Performance Benchmarking of Neural Networks Across CPU and GPU Environments

Ruturaj Vasant

November 14, 2025

Abstract

This report evaluates the performance of neural network workloads across local CPU, cloud CPU, and GPU environments. Using four neural network architectures and a standardized training configuration, we measure runtime, throughput, achieved FLOPs, and efficiency. Roofline models are constructed to analyze compute vs. memory bottlenecks, and a batch-size sweep experiment is conducted to study scaling effects.

Contents

1 Experiment Design

1.1 Objective

The objective is to compare the performance of neural network training workloads across different compute environments (local CPU, cloud CPU, GPU). The experiment measures throughput, latency, memory usage, and achieved FLOPs for multiple neural network models.

1.2 Hypothesis

- GPUs will deliver significantly higher throughput than CPUs, especially for deeper models with high arithmetic intensity.
- CPU vs GPU speedup will vary depending on model complexity.
- Increasing batch size should improve GPU utilization and move the workload closer to the compute roofline.

1.3 Experimental Scenarios

- **Scenario A: Environment Comparison** Four models run under identical configurations on:
  - Local CPU
  - Cloud CPU
  - GPU (local/cloud)
- **Scenario B: Batch Size Sweep** Model 4 is run with multiple batch sizes to analyze scaling.

2 Environment Setup

2.1 Hardware Configurations

2.1.1 Local Machine

- CPU: *[Fill in]*
- GPU: *[Fill in]*
- Memory: *[Fill in]*
- OS: *[Fill in]*
- CUDA / PyTorch Versions: *[Fill in]*

2.1.2 Cloud CPU (e2-standard-4)

- 4 vCPUs (Intel/AMD)
- 16 GB RAM
- Debian 12
- Python 3.11, PyTorch CPU

2.1.3 Cloud GPU (Tesla T4)

- GPU: Tesla T4, 16 GB GDDR6
- CUDA 12.4, Driver 550.xx
- PyTorch 2.8.x + cu126

2.2 Models Evaluated

- `resnet18`
- `mobilenet_v2`
- `resnet50`
- `squeezenet1_1`

2.3 Training Configuration

- Dataset: Tiny-ImageNet (200 classes, 64x64 images)
- Epochs: 1
- Workers: 2
- Batch sizes:
  - 128 for all models except SqueezeNet
  - 64 for SqueezeNet to avoid NaNs

3 Complexity Estimation

3.1 Model Complexity Summary

Model	Params (M)	FLOPs/Image (G)	Notes
ResNet18	[Fill]	[Fill]	–
MobileNetV2	[Fill]	[Fill]	Depthwise convs
ResNet50	[Fill]	[Fill]	Bottleneck blocks
SqueezeNet1.1	[Fill]	[Fill]	Fire modules

Table 1: Model Complexity Estimates

3.2 Arithmetic Intensity

Fill in after results.

4 Measurement

4.1 Metrics Collected

- Time per batch / epoch
- Throughput (images/sec)

- Achieved GFLOPs/sec or TFLOPs/sec
- GPU/CPU utilization
- Memory footprint

4.2 Data Collection Method

All runs were executed using the same Python benchmark script with CSV logging enabled.

5 Results

5.1 CPU vs GPU Results (Scenario A)

Insert your tables here — I will generate them once you give me the CSV.

Model	Env	Throughput img/s	Acc@1	GFLOPs/s
ResNet18	CPU	[ ]	[ ]	[ ]
ResNet18	GPU	[ ]	[ ]	[ ]
MobileNetV2	CPU	[ ]	[ ]	[ ]
MobileNetV2	GPU	[ ]	[ ]	[ ]
ResNet50	CPU	[ ]	[ ]	[ ]
ResNet50	GPU	[ ]	[ ]	[ ]
SqueezeNet	CPU	[ ]	[ ]	[ ]
SqueezeNet	GPU	[ ]	[ ]	[ ]

Table 2: CPU vs GPU Performance Comparison

5.2 Batch Size Sweep (Scenario B)

Insert your batch sweep results here.

Batch Size	Throughput	GPU Util %	GFLOPs/s
32	[ ]	[ ]	[ ]
64	[ ]	[ ]	[ ]
128	[ ]	[ ]	[ ]
256	[ ]	[ ]	[ ]

Table 3: Batch Size Scaling Results

6 Roofline Modeling

6.1 Hardware Roofline

Figures will be inserted after we compute AI and throughput.

6.2 Workload Placement

Discussion once values are available.

6.3 Batch Size Impact on Roofline

To be filled after sweep data.

7 Analysis

7.1 Environment Impact

To be written after tables are filled.

7.2 Model Differences

Explain why some models scale better on GPU.

7.3 Batch Size Effects

Impact on utilization, arithmetic intensity, throughput.

7.4 Bottlenecks Identified

Memory-bound, compute-bound, data-loading, etc.

8 Conclusion

Summary will be generated once all data is included.