
2. Ensure your data.yaml points to correct image/label folders
3. Run the script: python test_accuracy_with_labels.py
4. Choose option 1 (custom) or 2 (YOLO built-in)
5. View results in console and accuracy_results/ folder
```