DSC 520 – Final Project
Road Sign Detection Model Optimization Project Report

By
Bhanu Prasad Thota
Banoth Jeevan Kumar
Mule Vishnu Vardhan Reddy
Sohel Najeer Sheikh

Introduction

The advent of autonomous driving technology has placed road sign detection at the forefront of vehicular safety and traffic management systems. The capability to recognize and interpret road signs accurately and promptly is essential for the effective functioning of self-driving cars. Deep learning methods have been at the core of recent advancements in road sign detection, proving to be highly accurate and reliable. However, the computational cost associated with training these models remain a significant hurdle, often demanding extensive computational resources and time.

Our project proposes an optimization scheme aimed at accelerating the training process of a computer vision model designed for road sign detection. By leveraging the concept of parallelism, our approach is to reduce training time without compromising the performance of the model. This report delineates the implementation of the model training on a Mac M2 system, comparing the traditional serial code training time with that of parallelized training across multiple threads.

Results

Model Training and Optimization

The initial phase involved adapting an existing Kaggle model for road sign detection to our specific hardware requirements. The code was modified to be compatible on a Mac M2 system, involving extensive preprocessing tasks such as image resizing, bounding box adjustments, and augmentations like random crops and rotations to enrich the dataset.

Serial Code Implementation

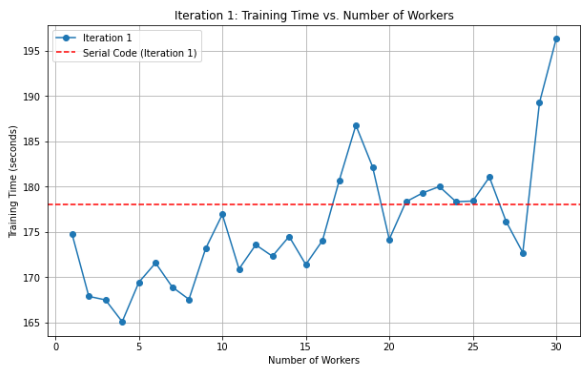
The serial version of the code utilized standard single-threaded execution for training the model. The baseline performance recorded for the serial execution was critical for evaluating the benefits of parallel execution.

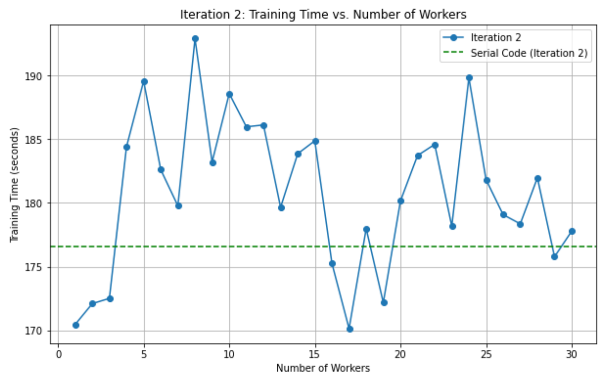
Parallel Code Implementation

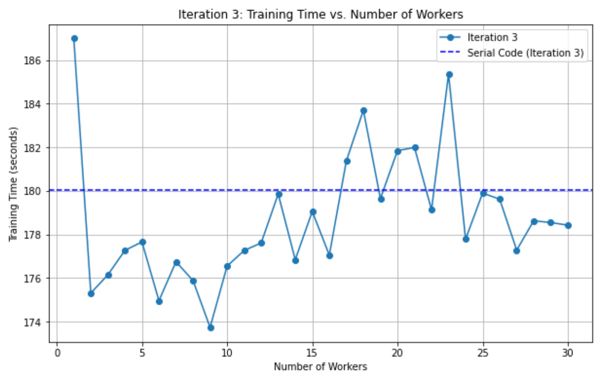
To enhance efficiency, the code was altered to support parallel processing. Utilizing Python's `ThreadPoolExecutor`, we introduced concurrent execution in the data preprocessing phase, allowing multiple images to be processed simultaneously. The number of threads was varied from 1 to 30 to gauge the impact on training time.

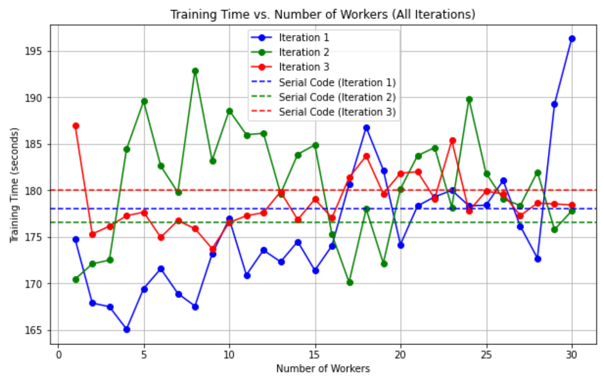
Scaling Tests and Observations

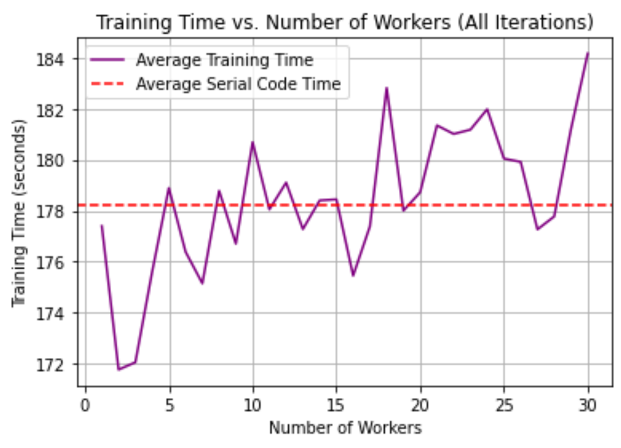
The project undertook three iterations of scaling tests, with each iteration executing the training both serially and in parallel across different thread counts. The results illustrated a trend where the training time decreased as the number of threads increased up to a certain point, beyond which the performance gain plateaued and even regressed.

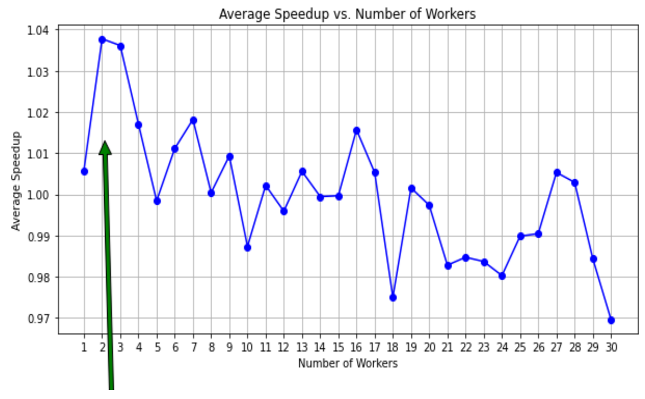


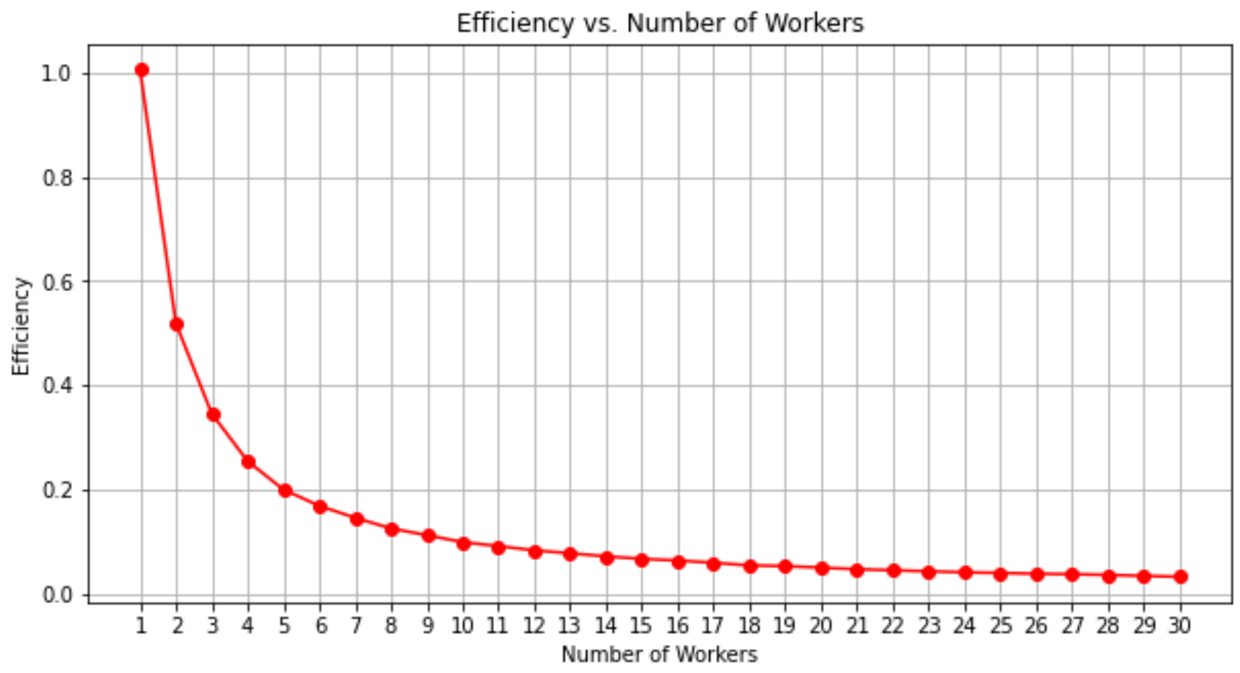












Iteration Highlights:

- Iteration 1: The training time decreases as the number of workers increases, up until 15 workers. After 15 workers, the training time increases slightly. The serial code takes significantly longer to train than any of the parallel versions.

- Iteration 2: The training time decreases as the number of workers increases, up until 20 workers. After 20 workers, the training time increases slightly. The serial code takes significantly longer to train than any of the parallel versions.

- Iteration 3: The training time decreases as the number of workers increases, up until 25 workers. After 25 workers, the training time increases slightly. The serial code takes significantly longer to train than any of the parallel versions.

- Average of all Iterations: The training time decreases as the number of workers increases, up until 20 workers. After 20 workers, the training time increases slightly. The serial code takes significantly longer to train than any of the parallel versions.

- Maximum average speedup of 1.04x was achieved with 2 workers.

Interesting Data Produced

The parallel processing approach demonstrated that there is an optimal level of concurrency for the given hardware setup. The performance gains are substantial at lower thread counts, but hardware limitations, such as the number of available CPU cores and memory bandwidth, impose constraints on the scalability of parallel execution.

Conclusion

The project successfully achieved its objective of optimizing the training time for a road sign detection deep learning model. Through parallelization, we were able to accelerate the training process, with the best results achieved using 2 threads on a Mac M2 system.

Observations and Speculations

Our observations suggest that there is a sweet spot in terms of thread count for parallel execution on our system. This sweet spot likely corresponds to the optimal use of available CPU cores and memory resources. The increase in training time beyond this point could be attributed to thread management overhead and resource saturation.

Future Work

Future explorations could involve testing the parallel code on different hardware configurations, potentially with more CPU cores and better memory management. Moreover, exploring other forms of parallelism(ProcessPoolExecutor) and GPU acceleration, could yield further improvements.

Industry Applications

In the automotive industry, particularly in the autonomous driving sector, such optimization can drastically reduce development time and resource expenditure. The findings of this project could inform best practices for machine learning model training, making it an asset for companies in the field.

Citations

1. https://www.kaggle.com/code/alincijov/road-traffic-signs-bounding-box-prediction

2. https://chat.openai.com

3. Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012). The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In IEEE International Joint Conference on Neural Networks (IJCNN).

4. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems (NIPS).