A Comparative Study on the Performance of Different Deep Learning Hardware

by

NI Ronghao

A thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science (Honours)
in Computer Science

Hong Kong Baptist University

April, 2020

Declaration

I hereby declare that all the work done in this Final Year Project is of my independent effort. I also certify that I have never submitted the idea and product of this Final Year Project for academic or employment credits.

_____

NI Ronghao

Date: _____

Hong Kong Baptist University
Computer Science Department

We hereby recommend that the Final Year Project submitted by NI Ronghao entitled "A Comparative Study on the Performance of Different Deep Learning Hardware" be accepted in partial fulfillment of the requirements for the degree of Bachelor of Science (Honours) in Computer Science.

Prof. Chu Xiaowen	Dr. Huang Xin
Supervisor	Observer
Date: _____	Date: _____

Acknowledgement

First of all, I would like to convey my sincerest gratitude to my supervisor Prof. Chu Xiaowen, who had not only given me careful guidance and help in the Final Year Project, but also supported me and introduced me to many activities throughout the four year undergraduate study, which enriched my knowledge and broadened my horizons.

I would also like to thank Dr. Huang Xin for his kind comments and suggestions on my Final Year Project.

I also want to thank Mr. Wang Qiang for helping set up the environments at the early stage and let me start the project.

I would like to take this opportunity to thank all my teachers, friends and classmates. Without you, I would not have this happy and memorable time at Hong Kong Baptist University.

Contents

Abstract

In the last decade, with the advent of the era of big data, and the growth of computing power, deep learning has been developing rapidly and have a wide application in many different fields such as image classification, object detection, and semantic interpretation [1]. Many of these applications run on mobile or embedded systems. To meet the demands, many kinds of deep learning accelerate hardware have been launched for consumer markets. Due to the portability feature of such mobile systems, those different kinds of mobile deep learning hardware needs to make a trade-off between the computation performance, and the energy consumption and size of the hardware. In this report, I present comprehensive comparisons among three different typical edge ML devices and Raspberry Pi (the baseline device). Inference speeds, stability in stress tests, power consumption and many other specifications are taken into considerations. Besides, several model compression tools and run-time optimization tools targeted at these devices are compared as well regarding their optimization effectiveness. In general, NVidia Jetson Nano and Coral USB Accelerator are overall best devices and Intel Neural Compute Stick is slightly inferior in some respects. Compared with the baseline device (Raspberry Pi), however, all these three ML devices can obtain considerable acceleration and performance improvement. To perform the benchmarks, I build and use a Python API for edge ML device benchmark which provides a uniform interface for all 4 types of devices and makes the benchmark much easier.

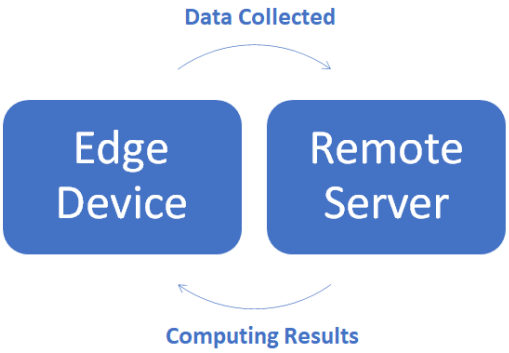
CHAPTER 1

Introduction

1.1 Backgrounds

Deep learning algorithms have a wider and wider applications in recent years, especially for end-user usages. For example, an autonomous vehicle should be capable of object detection function and a smartphone may be able to understand what the user says.

Traditionally, end user applications which adopted deep learning algorithms would require the support of high-performance servers just because the computation power of mobile processors at that time was not sufficient for deep learning algorithms. The data collected from end user devices was required to upload to the servers and then the servers would respond with the results. This traditional way, however, had many drawbacks such as high network occupations and high latency, which made it not a satisfactory method in many situations.



(A) Old implementation

(B) Edge AI

FIGURE 1.1. Comparisons of how we use AI before and now

As the demands of conducting deep learning algorithms on edge devices increase, many edge machine learning accelerators are launched into the consumer markets recently. Compared to high-performance

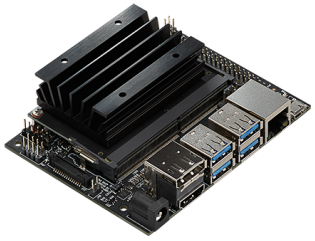
PC, these mobile or embedded systems should have the features of high portability and low power consumption. The needs of adopting deep learning on mobile and embedded systems emerge in recent years along with the popularization and wide applications of these devices.

1.1.1 Edge Machine Learning Devices

I select three popular edge machine learning devices: Google's Coral USB Accelerator (Coral) (shown in image 1.2c), NVIDIA Jetson Nano (Jetson Nano) (shown in image 1.2b) and Intel Neural Compute Stick 2 (NCS2) (shown in image 1.2d) for the experiments. Besides, Raspberry Pi 3B model (shown in image 1.2a) is selected as a baseline device. Detailed specifications are shown in table 1.1. Besides, as Raspberry Pi and Jetson Nano are single-board computers, other specifications about these two devices are shown in table 1.2.

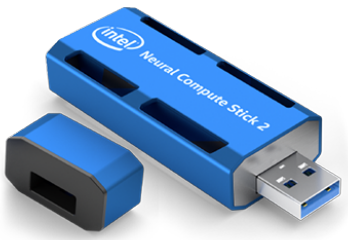
Generally, the three edge ML devices I selected are all aiming at accelerating the computations of deep learning models. Besides deep learning models, NVIDIA Jetson Nano also supports to accelerate pure math calculations (like matrix multiplications). However, for Coral USB Accelerators and Intel Neural Compute Stick 2, they only supports deep learning inferences. Not like CUDA provided by NVIDIA, APIs provided by Google and Intel for Coral USB Accelerator and Intel Neural Compute Stick 2 both require compiled deep learning models as input. Pure mathematics calculation is not supported.

(A) Raspberry Pi 3B model



(B) NVIDIA Jetson Nano

(C) Coral USB Accelerator



(D) Intel Neural Compute Stick 2

FIGURE 1.2. Devices in the experiments

1.1.2 Optimization Tools and Inference Frameworks

In order to achieve the accelerating performance on the three devices, specific frameworks should be used to do deep learning computation. For Raspberry Pi, as it does not have any ML accelerator, TensorFlow Lite may be the best framework (Figure 1.3a). For NVIDIA Jetson Nano, NVIDIA officially provides an accelerating tool called TensorRT (Figure 1.3b). Models from PyTorch (Figure

Device	Raspberry Pi	Jetson Nano	Coral	Intel NCS2
Needs host machine?	No	No	Yes	Yes
ML accelerator	None	128-core Maxwell GPU	Edge TPU ASIC (application-specific integrated circuit)	Intel Movidius Myriad X Vision Processing Unit
Connections to host	Built-in	Built-in	USB 3.1 (5Gb/s max.)	USB 3.1
Inferencing frameworks	TensorFlow Lite	TensorRT (with PyTorch and TensorFlow)	TensorFlow Lite	OpenVINO
Dimensions (mm)	85 x 56 x 17	69 x 45	30 x 65 x 8	72.5 x 27 x 14
Power consumption	5V 2.5A max. (700-1000mA in usual cases)	5V 4A max.	5V 0.9A max.	Not officially claimed, 1.5 watt as claimed in third party reports
Price	USD 35	USD 99	USD 74.99	USD 99

TABLE 1.1. Technical Specifications

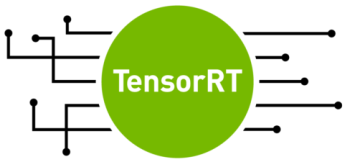
Device	Raspberry Pi	Jetson Nano
CPU	Broadcom BCM2837B0 Cortex-A53 64-bit SoC @ 1.4 GHz	Quad-core ARM A57 @ 1.43 GHz
Memory	1GB LPDDR2 SDRAM	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD	microSD

TABLE 1.2. Other Specifications of Raspberry Pi and Jetson Nano

1.3c) and TensorFlow (Figure 1.3d) can be easily imported into TensorRT to achieve considerable acceleration. For Coral USB Accelerator, TensorFlow Lite models should be first compiled using Coral Compilers (Figure 1.3e) and then run using TensorFlow Lite framework. The compiled models

are still standard TensorFlow Lite models that can be run anywhere. For Intel Neural Compute Stick 2, OpenVINO (Figure 1.3f) is the only supported framework.

(A) TensorFlow Lite



(B) TensorRT

(C) PyTorch

(D) TensorFlow

(E) Coral Compiler

(F) OpenVINO

FIGURE 1.3. Frameworks in the experiments

1.1.2.1 TensorRT

NVIDIA provides a high-performance deep learning inference optimization tool and also runtime in TensorRT targeted at NVIDIA graphics processing units (GPU). It is built for low-latency and high-throughput deep learning inference. The main theory of reducing inferencing time is to reduce the computing precision. Both INT8 and FP16 are supported for optimizations. Besides, layers and tensors can be fused to optimize the use of GPU memory and bandwidth. Dynamic Tensor Memory is also a feature of TensorRT where memories can be reused and memory footprint is minimized. Both TensorFlow and PyTorch models can be easily converted to TensorRT models through provided API.

1.1.2.2 TensorFlow Lite & Edge TPU Compiler

TensorFlow Lite [2] is a deep learning inferencing framework for mobile devices. It includes the tool to convert a TensorFlow model into TensorFlow Lite model. During conversion, FP32 precision is converted to INT8 for inference efficiency on mobile devices. Google also provides an easy-to-use Edge TPU Compiler for optimizing any TensorFlow Lite model into Edge TPU compatible version. The main idea is like TF Lite which is to reduce computation precision for lower memory consumption, but the compiler is more specified for Edge TPU. An Edge TPU has around 8MB SRAM and the Edge TPU Compiler will compress and try to fit the whole model into the SRAM.

1.1.2.3 OpenVINO

OpenVINO (Open Visual Inferencing and Neural Network Optimization) is a toolkit offered by Intel for Computer Vision development. It supports multiple Intel computer vision accelerator including Movidius Neural Compute Stick. OpenVINO is capable of not only optimizing deep learning algorithms, but also traditional computer vision algorithms with OpenCV library. OpenVINO supports conversion of Caffe, TensorFlow [3] [4], MXNet, Kaldi and ONNX models.

Besides Intel Neural Compute Stick 2 which is equiped with a Intel Movidius Myriad X Vision Processing Unit (VPU), OpenVINO is also capable of accelerating AI and machine learning tasks on Intel CPU and Intel GPU.

1.1.3 Image Classification Models

Image classification is the task of identifying and classifying the objects in an image by its visual content. Over the past years, the application of deep learning on image classification has been actively explored. Starting from the fully connected neural network (FNN) to convolutional neural network (CNN), the structures of image classification models are constantly changing. Meanwhile, we human are getting better and better outcomes on this task. Most of the models today depend on CNN to extract features from the image and use multiple FNN layers to do regression.

- EfficientNet
- AlexNet
- ResNet
- MobileNet v1
- MobileNet v2
- Inception v1 – v4
- SqueezeNet

1.1.4 Testing Dataset

ImageNetV2 [5] is a new test dataset for ImageNet benchmark. The dataset contains totally 10,000 new images in 1,000 categories corresponding to the 1,000 classed in ImageNet [6]. There are three versions of ImageNetV2 including Threshold0.7, MatchedFrequency and TopImages. I choose the MatchedFrequency set to obtain the most accurate experiment results where images are sampled such that the MTurk selection frequency distribution is consist with the original ImageNet dataset. During experiments, all tests (except a few tests that may take too long time) are conducted against the whole ImageNetV2 dataset in order to make the results comparable.

1.2 My Thoughts

As a newly emerged field, there are still few benchmarks on those mobile machine learning devices. Besides, many existing benchmark reports are out of date due to the constantly update of the optimization tools and inference engines for these devices. To make a more comprehensive report, there are several points that I am currently curious about.

- What are the impacts of the computation power of the host machines (Jetson Nano and PC) on the inference speeds of the devices?
- How fast are these devices as for the tasks of image classification?
- What are the effects of different model optimizers (compression tools)? (How much improvement on inference speed can we gain and how much accuracy should we lose? What does the tools really do?)
- How stable are the devices under stress testing (in the situations where devices are busy in long hours)?

To address these curiosities, multiple experiments are conducted. As different devices utilize different structures in machine learning accelerating units, it is impossible to compare the performance of these devices only by comparing the claimed computing clock rates. The performance largely depends on how to use the devices. In this report, three popular edge machine learning devices are benchmarked by real-world tasks. Different machine learning frameworks and different models are compared.

Apart from the comparisons, a Python API for running benchmark tasks on these four devices is also built and introduced in this report. The API provides a uniform interface for users to easily run benchmark tasks on four different edge ML devices with the same line of code. Given any deep learning models (now only image classification models), the tool can test the input models against all model compressors and devices.

CHAPTER 2

Related Works

Several benchmark reports of these devices are published officially and also by third-party. Here, I will review some of these reports related to my experiments.

2.1 Official Benchmarks

Both NVIDIA and Google release official benchmark results of their NVIDIA Jetson Nano and Coral Edge TPU. The benchmarks are both aiming at comparing the running speed but not inference accuracy. Intel does not provide any benchmark result of its Intel Neural Compute Stick 2.

2.1.1 NVIDIA

NVIDIA officially provides a set of benchmark results[1] involving NVIDIA Jetson Nano, Raspberry Pi 3, Intel Neural Compute Stick 2 and Coral Edge TPU Dev Board[2]. The experiment is conducted in synchronous mode with batch size 1. Totally 13 models are tested, 4 of which are image classification models. Here I extracted the experimental results of 3 models that are also selected in my experiments in Table 2.1.

Model	Framework	Jetson Nano	Raspberry Pi 3	Intel Neural Compute Stick 2	Coral Edge TPU Dev Board
ResNet-50	TensorFlow	36 FPS	1.4 FPS	16 FPS	DNR
MobileNet V2	TensorFlow	64 FPS	2.5 FPS	30 FPS	130 FPS
Inception V4	PyTorch	11 FPS	DNR	DNR	9 FPS

TABLE 2.1. FPS of each device in NVIDIA's official report

DNR indicates that this model is not supported on the relevant device due to multiple reasons[3].

[1] Details of this benchmark can be found at https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks

[2] Coral Edge TPU Dev Board is equiped with the same Tensor Processing Unit (TPU) as Coral USB Accelerator

[3] Reasons include limited memory capacity, unsupported network layers, or hardware/software limitations as described by NVIDIA

2.1.2 Coral Edge TPU

The benchmark results[4] provided officially by Google does not involve other machine learning accelerators such as NVIDIA Jetson Nano or Intel Nerual Compute Stick 2. It only compares Coral Edge TPU to Desktop CPU[5] and Embedded CPU[6]. All the tests are performed using C++ API which may be faster than Python API due to the larger overhead of Python than C++. The timing is based on pure device time (same as my experiments described in Chapter 3). Here, I extract some of the results which are related to my own experiments in Table 2.2.

Model	Coral USB Accelerator with Desktop CPU	Embedded CPU	Desktop CPU
Inception V1	3.4	392	90
Inception V4	85	3157	700
MobileNet V1	2.4	164	53
MobileNet V2	2.6	122	51
EfficientNet S	5.1	705	5431
EfficientNet M	8.7	1081	8469
EfficientNet L	25.3	2717	22258

TABLE 2.2. Inference time (ms) of each device in Google's official report

2.2 Third-party Benchmarks

Beside the official reports, there are also many third-party benchmarks available on the Internet. I select some comprehensive and impressive reports here for later comparisons.

2.2.1 Juan's Benchmark

Juan Pablo published a set of benchmark results of multiple machine learning accelerators (including Raspberry Pi 3B, Coral Dev. Board, Coral USB Accelerator, NVIDIA GTX-2080ti, Intel NCS2, NVIDIA Jetson Nano and Raspberry Pi 3B) on "tryo labs" entitled "Machine learning edge devices: benchmark report"[7] in October 2019. Several image classification models are tested on some of these devices. The top-1 and top-5 accuracy against ImageNet V2 test-set and inference speed are the criteria of the experiments. Parts of the results that involve the devices tested in my experiment are shown in Table 2.3 and Table 2.4.

[4]Details of this benchmark can be found at https://coral.ai/docs/edgetpu/benchmarks/

[5]Desktop CPU: Single 64-bit Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz

[6]Embedded CPU: Quad-core Cortex-A53 @ 1.5GHz. This is the CPU in Raspberry 3B+ model

[7]Report can be found at https://tryolabs.com/blog/machine-learning-on-edge-devices-benchmark-report/

Device	Framework	**Model**	Inference Time (ms)	Accuracy (Top-1) %	Accuracy (Top-5) %
Jetson Nano	TF-TensorRT-PyTorch	**Resnet-50**	2.67	64.30	NaN
Coral (USB Accelerator)	Edge TPU	**Efficientnet-S**	13.18	33.79	55.45
Jetson Nano	PyTorch	**Resnet-50**	35.67	64.30	NaN
Intel NCS2	OpenVINO	**Resnet-50**	58.40	48.00	72.80
Coral (USB Accelerator)	Edge TPU	**Efficientnet-M**	59.62	45.53	67.67
Jetson Nano	PyTorch	**Efficientnet-B0**	59.97	61.64	NaN
Jetson Nano	PyTorch	**Efficientnet-B3**	96.94	67.97	NaN
Jetson Nano	TensorFlow	**Efficientnet-B0**	98.78	79.61	NaN
Jetson Nano	TF-TensorRT	**Efficientnet-B0**	154.13	79.61	NaN
Jetson Nano	TF-TensorRT	**Resnet-50**	176.50	54.10	NaN
Jetson Nano	TensorFlow	**Resnet-50**	223.73	54.10	NaN
Coral (USB Accelerator)	Edge TPU	**Efficientnet-L**	225.28	42.92	65.06
Jetson Nano	TensorFlow	**Efficientnet-B3**	246.26	85.44	NaN
Jetson Nano	PyTorch	**Efficientnet-B5**	261.27	72.03	NaN
Jetson Nano	TF-TensorRT	**Efficientnet-B3**	327.28	85.44	NaN
Raspberry Pi 3B	TensorFlow	**Efficientnet-B0**	539.15	80.00	NaN
Raspberry Pi 3B	TensorFlow	**Resnet-50**	1660.55	67.00	NaN

TABLE 2.3. Juan's Benchmark Results

Device	Framework	**Model**	Inference Time (ms)	Accuracy (Top-1) %	Accuracy (Top-5) %
Raspberry Pi 3B	TensorFlow	**Efficientnet-B3**	1891.05	81.00	NaN
Raspberry Pi 3B	PyTorch	**Resnet-50**	3915.22	79.00	NaN
Raspberry Pi 3B	PyTorch	**Efficientnet-B0**	6908.80	78.00	NaN
Raspberry Pi 3B	PyTorch	**Efficientnet-B3**	13685.62	79.00	NaN

TABLE 2.4. Juan's Benchmark Results

Juan provided many results in the report. However, most of the experiments are on NVIDIA Jetson Nano and Raspberry Pi. Only 3 experiments are on Coral USB Accelerator (which are all tests for EfficientNet) and only 1 is on Intel Neural Compute Stick 2. Besides, for most of the experiments, the top-5 accuracy is missing.

2.2.2 Alasdair's Benchmark

Alasdair's report[8] is the most early third-party benchmark I can find on the Internet which was published on 7 May 2019. One thing to clarify first is that this benchmark is sponsored by Coral from Google. The experiment involves Coral USB Accelerator (early version)[9], NVIDIA Jetson Nano, Intel Neural Compute Stick (generation 1 and 2)[10], Raspberry Pi. The models that are tested are limited to only MobileNet series object detection model (including MobileNet V1 SSD and MobileNet V2 SSD). The criteria are inference latency and power consumption. Here I extract some related results in Table 2.5.

[8]Details of this benchmark can be found at https://medium.com/@aallan/benchmarking-edge-computing-ce3f13942245

[9]Alasdair used an early version Coral USB Accelerator which is different from mine in appearance but unsure whether the hardware is different. Coral USB Accelerator is attached to Raspberry Pi for testing

[10]Intel Neural Compute Stick is attached to Raspberry Pi for testing

Board	MobileNet v1 SSD (ms)	MobileNet v2 SSD (ms)
Coral USB Accelerator	49.3	58.1
NVIDIA Jetson Nano (TF)	276.0	309.3
NVIDIA Jetson Nano (TF-TRT)	61.6	72.3
Intel NCS2	87.2	118.6
Raspberry PI	480.3	654.0

TABLE 2.5. Alasdair's Benchmark Results

Although models involved in this experiment are different from mine, the related latency is still important for comparison.

CHAPTER 3

Methods

3.1 Inference Speed and Model Optimization Tests

Multiple DNN models for image classification are selected for different devices in the experiments. In all experiments, the average inferencing time, top-1 and top-5 accuracy are criteria.

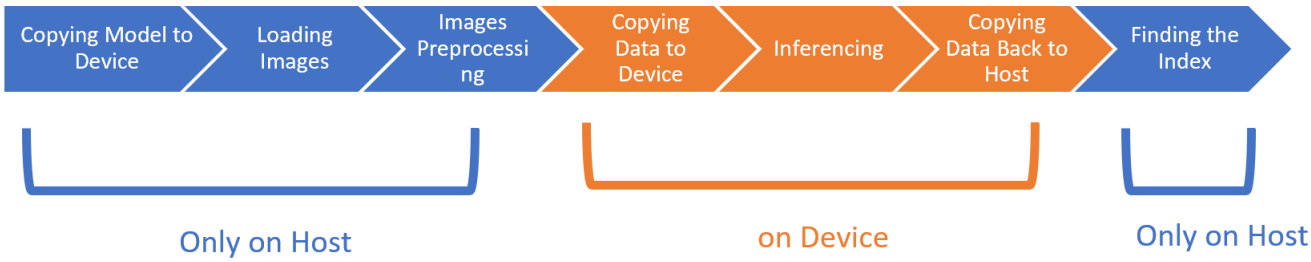


FIGURE 3.1. Common procedure of infer on Device

Average Inference Time[1]: I calculate the latency between sending the input tensor to the device and copying the results back to host. The experiments are conducted in two ways. In the first way, all images in ImageNetV2[2] are tested once and the average inference time is reported. In the second way, models are tested against a large number of random inputs and the average inference time is reported. In Figure 3.1, the timing criterion is shown visually.

Top-1 Accuracy[3]: Only if the top-1 answer (with highest probability) is consist with the ground truth, we regard it correct.

Top-5 Accuracy: If the top-5 answers (with highest 5 probabilities) contain the ground truth, we regard it correct.

Note: for timing, Python provides multiple methods to retrieve the current time. The most common two methods are `time.time()` and `time.clock()`. However, these two functions return different elapsed times in my experiments. I look into this issue and find that the time returned by the two functions are different and even different across platforms. Generally, on Linux (or Unix) platform, `time.clock()` returns the processor (CPU) time and is not accurate due to the Jiffy effects. However, on Windows, `time.clock()` returns wall time which has accuracy less than one millisecond.

[1]The timing is based on a Python program which may involve larger overhead than pure C program but compared with actual computation elapse, the overhead can be ignored.

[2]Totally 10,000 images are included in ImageNetv2 dataset.

[3]As Raspberry Pi takes a very long time for nearly all models, only inference times are tested on Raspberry Pi but not accuracy. 10 iterations are tested on one model and the average times are calculated.

For Raspberry Pi, NVIDIA Jetson Nano and Coral USB Accelerator, the benchmarks are run with the Python API I wrote. For Intel Neural Compute Stick 2, the experiments which are finished before March were also run with the Python API I wrote. However, in February, Intel officially launched a GUI benchmark and profiling tool called "Deep Learning Workbench" (interface shown in Figure 3.2).

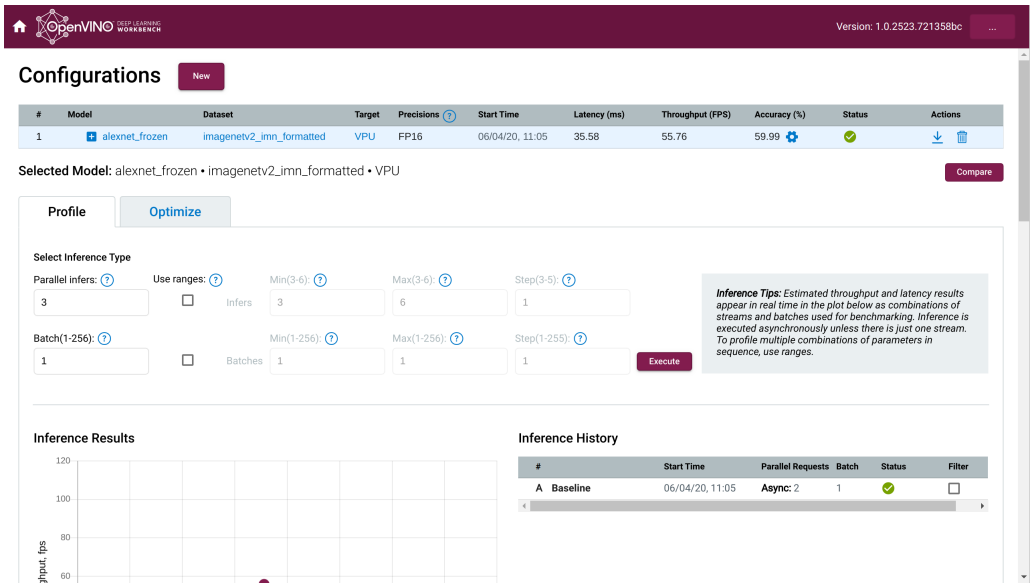


FIGURE 3.2. Deep Learning Workbench: A Intel Official Benchmark Tool

Detailed configurations of experiments are shown in table 3.1.

Device	Framework	Data Type	Layer Fusion
Raspberry Pi	TensorFlow Lite	8-bit INT	No
Jetson Nano	TensorRT	FP16	Yes
Coral USB Accelerator	TensorFlow Lite	8-bit INT	Yes
Intel NCS2	OpenVINO	FP16	Yes

TABLE 3.1. Detailed configurations

3.2 Stress Tests

To address the concern whether the devices can run in a stable frequency in long hours, I conducted stress tests for each device. In the test, each device is continuously doing inferences in a three-hour-long time. The time of each inference in that period is recorded. In Chapter 4, the results of the experiments are shown. In Chapter 5, an analysis of the stress tests are presented.

All the tests are based on EfficientNet-b0 model (or EfficientNet-S model for Raspberry Pi and Coral USB Accelerator) and the inputs are random-generated vectors. Because the inference times on different devices are different, the total times of inferences are also different.

```
# Sample codes for generating random input for TensorFlow Lite with
   datatype FP32

input_t = np.random.rand(1, width, height, 3).astype(np.float32)
```

```
# Sample codes for generating random input for TensorFlow Lite quantized
   model

input_t = np.random.randint(255, size=(1, width, height, 3),
   dtype='uint8')
```

```
# Sample codes for generating random input for OpenVINO

input_t = np.random.rand(n, c, h, w).astype(np.float16) # n for batch
   size, c for color channels, h for height and w for width
```

```
# Sample codes for generating random input for PyTorch with NVIDIA GPU
   supported

x = torch.rand((1,3,224,224)).cuda() # for a fixed input tensor shape
```

One thing to emphasize is that the three hours are the total time (wall time) but not the total inference time (which is smaller than 3 hours). This means that the pre-processing and post-processing times are also included in the 3 hours timing.

3.3 A Python API for Running Benchmarks

In order to do benchmarks on these devices more conveniently, I write a Python API for running benchmarks. The API supports TensorFlow Lite on generic CPU, TensorFlow Lite on Coral TPU, TensorRT on NVIDIA GPU and OpenVINO on Intel Neural Compute Stick 2.

The API can automatically import and convert models to target frameworks. Besides, it can generate fake input values for benchmarks. Benchmark progress can also be visualized with a progress bar.

Here is a brief manual for this program:

```
$ python image_class.py --help
usage: image_class.py [-h] --model MODEL [--imagedir IMAGEDIR]
   [--edgetpu] [--trt] [--openvino]
                [--partial] [--ratio RATIO] [--fakein] [--ftimes FTIMES]
                [--record] [-v]

optional arguments:
  -h, --help       show this help message and exit
  --model MODEL    location of the .tflite file or tensorflow model folder
  --imagedir IMAGEDIR the folder containing all testing images, folder
```

```
                structure is specified in README
  --edgetpu     whether using Coral Edge Accelerator
  --trt         whether using TensorRT and running on an NVIDIA GPU
  --openvino    whether using Intel Neural Compute Stick 2
  --partial     only do partial testing
  --ratio RATIO define how many images to test for speed testing
  --fakein      generate a fake input to test the correctness of
                configuration
  --ftimes FTIMES how many fake images to test inferencing latency
  --record      record the inferencing latency of each inference and
                log into a csv file
  -v            show progress bar
```

Note: all experiments in this project (except some experiments about OpenVINO) are conducted using this program.

CHAPTER 4

Results

4.1 Accuracy and Speed Test

4.1.1 Raspberry Pi

As described in Chapter 3, the accuracy is not a criteria for the experiments. The main reasons are that the inferences take too long a time and as a baseline device, I am mainly focusing on the acceleration of other three devices compared with it but not accuracy.

Tests results for Raspberry Pi are shown in table 4.1. Total 12 sets of experiments are conducted.

Model	Framework	Average Inference Time (ms)
EfficientNet S [7]	TensorFlow Lite	2074.80
EfficientNet M	TensorFlow Lite	3201.42
EfficientNet L	TensorFlow Lite	7793.95
MobileNet v1 [8]	TensorFlow Lite	584.96
MobileNet v2	TensorFlow Lite	523.70
Inception v1	TensorFlow Lite	1193.93
Inception v2	TensorFlow Lite	1720.32
Inception v3	TensorFlow Lite	4339.28
Inception v4	TensorFlow Lite	9248.16
SqueezeNet	TensorFlow Lite	596.62

TABLE 4.1. Results of Raspberry Pi

4.1.2 Jetson Nano

Tests results for NVIDIA Jetson Nano are shown in table 4.2. Total 12 sets of experiments are conducted.

Model	Framework	Average Inference Time (ms)	Top-1 Accuracy (%)	Top-5 Accuracy (%)
EfficientNet B0	TensorFlow	52.21	63.98	85.21
EfficientNet B3	TensorFlow	154.88	68.53	88.57
EfficientNet B5	TensorFlow	592.63	72.75	91.06
EfficientNet B0	PyTorch [9]	48.12	63.98	85.21
EfficientNet B3	PyTorch	98.97	68.53	88.57
EfficientNet B5	PyTorch	350.46	72.75	91.06
EfficientNet B0	TRT	Not Supported	N/A	N/A
EfficientNet B3	TRT	Not Supported	N/A	N/A
EfficientNet B5	TRT	Not Supported	N/A	N/A
AlexNet [10]	PyTorch	5.82	43.85	67.26
AlexNet	PyTorch-TRT	1.50	43.85	67.26
ResNet 50 [11]	PyTorch	35.65	63.22	84.76
ResNet 50	PyTorch-TRT	2.92	63.22	84.76
MobileNet v2 [12]	PyTorch	30.11	59.03	81.33
MobileNet v2	PyTorch-TRT	2.63	59.03	81.33

TABLE 4.2. Results of Jetson Nano

4.1.3 Coral USB Accelerator

The experiments of Coral USB Accelerator are divided into two parts. One is conducted on a more powerful PC. The other one is conducted on Jetson Nano. The specifications of the two hosts are shown

in table 4.3. Here we want to test how much computation is actually done on the USB Accelerator. Generally speaking, the PC is about 2.6 times faster than the Jetson Nano in performing single-core computation. All the models tested on Coral USB Accelerator are optimized by the Edge TPU Compiler for faster inferencing speed as the un-optimized models take too long time to infer on the USB Accelerator.

Host	PC	Jetson Nano
CPU	Quad-core 8-thread Intel Core i5-8250U @ 1.60GHz	Quad-core ARM A57 @ 1.43 GHz
Max CPU Clock Rate	3.40 GHz	1.43 GHz
ISA	x86_64	ARMv8-A 64-bit
CPU Cache	L1: 256 KB, L2: 1 MB, L3: 6MB	L1: 80 KiB, L2: 2MB
CPU Bus Speed	4 GT/s	Not reported
CPU Single-core benchmark[1]	3243	1237.5
RAM	16 GB DDR4	4 GB LPDDR4
RAM Speed	2400 MHz	25.6 GB/s

TABLE 4.3. Specifications Comparison of PC and Jetson Nano

Tests results for Coral USB Accelerator are shown in table 4.4 (for experiments on PC) and table 4.5 (for experiments on NVidia Jetson Nano). Total 12 sets of experiments are conducted.

Model	Framework	Average Inference Time (ms)	Top-1 Accuracy (%)	Top-5 Accuracy (%)
EfficientNet S	TensorFlow Lite	5.47	39.07	61.82
EfficientNet M	TensorFlow Lite	8.79	49.76	72.12
EfficientNet L	TensorFlow Lite	26.01	46.75	69.20

TABLE 4.4. Results of Coral USB Accelerator on PC

[1]The benchmark is based on Geekbench 3 32 Bit Single-Core Score, tested by NotebookCheck.Net.

Model	Framework	Average Inference Time (ms)	Top-1 Accuracy (%)	Top-5 Accuracy (%)
EfficientNet S Quant	TensorFlow Lite	6.41	39.07	61.82
EfficientNet M Quant	TensorFlow Lite	10.72	49.76	72.12
EfficientNet L Quant	TensorFlow Lite	28.88	46.75	69.20
Inception v1 [13]	TensorFlow Lite	4.76	54.95	78.17
Inception v2 [14]	TensorFlow Lite	16.70	58.47	80.36
Inception v3 [15]	TensorFlow Lite	50.83	64.37	84.30
Inception v4 [16]	TensorFlow Lite	98.64	67.90	86.46
MobileNet v1	TensorFlow Lite	3.37	54.61	76.32
MobileNet v2	TensorFlow Lite	4.07	54.90	77.75

TABLE 4.5. Results of Coral USB Accelerator on Jetson Nano

4.1.4 Intel Neural Compute Stick 2

Tests results for Intel Neural Compute Stick 2 are shown in table 4.6. Total 11 sets of experiments are conducted.

Unfortunately, EfficientNet is not supported on Intel Nerual Compute Stick 2. The MeanReduce layer is not supported on this device. However, the converted IR model (OpenVINO format) can run on Intel X86 CPU. So, the accuracy of the converted models is still tested for later comparisons of different frameworks and tools.

4.1.5 OpenVINO on Intel NCS 2, Intel X86 CPU and Intel GPU

Based on the official documents of OpenVINO and Intel Neural Compute Stick 2, we can clearly find that one of the main targets of Intel NCS2 is for accelerating PC applications for running AI and machine learning algorithms. Here, I test the inference latency and throughput of OpenVINO running

Model	Framework	Average Inference Time (ms)	Top-1 Accuracy (%)	Top-5 Accuracy (%)
EfficientNet B0	OpenVINO	N/A	41.88	66.91
EfficientNet B3	OpenVINO	N/A	50.80	75.09
EfficientNet B5	OpenVINO	N/A	63.65	85.68
AlexNet	OpenVINO	35.6	35.67	59.99
SqueezeNet	OpenVINO	11.55	44.91	68.07
Resnet 50	OpenVINO	51.75	48.43	72.08
Inception v1	OpenVINO	22.76	49.81	73.32
Inception v2	OpenVINO	24.60	50.08	74.65
Inception v3	OpenVINO	83.74	52.05	76.08
Inception v4	OpenVINO	185.60	60.63	83.08
MobileNet v2	OpenVINO	23.69	35.00	59.83

TABLE 4.6. Results of Intel NCS2

on Intel NCS 2, Intel CPU and Intel GPU to see how Intel Neural Compute Stick 2 accelerates AI applications compared with normal PC CPU and GPU.

The results are shown in table 4.7. For each device (VPU, CPU and GPU), a minimum latency (min. latency) and a maximum throughput (max. throughput) are recorded. For VPU (Intel NCS2), we can get a min latency when parallel stream is 2 and get a max throughput when parallel stream is 3. For CPU (Quad-core 8-thread In-tel Core i5-8250U @1.60GHz), we can get a min latency when parallel stream is 1 and get a max throughput when parallel stream is 8. For GPU (Intel UHD Graphics 620), we can get a min latency when parallel stream is 1 and get a max throughput when parallel stream is 2.

4.2 Stress Test

The stress tests are conducted according to the methods introduced in Chapter 3. Scatter plots of inference times of four devices are shown in this chapter. For detailed analysis, please refer the Chapter 5. In each scatter plot, obvious outliers are removed and not included in the both the scatter plots and the histograms that are shown in Chapter 5.

Model	INCS2		CPU		GPU	
	min. latency (ms)	max. through-put (FPS)	min. latency (ms)	max. through-put (FPS)	min. latency (ms)	max. through-put (FPS)
Inception v1	22.8	87.0	10.9	92.5	12.7	94.8
Inception v2	24.6	87.8	10.9	92.5	12.7	94.8
Inception v3	83.7	20.3	48.0	20.4	33.3	28.2
Inception v4	185.60	10.9	94.4	9.8	65.6	16.0
MobileNet v2	23.7	83.7	3.3	269.7	4.6	308.3

TABLE 4.7. OpenVINO results on INCS2, CPU and GPU

4.2.1 Raspberry Pi

A summarized scatter plot is shown in Figure 4.1.

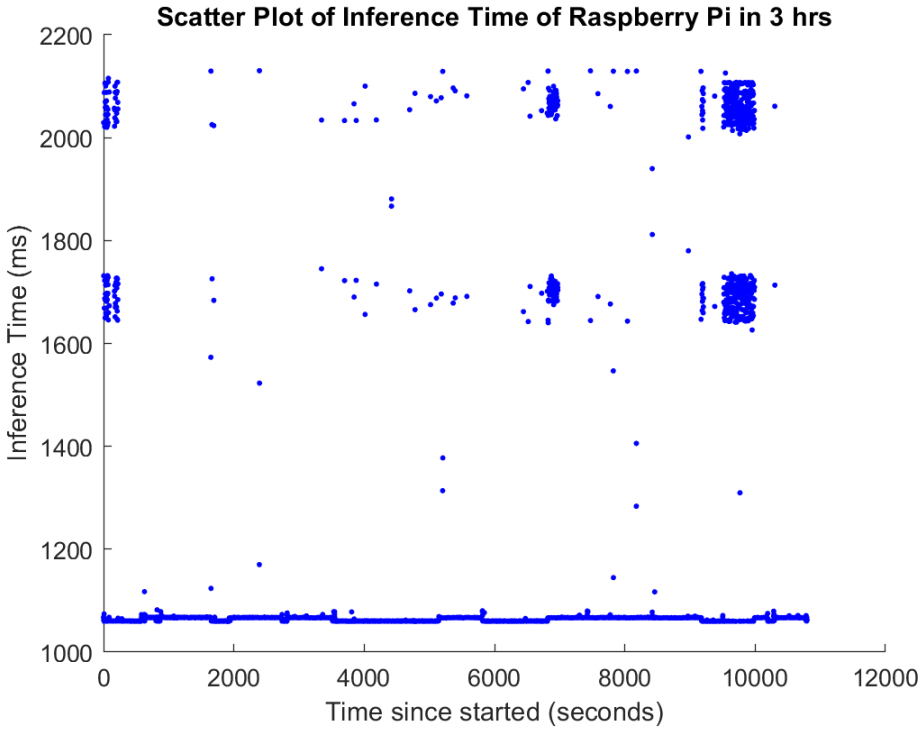


FIGURE 4.1. Scatter plot of Raspberry Pi

4.2.2 Jetson Nano

A summarized scatter plot is shown in Figure 4.2. Any point with inference time larger than 5 ms is regarded as outliers and are discarded.

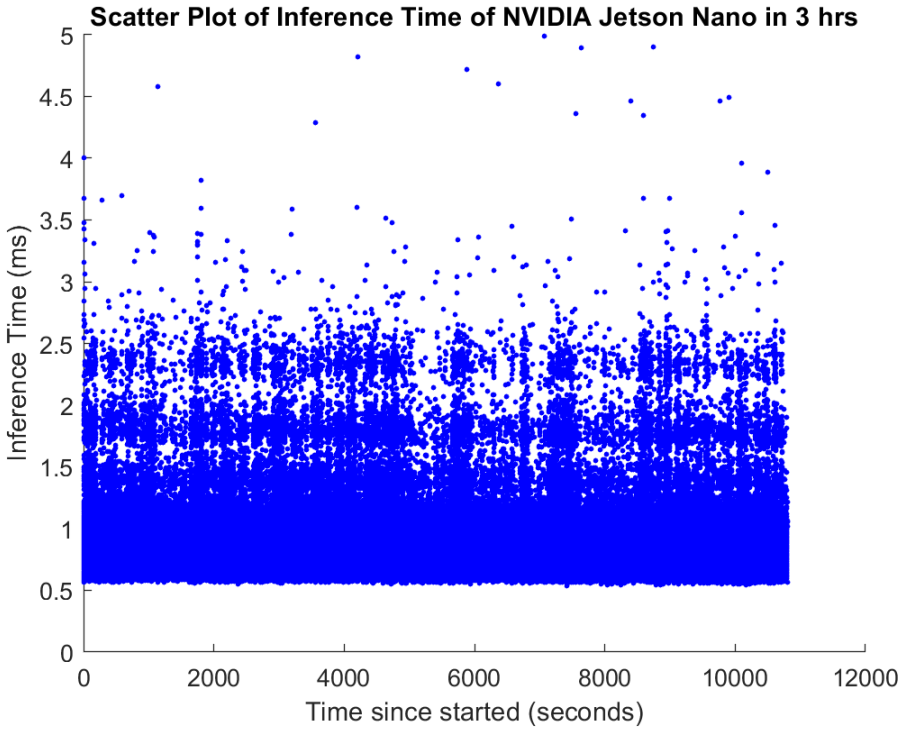


FIGURE 4.2. Scatter plot of NVIDIA Jetson Nano

4.2.3 Coral USB Accelerator

A summarized scatter plot is shown in Figure 4.3. Any point with inference time larger than 50 ms is regarded as outliers and are discarded.

4.2.4 Intel Neural Compute Stick 2

A summarized scatter plot is shown in Figure 4.4. Ouliers are not removed as they do not affect the visualization of the informative data.

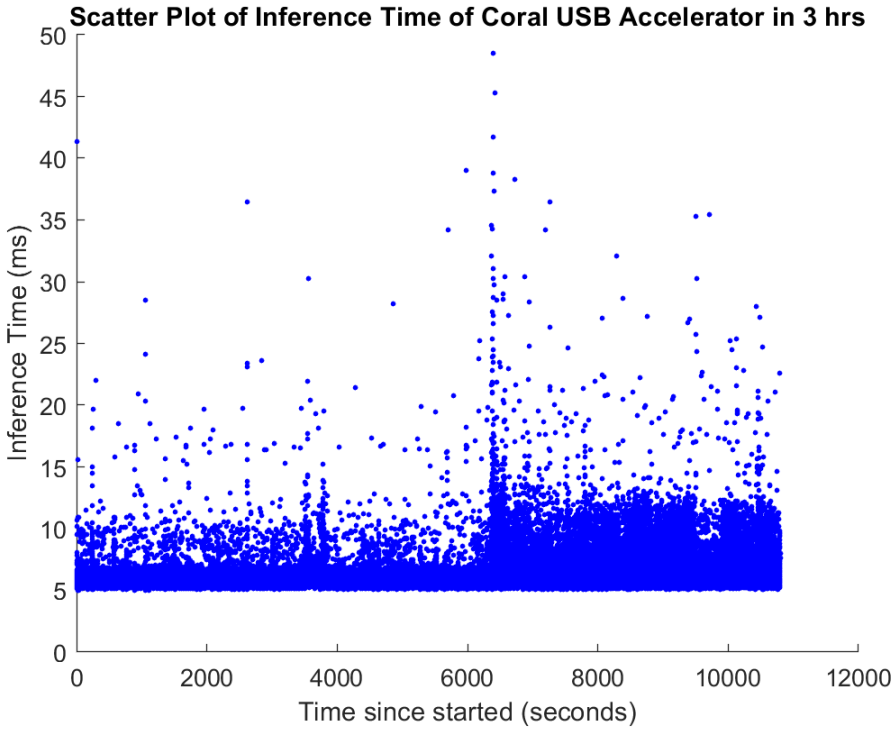


FIGURE 4.3. Scatter plot of Raspberry Pi

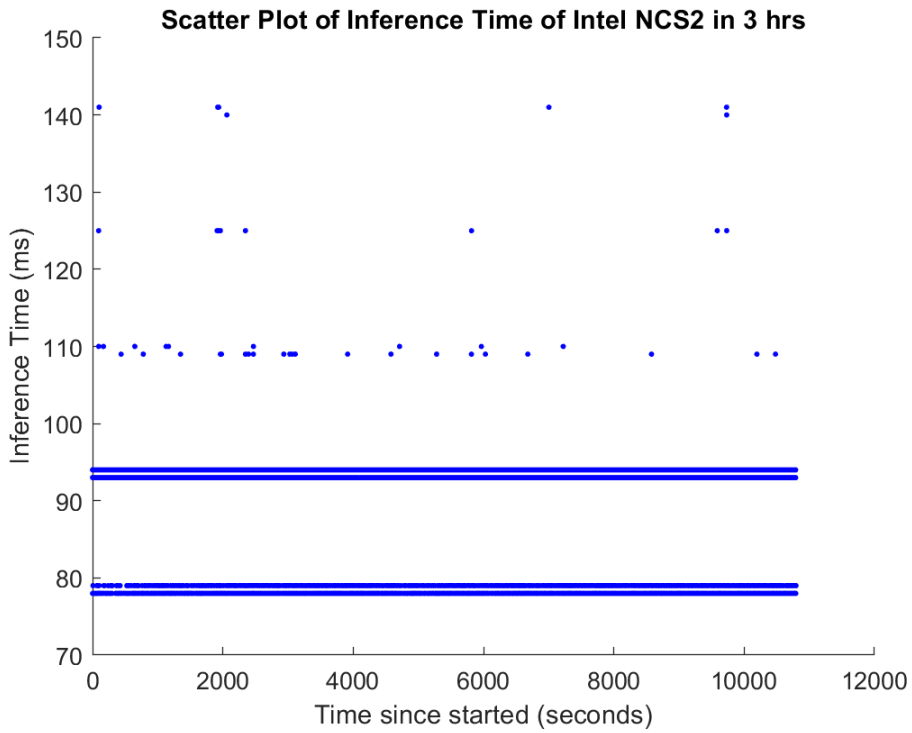


FIGURE 4.4. Scatter plot of Intel Neural Compute Stick 2

CHAPTER 5

Analysis of Experimental Results

Here, I will address all my questions raised in Chapter 1 based on the experiments results presented in Chapter 4.

5.1 Comparison with Previous Works

First, I will compare my results with the results in previous works and show that my results are sound.

In NVIDIA's result shown in Table 2.1, NVIDIA provides the Frame-Per-Second (FPS) to measure the performance of different hardware. Also, the FPS is calculated as the reciprocal of the turnover times. As the measurements are different from mine, we can hardly compare the results.

In Google's result shown in Table 2.2, Google provides the results in milliseconds. Described by Google, the timing method is consistent with my experiments. Comparing all the inference times in Google's results with mine, we can find that they are generally 20% faster than my results. There are two reasons for that result. First, in Google's experiments, the Coral USB Accelerator is attached to a Desktop PC but in my experiments, the Accelerator is attached to NVIDIA Jetson Nano which has a much weaker configurations compared with a Desktop PC. Second, Google uses C++ API for inference but I use Python API which has a significant higher overhead than C++.

Juan's benchmark is much more comprehensive than NVIDIA's and Google's reports. All the criteria in Juan's benchmark are consistent with my experiments so the two sets of experiments are more comparable. In Table 2.3 and Table 2.4, most of the inference time results are consistent with my results. However, the accuracy has a large difference. Even within Juan's own benchmark results, the accuracy of the same model on different platforms differ a lot. It seems that Juan uses different pretrained model for different tests. I believe this will break the comparability of the results.

Alasdair's Benchmark is short in content. The results are basically consistent with my results. But one problem is that for both MobileNet v1 and MobileNet v2, NVIDIA Jetson Nano is slower than Coral USB Accelerator. This may be caused by the problem of TensorFlow. In my tests, I find that TensorFlow plus TensorRT is generally slower than PyTorch plus TensorRT.

In sum, the previous works basically agree with my results.

5.2 Analysis and Comparisons of Different Frameworks and Optimization Tools

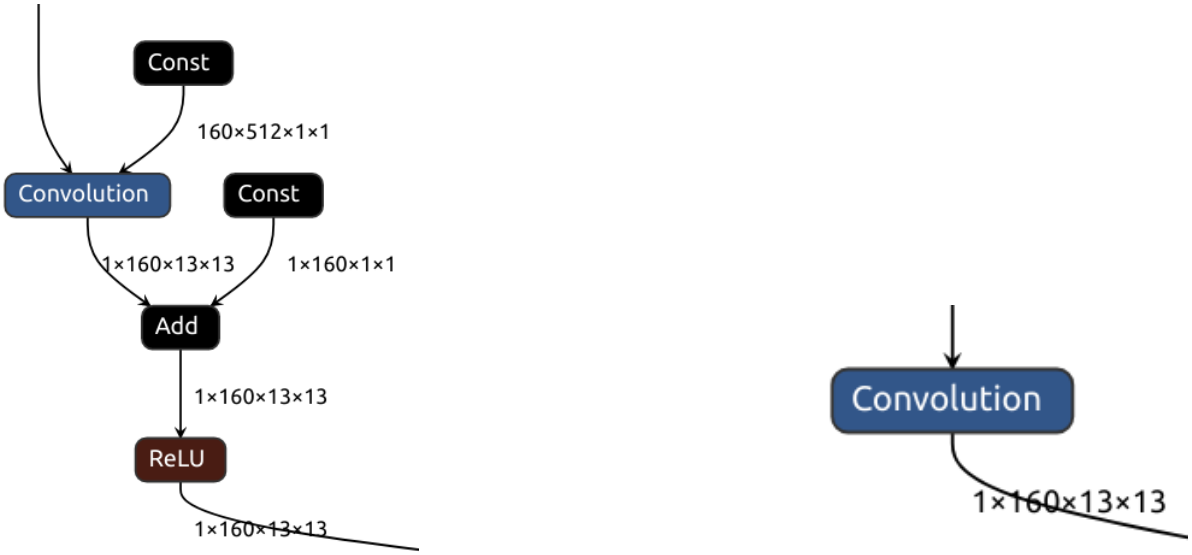
5.2.1 Model Structures

In Chapter 4, raw data of all experiments are shown. In order to further analyze the reason behind the results why different frameworks and tools have different optimization effects, I take Inception v1 as an example to show how each framework or tool work. For Inception v1, all operations are common (such as convolution, etc.) and supported by the Edge Compiler, TensorRT and OpenVINO.

5.2.1.1 OpenVINO

For Intel's OpenVINO, as it is an open source software, there is hardly any secret in its optimization techniques. Besides, the optimization theory of OpenVINO is also simple and traditional. Described by Intel [1], the model optimizer mainly works in the way of fusing adjacent layers. Here, I test the model optimizer with Inception v1 and see what happens.

Generally, the overall structures of the model remain unchanged. The only difference is that the layers around convolutions are fused into a single layer as shown in Figure 5.1 [2]. Compared with Edge-TPU Compilers and NVIDIA's TensorRT, OpenVINO seems to adopt an old-fashioned and relatively simple optimization algorithms which may be the main cause for its low acceleration performance compared with the other two devices.



(A) Subgraph of original IR (B) Subgraph of Runtime

FIGURE 5.1. OpenVINO Optimization comparison

5.2.1.2 Edge TPU Compiler

Unfortunately, Google's Edge TPU Compiler is not an open source software, especially for the core of the libraries. Although the compiled models are still valid and standard TensorFlow Lite models,

[1] Intel has an introductory page for its model optimizer:
https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_Model_Optimization_Techniques.html

[2] The graphs are generated by Netron: A Visualizer for neural network, deep learning and machine learning models.

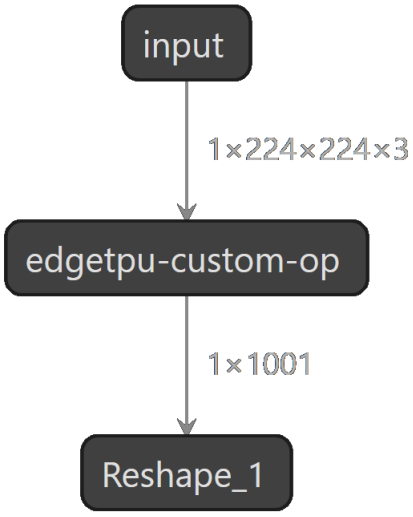


FIGURE 5.2. Resulted graph of Edge TPU Compiler

the files are serialized and can hardly be hacked and looked into. Here I shows what a compiled model's structure looks like in Figure 5.2. All supported layers and operations are wrapped into a single operation (edgetpu-custom-op).

5.2.1.3 TensorRT

The core part of TensoRT is not open-sourced either. From the documents [3] of NVIDIA, it is clear that TensorRT also depends on layer fusion. However, the more important techniques NVIDIA adopts should be Dynamic Tensor Memory and Kernel Auto-Tuning.

5.2.2 Accuracy Lost

As a matter of lower precision and fused layers, the accuracy of the models will become lower after optimization or compression of the models. Here, I show charts that compare the accuracy of different models after the optimizations of different tools. Figure 5.3 is a comparison of Top-1 Accuracy among models and frameworks. Figure 5.4 is a comparison of Top-5 Accuracy among models and frameworks.

Basically, TensorRT will not lose accuracy after optimization so I take those optimized by TensorRT as ceilings for other results. From the charts, not surprisingly, as the state-of-the-art image classification model, several versions of EfficientNet have the highest Top-1 and Top-5 accuracy among the models I tested. For EfficientNets, Intel's OpenVINO shares a general higher accuracy than Google's Edge TPU compiler. This is possible due to the memory restriction of Coral TPU of 8 MB. The Edge TPU compiler will compress and shrink the models for it to fit into the memory of device. Also, EfficientNets are much larger than other models I tested. Another noteworthy point is that Coral-optimized EfficientNet B5 has a lower accuracy than that of EfficientNet B3, which can be regarded as another evidence to illustrate that Coral TPU is restricted by its low memory.

[3] NVIDIA provides some introductory documents on the theory of TensorRT but the core parts are not open.

In general, TensorRT has the highest accuracy among all. Edge TPU Compiler and OpenVINO have their onw strengths and drawbacks based on different models, especially considering the size of models.

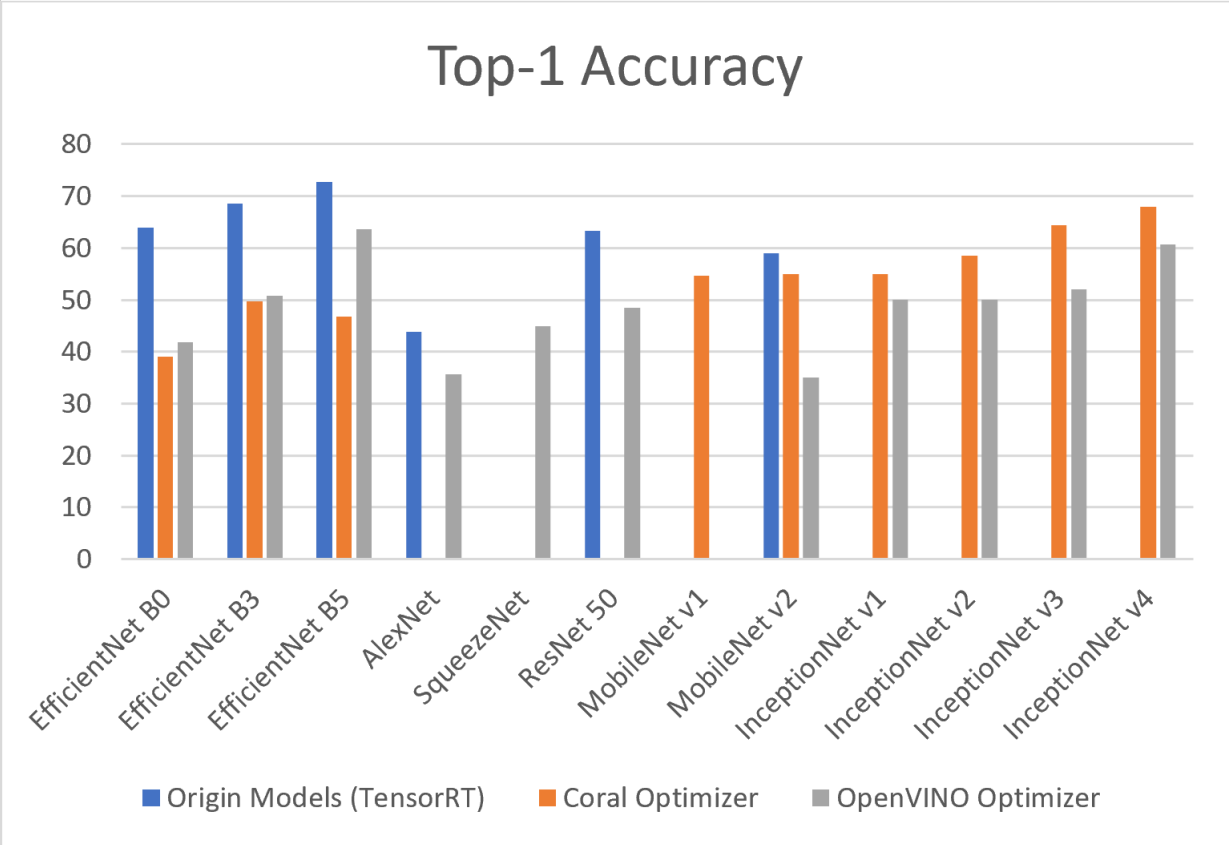


FIGURE 5.3. Top-1 Accuracy among models and frameworks

5.3 Analysis and Comparisons of Different Hardware

5.3.1 Impact of the host on inference speed

For Google Coral USB Accelerator, some part of computation is actually done on the host machine. As a result, it is unclear how much portion of computation will be conducted on device and how much on host. Through the experiments, we can have a general idea about the impact of the performance of host machine on the inference speed. Table 5.1 compares the inference time of Coral USB Accelerator on PC and Jetson Nano. Table 5.5 shows the reciprocals of the time which is regarded as speed.

It is assumed that the relationship of the inferencing speeds (reciprocal of inferencing times) on PC and Jetson Nano is linear. We can use regression to approximate the relationship.

We get the result of regression as follow:

$$Speed_{Jetson} = a \times Speed_{PC} + b$$

Where a = 0.8398 ± 0.03731 and b = 0.0008498 ± 0.004712. As we cannot compare the computation speed of the Coral USB Accelerator with either PC or Jetson Nano, we cannot compute the exact

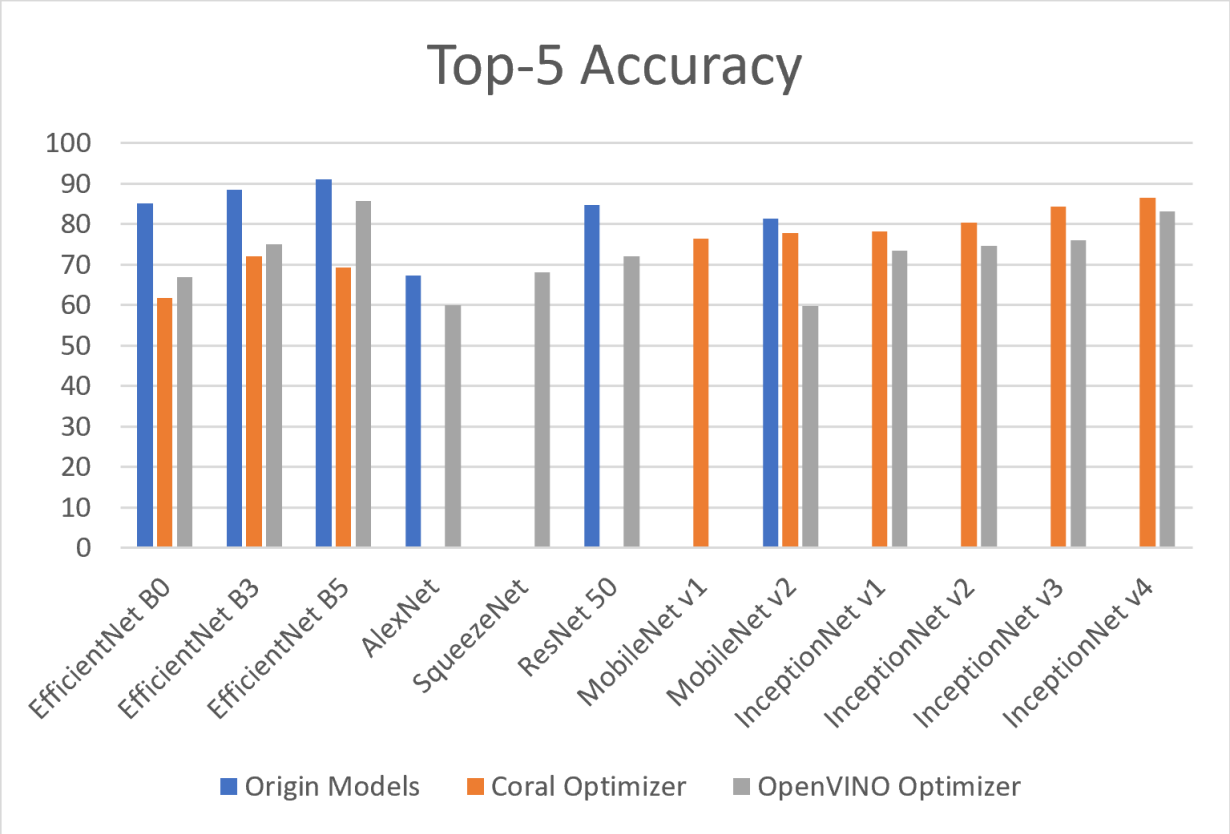


FIGURE 5.4. Top-5 Accuracy among models and frameworks

portion of computation that is done on the Accelerator. However, based on the results we gain, we can conclude that inferencing on an embedded device such as Jetson Nano is approximately 0.84 times as that on PC, which is an acceptable result.

Models	Average Inference Time (ms)	
	PC	Jetson Nano
EfficientNet S	5.47	6.41
EfficientNet M	8.79	10.72
EfficientNet L	26.01	28.88

TABLE 5.1. Inference Time Comparison

Models	Average Inference Speed (1/ms)	
	PC	Jetson Nano
EfficientNet S	0.182815	0.156006
EfficientNet M	0.113766	0.093284
EfficientNet L	0.038447	0.034626

TABLE 5.2. Inference Speed Comparison

5.3.2 Comparisons of Inference Speed

Device	Best Inference Time (ms)	Corresponding Model
Raspberry Pi	523.70	MobileNet v1
Jetson Nano	1.50	AlexNet
Coral USB Accelerator	3.37	MobileNet v1
Intel NCS2	11.55	SqueezeNet

TABLE 5.3. Best Inference Time

Jetson Nano gets the best result among the three device regarding the inference time. Using AlexNet with PyTorch and TensorRT optimization, it reaches 1.50 milliseconds per inference which is around 667 frames per second. However, the time does not include the latency of image preprocessing. Considering this factor, the inference can still be fast enough for any real-time image classification task.

Coral USB Accelerator is the runner up in the comparison. Using MobileNet v1 and Edge TPU compiler to optimize, it reaches 3.37 milliseconds per inference which is around 297 frames per second. Even considering the image preprocessing time, Coral USB Accelerator can still be an amazing device which is suitable for edge image classification tasks.

Intel Neural Compute Stick 2 seems to be poor at conducting deep learning tasks, compared with the above two devices. Even after the optimization, the least inference time of all models that are tested is still no less than 10ms. This is a quite slow operation even compared with Google Coral USB Accelerator which is similar to the Intel NCS2 in the aspect of size and price. The worse point is that the configuration of OpenVINO is much more complex than the framework used by the other two devices, but it does not give good outcomes in return.

In general, all these devices achieve a great accelerations compared with Raspberry Pi (the baseline device) with the fastest inference taking 523.70 ms.

5.3.3 Comparisons of Accuracy

5.3.3.1 Best accuracy of each device

Device	Best Top-1 Accuracy (%)	Best Top-5 Accuracy (%)	Corresponding Avg. Inference Time (ms)	Corresponding Model
Jetson Nano	72.75	91.06	592.63	EfficientNet B5
Coral USB Accelerator	67.90	86.46	98.64	Inception v4
Intel NCS2	60.63	83.08	185.60	Inception v4

TABLE 5.4. Best Accuracy

Again, Jetson Nano gets the best result regarding the inference accuracy. EfficientNet B5 model reaches 72.75% top-1 accuracy against ImageNetV2 test set. Coral USB Accelerator and Intel NCS2 gets fair accuracy as well using quantized Inception v4 model. However, the inference time of these models are pretty long.

5.3.3.2 Best Accuracy in Limited Inference Time

Now we set thresholds of 10 and 100 milliseconds on inference time and see which device can get best accuracy in the limited time.

Device	In 10 ms		In 100 ms	
	Top-1 Accuracy (%)	Top-5 Accuracy (%)	Top-1 Accuracy (%)	Top-5 Accuracy (%)
Jetson Nano	63.22	84.76	68.53	88.57
Coral USB	54.95	78.17	67.90	86.46
Intel NCS2	N/A	N/A	52.05	76.08

TABLE 5.5. Inference Speed Comparison

NVIDIA Jetson Nano wins in both thresholds. Coral USB Accelerator gets very close to Jetson Nano's results. Intel Neural Compute Stick 2 cannot finish work in 10 ms. One thing noticeable is that increasing time limits from 10 ms to 100 ms does not increase the accuracy so much in all devices.

5.3.4 Inference Time vs. Accuracy

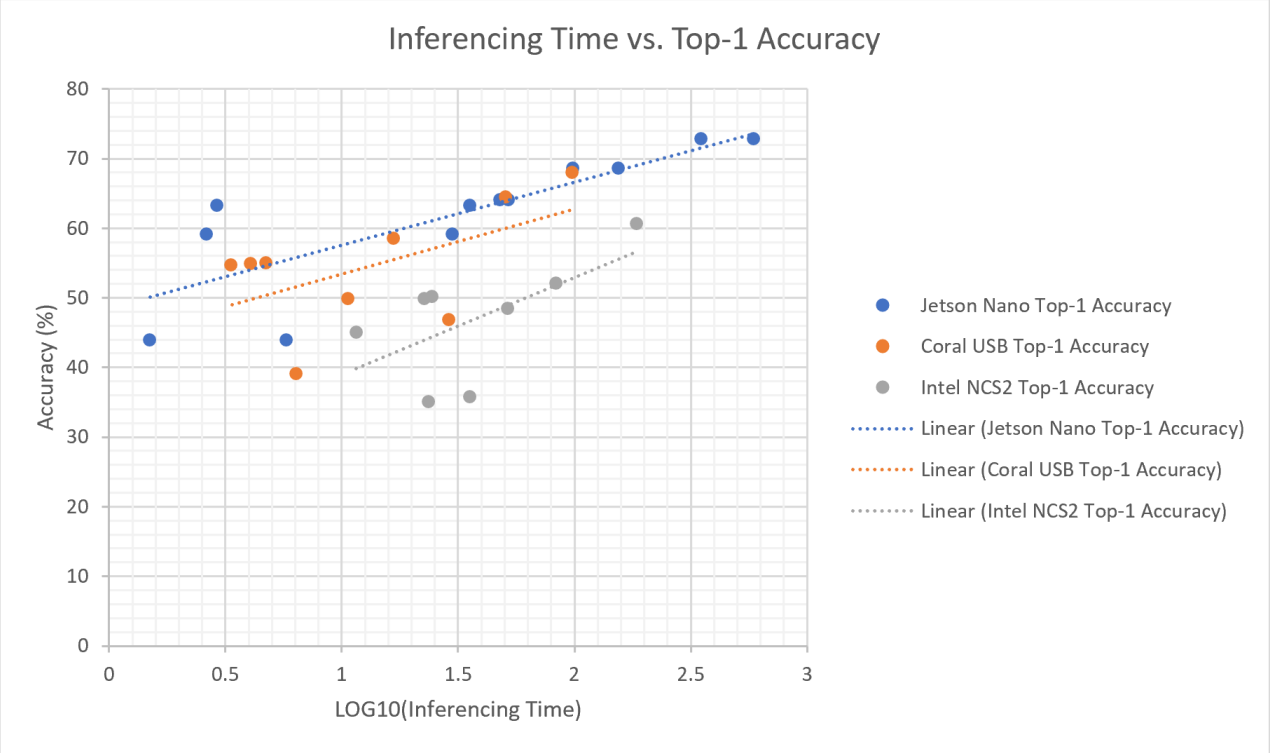


FIGURE 5.5. Inference Time vs. Top-1 Accuracy

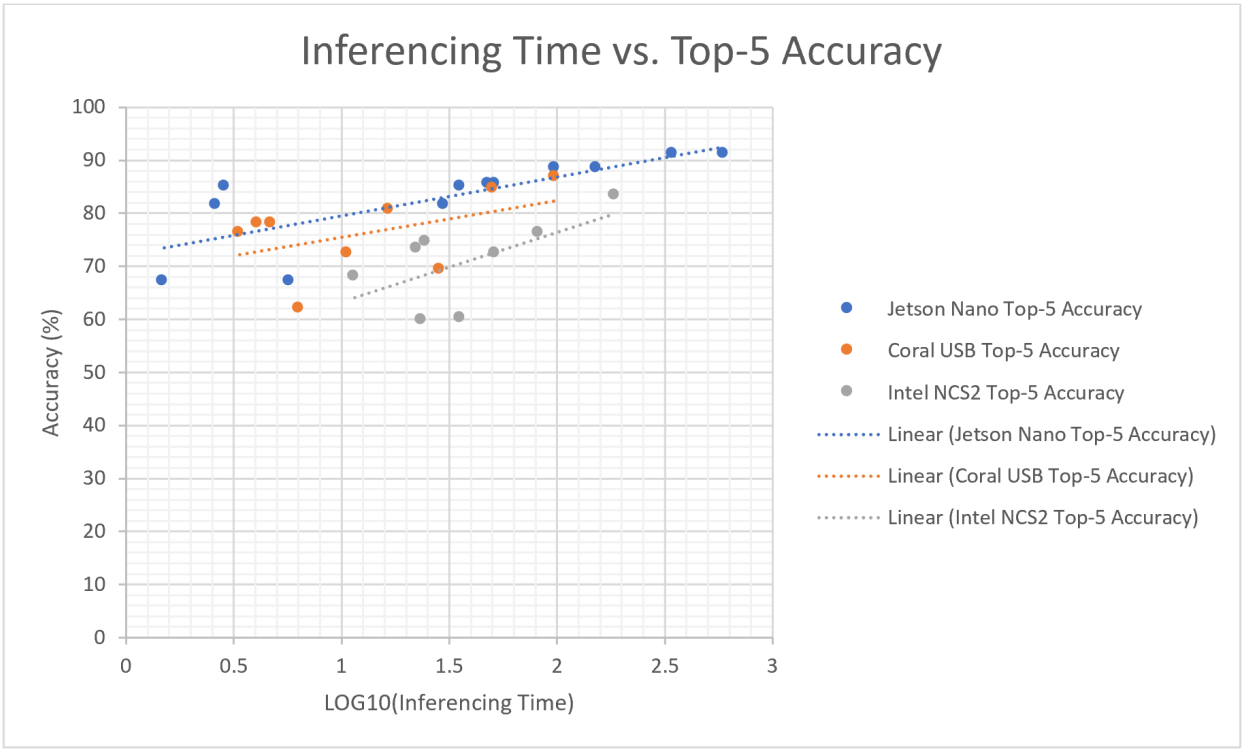


FIGURE 5.6. Inference Time vs. Top-5 Accuracy

As shown in the graph, Jetson Nano has the highest inference accuracy given the same inference time. Google's Coral USB Accelerator gets very close to the Jetson Nano in terms of the efficiency of

inference. Besides, both devices show strong linear relationships between the accuracy and inference speed. As for Intel Neural Compute Stick 2, the relationship between the accuracy and inference speed is a little weaker. Besides, the efficiency of Intel NCS2 is obviously lower than that of Jetson Nano and Coral USB Accelerator.

5.3.5 OpenVINO on VPU, CPU and GPU

Table 4.7 shows the results of running OpenVINO on three types of processors (VPU, CPU and GPU). All the three processors are built by Intel and compatible of using OpenVINO to accelerate deep learning tasks. Generally, Intel Neural Compute Stick 2 (with VPU) has a higher latency and lower throughput than CPU and GPU. Based on this test, Intel Neural Compute Stick 2 is not a good choice to accelerate PC AI and machine learning applications.

5.4 Stress Test

5.4.1 Analysis of Scatter Plot

In Chapter 4, the results of stress tests on the four devices are summarized into scatter plots. Detailed methods are discussed in Chapter 3.

From the scatter plots, we can get brief views of how each device performs in a three-hour stress tests. From Figure 4.1, we can conclude that Raspberry Pi 3B model can generally run in a stable frequency (with most inferences finished in around 1100 ms). From Figure 4.2, it is clear that for NVIDIA Jetson Nano, most of inferences are finished within 0.5 to 2.5 ms. The upper bound seems to be five times longer than the lower bound. However, continuous running does not seem to affect the inference speed. The histogram analysis below will reveal this problem more detailed. From Figure 4.3 of Coral USB Accelerator, we can clearly find a clear distinguishing point at around 6,000 seconds. Before that point, the changing range of inference times is from 5 to around 10 ms. After that point the upper bound increases to around 13 ms. This can be regarded as a sign of frequency decreasing due to overheating. From Figure 4.4 of Intel Neural Compute Stick 2, we can find that most of the inferences are finished in four outstanding times (corresponding to the four lines in Figure 4.4). As most points are exactly located in these four lines, this phenomenon can be regarded as a sign of active frequency adjustment.

5.4.2 Analysis of Histogram

As the scatter plots emphasis more on changes in computing speeds over time, we need to use histograms to determine the interval ratio of the times required for one iteration.

From Figure 5.7 to 5.10, four histograms are generated for our analysis.

5.4.2.1 Raspberry Pi

From Figure 5.7, Raspberry Pi shows a great stability in the three hours stress test. Up to 93% of the inferences are finished at a mean time of 1065 ms ($\pm$ 15 ms, which is 1.41% of the mean). However, as Raspberry Pi use CPU to do the inferences, there are chances that CPU resource is not fully occupied.

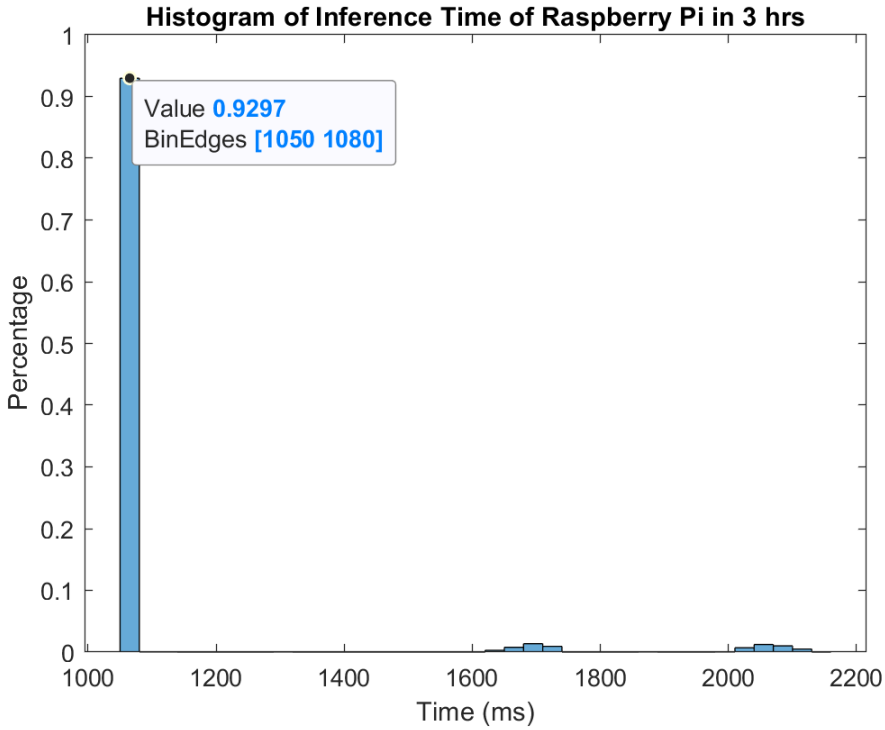


FIGURE 5.7. Histogram of Raspberry Pi

5.4.2.2 NVIDIA Jetson Nano

From Figure 5.8, NVIDIA Jetson Nano has a great accelerations compared with Raspberry Pi. Most (62.03%) of the inferences are expected to take 0.626 ms to 0.752 ms to finish. Around 95% of the inferences are finished at a mean time of 0.752 ms ($\pm$ 0.252 ms, which is 33.51% of the mean).

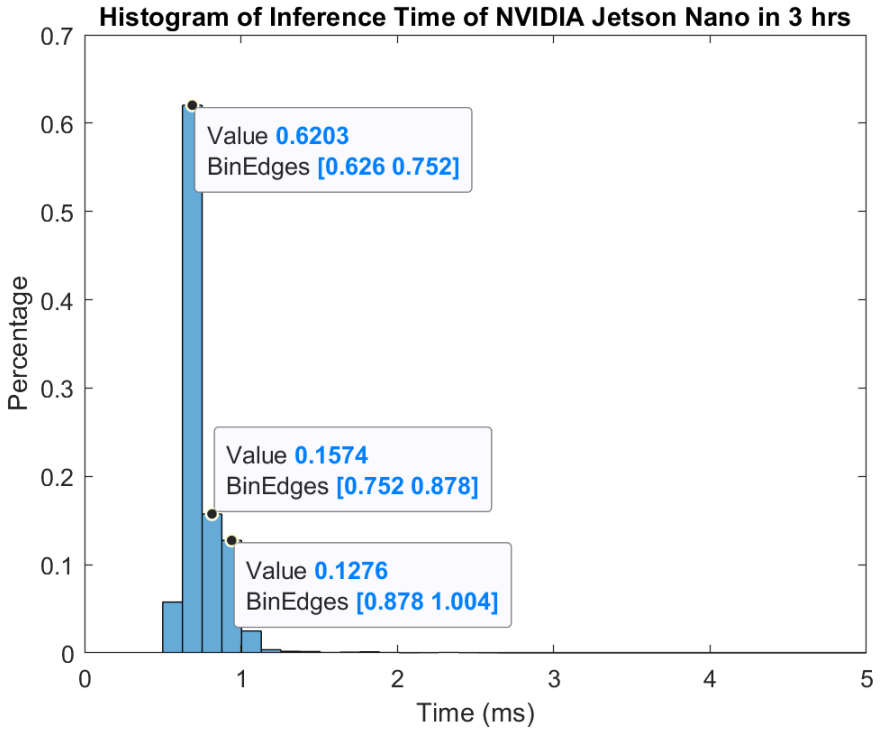


FIGURE 5.8. Histogram of NVIDIA Jetson Nano

5.4.2.3 Coral USB Accelerator

From Figure 5.9, Coral USB Accelerator is much faster than Raspberry Pi as well and also maintains a computing speed. 98.65% of the inferences are finished at a mean time of 5.98 ms ($\pm$ 0.98 ms, which is 16.39% of the mean)

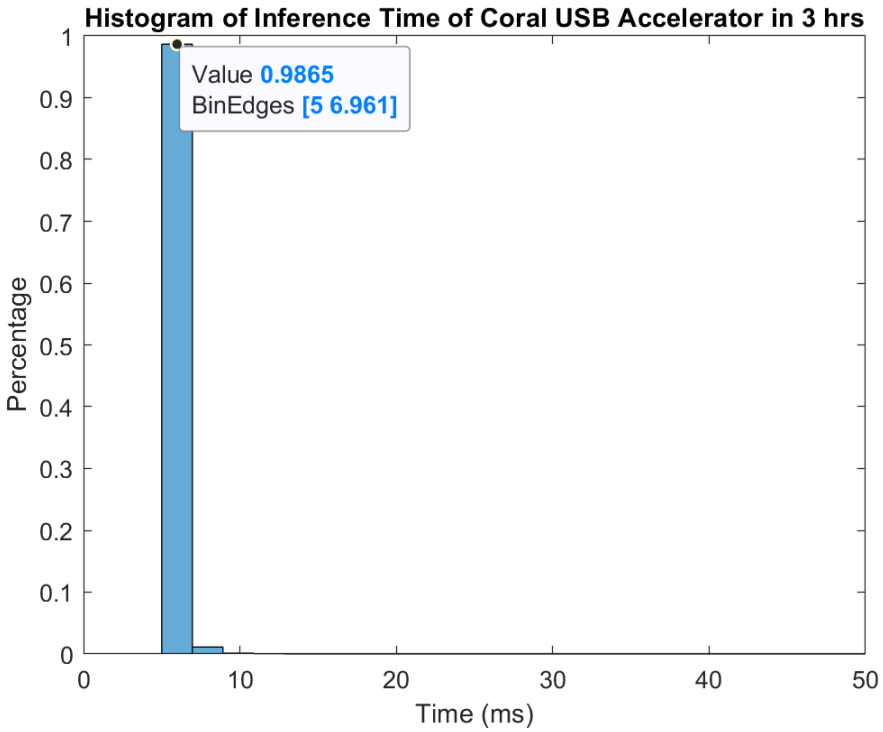


FIGURE 5.9. Histogram of Coral USB Accelerator

5.4.2.4 Intel Neural Compute Stick 2

From Figure 5.10, Intel Neural Compute Stick 2 is not that stable compared with the previous three devices. The inference time costs are precisely distributed in four intervals (as shown in Figure 5.10). It can be inferred from this result that Intel may adopt adaptive frequency control on this device.

However, we can find that only about 11.85% of all inferences are operated in the maximum frequency and around 86.40% are operated in lower frequencies. In actual use, this may cause unstable throughput.

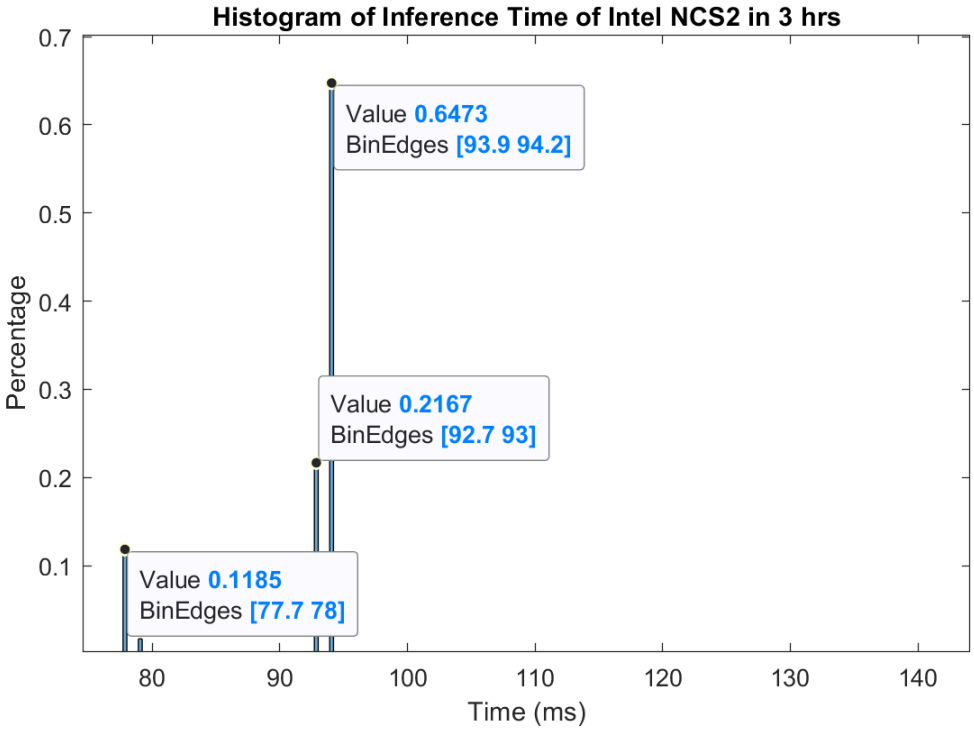


FIGURE 5.10. Histogram of Intel Neural Compute Stick 2

CHAPTER 6

Conclusion

In this report, I present a comparative study on three different edge machine learning devices. Both the software and the hardware of these devices are involved in the comparisons. Generally speaking, all these three devices can achieve a considerable acceleration in conducting machine learning (especially deep learning) tasks compared to the baseline device - Raspberry Pi 3B model.

However, looking into these three devices in details, we can find clearly which one is better than another in some aspects. NVIDIA Jetson Nano can be considered as the overall winner in the comparisons considering its speed, accuracy and ease of use with the standard CUDA environment. Coral USB Accelerator is impressive for speed and accuracy despite its small size and also for its simple configurations. Intel Neural Compute Stick 2 is somewhat inferior compared with the previous two. The configurations of this device is a little complex and the outcomes are not satisfactory (considering the slower computation speed, lower accuracy inference and instability). There is still big room for improvement for Intel.

Compared with previous benchmark reports as I represent in Chapter 2, my benchmark results are basically consistent with the previous results. Besides, I also provide some more experiments which are never mentioned in previous works. Thus, my work here is sound and meaningful for the results I provide.

In sum, a comparison study of some popular edge machine learning devices is presented in this report for further reference.

6.1 Future Work

In this report, only image classification tasks are selected in the experiments. However, there are also many other applications that involves deep learning algorithms and may even involve other deep neural network (DNN) structures like recurrent neural network (RNN) which has a wide application in time series forecasting (speech recognition). For those models, as the model size and memory consumption differs from image classification tasks, the performances of these devices may be different from the results in this report. Future work can focus on other different kinds of models.

Second, in this report, all experiments are conducted in synchronous execution, which means that inference is done in a one-by-one or batch-by-batch mode. However, all these three devices support asynchronous execution which may improve the performance.

References

[1] Y. Deng, 'Deep learning on mobile devices: a review', ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 10993, May 2019, 109930A. DOI: 10.1117/12.2518469. arXiv: 1904.09274 [cs.LG].

[2] J. Lee, N. Chirkov, E. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik and M. Grundmann, *On-device neural net inference with mobile gpus*, 2019. eprint: arXiv:1907.01989.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, 2016. eprint: arXiv:1603.04467.

[4] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, *Tensorflow: A system for large-scale machine learning*, 2016. eprint: arXiv:1605.08695.

[5] B. Recht, R. Roelofs, L. Schmidt and V. Shankar, *Do imagenet classifiers generalize to imagenet?*, 2019. eprint: arXiv:1902.10811.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, *Imagenet large scale visual recognition challenge*, 2014. eprint: arXiv:1409.0575.

[7] M. Tan and Q. V. Le, 'Efficientnet: Rethinking model scaling for convolutional neural networks', *arXiv preprint arXiv:1905.11946*, 2019.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, 'Mobilenets: Efficient convolutional neural networks for mobile vision applications', *arXiv preprint arXiv:1704.04861*, 2017.

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, 'Pytorch: An imperative style, high-performance deep learning library', in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[10] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks', in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition', *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. DOI: 10.1109/cvpr.2016.90.

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, 'Mobilenetv2: Inverted residuals and linear bottlenecks', *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. DOI: 10.1109/cvpr.2018.00474.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, 'Going deeper with convolutions', *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. DOI: 10.1109/cvpr.2015.7298594.

[14] S. Ioffe and C. Szegedy, 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', *arXiv preprint arXiv:1502.03167*, 2015.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, 'Rethinking the inception architecture for computer vision', *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. DOI: 10.1109/cvpr.2016.308.

[16] C. Szegedy, S. Ioffe, V. Vanhoucke and A. A. Alemi, 'Inception-v4, inception-resnet and the impact of residual connections on learning', in *Thirty-first AAAI conference on artificial intelligence*, 2017.