Task: Custom Object Detection and Novel Bounding Box Metric with YOLO

Overview

Choose a small, **manually labeled** image dataset of two object classes (e.g., *cats* and *dogs*). Your task is to set up a YOLO (You Only Look Once) based object detection pipeline from scratch or adapt an existing, minimal YOLO codebase and then implement **a custom bounding box similarity metric** that is used *in addition* to the traditional Intersection over Union (IoU). Finally, you will report both qualitative and quantitative results.

Requirements

1. **Setup YOLO**
    - Use YOLOv5.
    - Train on the selected labeled dataset of images containing *cats* and *dogs*.
    - If you do not have a ready YOLO codebase, you may adapt a publicly available minimal PyTorch/TensorFlow YOLO implementation. However, do *not* rely entirely on an automated off-the-shelf training procedure, show you can modify/troubleshoot the pipeline yourself.
2. **Custom Bounding Box Similarity Metric**
    - Create a new similarity measure between two bounding boxes that takes into account *not only their overlap* but also *aspect ratio, center alignment, or any other geometric properties* you find relevant.
    - **Explain mathematically** how your metric works (i.e., how it's calculated), how it differs from IoU, and *why* it might be beneficial for certain object detection tasks.
3. **Incorporate Your Metric into the Training or Evaluation Loop**
    - Decide if your new metric will be used:
        - As an additional term in the loss function (replacing or augmenting standard IoU/GIoU/CIoU losses).
        - Or purely as a post-training evaluation metric (i.e., still train with standard YOLO loss but evaluate with your new metric).
    - Show or discuss how you integrate it into the code (loss function or evaluation script).
4. **Experimental Results and Analysis**
    - **Train** your model (it does not have to converge perfectly on the small dataset, but do demonstrate you can run the training loop).
    - **Evaluate** on a small test set.

- **Report**:
    1. Standard detection metrics (mAP, IoU) from YOLO.
    2. Your custom bounding box similarity metric.
    3. Qualitative results (sample images with bounding boxes).

2. **Reflective Questions**
   In your write-up, address the following:
    - **Performance**: Did your custom similarity metric improve or degrade performance (either qualitatively or quantitatively)?
    - **Trade-offs**: Discuss any computational or conceptual trade-offs of your metric vs. standard IoU-based metrics.
    - **Further Ideas**: Briefly propose how you might extend or refine this new metric (e.g., weighting for different object classes, distance-based penalties, etc.).
3. **Submission Guidelines**
    - Provide a **short written report** (Markdown or PDF) answering the above questions.
    - Share your **code** in a single ZIP or Git repo with clear instructions on how to run it (including environment setup, dependencies, commands).
    - Make sure to include a **README** that briefly describes:
        0. Your YOLO setup/architecture
        1. The custom metric definition
        2. Instructions for training & evaluation

Hints and Constraints

- You have **one day** (24 hours) to complete this. Tailor your solution accordingly.
- Do not overcomplicate the dataset or training. The main focus is on:
    1. Your ability to set up or adapt a YOLO pipeline.
    2. Your mathematical reasoning in defining a custom metric.
    3. Showing you can analyze results beyond standard tutorials.
- Assume about ~50–100 images with bounding box annotations is enough for a proof-of-concept.
- You are **not** expected to produce a perfectly accurate YOLO model—this is a short challenge. Your approach, problem-solving, and clarity of explanation matter most.

What We're Looking For

1. **Technical Proficiency**: Can you comfortably navigate Python, PyTorch/TensorFlow, YOLO code, etc.?
2. **Mathematical Reasoning**: Did you propose a metric that demonstrates original thought and correct math derivation or justification?
3. **Problem-Solving Skills**: Did you handle any obstacles, error messages, or code modifications logically and effectively?
4. **Clarity of Communication**: Are your explanations and code well-structured, easy to follow, and well-commented?