Benchmarking YOLOv8 Object Detection

Nathan Hirata, Carlos Ramirez, Daniel Carrera

Electrical and Computer Engineering Department
California State Polytechnic University Pomona

3801 W Temple Ave, Pomona, CA 91768

nahirata@cpp.edu , carrea@cpp.edu , carlosr3@cpp.edu

***Abstract*—This project aims to benchmark the behavior of various computer architectures while running YOLOv8 object detection. YOLOv8 (you only look once) was used to test different hardware: Intel Core i7 10th Gen, ARM A57, Intel i7 8th Gen, and Nvidia Geforce RTX 2060. This was done by implementing several benchmark tests with a focus on inference time, frames per second, and memory usage. The experiment shows that the computer's architecture is essential to a model's performance but comes at a trade-off between the types of hardware used.**

Keywords - CPU, GPU, TPU, YOLOv8, Perf, Ubuntu, Jetpack, Jetson, Object Detection

I. INTRODUCTION

There are many Object Detection methods that have been used since computer vision has evolved over the years. To name a few, there is R-CNN, Fast R-CNN, Faster R-CNN, SSD, and RetinaNet. As you can see in a lot of these abbreviations, CNN is mentioned, this stands for Convolutional Neural Network, which is a type of deep learning model that was designed for processing grid-like data. The reason this is important is because an image when converted into data is grid-like, so for computer vision and object detection, these CNNs process the grid-like data through several layers that perform operations such as convolution, activation, pooling, flattening, linear transformation, and softmax. In many CNN architectures, each one of these operations would act as a different layer of the network, when training a new CNN model, weights and biases are constantly updated to try and find a relationship to make more accurate predictions.

In 2015 YOLO(you only look once) was introduced as a research paper on object detection written by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. YOLO has gained popularity in recent years because the speed and accuracy of its models far outperform the other object detection models that were formerly used. YOLO achieves its faster speed and accuracy because the architecture of its model differs from past methods, for example, in R-CNN a two-stage approach is used where the data has to pass through twice; once for region proposals of what area is to be checked, then again through a separate CNN for classification. Alternatively, YOLO was able to combine this into a single unified detection network that can perform all steps needed for object detection through a single pass of 1 CNN. This same difference in single vs multi-stage CNNs applies to training a new model between YOLO and other methods. Since YOLO predicts the bounding boxes and class probabilities in a single pass-through of the CNN, it allows for real-time object detection by reducing the inference times of past computer vision models. YOLO is less computationally expensive compared to other models as well since it only needs to work through a single CNN.

YOLO works by dividing our image or grid-like data into fixed grid sizes, where each grid is responsible for predicting bounding boxes and class probabilities for objects that are contained within that grid area. Some older object detection models such as SSD and RetinaNet use a different approach, they use anchor boxes and predefined aspect ratios to predict the bounding boxes. A higher level look at YOLO's process would show that it begins with a preprocessing stage, where it resizes the image or frame being looked at to a fixed size and normalizes it, then that preprocessed image is pass through a series of layers in the

CNN where the appropriate operations are performed to extract the features from the image. Then a feature map is produced from the results of the prior step, where it is passed through even more convolutional layers to predict things like the bounding box coordinates, objectness scores, and class probabilities. For the prediction stage, the data has to be generated for multiple scales or sizes to increase the chances of detecting an object. Last, there are some post-processing steps done to filter our low confidence or repeated results that were found when checking for different image scales. Each generation of YOLO has added and taken away certain layers of the CNN which has consistently improved performance in terms of inference time, accuracy, and power efficiency. The YOLO version that will be used in this paper is YOLOv8 by Ultralytics, at the time of this paper Ultralytics has not released a paper with the updates to the architecture they have made between this version and the prior Yolov5 they designed.

The operations used in the Convolutional Neural Network for YOLO introduce a unique opportunity for the field of computer architecture and hardware. Prior to deep learning or machine learning, computer architectures seemed to have been primarily designed to support general computing tasks that covered a broad scope of tasks that would benefit the average user, while balancing performance, cost, and power efficiency. With this approach to computing, the computer architecture that was widely used in computers was in the form of CPUs or Central Processing Units. CPUs contain some cores that are each capable of executing instructions, they can also handle control flows such as looping or branching. At the peak of the CPUs efforts to increase performance, techniques such as branch prediction, pipelining, and out-of-order execution are utilized. While the CPU has served well since computers have been used, it is not ideal for machine learning and deep learning [1]. A computer architecture that can handle the operations commonly used in these areas is needed, as an architecture that can perform matrix operations and massive parallel calculations. In order to improve the performance of machine learning, computer vision, or deep learning, two more types of hardware architectures are widely used, the GPU, or Graphics Processing Unit, and the TPU, the Tensor Processing Unit. These are both considered to be hardware-accelerated architectures. Hardware acceleration is when a hardware component or module is used to accelerate a task that used to be performed in a general-purpose processor such as inside of the CPU. While the CPU would handle these tasks slowly for reasons that have been covered, the hardware accelerator is designed to perform these same tasks faster and more efficiently.

The GPU was originally designed to render graphics like videos and 3D images, it has become very popular for video game graphics, but has also started to be used for machine learning, deep learning, and computer vision. GPUs have thousands of ALUs that allow them to perform an incredible amount of arithmetic operations per cycle [2]. GPUs have much smaller cores than the ones found in CPUs, but they also have thousands of them in comparison, because of this they are able to perform an extremely large number of tasks simultaneously, or in parallel. This comes as a benefit in the form of handling the large-scale matrix operations performed by the CNNs in YOLO. Another benefit of GPUs is that they allow for scalability, so multiple GPUs can be used to distribute the computing across however many GPUS are being used, which would allow for larger more complex deep learning models to be used or trained. Another way that GPUs are good for these workloads is because of their high bandwidth memory interfaces, these affect the throughput or data processed over time, in comparison to the amount of data a CPU can handle, the GPU can handle much more that allows its amount of data processed in parallel to increase even greater. GPUs also contain some specialized hardware units, such as Tensor Cores that are widely used in Nvidia's current lines of graphics cards; these tensor cores are designed specifically for deep learning operations, which makes the GPU a great candidate for accelerating YOLO.

The TPU was developed by Google as an accelerator to provide high-performance computing capabilities for tensor operations, such as those used in YOLO. In comparison to the GPU and CPU, the TPU is more efficient when it comes to convolutions and large-scale matrix multiplications. There is a processing unit in the TPUs that acts as a hardware accelerator dedicated to matrix processing and convolutions in parallel [3]. TPUs also have scalability and high memory bandwidth which benefits them in deep learning for the same reasons listed in covering the GPU's high memory bandwidth and scalability. There is also a version of integer operations that are low precision used by TPUs that are called quantized arithmetic, these operations can lower the memory bandwidth requirements and computational complexity without significant loss in the model's accuracy. For these reasons, the TPU can deliver better performance and power efficiency for deep learning and computer vision.

Despite the benefits of GPUs and TPUs for computer vision, they still have their drawbacks. They consume more power than CPUs and produce more heat during heavier workloads, they also are not as flexible for general-purpose computing tasks, and last, on average have limited on-chip memory which can bottleneck larger computer vision models.

In this project, laptops using x86 architecture and CPU, a laptop using a Virtual Machine through x86 architecture and a CPU, and an edge device using an Arm CPU were used to benchmark YOLOv8 over multiple architectures. After comparing those devices, a laptop used prior with the CPU is instead tested while running YOLOv8 on its GPU instead, to demonstrate the trends outlined above.

II. EXPERIMENTAL METHODOLOGY

A. Materials Specifications

Device 1 Hardware: MSI Creator 15 Laptop

- GPU: Nvidia Geforce RTX 2060
- CPU: Intel® Core™ i7-10875H × 16
- Operating System: Ubuntu 22.10

Device 2 Hardware: Jetson Nano Developer Kit

- GPU: 128-core Maxwell
- CPU: Quad-core ARM A57 @ 1.43 GHz
- Memory: 4 GB 64-bit LPDDR4 25.6 GB/s
- Operating System: Ubuntu 20.04

Device 3 Hardware:

- GPU: N/A
- CPU: Intel i7 8th Gen
- Operating System: Ubuntu 22.10

MP4 Tested Video:

- Size of video file: 5.5MB
- Resolution of video file: 1280 x 720
- Duration of video file: 20 seconds
- Frame Rate: 25 fps

Yolov8 configuration:

- Model: Pretrained COCO dataset
- Version: nano
- Image size: default (600X600)
- Confidence of 0.50

Benchmarking Software:

Operf , Perf , flamegraph , Jetson-Stats, htop

B. Experimental Setup

1. Setup operating system: For the Jetson Nano, flash the JetPack image onto an SD card and install it into Jetson Nano. For Laptops, install VirtualBox and download Ubuntu 22.10 images. Create Virtual Machines for testing.
2. Pip install ultralytics: this will retrieve all libraries and additional software required for YOLOv8 to execute.
3. Upload targeted video and generate a Python script using ultralytics libraries and commands.
4. Benchmark using Operf, Perf, flamegraph, Jetson-Stats, and htop targeted at the Python script.

C. Benchmarking Computer Vision

In order to benchmark the computer vision performance on multiple devices and architectures, it was important to identify what resources and metrics were to be considered important. In this project, the most important metrics being observed were inference time and frames per second while using the nano model. Other metrics that were considered as important while running the Python script were the number of instructions or cycles used while the office video was being benchmarked. The last metric that was used was comparing the system's memory or core activity while the system is idle compared to when it was running our script. It can be seen in the additional images posted with this project's GitHub repository that YOLOv8 causes each system to utilize all of the cores to near their max potential, while the memory plateaus around 5GB of use.

When running the office mp4 file through YOLOv8, the system starts by identifying what version of PyTorch and what model is being used, then it breaks the 20-second video into 541 pieces and proceeds to detect objects in every frame, In order to isolate false positives and overload the edge device, a threshold of 50% was set to not detect an object under that level of confidence. The output from the terminal shows that after each of the 541 pieces is finished being processed, a list of identified objects is listed. After all 541 pieces finish, some resource usage and useful average times are provided that were used to evaluate our benchmarks.

III. RESULTS

For the outcome of the benchmarking tests, we measured inference time or frames per second, memory usage, and power consumption. The inference time was retrieved from the YOLOv8 software and the frames per second can be calculated using the equation: 1 second ÷ inference time. Memory usage was gathered from Jtop, a Jetson Nano profiler, and htop, a system-monitor process-viewer. See Table 1 for the results below:

Device	Video Resolution	Inference Time	FPS	Memory Usage
Msi Laptop using CPU	640x640	25.9ms	38.61	0.6GB
Msi Laptop using GPU	640x640	4.8ms	208	2GB
X360 laptop using CPU	640x640	70ms	14.29	1.2GB
Jetson Nano using CPU	480x384	454.9ms	2.2	1.4GB
Jetson Nano using CPU	640x384	922.9ms	1.1	1.4GB

Table 1: Benchmark performance

The MSI and X360 laptops utilizing the CPU performed mediocre at about 38 and 14 frames per second a piece. One statistic to note is memory usage. The CPU of the MSI laptop has double the cache memory which is why the memory usage was lower. When comparing the CPUs of each device there are notable differences in their specifications. The X360 laptop's CPU has a clock speed of around 4.5GHz compared to the MSI laptop's CPU's 5.1GHz clock rate. This does not always matter in terms of execution time since different variables must be taken into calculation such as instruction count and cycles per instruction (CPI). The other primary specification is the amount of cache each CPU contains. The X360 CPU has 8MB of cache compared to the MSI CPU which contains 16MB of cache. This allows the processor to store double the amount of data for fast and easy access. This can definitely affect the performance greatly in either CPU. Lastly, the amount of cores in each CPU is most important. The X360 CPU has 4 multithreaded cores which contain a total of 8 threads compared to the MSI CPU which has 8 multithreaded cores boasting a total of 16 threads. The MSI CPU is much more powerful compared to the X360 CPU which is why the results show its performance being more than double the efficiency.

The Jetson Nano's performance was not impressive. Around 1 to 2 frames per second of rendering would not be great for live video testing. Since the results are based on the performance of the Jetson Nanos CPU, it is understandable that these were the results we retrieved. The performance of the Jetson Nano utilizing its full capabilities would have garnered far better results. When comparing the performance of image sizes, using a rendering resolution of 480x384 is two times faster than a rendering resolution of 640x384. Since every frame is smaller it makes sense that the efficiency is greater. The memory usage and power consumption of both tests were the same, but since one test took longer it was consuming more overall power. Memory usage was clocked in at around 46 percent of the available 3.86GB of RAM. When comparing the Jetson Nano's CPU to the other laptops tested, it is no competition. The Jetson Nano has a quad-core arm processor without multithreading, meaning a total of 4 threads. The clock rate of the Jetson CPU is around 1.5 GHz and its CPU cache only contains up to 2MB. These specifications are why the Jetson Nano had poor performance compared to the CPUs used in the laptops.

Lastly, the performance of YOLOv8 on the MSI laptop's Nvidia Geforce RTX 2060 GPU was by far the fastest rendering measured. The MSI GPU produced a rendering speed of about 208 frames per second. This is more than five times faster than the next fastest device tested. The reason for such performance while running YOLOv8 utilizing the GPU is due to various components. The first is because of parallel processing. GPUs are designed to handle multiple tasks simultaneously, making them ideal for parallel processing. Computer vision algorithms involve mathematical operations that can be divided into smaller tasks and executed in parallel. GPUs have thousands of more cores compared to the amount in CPUs. This gives GPUs the power to perform tasks more efficiently than CPUs. GPUs also have much higher memory bandwidth compared to CPUs. They can move data in and out of the memory a lot faster. Finally, GPUs are designed to handle the type of calculations that are contained in YOLOv8's object detection software. For instance, floating point operations and CNNs. CPUs, are designed to handle a wider range of tasks and are not optimized for computer vision-specific calculations. So the results of the tested GPU were expected and are far more efficient than any of the CPU-utilized devices.

IV. CHALLENGES

The NVIDIA Jetson Nano edge device presented various issues while attempting to benchmark the YOLOv8 object detection software. To start, YOLOv8 requires a list of prerequisite libraries in order for the software to function properly. One of the primary requirements was Python version 3.7 or higher [4]. The Jetson Nano has a specific operating system setup that requires multiple customized boot partitions. This unique operating system is acquired from NVIDIA's JetPack SDK. JetPack version 4.6 is the latest production release that is officially supported for the Jetson Nano [5]. The critical drawback is JetPack 4.6 uses Ubuntu version 18.04 Linux operating system, which means only Python 3.6 and below can operate on this specific version of JetPack. This was the first issue trying to execute the YOLOv8 software. It was able to be worked around by installing an NVIDIA developer-made custom JetPack version that implements Ubuntu Linux 20.04 [6]. The custom JetPack was very useful for accomplishing the YOLOv8 requirements, but since it is not officially supported, it wasn't as functional for additional customizations.

After installing all requirements for the YOLOv8 software, performing object detection on images and video was successful. When analyzing the efficiency of the Jetson Nano executing YOLOv8 object detection, it could be noticed that the GPU of the device was being utilized minimally. The Jetson Nano was utilizing 95% of the CPU. This was due to the custom JetPack implementation not containing CUDA. CUDA is a parallel computing platform and programming model developed by NVIDIA for use with their GPUs [7]. Without this infrastructure implemented, the GPU can not be utilized to its full potential while performing high-speed video processing. While attempting to install CUDA using numerous methods, errors consistently blocked implementation. It was discovered that CUDA with Ubuntu 20.04 required a specific Linux kernel version. The kernel version required is 5.10.120-tegra or higher [8]. The custom JetPack installed on the Jetson Nano contained kernel version 4.9.140-tegra, which is incompatible. These obstacles unfortunately blocked the full utilization of the GPU on the Jetson Nano. The benchmarking statistics acquired are powered mostly by the CPU on the device, which is not as impressive as if it were powered by the GPU.

V. CONCLUSION

In this project, the rapidly emerging field of Computer Vision was covered with a specific focus on the YOLO(You Only Look Once) model. A prediction was made on how different CPU architectures would significantly affect the model's performance, which was tested by benchmarking a video through the YOLO model over several devices. The findings of the benchmarking provided us with conclusive proof that the computer architecture used for computer vision algorithms is critical in its performance. After this was confirmed across multiple CPUs, a GPU was used to benchmark the same video using the YOLO model. The results of the GPUs benchmarking showed a substantial performance gain in inference time and frames per second, which also followed the initial prediction of GPUs architecture being better for computer vision over CPUs. The overall findings of this project point to the conclusion that architecture is incredibly important when deploying a computer vision model, and there isn't a one size fits all approach that can be taken when selecting the hardware to run these models, things like power efficiency, memory bandwidth, amount of cores, and what specialized hardware accelerators they contain, are all things that need to be taken into account on a case by case basis. Additional details are shared on the associated GitHub repository where a deeper level of metrics is provided for additional analysis.

REFERENCES

[1] P. by&nbsp; &nbsp; &nbsp; Nova and * N., "TPU vs GPU VS CPU: Which hardware should you choose for deep learning?," AITechTrend, https://aitechtrend.com/tpu-vs-gpu-vs-cpu-which-hardware-should-you-choose-for-deep-learning/

[2] "TPU vs GPU: Pros and cons," OpenMetal IaaS, https://openmetal.io/docs/product-guides/private-cloud/tpu-vs-gpu-pros-and-cons/

[3] Yu, Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU platforms for Deep Learning," – arXiv Vanity, https://www.arxiv-vanity.com/papers/1907.10701/

[4] "Xubuntu 20.04 Focal Fossa L4T R32.3.1 - Custom Image for the Jetson Nano," NVIDIA Developer Forums, May 02, 2020. https://forums.developer.nvidia.com/t/xubuntu-20-04-focal-fossa-l4t-r32-3-1-custom-image-for-the-jetson-nano/121768

[5] "JetPack SDK 4.6 Release Page," NVIDIA Developer, Jul. 27, 2021. https://developer.nvidia.com/embedded/jetpack-sdk-46

[6] Ultralytics, "YOLOv8" GitHub, May 08, 2023. https://github.com/ultralytics/ultralytics

[7] "CUDA Zone - Library of Resources," NVIDIA Developer, Jul. 18, 2017. https://developer.nvidia.com/cuda-zone

[8] "NVIDIA CUDA Installation Guide for Linux," Nvidia.com, 2021. https://docs.nvidia.com/cuda/cuda-installation-guide-linux