YOLOv5 Performance Analysis & Optimization

Software Programming for Performance, Spring 2025

March 26, 2025

Overview

In this assignment, you will explore the performance characteristics of the YOLOv5 object detection model. Your task is to measure inference performance, analyze computational characteristics, profile the implementation, and explore optimization strategies. You will gain practical experience with profiling tools, understand compute and memory behavior, and come up with performance improvements.

Problem 1: Introduction to YOLOv5 & Basic Benchmarking

YOLO (You Only Look Once) is a family of real-time object detection models. YOLOv5 is implemented in PyTorch and provides various versions with tradeoffs between speed and accuracy:

- **yolov5n** – Nano (fastest, least accurate)
- **yolov5s** – Small
- **yolov5m** – Medium
- **yolov5l** – Large
- **yolov5x** – Extra Large (most accurate, slowest)

Earlier YOLO versions (YOLOv3, YOLOv4) were implemented in Darknet (C++), but YOLOv5 transitioned to PyTorch, enabling easier experimentation.

Objective

Measure and compare latency (ms) and throughput (FPS) of different YOLOv5 models on test images.

Tasks

1. A Jupyter notebook that allows you to experiment with different YOLOv5 model variants and visualize detection results on sample test images. Run inference on multiple test images using: `yolov5n`, `yolov5s`, `yolov5m`, `yolov5l`, and `yolov5x` models.
2. Measure and report.
   (a) Per-image latency (milliseconds)

(b) Throughput (frames per second)

Prepare a dataset with at least 100 images. You may use 'data/images', download a subset of COCO/VOC, or prepare your own image set.

Hints

1. Use the official YOLOv5 repository:

```
git clone https://github.com/ultralytics/yolov5
cd yolov5
pip install -r requirements.txt
```

2. For terminal testing:

```
python detect.py --weights [MODEL].pt --source [IMAGE_FOLDER] --device [0/CPU]
python detect.py --weights yolov5s.pt --source data/images --save-txt
```

Expected Output Format

Model	Latency (ms)	FPS
yolov5s		
yolov5m		
yolov5l		
yolov5x		

Problem 2: Roofline Bound Analysis

Objective

Determine whether the models are compute-bound or memory-bound on your hardware.

Tasks

1. Calculate for each model. Report it in the form of a table.
   - Total parameters
   - Model size (MB)
   - GFLOPS per inference pass
2. Clearly explain if the models are compute or memory bound. Give the peak GFLOPS utilization.

Extra Credit: Extend the above analysis to a per-layer granularity. Identify the layers with the highest and lowest compute utilization, and provide insights into their performance characteristics.

Hints

- Find model information:

```
from torchinfo import summary
summary(model, input_size=(1, 3, 640, 640))
```

- Compute utilization formula:

$$\text{Utilization} = \frac{\text{Actual GFLOPS/sec}}{\text{Peak GFLOPS/sec}}$$

Problem 3: Code Profiling & Hotspot Identification

Use **at least two** of the following profiling tools:

- **cProfile:** Add the following wrapper to profile 'main()':

```
import cProfile
cProfile.run('main()', 'profile_output')
```

- **line_profiler:** Install and run as:

```
pip install line_profiler
kernprof -l detect.py --weights yolov5s.pt --source data/images
python -m line_profiler detect.py.lprof
```

- **PyTorch Profiler:** Add around inference step:

```
with torch.profiler.profile(
    activities=[torch.profiler.ProfilerActivity.CPU, torch.profiler.ProfilerAc
    record_shapes=True,
    with_stack=True
) as prof:
    results = model(im)
print(prof.key_averages().table(sort_by="cpu_time_total"))
```

- **torch.utils.bottleneck:**

```
python -m torch.utils.bottleneck detect.py --weights yolov5s.pt --source data/i
```

Identify and report:

- Percentage of time in each pipeline component (loading, preprocessing, inference, post-processing).
- CPU vs GPU time (if using GPU).
- Memory or threading issues if observed.

Problem 4: Performance Optimization

Objective

Implement and measure performance improvements.

Tasks

Try at least 3 optimization techniques from:

- Mixed precision (FP16)
- Batch processing
- Multi-threading
- Model quantization
- ONNX/TensorRT conversion

Hints

- FP16 implementation:

```
model.half()  # Convert model to FP16
```

Extra Credit: Performance per Watt and Cache Analysis.

Deliverables (PDF)

1. System specs (CPU/GPU, RAM)
2. Dataset used
3. Profiling tool outputs (tables, screenshots, graphs)
4. Time breakdown across pipeline stages
5. Optimization suggestions and reasoning
6. Optimization results with comparison
7. A 5-slide summary inside the PDF

Grading Rubric

Category	Weight
Completeness	20%
Analysis Depth	30%
Optimization	30%
Report Quality	20%