Scholarly Paper

Evaluating the Performance Comparison of NVIDIA Jetson Nano with different Object Detection Applications

Wenzhe Luo
ECE 618: Hardware Accelerators for Machine Learning

Abstract:

Object detection is a challenging task that requires efficient hardware to achieve satisfactory performance, including autonomous driving, security surveillance, and robotics. In recent years, hardware accelerators such as NVIDIA Jetson Nano have gained popularity for their ability to improve the performance of object detection models. In this paper, I evaluate the performance of NVIDIA Jetson Nano with different object detection applications, specifically YOLOv5 and YOLOv7, and compare it with the performance of a personal laptop with 4080 Ti GPU and CPU. I conduct experiments using a webcam and a dataset of images and evaluate the models using standard evaluation metrics (fps, CPU consumption). The results show that NVIDIA Jetson Nano outperforms the laptop's i7-12700H CPU for object detection tasks. I also find that the performance of YOLOv7 is better than YOLOv5 on both hardware platforms. My study provides insights into the benefits of hardware accelerators for object detection and highlights the potential applications of NVIDIA Jetson Nano in real-world scenarios.

Key words: Object Detection, Hardware Acceleration, NVIDIA Jerson Nano, YOLOv5 & v7, Machine Learning, GPU, CPU, Image Processing

1 INTRODUCTION

Object detection is a critical task in computer vision that involves identifying and localizing objects of interest within an image or video. This task has many important applications, such as in security and surveillance systems, traffic monitoring, robotics, and autonomous vehicles [1, 2, 3]. Object detection can be used to identify objects in real-time, making it an essential component of many real-world applications. Most algorithms of Object detection typically involve multiple stages, including feature extraction, object proposal generation, and classification [4]. These algorithms use complex mathematical models and require significant processing power and memory. This can be a significant challenge [5], particularly in real-time applications that require fast and accurate detection of objects. To address this challenge, hardware accelerators have been developed to offload some of the computation to specialized hardware. These accelerators can significantly speed up the processing of object detection algorithms, enabling real-time detection of objects in a variety of applications.

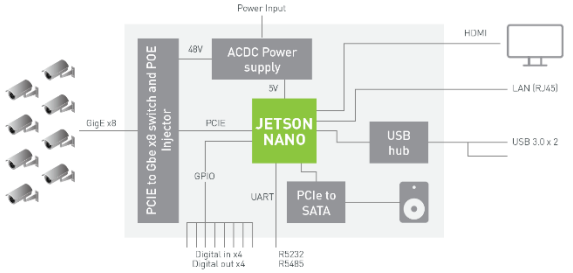

Figure 1. Reference NVR system architecture with Jetson Nano and 8x HD camera inputs [6]

Overall, object detection is an essential task in computer vision, and its importance continues to

grow as more real-world applications emerge. The development of hardware accelerators, such as the NVIDIA Jetson Nano, provides a promising avenue for improving the speed and accuracy of object detection algorithms, enabling more robust and efficient object detection systems.

This article aims to evaluate the performance of the NVIDIA Jetson Nano when used in object detection applications. Specifically, comparing the performance of Jetson Nano on GPU and CPU using YOLOv5 and YOLOv7 object detection models. For this, we will use a webcam connected to a Jetson Nano (for the real-time detection implementation) and compare the results with those obtained using a laptop equipped with an NVIDIA 4080 Ti GPU.

The main research question I aimed to answer was: How does the NVIDIA Jetson Nano perform compared to high-end GPUs when used in object detection applications? The main objective is to determine the speed and CPU usage of the Jetson Nano for object detection and evaluate the benefit of using a hardware accelerator for this task. By doing so, we hope to gain insight into the potential of the Jetson Nano for practical object detection applications.

In the rest of this paper: Section 2 reviews the relevant experimental background and provides motivation; Section 3 explains the internal structure of NVIDIA Jetson Nano and the evolution of the YOLO model family for object detection. Section 4 defines the experimental method. Experimental results are shown in Section 5 and concluding comments are given in Section 6.

2 RELATED WORK AND MOTIVATION

Cause: Hardware depends on the specific requirements of the application

Last semester, in the final project (Human Shape Recognition) of ECE554, our group designed and practiced the MobileNetv2 object recognition model around Arduino 33 BLE and machine learning. Sandler et al. (2018) [7] introduced the original MobileNet architecture, which uses depth wise separable convolutions to reduce the number of parameters and computation required for image classification tasks. They evaluated the performance of the network on several benchmark datasets, including the COCO dataset, and showed that MobileNet achieves high accuracy with low computational cost. However, The Arduino 33 BLE has limited memory, which can make it difficult to store large datasets like COCO. This may require down sampling or reducing the size of the dataset, which can affect the accuracy of the model. In addition, The Arduino 33 BLE is a low-power microcontroller board, which means that it has limited processing power compared to more powerful devices like desktop computers or specialized hardware like GPUs. This makes the whole project inconsistent with the topic of hardware accelerators.

Motivation: Choosing Nvidia Jetson Nano as the hardware basis for the whole experiment

Regarding the choice between Jetson Nano and Arduino 33 BLE, Arduino 33 BLE is designed for low-power applications and has limited computational resources compared to Jetson Nano that is a more powerful device than Arduino 33 BLE and is better suited for applications that require high-performance computing, such as object detection at higher resolutions or with larger datasets. As Figure 1 shows, Jetson Nano has several connectivity options, including Ethernet, USB, and Wi-Fi, which enable easy streaming of data to and from the device. This makes it well-suited for real-time applications like object detection. Jetson Nano is also equipped with a powerful NVIDIA GPU that enables high-performance machine learning applications. This GPU can accelerate deep

learning algorithms by several orders of magnitude, making it well-suited for complex tasks like object detection.

Observation: Models and Dataset change experimental direction

Machine learning models and datasets play a crucial role in the direction and success of experimental outcomes in various fields. The choice of model and dataset can greatly impact the accuracy and performance of a machine learning system. Redmon and Farhadi (2017) state that "the choice of architecture for object detection can have a significant impact on both accuracy and speed." By choosing a well-designed model, researchers and practitioners can achieve better accuracy and faster processing times, ultimately leading to more successful machine learning and object detection applications.

Motivation: YOLO model can provide fast and accurate object detection

The choice between MobileNetV2 and YOLOv5 depends on the specific requirements of the application, such as processing power, real-time performance, accuracy, and customization options. They are both deep neural network architectures used for object detection. However, MobileNetv2 uses depth wise separable convolutions to reduce the number of parameters and computational cost of the network while maintaining high accuracy. In comparison, YOLOv5 and other latest version model use a fully convolutional architecture with anchor-based box prediction to achieve high accuracy. In addition, YOLO model typically used on more powerful devices such as desktop computers or edge devices with dedicated hardware acceleration. In order to better study hardware accelerators, I chose Jetson nano. On this basis, I decided to experiment with the project around the YOLO model.

3 JETSON NANO AND YOLO MODEL

Section 2 has briefly explained why jetson nano is used to replace arduino 33 ble in this article. This section explains the structure of jetson nano and the significance of its creation for the IoT, ML, and object detection.

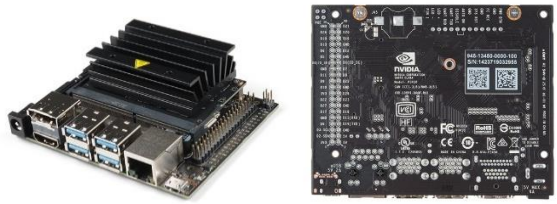

Figure 2. NVIDIA Jetson Nano

NVIDIA Jetson Nano (as Figure 2 shows) is a small, powerful computer designed for embedded systems and AI applications. It is part of the NVIDIA Jetson family of devices designed to accelerate AI and machine learning tasks. According to Lofqvist (2020) [10], “This device comes with a 128-core integrated NVIDIA Maxwell GPU and a quad-core 64-bit ARM CPU. It has 4GB of LPDDR4 memory at 25.6GB/s and 5W/10W power modes” (p. 07). Processing power is the key advantage, which provides significant processing capabilities for AI and machine learning workloads. In addition, 4GB of LPDDR4 memory could make this device ideal for processing large amounts of data, such as COCO dataset. [10, 11]

Another advantage of the Jetson Nano is its size and power consumption. It's small enough to fit in the palm of your hand, and it only requires a 5V power supply, which means it can be used in a wide range of applications where power and space are limited. This makes it a popular choice for robotics, drones, and other embedded systems. NIVIDA Jetson Nano is also designed to work seamlessly with NVIDIA's deep learning software libraries, including ***TensorRT*** and ***DeepStream***. This makes it easy to develop and deploy machine learning applications on the

device, without needing to spend time optimizing the code for the hardware.

The purpose of this study is to compare the performance of machine learning models (two cases: the case using hardware accelerators and the case using only CPU), especially the performance of YOLOv5 and YOLOv7 on different hardware platforms.

3.1 YOLOv5

YOLOv5 (You Only Look Once version 5) is an object detection algorithm that is designed to detect and localize objects in an image. It is a convolutional neural network (CNN) that uses a single neural network to perform object detection in real-time. Jocher (2021) [12] proposed a new object detection algorithm, YOLOv5, which achieved state-of-the-art performance on several benchmark datasets.

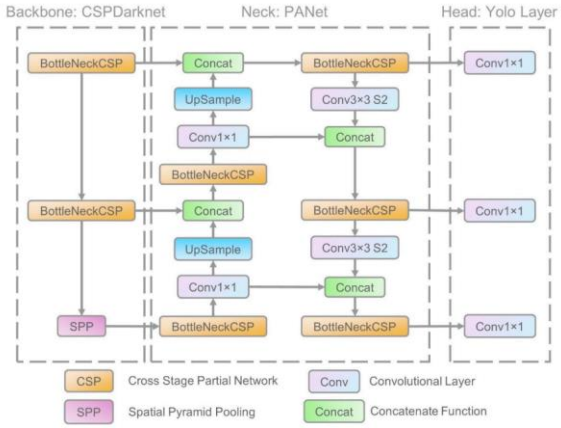

Figure 3. The network architecture of YOLOv5

According to the paper [13] "A Forest Fire Detection System Based on Ensemble Learning", the network architecture of Yolov5 shown in Figure 3 is composed of three parts: the Backbone (CSPDarknet), the Neck (PANet), and the Head (Yolo Layer), where the data undergoes feature extraction by CSPDarknet, feature fusion by PANet, and finally, detection results (class, score, location, size) output by Yolo Layer (Xu, R, 2021). YOLOv5 is a powerful and efficient object detection model that achieves state-of-the-art accuracy while remaining fast and easy to use. Through the application of jetson nano and 4080Ti GPU, I found that YOLOv5 is a powerful and efficient object detection model that achieves state-of-the-art accuracy while remaining fast and easy to use. By using a more powerful backbone network (CSPDarknet53) and Spatial Pyramid Pooling (which is a more efficient feature aggregation method), YOLOv5 improves upon its predecessors (version 3 & 4) in terms of accuracy.

The Backbone is responsible for extracting features from the input image. In YOLOv5, the backbone is implemented using a variant of the Darknet architecture called CSPDarknet, which uses cross-stage partial connections to improve feature representation. The neck is responsible for feature fusion at different scales. In YOLOv5, the neck is implemented using the PANet architecture, which uses feature map pyramids of different resolutions to capture object details at different scales. The head is responsible for object detection and localization. In YOLOv5, the head is implemented using a YOLO layer, which outputs a set of bounding boxes, confidence scores, and class probabilities for objects in the input image.

In the past few years since yolov5 came out, it has gradually developed into a popular machine learning model for object recognition and real-time detection. The benefits of YOLOv5 extend beyond its impressive performance metrics. YOLOv5's real-time object detection capabilities can be used in various applications, including public safety, agriculture, manufacturing, and autonomous vehicles. For example, according to Nepal (2022) [14], YOLOv5 can be used to detect and identify potential landing spots in real-time, improving the safety and reliability of the landing process. In the event of a faulty UAV, YOLOv5's object detection capabilities can help the UAV navigate and land safely, even in situations where human intervention may not be possible or

practical, in addition, YOLOv5 could help improve reducing the risk of damage or injury and increasing the effectiveness of UAV operations.

YOLOv5 model [12] includes lots of pre-trained weights that can be used to initialize the model for object detection tasks. Figure 4 describes different YOLOv5 weights with their size and mAP.

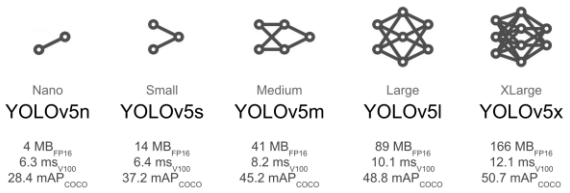

Figure 4. Different YOLOv5 model (weights) description

In this experiment, I picked yolov5s.pt and yolov5l.pt as the conditions, where ***yolov5s.pt*** is the smallest YOLOv5 variant, with fewer parameters and lower accuracy than the larger models. It is useful for real-time applications that require fast processing speeds and can detect objects in images at a resolution of 640x640. As the opposite side of this smallest variant, I picked ***yolov5l.pt*** as the larger size weight which offers higher accuracy than the smaller models, but at the cost of increased processing time. It can detect objects in images at a resolution of 1280x1280 and is useful for more complex object detection tasks.

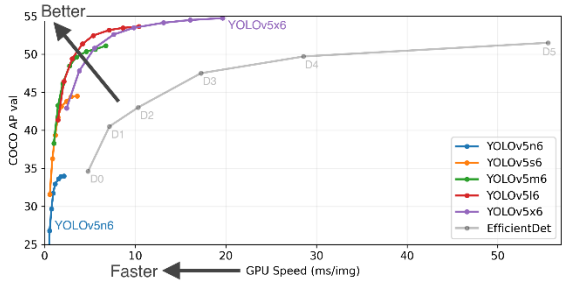

Figure 5. GPU Speed performance between different models of YOLOv5

The main differences between the different YOLOv5 variants are the number of layers and the number of parameters in the network. As the number of layers and parameters increase, the network becomes more complex and capable of capturing more detailed features from the input image, but also requires more computational resources and may be slower. As Figure 5 shows [12], in terms of GPU speed performance, the YOLOv5 variants have different requirements depending on their size and complexity, the professional website of YOLOv5 [12] explained the performance for *GPU Speed measures end-to-end time per image averaged over 5000 COCO val2017 images using a V100 GPU with batch size 32, and includes image preprocessing, PyTorch FP16 inference, postprocessing and NMS.*

3.2 YOLOv7

According to Wang (2022) [16], YOLOv7 is a new object detection algorithm that builds upon the YOLOv5 architecture, but with several modifications and improvements to achieve state-of-the-art performance in terms of accuracy and speed. YOLOv7 greatly improves the accuracy of real-time object detection without increasing the cost of inference. Compared with other known object detectors, YOLOv7 can effectively reduce state-of-the-art real-time object detection by about 40% of the parameters and 50% of the calculation and achieve faster reasoning speed and higher detection accuracy.

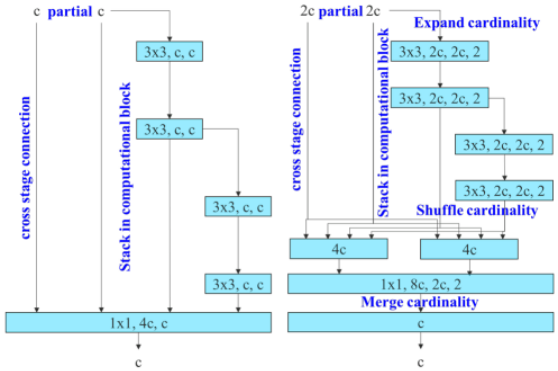

Figure 6. Extended efficient layer aggregation networks, left (ELAN), right (E-ELAN)

The YOLOv7 architecture is based on the previous YOLO model architecture, Figure 6

shows the ELAN architecture and E-ELAN architecture, where the extended high-efficiency layer aggregation network (E-ELAN), and the computing block in the YOLOv7 backbone network is named E-ELAN, which stands for the extended high-efficiency layer aggregation network. Shown in Figure 6 [17]. According to Jiang (2022) [17], “extended ELAN (E-ELAN) based on ELAN is proposed. Expand, shuffle, and merge cardinality are used to continuously enhance the learning ability of the network without destroying the original gradient path.”

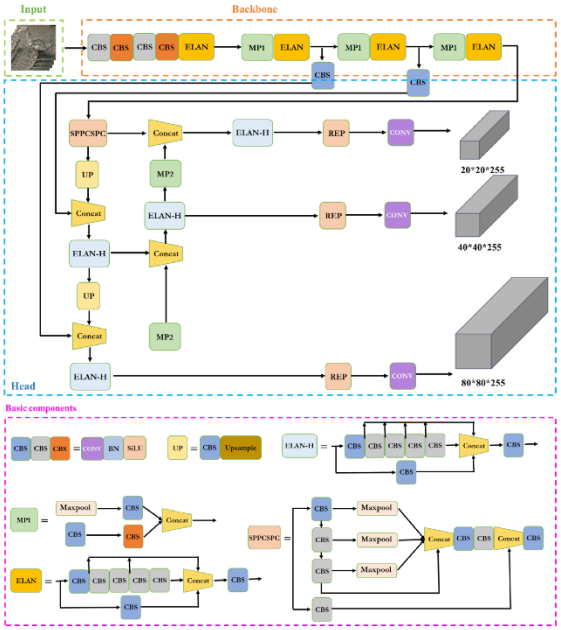

Figure 7. The network architecture of YOLOv7

By traversing Figure 7, it could be analyzed: firstly, resize the input picture to 640x640 size, input it into the backbone network, then output three layers of **feature map** with different sizes through the head layer network, and output it through REP and CONV forecast result. Taking the COCO dataset as an example, the output is 80 categories, and then each output (x, y, w, h, o) is the coordinate position and background, 3 refers to the number of anchors, so the output of each layer is (80 + 5) * 3 = 255, and then multiplied by the size of the feature map is the final output.

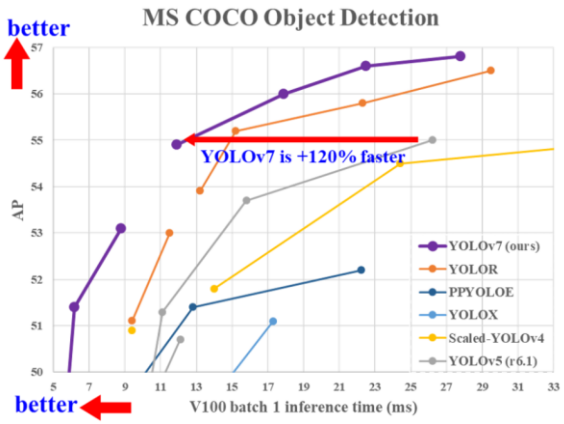

Figure 8. GPU V100 batch 1 inference time performance between different object detectors

The above Figure 8 [15, 16] explained the performance comparison between different object detectors for object detection by using V100 GPU with batch 1 in MS COCO dataset. It could be found that YOLOv7 is faster than YOLOv5, where gray represents YOLOv5, and YOLOv7 is purple. When the demand for AP is both reached to 55, YOLOv7 has a better FPS than YOLOv5, and the inference time is 120% faster. According to Wang (2022) [16], In the range of 5FPS to 160FPS, whether it is speed or accuracy, YOLOv7 exceeds the currently known detectors, and tested on GPU V100, the model with an accuracy of 56.8% AP can reach 30 FPS (batch=1). At the same time, this is the only detector that can still exceed 30FPS with such high precision.

3.3 DeepStream and TensorRT

This section introduces DeepStream and TensorRT from NVIDIA. DeepStream provides a platform for building intelligent video analytics applications, while TensorRT provides a library for optimizing deep learning models for efficient inference on NVIDIA GPUs. DeepStream and TensorRT are often used together to enable high-performance, real-time video analytics applications that require efficient processing of large amounts of data. In this project, the combination of DeepStream and TensorRT on the

NVIDIA Jetson Nano can be utilized to deploy YOLOv5 & YOLOv7 object detection models on edge devices with high efficiency and real-time performance.

3.3.1 DeepStream

DeepStream is an NVIDIA software platform for building and deploying intelligent video analytics (IVA) solutions at scale. According to Liu (2022) [18], DeepStream provides a unified development environment and runtime for creating and deploying deep learning-based applications for video analysis. As Figure 9 [19] shows, DeepStream includes pre-trained models and libraries optimized for NVIDIA GPUs, as well as a range of tools for video processing, analysis, and visualization.

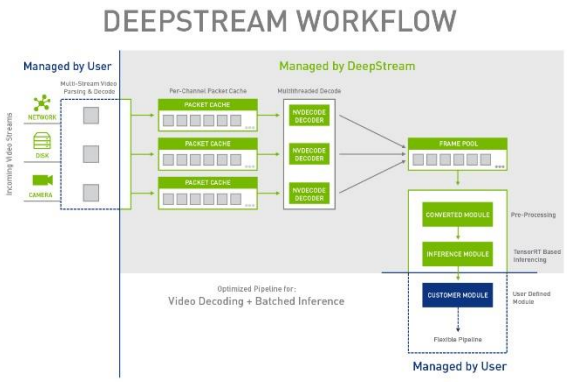

Figure 9. The DeepStream pipeline

DeepStream plays the key role to make video deep learning accessible, and it focuses on quickly building and customizing efficient and scalable video analytics applications.

3.3.2 TensorRT

Tensor is a library that facilitates high-performance inference on NVIDIA Graphics Processing Units (GPUs). It is designed to work in a complementary manner with training frameworks such as TesnsorFlow, Caffe, Pytorch, and MXNet, and is dedicated to fast and efficient network inference on GPUs. After training the neural network, TensorRT can compress, optimize, and deploy the network at runtime without the overhead of the framework. TensorRT improves the latency, throughput, and efficiency of the network by combining layers, optimizing kernel selection, and performing normalization and conversion into the optimal matrix math method according to the specified accuracy.

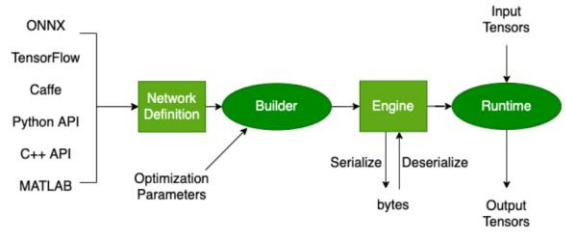

Figure 10. The TensorRT workflow [20]

Overall, TensorRT enables developers to accelerate and optimize their deep learning inference workloads, making it a popular choice for applications such as object detection, natural language processing, and recommender systems.

4 EXPERIMENTS

The Jetson Nano is a small, low-power single-board computer powered by an NVIDIA GPU that can be used to accelerate inference on deep learning models such as YOLOv5. Therefore, we used jetson nano as the edge device, yolov5 and yolov7 as our two sets of test models, and analyzed the performance with hardware acceleration through the experimental results. And using the basic GPU of Google Colab as a reference to analyze the results, it is concluded that jetson nano's performance has increased by 6.x times with the assistance of DeepStream and TensorRT.

4.1 Setting-up experiment environments

The authors of the YOLO family used the PyTorch deep learning framework to implement the algorithm. PyTorch is a widely used deep learning framework that provides a wealth of tools and functions for defining, training, and deploying neural network models. PyTorch has

the feature of dynamic calculation graph, which makes the construction and debugging of models more flexible and intuitive. These features make PyTorch the framework of choice for many researchers and engineers. Torchvision is an extension library of PyTorch that provides convenient tools and functions for computer vision tasks. Their combination makes developing and researching computer vision models more efficient and convenient.

To compare the performance of different hardware accelerators, I decided to use Jetson Nano as an experimental baseline and my laptop as a reference for the highest level results and use these two baselines to compare the jetson nano's hardware accelerator performance on the machine.

I. NIVIDA Jetson Nano devices:

Through Section 3, we understand the specifications of jetson nano: CPU: Quad-core ARM Cortex-A57, clock frequency 1.43GHz; GPU: 128-core NVIDIA CUDA GPU, using NVIDIA Maxwell architecture; Memory: 4GB LPDDR4 memory. Jetson Nano provides powerful computing and graphics processing capabilities, suitable for developing and deploying various AI applications, but as an entry-level edge hardware accelerator, jetson nano can only support pytorch and torchvision versions 1.10.0 and 0.11.0 (respectively). CUDA 11 cannot be installed on the Nano due to low-level GPU incompatibilities. Pytorch 2.0 and above is required to use CUDA 11. The Jetson Nano only has CUDA 10.2.

II. Laptop & Google CoLab:

II.1 Laptop

Different from the version of jetson nano, when using a notebook as an experimental device, the version of pytorch is 2.0.1, the version of CUDA is 11.8, and the version of torchvision is 0.15.1, as Figure 11 shows.

```
>>> import torch
>>> torch.__version__
'2.0.0+cu118'
>>> import torchvision
>>> torchvision.__version__
'0.15.1+cu118'
```

Figure 11. The version of torchvision and PyTroch

II.2 Google CoLab

Google Colab provides users with free GPU acceleration, usually an NVIDIA Tesla T4 GPU. This is very valuable for users who need to do deep learning or other computationally intensive tasks, because the GPU can significantly accelerate the speed of model training and inference.

Table 1. Different hardware specifications

Model	CPU	GPU	Memory
Jetson Nano	Quad-core ARM A57 @ 1.43 GHz	128-core Maxwell	4 GB 64-bit LPDDR4
Laptop	12th Generation Intel Core i7-12700H	NVIDIA GeForce RTX 3080 Ti	32 GB 64-bit LPDDR5
Google CoLab	Intel Xeon CPU 2.30GHz	Tesla T4	12.72 GB

III. Dataset:

According to Lin (2014) [22, 23], The COCO (Common Objects in Context) dataset is an important dataset widely used in the field of computer vision. Its main purpose is to advance the development and evaluation of computer vision tasks such as object detection, object recognition, and semantic segmentation. The COCO dataset contains hundreds of thousands of images and annotations of millions of objects instances and has a considerable scale. The COCO dataset contains many images in real-world scenes, covering various categories of objects, and annotating the location and category information of these objects. The images in the dataset have different backgrounds, viewing angles, scales, and occlusions, which enable the

model to have better generalization capabilities in different scenarios.

Figure 12. Samples of MS COCO dataset [21]

YOLOv5 chose to use the COCO dataset because it provides large-scale, diverse, and high-quality labeled data, has a unified evaluation standard, and contains some challenging scenes and objects. Using the COCO dataset can help the YOLOv5 model learn objects of various categories and scenes, thereby improving the generalization ability and recognition accuracy of the model. Many researchers and developers use the COCO dataset to evaluate and improve their models. Using the COCO dataset, YOLOv5 can be compared with other algorithms and validate its own performance and improvements by referring to benchmark results.

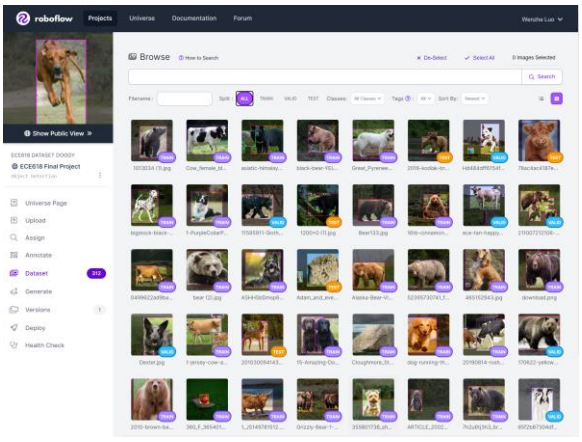

Figure 13. page of the Dataset created [23]

At the same time, I will also mark my own data set (312 pictures of animals (including dogs, bears, cows) of which 252 are used for training, 35 are used for validation, and 32 are used for testing), the platform are roboflow (showing in Figure 13) and Google Colab.

5 IMPLEMENTATION & PERFORMANCE

When applying the YOLO model, I divided it into two experimental steps. The first part is to compare the object detection performance on different devices using pre-trained YOLO weights. The second part involves creating a custom database and evaluating the model using appropriate evaluation metrics (mAP). Pre-trained weights are typically trained for a long time on large-scale datasets, resulting in good initial weights. These weights capture rich visual features and object class information, helping the model converge faster and improve its initial detection performance. Using trained weight files, perform model inference, real-time object detection, inference on images/videos, and visualize the results. Adapt to the specific hardware and software environments according to the platform and task requirements to ensure the successful application of the YOLOv5 model. Using the weights of YOLOv5s and YOLOv5l on different devices, as shown in Table 2, we can observe that under the same size of 640 pixels, the fps result of YOLOv5l weight is significantly better than YOLOv5s. YOLOv5l has a deeper network structure compared to YOLOv5s. It has more convolutional layers and parameters, allowing it to extract richer feature representations. This makes YOLOv5l more expressive and capable of modeling in object detection tasks, enabling it to better capture the details and contextual information of the targets. The advantage of YOLOv5l over YOLOv5s lies in its higher performance and accuracy, but it also comes with higher computational complexity and

Table 2. Comparison of the three devices under three different weights (since the 4080Ti and Tesla T4 are several levels better than the Jetson nano GPU, we use the performance of Jetson Nano as a baseline)

Model (weights)	Device	Size (pixels)	mAP 50	Speed (CPU) ms	Speed (GPU) ms	Speedup CPU to GPU	Params (M)	Layers
YOLOv5s	Jetson Nano*	640	73.5%	0.78 - 3.76	2.98 - 32	8.51X	7.23	213
	Laptop			9	128	14.22X		
	Google Colab			4.69	100	21.32X		
YOLOv5l	Jeson Nano*		83.6%	0.36 - 2.83	2.17 - 26	9.18X	46.53	367
	Laptop			6	96	16X		
	Google Colab			3.31	59	25.54X		
YOLOv7	Jeson Nano*		85.4%	0.21 - 1.72	1.58 - 11	6.39X	36.90	306
	Laptop			4	72	18X		
	Google Colab			1.78	44	25.28X		

memory consumption. Therefore, when selecting a model, it is important to consider, and balance based on the specific application scenario and hardware resources.

5.1 mAP for different YOLO weights & Train Custom Dataset

In machine learning, mAP stands for mean Average Precision. It is a metric used to evaluate the performance of tasks such as object detection and image classification. In object detection tasks, mAP measures the balance between precision and recall of a model on different classes of objects. Precision refers to the ability of the model to correctly detect objects, and recall refers to the ability of the model to find all objects. Compared with YOLOv5s from Table 2, YOLOv5l performs better on mAP mainly due to its deeper network structure and higher receptive field. YOLOv5l has a deeper network structure than YOLOv5s, and it has more convolutional layers and parameters. This enables YOLOv5l to extract richer feature representations for more accurate object detection and recognition. Since the YOLOv5l network is deeper, the receptive field of each convolutional layer is larger, which can capture broader context information. This helps the YOLOv5l model to better understand the background and environment around the target and improve the accuracy of detection.

When training my own dataset with the YOLOv5 algorithm, I imported my image data using the Roboflow platform and annotated the images with their annotation tool to label the object bounding boxes. Then, I applied appropriate data preprocessing techniques such as scaling and cropping. Next, I converted the annotated data into the required format for YOLOv5 and downloaded it along with the image data. Finally, I trained the model using the YOLOv5 training script and the prepared dataset, adjusting and optimization as needed. This process enabled me to perform data annotation and model training for object detection tasks, improving accuracy and performance.

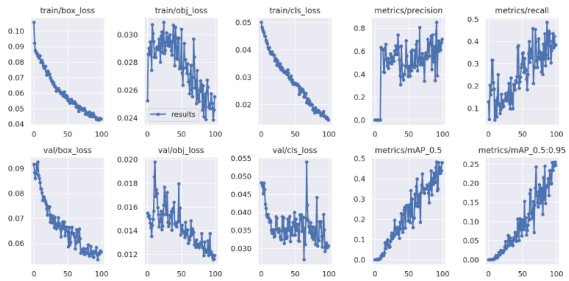

Figure 14. Evaluate Custom YOLOv5 Detector Performance

Figure 14 illustrates the performance of the custom dataset in the YOLOv5 detector. During training, the ideal outcome is both “box_loss” and

"obj_loss" to be as low as possible to ensure bounding box accuracy and object detection performance. At the same time, both precision and recall can be high, to achieve the goal of accurate recognition and comprehensive detection of targets. The comprehensive performance of these indicators can help us evaluate the performance of the model and adjust and optimizations.

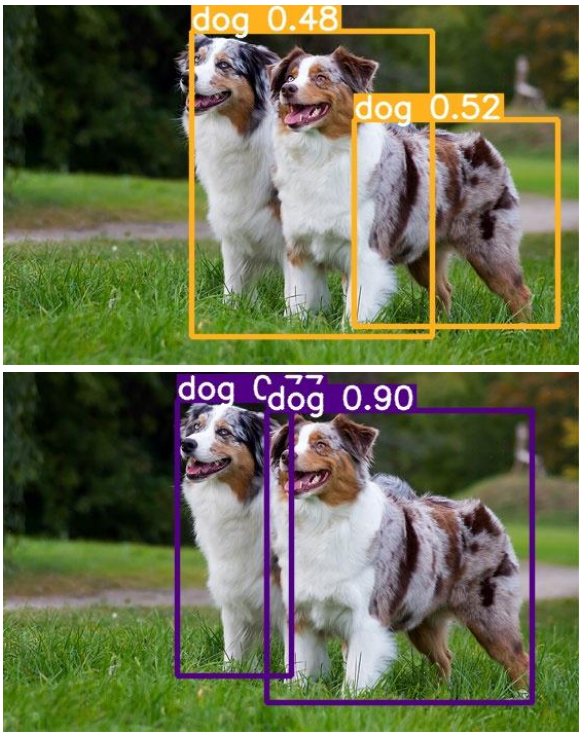

Figure 15. Comparison of custom dataset (Top) and COCO dataset (Bottom) test results under YOLOv5 algorithm

The reason for the significant discrepancy between the results of testing (shows as Figure 15) with the COCO dataset and the custom dataset model is due to inadequate training of the custom model, as custom dataset only consists of 501 images. The dataset size is an important factor that affects model performance. The COCO dataset is relatively large, containing a vast number of images and object categories, whereas your dataset is limited to only 501 images. This difference in dataset size can result in significant gaps between the testing results on your dataset and the COCO dataset. Training deep learning models typically require a large amount of data to capture various object variations and scenes. Smaller datasets are more prone to overfitting, where the model performs well on the training set but lacks generalization ability on other datasets.

5.2 Factors affecting inference time (fps)

FPS (frames per second) are an important indicator to measure the performance of an object recognition system. FPS measures the processing speed of an object recognition system, which is the number of image frames per second it can process. For many real-time application scenarios, such as video surveillance, autonomous driving, and real-time face recognition, fast object recognition enables timely response and real-time decision making. A higher FPS value means that the system can process images faster and provide real-time results.

Factors that affect the inference time of YOLOv5 and YOLOv7 models when running on GPU (graphics processing unit) and CPU (central processing unit) mainly include the following aspects:

I. Hardware Performance

GPU has higher computing power in parallel computing than CPU and is suitable for deep learning tasks. In general, the inference time for running YOLOv5 models using GPUs is lower because GPUs can process multiple data at the same time and perform parallel computations. Experimental results show that GPU brings many advantages to object recognition. Under the Jetson nano device, compared to only using the CPU, using the GPU can speed up the object recognition application by an average of more than 8 times.

II. Hardware-accelerated Libraries

Using appropriate deep learning acceleration libraries (such as CUDA, cuDNN, etc.) can increase the speed of inference running on GPUs. In addition, reasonable model optimization techniques (such as batch processing during

inference, quantization, etc.) can also improve the inference efficiency of the model. For example, in using jetson nano, I also used two optimizations of Deepstream and tensorRT.

III. Model (Weights) Complexity

The complexity of the YOLOv5 model has a direct impact on inference time. Larger models (such as yolov5l) typically contain more layers and parameters, thus requiring more computing resources and time to complete inference. In contrast, smaller models such as yolov5s are faster at inference time.

5.3 Comparison Between Models (Weights)

YOLOv7 provides different model variants, including YOLOv7.pt and YOLOv7-tiny.pt. These variants have some differences in terms of model size and performance. As shown from Figure 16, YOLOv7-tiny.pt (right, 200 layers) is the smallest model in the YOLOv7 series, with relatively few parameters, suitable for resource-constrained environments. YOLOv7.pt (left, 306 layers) is a larger model with more parameters and thus requires more computing resources.

YOLOv7-tiny.pt is relatively small, so the inference speed is fast. From Figure 14, the inference time by using YOLOv7-tiny.pt just taken 0.171s, which is faster than the larger weight YOLOv7.pt (taking 0.312s) which is suitable for real-time applications.

Deeper models can capture richer semantic information through multiple levels of feature extraction and cascaded processing. This enables the model to better understand details such as object shape, texture, context, etc. in the image, leading to more accurate object detection and classification. YOLOv7.pt is relatively slow in terms of inference speed, but has higher detection precision and recall, which is suitable for tasks that pay more attention to accuracy. Figure 16 clearly tells us the impact of different model-

Figure 16. Different recognition results under different weights (YOLOv7)

depths. It can be found that the accuracy of YOLOv7-tiny (with 200 layers and 6.22 params) models is very low, such as defining blurred objects as traffic lights. In comparison, the deep model (YOLOv7 with 306 layers and 36.9 M params) can accurately detect the cyclist.

6 CONCLUSIONS

In this project, I used the Jetson Nano as an embedded computing platform and applied two different object detection models, YOLOv5 and YOLOv7 (with three different weights). I analyzed the value of hardware accelerators by using the GPU of the Jetson Nano as a reference. Additionally, I trained a custom dataset using the YOLOv5 algorithm, focusing on detecting dogs. Building upon that, I explored the use of DeepStream and TensorRT on the Jetson Nano as acceleration tools to reduce inference time and increase frames per second (FPS).

Through this practical work, I gained an understanding of the significance of hardware accelerators. They leverage specially designed hardware circuits and optimized algorithms to accelerate complex computational tasks, thereby improving the performance and efficiency of machine learning applications. The Jetson Nano, as an entry-level hardware accelerator, greatly speeds up the processing of object detection and recognition tasks. By combining optimized hardware and software algorithms, the accelerator can efficiently perform convolutions, matrix computations, and other complex operations required by neural networks, enabling real-time object detection and tracking.

Furthermore, the YOLO model, as an efficient and accurate object detection algorithm, provides a powerful tool for the machine learning community. It assists developers in building higher-performance and more practical object detection applications, thereby advancing the application and development of artificial intelligence technology across various fields.

REFERENCES

[1] Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2023). Object detection in 20 years: A survey. Proceedings of the IEEE.

[2] Hossain, S., & Lee, D. J. (2019). Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with GPU-based embedded devices. Sensors, 19(15), 3371.

[3] Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z., & Qu, R. (2019). A survey of deep learning-based object detection. IEEE access, 7, 128837-128868.

[4] Padilla, R., Netto, S. L., & Da Silva, E. A. (2020, July). A survey on performance metrics for object-detection algorithms. In 2020 international conference on systems, signals and image processing (IWSSIP) (pp. 237-242). IEEE.

[5] Mariano, V. Y., Min, J., Park, J. H., Kasturi, R., Mihalcik, D., Li, H., ... & Drayer, T. (2002, August). Performance evaluation of object detection algorithms. In 2002 International Conference on Pattern Recognition (Vol. 3, pp. 965-969). IEEE.

[6] Franklin, D. (2022, August 21). Jetson Nano brings AI computing to everyone. NVIDIA Technical Blog. from https://developer.nvidia.com/blog/jetson-nano-ai-computing/

[7] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

[8] Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

[9] Iyer, R., Ringe, P. S., & Bhensdadiya, K. P. (2021). Comparison of YOLOv3, YOLOv5s and MobileNet-SSD V2 for real-time mask detection. Artic. Int. J. Res. Eng. Technol, 8, 1156-1160.

[10] Lofqvist, M., & Cano, J. (2020). Accelerating deep learning applications in space. arXiv preprint arXiv:2007.11089.

[11] Othman, N. A., Saleh, Z. Z., & Ibrahim, B. R. (2022, March). A Low-Cost Embedded Car Counter System by using Jetson Nano Based on Computer Vision and Internet of Things. In 2022 International Conference on Decision Aid Sciences and Applications (DASA) (pp. 698-701). IEEE.

[12] https://github.com/ultralytics/yolov5

[13] Xu, R., Lin, H., Lu, K., Cao, L., & Liu, Y. (2021). A forest fire detection system based on ensemble learning. Forests, 12(2), 217.

[14] Nepal, U., & Eslamiat, H. (2022). Comparing YOLOv3, YOLOv4 and YOLOv5 for autonomous landing spot detection in faulty UAVs. Sensors, 22(2), 464.

[15] https://github.com/WongKinYiu/yolov7

[16] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv preprint arXiv:2207.02696.

[17] Jiang K, Xie T, Yan R, Wen X, Li D, Jiang H, Jiang N, Feng L, Duan X, Wang J. An Attention Mechanism-Improved YOLOv7 Object Detection Algorithm for Hemp Duck Count Estimation. Agriculture. 2022; 12(10):1659. https://doi.org/10.3390/agriculture12101659

[18] Liu, S., Feng, J., Zhang, Q., & Peng, B. (2022, August). A Real-Time Smoke and Fire Warning Detection Method Based on an Improved YOLOv5 Model. In 2022 5th International Conference on Pattern Recognition and Artificial Intelligence (PRAI) (pp. 728-734). IEEE.

[19] https://developer.nvidia.com/blog/deepstream-video-analytics-smart-cities/

[20]https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html
[21] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In European conference on computer vision (pp. 740-755). Springer.
[22] https://cocodataset.org/#home
[23] https://app.roboflow.com/ece618-dataset-doggy