Performance Analysis and Optimization on Edge AI devices

Reeva Mishra

June 6, 2021

1 Introduction

The application of Edge Artificial Intelligence (AI) can be found in various computer vision applications such as digital assistants and self-driving cars where a prompt response is necessary, and a small amount of latency could be very devastating. For this reason, it is necessary to fasten the edge device's processing time. Internet of Things (IoT) devices has alleviated the problem to an extent by processing the data closer to the data generation source and this mechanism is called "Edge Computing". NVIDIA Jetson Nano and Raspberry Pi 3B are embedded SoM (System on Module) devices that encompass GPU, CPU, DRAM and flash storage on a compact platform. These are commonly used affordable devices for edge computing, AI and machine learning. By measuring the application workload performance and identifying areas of inefficient programming, performance bottlenecks and memory I/O leaks, performance could be further improved programmatically. This project uses three methods to profile edge devices (e.g.: NVIDIA Jetson embedded GPU) – NVIDIA Nsight Systems, NVPROF and tegrastats GUI version jtop to analyze a sample program. Various optimization ways have been applied and examined closely to observe the performance improvement. Additionally, a performance comparison of various benchmarking models and two edge devices has been performed to select on the best models for the respective hardware architectures.

2 Reeva's Analysis (GitHub Folder)

2.1 Environmental Setup

2.1.1 Nvidia Nsight Tools

Profiling is an iterative process of analyzing profile data, optimizing the application and profiling the application again to check if desired behavior is achieved (Fig 1 Left). This process is continued until the desired performance is achieved.

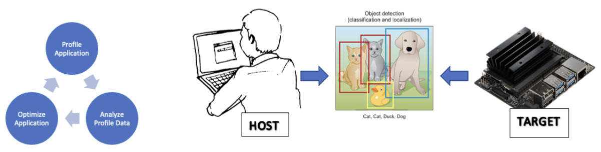


Figure 1: Left:Profiling Cycle. Right:Edge Profiling

Most deep learning applications on edge devices are based on ML frameworks written in python. There are many python-based profiling tools available such as cProfile, PySpy, VizTracer, PyInstrument and Yappi but none of these tools can profile code running on GPU for python. However, NVIDIA's Nsight Tools (comprising of Nsight System and Nsight Compute/Graphics) can perform both system and kernel level analysis for python programs. Edge profiling is time-consuming and cumbersome. The profiling tools generate a lot of data which is complex to understand. The dependency on a separate host device to perform profiling on the target edge device adds to the complexity.

To profile a sample deep learning model, NVIDIA Jetson Nano (4GB RAM) is used as the target device where the application that we want to profile is running. It comprises of a NVIDIA Maxwell GPU with 128 NVIDIA CUDA Cores and a quad-core ARM Cortex-A57 MPCore CPU processor. A virtual machine of Ubuntu 18.04 installed on a MacBook Pro is used as the host device where visual profiling tools such as Nvidia Nsight Systems and Visual Profiler have been setup. Details of the steps for profiling with Nsight Systems can be found under my GitHub link(NsightSystems) and with

Visual Profiler can be found under my GitHub link(VisualProfiler). First, Nsight Systems is used to perform the initial system level analysis to get rid of system-level bottlenecks such as unnecessary data transfers and improve system-level parallelism. Once, satisfied with the performance, Nsight Compute/Graphics/Visual Profiler can be used to perform deeper kernel-level analysis to analyze each CUDA thread.

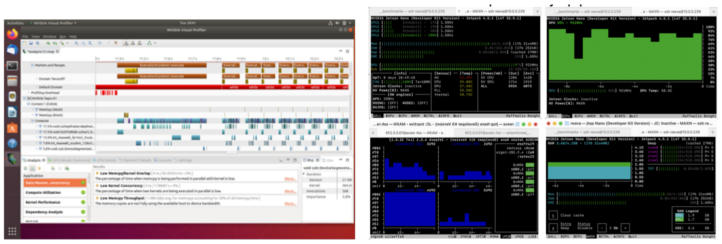


Figure 2: Visual Profiler and jtop Screenshot

The jetson_stats can also be used to visualize an output as it combines both tegrastats and NVP-model into a GUI to profile and control Jetson. Its most used tool is jtop and it provides summary of Jetson CPU and GPU utilization, power consumption and thermal dissipation in a coarse time scale. It can directly run on the jetson device. In contrast, Nsight Systems provides us with individual activities across a fine time scale making it better tool for performance optimization.

2.2 Implementation (GitHub)

2.2.1 Sample edge AI program

NVIDIA's SSD300 (Link) is an object detection model which has been trained on COCO dataset. The output of this model consists of bounding boxes with probabilities for 81 object classes. Now, we shall see how to profile and optimize a video inferencing pipeline program based on SSD300 model. There are 4 important parts of the code- buffer_to_image_tensor that converts video frames into pytorch tensors, preprocess that scales 0-255 RGB pixel values to -1 to +1 float values, detector that runs the SSD300 model and postprocess that turns model output into bounding boxes.

2.2.2 Tracing with Nsight System and Optimization

On running the baseline program on the edge device (Jetson Nano used here), an FPS of 1.09 was achieved. On analyzing the qdrep file (as can be seen below), it was concluded that CPU usage is very high, GPU is used only during inference and there are constant memory transfers resulting from constant synchronization between CPU and GPU.

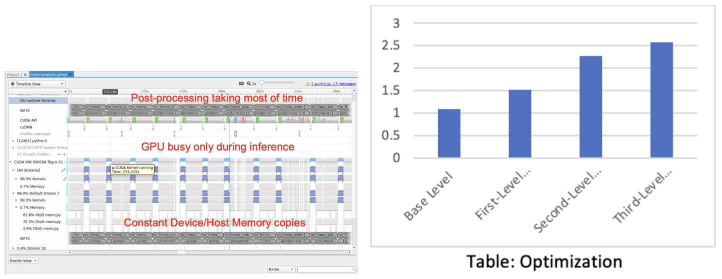


Figure 3: **Left:**Base Program on Nsight System. **Right:**Optimization and Performance Improvement

First level Optimization – The tensors (in the postprocessing function) were accessed on GPU and the computation was performed on CPU. So, when these tensors were moved to CPU with ".cpu()", FPS increased to 1.53 (40% improvement). Second level optimization – Here, all the preprocessing and postprocessing parts of the application is moved to GPU and the FPS increased to 2.27 (48% improvement) without the profiling overhead. Third level optimization - In this step of batch processing i.e., multiple frames are sent to the neural network. By this, it was observed that the FPS increased to 2.57 (13% improvement). Fourth level optimization – In this step, half-precision data was adopted to check if it improves the performance by reducing memory usage of neural network. Reducing to fp16 cuts down the size of weights of neural network models by half but reduces the accuracy by a minor amount. However, it was observed that the FPS got dropped to 1.20. The performance improvement can be seen from the Fig **3 Right**. Overall, there was an improvement of 68%. All the profile files (qdrep) after each level of optimization can be found under my GitHub link(ProfileFiles).

2.2.3 Other Benchmark Models

To test more on the dependency of data precision with performance, I tested the other benchmark models with data precision of fp32 and fp16, and the results are as follows:

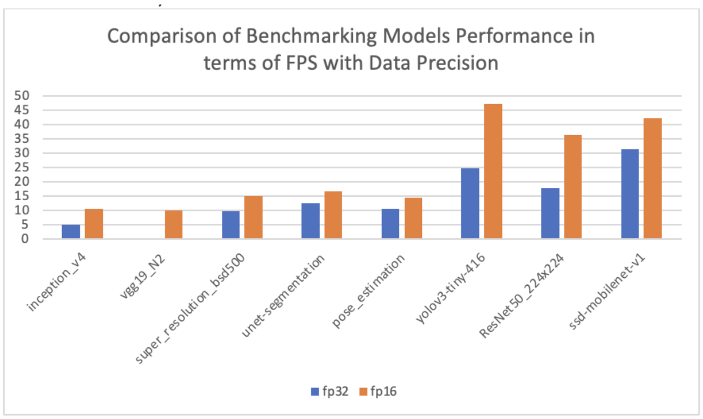
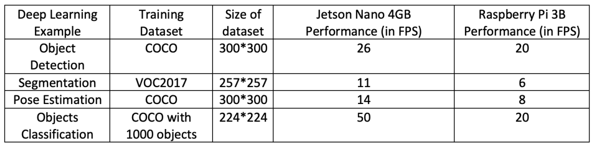


Figure 4: Data precision vs Performance

Finding: The above deep-learning models were executed on Jetson Nano with data precision fp32 and fp16. Reducing to fp16 cuts down the size of weights of neural network models by half but reduces the accuracy by a minor amount. The rate of variation in performance with half-precision data is dependent on both the application's model architecture as well as on the hardware specifications (understood from table and observation above). The background processes for some models was observed further closely (difference in processing time of different threads due to variation in data precision) with Nsight systems whose qdrep files can be found under my GitHub link(BenchmarkModels).

2.3 Hardware Platform: Raspberry pi vs Jetson Nano

To evaluate application's performance dependency on hardware specifications, I executed the same applications both on Jetson Nano and Raspberry Pi 3B. The results can be found in figure 5.



Deep Learning Example	Training Dataset	Size of dataset	Jetson Nano 4GB Performance (in FPS)	Raspberry Pi 3B Performance (in FPS)
Object Detection	COCO	300*300	26	20
Segmentation	VOC2017	257*257	11	6
Pose Estimation	COCO	300*300	14	8
Objects Classification	COCO with 1000 objects	224*224	50	20

Figure 5: Hardware Comparison

Finding: The major performance gap between the 2 edge devices is the availability of GPU delegates in Jetson Nano (not present in Raspberry Pi) which TensorFlow Lite deploys at the time of execution. The AI performance of Jetson Nano board is rated 472 GFLOPs which is just 21.4 in case of Raspberry Pi 3B. While the Jetson Nano board consists of 128-core NVIDIA Maxwell GPU architecture and a 4GB LPDDR4 RAM, Raspberry Pi has a 1GB RAM and runs on CPU mostly. The Raspberry Pi 3B comes at $35 whereas the Jetson Nano is $108. So, there is a tradeoff between price and performance.

2.4 Use Case

Based on the comparison of various deep-learning models, I selected tiny-yolo as the object-detection application for my autonomous home surveillance robot "Oni"(Video Link), which uses edge AI to autonomously move around the home and monitor things and send alerts when it detects emergencies or problems (e.g.: fall detection, intrusion detection etc.)