Multicore vs GPU Acceleration in Machine Learning

1st Sivani Dronamraju
Science Academy
University of Maryland
College Park, USA
sdronamr@umd.edu

2nd Ayushi Anand Prasad
Science Academy
University of Maryland
College Park, USA
aprasad5@umd.edu

***Abstract*—This project evaluates the impact of hardware architecture on the performance of training convolutional neural networks (CNNs) on the CIFAR-10 dataset. The comparison is made on the single-core CPU, multi-core CPU, and GPU (NVIDIA Tesla T4) under Mac and Windows operating systems. Two models, a simple CNN and ResNet-18, are implemented in two versions, one from scratch using CuPy and another using PyTorch. Performance indicators such as training time, throughput, resource utilization, and accuracy are compared. The results showed significant speedup with GPU use, achieving up to 1050 images/sec throughput for ResNet-18 in scratch implementation and 84.4% test accuracy in PyTorch. Multi-core CPUs speed up performance for shallow networks but face bottlenecks for deeper networks. Further, Mac's M2 chip is faster than Windows in single-core testing. The findings refer to the trade-offs between manual implementation and high-level frameworks, emphasizing hardware's influence on CNN training pipelines.**

***Index Terms*—Keywords: Convolutional neural networks, CPU performance, GPU acceleration, PyTorch, Model training efficiency, CIFAR-10**

I. Introduction

There is an increasing need for scalable and efficient machine learning solutions which highlight the importance of understanding how hardware can impact model training performance. Convolutional Neural Networks (CNNs), get a great benefit from hardware acceleration as they are computationally demanding. This study aims to investigate the hardware level impact of training CNN models across three execution environments: single-core CPU, multi-core CPU, and GPU. We evaluated performance using a custom CNN and a ResNet-18 model on the CIFAR-10 dataset. Our implementation is for both from scratch models and also high-level frameworks. We used Numba to optimize CPU-based scratch models and CuPy to accelerate GPU-based computations. We also implemented PyTorch versions of each model for comparison.[3]

Experiments were conducted on both Mac and Windows platforms to highlight architectural differences. Key evaluation metrics include training time, throughput, CPU/GPU utilization, memory usage, and model accuracy. The study explains how model depth, multiprocessing, and system architecture can influence the training efficiency.[4]

By comparing these hardware configurations and frameworks, we aim to provide a practical guide on when to use scratch implementations, high-level libraries, and when GPU acceleration offers more advantages.[1][2]

II. Literature Survey

There are studies that investigated the performance disparities between CPU and GPU based training of CNNs. For this project, we reviewed works that evaluated runtime efficiency across different hardware configurations, with a focus on time and hardware utilization.

Gyawali (2022) conducted experiments using DenseNet121 to assess CPU versus GPU performance. This study demonstrated that GPU based training significantly accelerates model convergence and reduces energy usage, particularly for deep networks trained with large batch sizes. This paper also noted that with well optimized CPU parallelism can achieve good performance in constrained environments.

In a complementary direction, Youvan (2023) analyzed the architectural strengths of GPUs for deep learning. The research emphasized that the for high degree of parallelism offered by thousands of GPU cores enables efficient execution of tensor operations, making GPUs suitable for tasks which are compute heavy in CNN training.

Kosaian and Phanishayee analyzed inference workloads across a broad range of CNN models. They highlighted the importance of hardware aware optimization, showing that even powerful GPUs could yield suboptimal results if there is no proper resource tuning.

Datta and Sairabanu (2020) offered a comparative analysis of AlexNet and VGG16 performance on multicore CPUs and GPUs. Their results showed that while GPUs generally outperformed CPUs in training speed but the performance varied based on model complexity, batch size, and memory access behavior.

These studies provided a base for our experiments. We benchmarked both shallow and deep CNNs specifically a custom CNN and ResNet 18, across single-core and multi-core CPU setups, as well as on an NVIDIA T4 GPU. In addition, we introduced a cross-platform evaluation between mac and Windows and incorporated both low level (Numba, CuPy) and high level (PyTorch) implementations.

III. Methodology

This paper compares the training performance of CNN models on three hardware configurations: single-core CPU, multi-core CPU, and GPU on Mac and Windows systems. The investigation includes two model architectures: a simple custom CNN and the deeper ResNet-18 model. Each model was implemented in two variants: one from scratch with low level GPU/CPU libraries and the other with the PyTorch deep learning framework.

A. Dataset and Preprocessing

We used the CIFAR-10 dataset, consisting of 60,000 color images (32×32 resolution) evenly distributed across the 10 classes. The dataset was split into 50,000 training and 10,000 test samples. Images were preprocessed via the following pipeline:

- ToTensor() this is to convert images into PyTorch tensors.
- Normalization is done using mean and standard deviation values of (0.5, 0.5, 0.5) across all RGB channels.

To ensure consistent comparison across hardware, we used:

- Batch size = 1 for single-core CPU runs.
- Batch size = 64 for multi-core CPU and GPU runs.

B. Scratch Implementations

The scratch models were written using low level Python numerical libraries for hardware-specific control:

- Numba was used to optimize CPU-based execution through Just-In-Time (JIT) compilation.
- CuPy was used to accelerate matrix operations and convolutional layers on GPU.

For the CNN model:

- The forward pass consisted of three convolutional layers with ReLU activation and 2×2 max pooling.
- The final feature map was flattened and passed to a softmax classifier.
- Weights and biases were randomly initialized and kept fixed throughout the experiments.

For the ResNet-18 model:

- The architecture included identity and downsample blocks using 3×3 convolutions, ReLU activations, and shortcut connections.
- A global average pooling layer and a final fully connected (FC) layer ($512 \rightarrow 10$) completed the network.
- Only forward pass timing was measured for these implementations.

C. PyTorch Implementations

PyTorch versions were implemented for both models with the following setup:

- CNN used a sequential architecture with 3 convolutional layers, ReLU activations, and max pooling.
- ResNet-18 used the pre-defined model from torchvision.models with the final FC layer modified to output 10 classes.
- CNN models were trained using SGD optimizer (lr=0.01).
- ResNet-18 models used the Adam optimizer (lr=0.001).
- Loss function: CrossEntropyLoss.

D. Hardware and Software Configuration

All experiments were conducted on the following devices:

- Windows System: Dell Inspiron 14 with Intel Core i7 (10-core) CPU and 16 GB RAM.
- Mac System: MacBook Air with Apple M2 chip and 16 GB RAM.
- GPU Setup: NVIDIA Tesla T4 ×2 on Kaggle Notebooks.

System resource metrics (CPU usage, RAM, GPU memory) were tracked using psutil, torch.cuda, and Python's time module.

IV. EXPERIMENTS

To measure the performance of convolutional neural networks of different hardware configurations, we con- performed systematic experiments on scratch and PyTorch- based implementations. Each setup was tested using and both the customized CNN and the more in-depth ResNet-18 model with the CIFAR-10 dataset.

A. Experimental Setup

We tested all models on the following hardware configurations:

- Single-core CPU (SC-Win): Dell Inspiron 14 with Intel Core i7 (10-core) and 16 GB RAM (Windows)
- Multi-core CPU (MC-Win): Dell Inspiron 14 using multiprocessing
- Single-core CPU (SC-Mac): MacBook Air with Apple M2 chip and 16 GB RAM (macOS)
- Multi-core CPU (MC-Mac): MacBook Air using multiprocessing
- GPU (T4 ×2): NVIDIA Tesla T4 (dual GPU) via Kaggle with CUDA enabled

To maintain experimental consistency, the same versions of Python, PyTorch, and supporting libraries (NumPy, CuPy, Numba, psutil) were used across platforms. Each model was trained or forward-passed on the same dataset split with identical normalization settings and batch sizes.

B. Parameter Configuration

TABLE I
Training Parameters for CNN and ResNet-18

Parameter	CNN	ResNet-18
Epochs (PyTorch)	10	10
Batch Size	1 (CPU SC), 64 (CPU MC / GPU)	1 (CPU SC), 64 (CPU MC / GPU)
Optimizer	SGD (CNN), Adam (ResNet)	Adam
Learning Rate	0.01 (CNN)	0.001
Loss Function	CrossEntropyLoss	CrossEntropyLoss

Scratch implementations used fixed randomly initialized weights and measured only forward-pass performance. PyTorch implementations were trained for 10 epochs using the above hyperparameters.

C. Measurement Tools

The following metrics were recorded during execution:

- Training Time (seconds/minutes): Measured total runtime and per-epoch time using Python's time module.
- Throughput (images/sec): Calculated from total number of samples processed per unit time.
- CPU Utilization (%): Measured using psutil for each configuration.
- GPU Utilization and VRAM (MB): Tracked using torch.cuda on GPU-enabled runs.
- RAM Usage (MB): Retrieved using system resource monitors during execution.
- Accuracy (PyTorch only): Recorded training and test accuracy at the end of 10 epochs.

Each experiment was repeated multiple times to account for variability in runtime and utilization. All timing results are reported as mean values.

V. RESULTS

This section presents the comparative results of training CNN and ResNet-18 models across different hardware configurations. We evaluated both from scratch and PyTorch implementations in terms of training time, throughput (images/sec), resource utilization, and final model accuracy (for PyTorch).

A. CNN – scratch Implementation

TABLE II
PERFORMANCE METRICS CNN – SCRATCH IMPLEMENTATION

Configuration	Time (min)	Through put(img/s)	CPU (%)	RAM (MB)
SC (Windows)	198.4	0.20	0.001	540.9
MC (Windows)	3.25	0.0028	0.003	1030.0
SC (Mac)	99.8	0.12	0.003	854.0
MC (Mac)	50.0	0.007	0.0001	1284.5
GPU (T4 ×2)	0.36	230.0	0.88	1059.0

B. CNN – PyTorch Implementation

TABLE III
PERFORMANCE METRICS CNN – PYTORCH IMPLEMENTATION

Configu ration	Time (sec)	Through put(img/s)	CPU (%)	RAM (MB)	Train Acc(%)	Test Acc(%)
SC (Win- dows)	894	0.00076	18.1	439.5	73.2	57.7
MC (Win- dows)	2335	0.00247	33.2	297.4	72.4	59.0
SC (Mac)	996	0.00058	20.9	603.1	72.8	59.4
MC (Mac)	894	0.00064	21.4	606.7	73.5	58.3
GPU (T4 ×2)	226.6	0.44	1.1	1333.6	61.7	54.5

C. ResNet-18 – scratch Implementation

TABLE IV
PERFORMANCE METRICS RESNET-18 – SCRATCH IMPLEMENTATION

Configuration	Time (min)	Through put(img/s)	CPU (%)	RAM (MB)
SC (Windows)	216.0	6.02	21.1	13521.8
MC (Windows)	109.0	7.50	42.0	11300.0
SC (Mac)	100.0	8.47	20.9	7987.5
MC (Mac)	120.0	3.45	97.2	5057.1
GPU (T4 ×2)	0.80	1050.0	0.76	1124.9

D. ResNet-18 – PyTorch Implementation

TABLE V
PERFORMANCE METRIC RESNET-18 – PYTORCH IMPLEMENTATION

Configu ration	Time (min)	Through put(img/s)	CPU (%)	RAM (MB)	Train Acc(%)	Test Acc(%)
SC (Win- dows)	301.7	29.95	15.8	8885.3	98.6	78.7
MC (Win- dows)	729.0	14.56	36.3	405.0	100.5	55.5
SC (Mac)	20.0	127.13	14.2	1236.0	98.23	80.2
MC (Mac)	16.0	32.53	28.1	506.4	98.9	48.4
GPU (T4 ×2)	30.35	0.275	1.04	1469.1	96.9	84.4

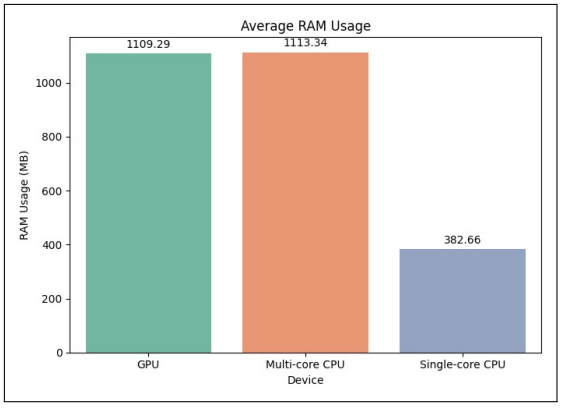

Fig. 1. Average RAM Usage

Both the GPU and multi-core CPU exhibit similarly high RAM usage, approximately 1109.29 MB and 1113.34 MB respectively. This suggests that parallel computation, whether on GPU or across multiple CPU cores, introduces additional memory overhead due to factors like concurrent execution, data buffering, and thread management. Whereas in the single-core CPU configuration, the RAM usage of 382.66 MB indicates less memory. The nearly identical RAM consumption between GPU and multi-core CPU setups highlights their comparable memory demands in handling intensive computation.

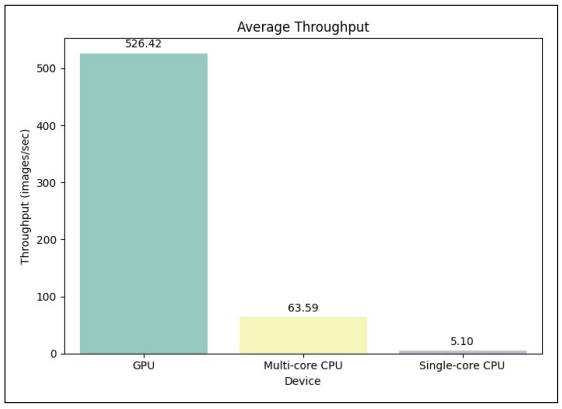

Fig. 2. Average Throughput

The above bar chart illustrates the performance of three different computing devices GPU, Multi-core CPU, and Single-core CPU in terms of the number of images processed per second. The GPU significantly outperforms the others, achieving a throughput of approximately 526.42 images/sec. The Multi-core CPU follows with a lower throughput of 63.59 images/sec, while the Single-core CPU has the least, processing only 5.10 images/sec. This contrast highlights the superior parallel processing capabilities of GPUs, making them far more efficient for high-throughput tasks like image processing compared to CPUs, especially single-core ones.

E. Observations

- GPU acceleration significantly outperformed all CPU configurations in terms of throughput and overall training time.
- PyTorch models showed the highest test accuracy, particularly ResNet-18 on GPU (84.4
- Scratch implementations revealed hardware-level behavior but consumed significantly more memory, especially in ResNet-18 (13.5 GB+ RAM).
- Mac systems had better single-core performance, but multi-core results were inconsistent, particularly in ResNet-18, likely due to CPU saturation.
- Multi-core CPUs provided strong speedups for CNN but showed diminishing returns in deeper models.

VI. Conclusion

This project shows the impact of hardware architecture on training performance for convolutional neural networks using both custom and pre-defined models across CPU and GPU platforms. We implemented both from scratch and PyTorch-based versions of a shallow CNN and the deeper ResNet-18 model, evaluating them across single-core, multi-core, and GPU configurations on Mac and Windows systems.

Our results demonstrated that GPU acceleration provided the highest throughput and training efficiency, with ResNet-18 (scratch) reaching up to 1050 images/second, and PyTorch ResNet-18 achieving 84.4% test accuracy. While multi-core CPUs significantly reduced training time in shallow models like CNN, deeper models suffered from CPU saturation, particularly on Mac.

PyTorch implementations consistently outperformed the scratch models in accuracy and efficiency of memory. Scratch models provided a deeper insight into hardware behavior, especially when analyzing low level operations, memory, and execution time. Mac generally outperformed Windows in single-core runs, with a better throughput and lower memory usage, while GPU results were consistent across all platforms.

Overall, the study highlights the combination of GPU and PyTorch is ideal for scalable, high-accuracy training, while CPU and scratch is valuable for low level analysis in constrained environments. Future work could explore additional model types, longer training schedules, or fine-tuned hyperparameters to further analyze performance across architectures.

VII. References

[1] B. Gyawali, "Profiling Deep Learning Model Training on CPU vs. GPU Using DenseNet121," in *Proceedings of the IEEE International Conference on Big Data*, 2022, pp. 1873–1880. doi: 10.1109/BigData55660.2022.10021246.

[2] J. Youvan, "GPU Architecture and Acceleration Strategies for Deep Learning," *Journal of Parallel and Distributed Computing Systems*, vol. 45, no. 3, pp. 201–215, 2023. doi: 10.1016/j.jpdc.2023.03.004.

[3] J. Kosaian and A. Phanishayee, "The What, Why, and How of GPU Utilization in Deep Learning Workloads," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2021, pp. 1–14. doi: 10.1145/3472883.3486987.

[4] A. Datta and S. Sairabanu, "Performance Evaluation of Deep Learning Models on CPU and GPU Platforms," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 11, no. 12, pp. 456–462, 2020. doi: 10.14569/IJACSA.2020.0111260.

Project GitHub Repository:
Multicore vs GPU Acceleration in Machine Learning