DEEP LEARNING FRAMEWORKS EVALUATION FOR IMAGE CLASSIFICATION ON RESOURCE CONSTRAINED DEVICE

Mathieu Febvay and Ahmed Bounekkar

Université de Lyon, Lyon 2, ERIC UR 3083, F69676 Bron Cedex, France

ABSTRACT

Each new generation of smartphone gains capabilities that increase performance and power efficiency allowing us to use them for increasingly complex calculations such as Deep Learning. This paper implemented four Android deep learning inference frameworks (TFLite, MNN, NCNN and PyTorch) to evaluate the most recent generation of System On a Chip (SoC) Samsung Exynos 2100, Qualcomm Snapdragon 865+ and 865. Our work focused on image classification task using five state-of-the-art models. The 50 000 images of the ImageNet 2012 validation subset were inferred. Latency and accuracy with various scenarios like CPU, OpenCL, Vulkan with and without multi-threading were measured. Power efficiency and real-world use-case were evaluated from these results as we run the same experiment on the device's camera stream until they consumed 3% of their battery. Our results show that low-level software optimizations, image pre-processing algorithms, conversion process and cooling design have an impact on latency, accuracy and energy efficiency.

KEYWORDS

Deep Learning, On-device inference, Image classification, Mobile, Quantized Models.

1. INTRODUCTION

Nowadays, mobile devices are in every human hand, replacing slowly but surely our way of life. Many mobile applications use artificial intelligence in diverse ways such as gaming, social media, artistic filters or augmented reality using different tasks like face detection, real-time image classification or object detection. Unfortunately, many artificial intelligence models run in the cloud due to the computational resources needed to execute their complexity with millions of parameters. Today, more than ever, data privacy represents a major concern for people. On-device inference is an alternative, protecting data, fixing loss of internet connectivity, and reducing computing costs. However, computing power on these devices is clearly insufficient to run effectively and submitted to energy limitations.

Recent improvements made on hardware like Neural and Tensor Processing Unit (NPU/TPU), Digital Signal Processor (DSP), and other accelerators [1] let Machine Learning and Deep Learning on-device execution possible [2, 3]. Several mobile deep learning frameworks have been developed by open-source community or industry leader with low-level software optimization like General Matrix Multiplication (GeMM), GPU libraries (e.g. OpenCL™, Vulkan® and OpenGL® ES) and most recently general hardware accelerators API like NNAPI letting on-device inference become a new opportunity [4].

David C. Wyld et al. (Eds): EMSA, SEA, AIFU, NLCAI, BDML, BIoT, NCOM, CLOUD, CCSEA, SIPRO - 2022
pp. 27-40, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.120603

But these features are implemented differently in frameworks and combination of both model, framework, hardware and device make performance assessment difficult.

Two smartphones and one tablet, based on the two most popular architecture, Qualcomm Snapdragon and Samsung Exynos, were chosen. Android devices were selected because of its easier framework deployment process compared to Apple iPhone. We used four different frameworks with different low-level software optimization techniques such as integration of Arm assembly language code portion, integration of GeMM libraries Eigen, OpenBLAS or custom, NPU support and different software graphic libraries (OpenCL, OpenGL, Vulkan). Our models are pre-trained on ImageNet dataset with both Tensorflow and PyTorch allowing us to easily convert them to our two other frameworks.

Our approach is to evaluate frameworks and models designed and developed for mobile devices with the objective of providing the community our inference latency, Top-1 and Top-5 accuracy and power efficiency results of different models allowing scientists to take the proper decisions and save time when choosing software libraries and hardware in order to run image classification, object detection, instance segmentation on resource constrained devices based on Arm Cortex A architecture. Our work differs from other as we developed an Android Java application for each framework where inference took place.

2. RELATED WORK

Bahrampour *et al.* [5] evaluated Deep Learning frameworks performance but they focused their work on desktop computer with a Titan X GPU.

Lu *et al.* [6] launched their benchmark on different mobile frameworks with a Nvidia TK1 and TX1 which are not smartphones or tablet used by customers.

Sehgal and Kehtarnavaz [7] offered a benchmark of multiple deep learning models inferring on mobile SoC but they tested TFLite and Core ML only.

MLPerf [8] and AI Benchmark [9, 10] provide an Android application to test various models on the device using different scenarios. Limitations are the inference engine which is based on TFLite only and the output result, approximated (MLPerf) or displayed as a weighted score (AI Benchmark).

Bianco *et al.* [11] and Almeida *et al.* [12] proposed the most related works. They evaluated multiple models on diverse architecture among which there are mobile SoCs. The main difference is they didn't run their test from an Android application.

Benchmarks and previous work to evaluate the performance of deep learning models or frameworks on different devices exist but we propose an alternative approach as we focused our test on mobile devices, either smartphone or tablet, with frameworks and models optimized for them.

3. ALGORITHMIC APPROACH

For our experiment, we chose two smartphones which had a SoC generation gap and one tablet with a boosted SoC. Four frameworks were implemented on which we executed seven models, five 32-bit floating point and two quantized (8-bit integer) used as image segmentation backbones. To simulate the most representative use cases for real-time image segmentation tasks, we needed a dataset with enough images. Our choice was to use the ImageNet 2012 validation dataset containing 50,000 images. We kept the results from this first benchmark to evaluate the device power consumption and the image inference latency from the device's camera.

3.1. Devices

We selected the latest Samsung Galaxy Tab S7 containing a Qualcomm Snapdragon 865+ SoC, the OnePlus 8 with a Qualcomm Snapdragon 865 and the newest Samsung Galaxy S21 with a Samsung Exynos 2100. The two Snapdragon are on the same architecture to explore if the extra 260 MHz on one big core and the 87 MHz boost on the GPU provided by the 865+ produce a significant impact on the latency. Recent release of the Exynos 2100 represents a generation gap with the Snapdragon 865. It's based on the new Arm Cortex X1 which, giving to Arm, is 30% faster and have twice the ML performance over the Cortex A77 [13]. The three devices have Android 11 operating system. Table 1 shows their specifications in-depth. Our experiment was launched on all hardware available on each device which was CPU, GPU and NPU/DSP with different hyper-threading scenarios. When we run on GPU, we inferred with OpenCL, OpenGL or Vulkan graphic libraries. Manufacturers consider CPU, GPU and NPU/DSP, as a whole, named the AI engine which can only run quantized models with specific software frameworks.

Table 1. Device SoC's specifications with quantity of RAM, type of cluster with number of cores in it, Arm reference and core frequencies

SoC	RAM (Gb)	Cluster	Number	Ref	Freq (GHz)
865	8	LITLLE	4	A55	1.80
		big	3	A77	2.42
		big	1	A77	2.84
865+	6	LITLLE	4	A55	1.80
		big	3	A77	2.42
		big	**1**	**A77**	**3.10**
2100	8	**LITLLE**	**4**	**A55**	**2.20**
		big	**3**	**A78**	**2.80**
		big	1	X1	2.90

3.2. Frameworks

We tested four open-source frameworks, TensorFlow Lite 2.4.0, MNN 1.1.0, NCNN 20201218 and PyTorch mobile 1.7.

They all had Arm NEON optimizations and OpenMP library integrated in their source code. TFLite [14] is, at the time of this paper, the only framework to have a general hardware accelerator library, NNAPI, which allow inference on the AI engine. MNN and NCNN use a custom GeMM implementation whereas PyTorch does not have a GPU and NPU inference option yet.

We selected these frameworks due to their mobile context. All of them are compatible with Android and iOS devices.

3.3. Models and Dataset

The inference was launched on ImageNet 2012 [15] pre-trained models commonly used as image segmentation backbone.

The main difficulty was to find different models available on both PyTorch and Tensorflow but we manage to download five 32-bits floating point models: SqueezeNet v1.1 (sqn11) [16], MobileNet v2 (mob2) [17], Inception v3 (inc3) [18], ResNet50 v1 (res50), ResNet101 v1 (res101) [19] and two TFLite quantized models: MobileNet v2 (mob2q) and Inception v3 (inc3q) to run on the AI engine.

Table 2 shows the Top-1 and Top-5 accuracy provided by Tensorflow and PyTorch Hub [20, 21, 22, 23].

Table 2. PyTorch and TensorFlow Top-1 and Top-5 model accuracies provided by the sources. Best accuracy for each model is in bold text

Framework	Model	Top-1 (%)	Top-5 (%)
PyTorch	**SqueezeNet v1.1**	**58.19**	**80.62**
	MobileNet v2	71.88	90.29
	Inception v3	77.45	93.56
	ResNet50 v1	**76.15**	**92.87**
	ResNet101 v1	**77.37**	**93.56**
Tensorflow	SqueezeNet v1.1	49.00	72.90
	MobileNet v2	**71.90**	**91.00**
	MobileNet v2 (quant)	70.80	89.9
	Inception v3	**78.00**	**93.90**
	Inception v3 (quant)	77.5	93.70
	ResNet50 v1	75.20	92.20
	ResNet101 v1	76.40	92.90

3.4. Model conversion process

The frameworks implemented for our experiment can't use the downloaded models, they need to be converted. TFLite and PyTorch mobile models were the easiest to switch because of the tools provided by their parent training framework but MNN and NCNN don't support all of the PyTorch and TensorFlow operations.

To be compatible, PyTorch models had to be converted in ONNX format. We run different converters to make them compatible with MNN and NCNN.

For Tensorflow models, the MNN and NCNN tools were unable to convert ResNet v1 and Inception v3 architecture.

3.5. Image pre-processing

During the training phase of our models, each image was transformed to fit in the input tensor. We had to reproduce the pre-processing steps to reproduce the best accuracy.

TensorFlow crops or pads the image to the littlest size followed by a scale down then it normalizes each image color channel, Red, Blue, Green, with mean and standard deviation equal to 127.5 for floating point models and mean to 0.0 and standard deviation to 1.0 for quantized models.

It is quite the opposite for PyTorch as it resizes the image before cropping or padding it. Its normalization parameters are respectively for red, blue and green channels and for mean: 0.485, 0.456, 0.406 and standard deviation: 0.229, 0.224, 0.225.

3.6. Algorithm

For each framework, we developed a Java Android application which looped on all the converted models and inferred each of the ImageNet 50,000 images for any hardware available (CPU, OpenCL, OpenGL, Vulkan or NNAPI) from one to ten threads.

At each inference the time elapsed by the device to output the probabilities was gathered. We compared the result to the image key contained in the ground truth file provided with the dataset to know if the highest probability and the five best were in it. Latency and accuracy are saved in a CSV file in the internal memory. When the test was launched, the device was plugged to the power source in plane mode and screen luminosity was at its minimum level. The energy consumption was not measure in this algorithm.

From the results collected in the previous algorithm, the same experiment parameters were executed from a camera stream acquired on the device. The energy efficiency of all the components as well as the image pre-processing time were evaluated. In addition, the screen and the camera power consumption were collected separately to isolate the hardware used during the inference.

Algorithm 1. Experiment algorithm

Input	image = 1,…,50000
Output	latency = inference latency of the image isInTop1 = ground truth compared to the best probability isInTop5 = ground truth compared to the five best probabilities
Parameters	hardware = CPU,…,NNAPI thread = 1,…,10 model = sqn11,…,inc3q

```
for hardware = CPU to NNAPI do
  for thread = 1 to 10 do
    for model = sqn11 to inc3q do
      for image = 1 to 50000 do
        preProcessedImage ← preProcessImage(image);
        startTime ← getSystemTime();
```

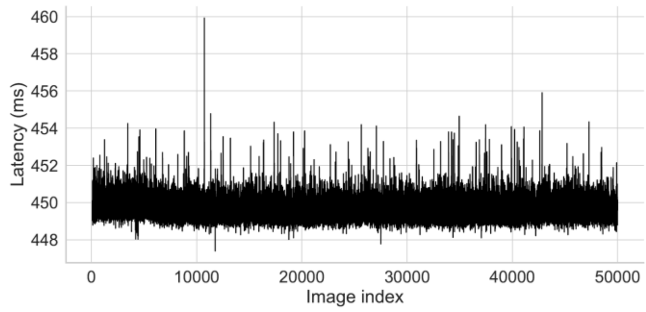
```
            probs ← infer(preProcessedImage);
            stopTime ← getSystemTime();
            latency ← (stopTime – startTime);
            descendantOrderSort(probs)
            isInTop1 ← false;
            isInTop5 ← false;
            if ground truth == probs[0] then
                isInTop1 ← true;
                isInTop5 ← true;
            end
            else if ground truth in probs[1:4] then
                isInTop5 ← true;
            end
            appendToCSV(latency, isInTop1, isInTop5);
        end
    end
  end
end
```

4. EXPERIMENTAL RESULTS

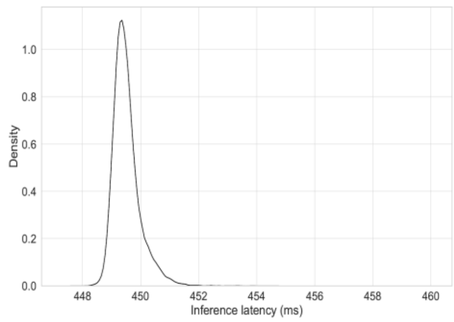
For our experiment, we chose two smartphones which had a SoC generation gap and one tablet with a boosted SoC. Four frameworks were implemented on which we executed seven models, five 32-bit floating point and two quantized (8-bit integer) used as image segmentation backbones.

4.1. ImageNet dataset latency

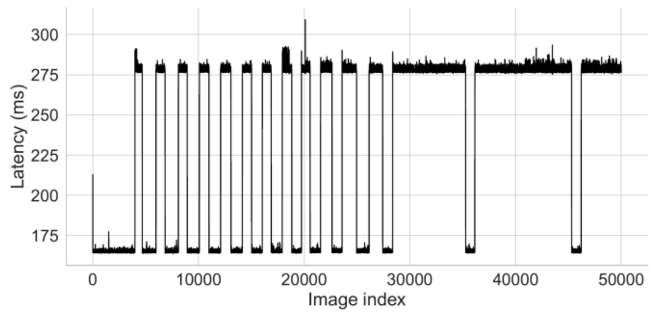
We ran Algorithm 1 on two smartphones and one tablet to get the closest real-world use case results. Our algorithm was looping on 50,000 images which could come closest to a video feed from the device camera to simulate an image segmentation backbone in real-time. An acceptable latency for this task is under 30 ms letting display around 30 frames per second while providing room for image pre-processing and decoding functions. One of the intrinsic limitations of our devices was the thermal protection mechanism also known as Dynamic Voltage Frequency Scaling (DVFS) or CPU throttling. The system downscales the CPU frequency to dissipate the heat. Figure 1 shows two different DVFS behaviours when we ran NCNN on the Snapdragon 865 with one CPU thread. DVFS effect of Inception v3 pre-trained with PyTorch (1a) is not obvious, resulting in a stable inference with a narrow range around 4 ms (1b). On the contrary, ResNet 50 v1 pre-trained with the TensorFlow framework (1c) shows two inference levels, 165 ms and 280 ms (1d). From the 30,000th image, the SoC is so hot it stands longer at 280 ms. DVFS is less present on the Snapdragon 865+ because it is an 11-inch tablet which contains more space to dissipate the heat, unlike the two other devices as shown on Figure 2.



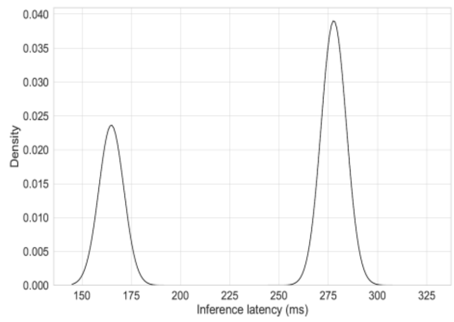
(a) Raw inference latency (in ms) of Inception v3 (PyTorch) without DVFS effect



(b) Kernel density estimation of Inception v3 (PyTorch) without DVFS effect



(c) Raw inference latency (in ms) of ResNet50 (Tensorflow) v1 with DVFS effect



(d) Kernel density estimation of ResNet50 (Tensorflow) v1 with DVFS effect

Figure 1. Inference without (a)(b) and with (c)(d) DVFS on Snapdragon 865 CPU with 1 thread

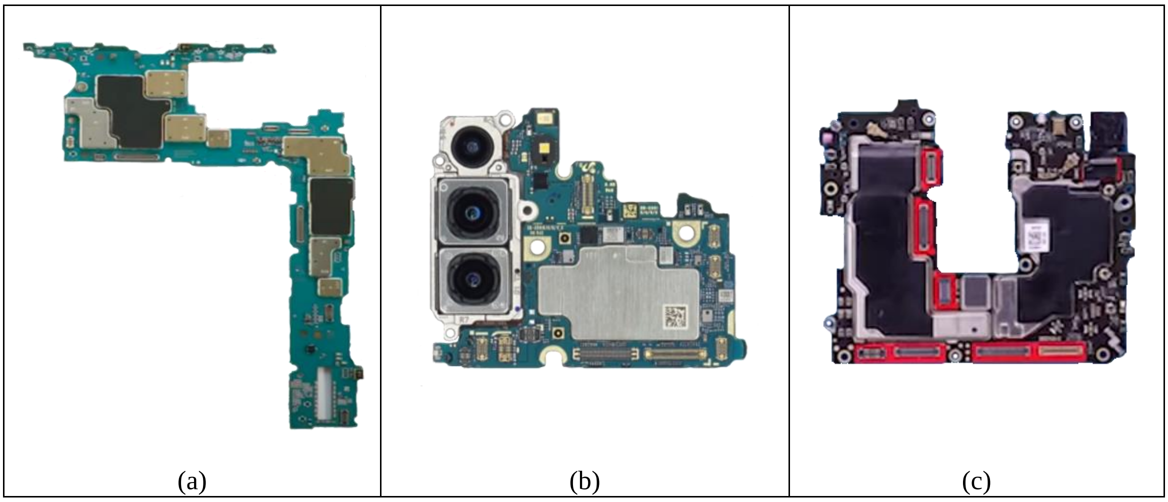
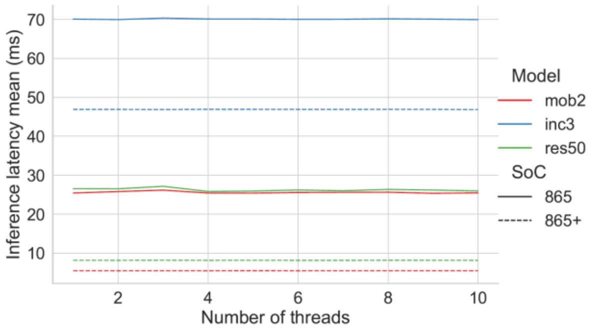


Figure 2. SoC's boards from Samsung Galaxy Tab S7 (a), Samsung Galaxy S21 5G (b) and OnePlus 8 (c) (not to scale)

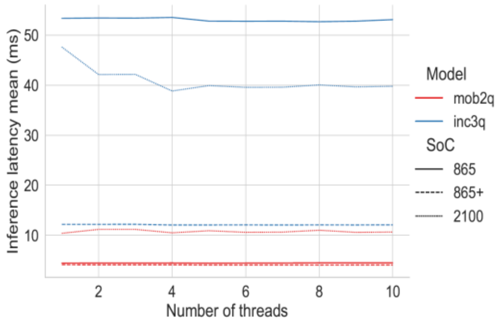
Figure 3a shows that the multi-threading mechanism didn't affect the GPU. Switching from 1 to 10 threads didn’t affect the latency.

Figure 3b shows the AI engine, which uses CPU, GPU and NPU.

We saw that MobileNet v2 and Inception v3 latencies were improved when switching from the GPU with floating point format to the AI engine with quantized one. Quantized version of Inception v3 on the Exynos 2100 is improved when running from 1 to 4 threads. The NNAPI library uses the best hardware in order to improve the latency. In our case, the library used the NPU, and the GPU excepted for Inception v3 model on the Exynos 2100.



(a)



(b)

Figure 3. Influence of multi-threading on GPU (a) and AI engine (b)

Table 3 represents the arithmetic mean (µ) and the standard deviation (σ) of the inference latency in milliseconds with the accuracy loss compared to their reference model in Table 2. For each row we reported the best results of our experiment.

TensorFlow model latencies are the best with TFLite OpenCL for floating point models. We greyed SqueezeNet v1.1, ResNet50 v1 and ResNet101 v1 Tensorflow models in our table due to conversion issue reported in Section 3.4. The new Exynos 2100 provides an improvement in comparison of the Snapdragon 865 especially for PyTorch models inferred with NCNN Vulkan. NCNN has a non-negligible accuracy drop on different models but on SqueezeNet v1.1 it is 18 % faster than the second best framework, MNN, with 13.91 ± 0.16 ms on Snapdragon 865 and 13.36 ± 0.43 ms on Snapdragon 865+.

Snapdragon 865+ outperforms the most recent generation due to its better CPU and GPU frequency. This increase in frequency should represent a problem due to the throttling mechanism however the device demonstrates an excellent capability to dissipate the heat making extra computational power efficient.

The inconclusive results of the TFLite NNAPI on the 2100 should be related to the driver compatibility of the Samsung NPU which was probably unimplemented yet.

Table 3. Best mean (μ) with standard deviation (σ) of inference time in milliseconds for each model trained by TensorFlow (model-tf) and PyTorch (model-pt) of all hardware and framework on the three devices with their accuracy loss compared to Table 2 Top-1 and Top-5

SoC	Model	Framework	Hardware	μ ± σ (ms)	Top-1 (%)	Top-5 (%)
Snapdragon 865	sqn11-pt	NCNN	CPU2	11.40 ± 3.89	-13.07	-10.67
	sqn11-tf	NCNN	CPU3	20.04 ± 4.13	-24.67	-28.25
	mob2-pt	MNN	CPU7	15.96 ± 2.32	-3.17	-1.49
	mob2-tf	NCNN	CPU3	13.00 ± 2.86	-3.79	-3.30
	mobq2-tf	TFLite	NNAPI	4.44 ± 0.69	-1.79	-1.15
	inc3-pt	MNN	CPU8	158.54 ± 33.94	-1.40	-0.68
	inc3-tf	TFLite	OpenCL	70.07 ± 1.69	-0.44	-0.24
	inc3q-tf	TFLite	NNAPI	52.67 ± 4.46	-0.26	-0.20
	res50-pt	NCNN	Vulkan	86.23 ± 2.87	-7.70	-4.22
	res50-tf	TFLite	OpenCL	26.54 ± 13.06	-47.08	-42.04
	res101-pt	NCNN	Vulkan	133.17 ± 2.24	-5.78	-3.09
	res101-tf	TFLite	OpenCL	25.98 ± 13.03	-48.28	-42.74
Snapdragon 865 +	sqn11-pt	NCNN	CPU3	10.95 ± 2.24	-13.07	-10.67
	sqn11-tf	TFLite	OpenCL	8.14 ± 0.59	-20.88	-22.74
	mob2-pt	NCNN	CPU3	14.54 ± 1.24	-9.51	-5.78
	mob2-tf	TFLite	OpenCL	5.47 ± 0.70	-1.70	-1.63
	mobq2-tf	TFLite	NNAPI	4.07 ± 0.81	-1.79	-1.15
	inc3-pt	MNN	CPU5	182.09 ± 23.17	-1.40	-0.68
	inc3-tf	TFLite	OpenCL	46.85 ± 0.60	-0.44	-0.24
	inc3q-tf	TFLite	NNAPI	12.06 ± 0.81	-0.26	-0.20
	res50-pt	NCNN	Vulkan	66.29 ± 2.75	-7.70	-4.22
	res50-tf	TFLite	OpenCL	8.13 ± 0.58	-47.08	-42.04
	res101-pt	NCNN	Vulkan	101.22 ± 2.3	-5.78	-3.09
	res101-tf	TFLite	OpenCL	8.17 ± 0.59	-48.28	-42.74
Exynos 2100	sqn11-pt	MNN	CPU4	12.88 ± 0.44	-4.17	-3.08
	sqn11-tf	TFLite	OpenCL	18.02 ± 7.46	-20.88	-22.74
	mob2-pt	MNN	CPU5	14.55 ± 3.43	-3.17	-1.49
	mob2-tf	TFLite	OpenCL	11.37 ± 6.80	-1.70	-1.63
	mobq2-tf	TFLite	NNAPI	10.77 ± 5.56	-1.79	-1.15
	inc3-pt	MNN	CPU4	194.73 ± 43.03	-1.40	-0.68
	inc3-tf	TFLite	OpenCL	93.05 ± 31.31	-0.44	-0.24
	inc3q-tf	TFLite	NNAPI	40.92 ± 9.53	-0.26	-0.20
	res50-pt	NCNN	Vulkan	81.48 ± 9.72	-7.70	-4.22
	res50-tf	TFLite	OpenCL	17.00 ± 6.71	-47.08	-42.04
	res101-pt	NCNN	Vulkan	117.25 ± 29.78	-5.78	-3.09
	res101-tf	TFLite	OpenCL	17.07 ± 7.01	-48.28	-42.74

4.2. Accuracy

An accuracy loss occurred when the model is converted. For Tensorflow models there was a drop of 1-2 % on Top-1 and 2-3 % on Top-5 on all frameworks with, from the lowest loss to highest: TFLite, MNN, NCNN. The same figures appeared for PyTorch ones except for NCNN which had a 7-13 % drop on Top-1 and a 5-11 % drop on Top-5 depending on models.

In addition, SqueezeNet v1.1, ResNet50 v1, ResNet101 v1 from TensorFlow were not operating the pre-processing parameters provided on TensorFlow Hub leading to an accuracy cap on both Top-1 and Top-5 with respectively 28.12 % and 50.16 % for them.

The accuracy loss from the quantized model is negligible regarding the latency gain. MobileNet v2 lost 1.89 % compared to its floating-point version but it reduced its latency by 5 % on the 2100 SoC, 26 % on the 865+ and 66 % on the 865. This gain is even bigger with Inception v3 model.

4.3. Camera stream latency

The camera stream from the device was integrated inside the Android application using Camera2 API. Images are acquired by the device camera in the YUV420 format and converted into ARGB8888 to make it compatible with models input. The image's size is 640 pixels width and 480 pixels height. These new outcomes integrate the image pre-processing latency executed by the framework and show CPU and GPU governors behaviour once the device is powered by its battery. These results are consistent with the ImageNet ones. There is a performance drop for all the frameworks as the device has to manage its energy. We observe that TFLite is more affected than MNN or NCNN. Once again, quantized models outperform the others on the three devices. It is particularly obvious for Inception v3 as it is approximately 3 times faster than its floating-point version on Snapdragon 865, 2.5 times on Snapdragon 865+ and 1.5 times on the Exynos 2100. This experiment confirms the performance of the Snapdragon 865+ related to a reduced DVFS effect.

4.4. Power efficiency

Before each test, devices were fully charged, screen brightness was set to medium, Bluetooth and Wi-Fi were turned ON to reproduce as much as possible real usage of the device. The test was stop once the device's battery reaches 97 % to avoid the nonlinear discharge of the lithium-ion battery.

We recorded the elapsed time for the device to go from 100 to 97 % with the help of Battery Historian software from Google [24]. We measured the screen consumption by setting the device in plane mode and recording the time for the device to reach 97 % when the screen is ON with medium brightness. Then we measured the camera consumption by doing the same process as for the screen but with launching the camera application. Then we subtracted the screen consumption to the observed one to have the camera.

The energy consumption for the Snapdragon 865, 865+ and Exynos 2100 screen are respectively 214 mAh, 619 mAh and 198 mAh. For the cameras, 438 mAh, 413 mAh and 792 mAh. The Snapdragon 865+ screen is bigger than the two others and the Exynos 2100 has the most powerful camera module.

Once again, our results show small models and quantized models are the more energy efficient. The faster it runs the less energy it consumes. Also, device screen and camera have a bigger impact on energy than the dedicated inference hardware.

Table 4. Latency and energy consumed in µA for processing one image from device camera stream after consuming 3 % of battery device. Hardware consumption is for the energy consumed by the hardware involved in the inference (CPU, GPU, NPU, RAM) and Device consumption represents the total of energy consumed by the device (screen and camera included).

SoC	Model	Framework	Hardware	µ ± σ (ms)	Hardware consumption (µA/img)	Device consumption (µA/img)
Snapdragon 865	sqn11-pt	NCNN	CPU2	11.40 ± 3.89	2.73	6.55
	sqn11-tf	NCNN	CPU3	20.04 ± 4.13	5.69	9.76
	mob2-pt	MNN	CPU7	15.96 ± 2.32	0.64	4.53
	mob2-tf	NCNN	CPU3	13.00 ± 2.86	3.24	6.49
	mobq2-tf	TFLite	NNAPI	4.44 ± 0.69	0.92	3.69
	inc3-pt	MNN	CPU8	158.54 ± 33.94	27.40	59.78
	inc3-tf	TFLite	OpenCL	70.07 ± 1.69	17.06	34.22
	inc3q-tf	TFLite	NNAPI	52.67 ± 4.46	2.47	7.40
	res50-pt	NCNN	Vulkan	86.23 ± 2.87	13.11	31.55
	res50-tf	TFLite	OpenCL	26.54 ± 13.06	7.14	15.58
	res101-pt	NCNN	Vulkan	133.17 ± 2.24	20.06	48.12
	res101-tf	TFLite	OpenCL	25.98 ± 13.03	8.50	25.48
Snapdragon 865 +	sqn11-pt	NCNN	CPU3	10.95 ± 2.24	3.92	7.57
	sqn11-tf	TFLite	OpenCL	8.14 ± 0.59	2.97	8.06
	mob2-pt	NCNN	CPU3	14.54 ± 1.24	4.85	9.37
	mob2-tf	TFLite	OpenCL	5.47 ± 0.70	2.16	6.49
	mobq2-tf	TFLite	NNAPI	4.07 ± 0.81	0.94	5.09
	inc3-pt	MNN	CPU5	182.09 ± 23.17	51.73	107.44
	inc3-tf	TFLite	OpenCL	46.85 ± 0.60	11.44	30.98
	inc3q-tf	TFLite	NNAPI	12.06 ± 0.81	2.69	10.35
	res50-pt	NCNN	Vulkan	66.29 ± 2.75	18.88	39.22
	res50-tf	TFLite	OpenCL	8.13 ± 0.58	8.01	19.73
	res101-pt	NCNN	Vulkan	101.22 ± 2.3	34.17	70.97
	res101-tf	TFLite	OpenCL	8.17 ± 0.59	14.28	35.13
Exynos 2100	sqn11-pt	MNN	CPU4	12.88 ± 0.44	1.15	5.39
	sqn11-tf	TFLite	OpenCL	18.02 ± 7.46	1.89	13.18

mob2-pt	MNN	CPU5	14.55 ± 3.43	1.67	7.82
mob2-tf	TFLite	OpenCL	11.37 ± 6.80	3.21	14.99
mobq2-tf	TFLite	NNAPI	10.77 ± 5.56	2.96	13.70
inc3-pt	MNN	CPU4	194.73 ± 43.03	10.57	73.67
inc3-tf	TFLite	OpenCL	93.05 ± 31.31	21.73	60.38
inc3q-tf	TFLite	NNAPI	40.92 ± 9.53	4.30	30.48
res50-pt	NCNN	Vulkan	81.48 ± 9.72	14.17	39.63
res50-tf	TFLite	OpenCL	17.00 ± 6.71	7.18	33.19
res101-pt	NCNN	Vulkan	117.25 ± 29.78	15.63	54.00
res101-tf	TFLite	OpenCL	17.07 ± 7.01	15.18	52.90

5. CONCLUSION

In this paper we presented an inference latency benchmark on mobile to help the community better deployed image classification/segmentation model on Android devices. Our results showed that quantized models on AI engine should be the de facto standard, especially for complex models like Inception v3. Quantized models are more energy efficient and performs better than floating point ones with a tiny loss of accuracy. If there is no other choice than floating points, developers should go for TFLite. It experiences an easy model conversion and integration process on Android. For PyTorch models, we saw NCNN is a notable candidate, but it needs to improve its conversion process to gain more accuracy. We are looking forward to GPU and NPU/DSP's support in the future PyTorch mobile framework.

MNN and NCNN integration of these frameworks inside Android application is not a straightforward task. The conversion step is not user-friendly as engineer need to compile or find the appropriate converter and execute commands to transform the original model to a compatible and optimized one. Additionally, framework libraries must be compiled and integrated with the Android NDK which is an error prone process.

To conclude, manufacturers should improve heat dissipation or cooling mechanism on small devices to avoid the DVFS effect resulting in an improved latency.

REFERENCES

[1] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators.2019 IEEE High Performance Extreme Computing Conference (HPEC), pages 1–9, 2019.

[2] Sahar Voghoei, Navid Hashemi Tonekaboni, Jason G Wallace, and Hamid Reza Arabnia. Deep learning at the edge. 2018 International Conference on Computational Science and Computational Intelligence (CSCI), pages 895–901, 2018.

[3] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim M. Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch,Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian,Sungjoo Yoo, and Peizhao Zhang. Machine learning at facebook: Understanding inference at the edge.2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pages 331–344, 2019.

[4] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. In WWW ’19, 2019.

[5] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative study of deep learning software frameworks arXiv: Learning, 2016.

[6] Zongqing Lu, Swati Rallapalli, Kevin S. Chan, and Thomas F. La Porta. Modeling the resource requirements of convolutional neural networks on mobile devices. Proceedings of the 25th ACM international conference on Multimedia, 2017.

[7] Abhishek Sehgal and Nasser Kehtarnavaz. Guidelines and benchmarks for deployment of deep learning models on smartphones as real-time apps. Machine Learning and Knowledge Extraction, 1:450–465, 2019.

[8] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark, 2019.

[9] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In ECCV Workshops, 2018.

[10] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seung soo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. Ai benchmark: All about deep learning on smartphones in 2019.2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pages 3617–3635, 2019.

[11] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. IEEE Access, 6:64270–64277, 2018.

[12] Mario Almeida, Stefanos Laskaridis, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. Embench: Quantifying performance variations of deep neural networks across modern commodity devices. In The 3rd International Workshop on Deep Learning for Mobile Systems and Applications, EMDL’19, page 1–6, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367714.doi: 10.1145/3325413.3329793. https://doi.org/10.1145/3325413.3329793.

[13] Arm Ltd. Introducing the arm cortex-x custom program, 2021. https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/arm-cortex-x-custom-program.

[14] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean,Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kud-lur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng.Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265–283, 2016. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C.Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[16] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J.Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. ArXiv, abs/1602.07360, 2016.

[17] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. ArXiv, abs/1801.04381, 2018.

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.
[20] Google. Tensorflow hub, 2020. https://tfhub.dev/.
[21] Google. TFLite hosted models, 2020. https://github.com/tensorflow/models/tree/master/research/slim.
[22] PyTorch. Pytorch model hub, 2020. https://pytorch.org/hub/.
[23] TensorFlow. TFLite hosted models, 2020. https://www.tensorflow.org/lite/guide/hosted_models.
[24] Google. Battery historian GitHub, 2021. https://github.com/google/battery-historian.

AUTHORS

Mathieu Febvay is currently a PhD candidate at ERIC laboratory at University of Lyon in France (UR 3083). His research focuses on Lightweight Deep Learning where he investigates performance and feasibility of neural networks model running on resource constrained devices. He also works as a Software Engineer on mobile devices. He holds a Master in Computer Science (MIAGE) from University of Lyon (2017) and is graduated in Software Development from University of Montpellier (2008). He has interest in the field of health and mobile medical devices.

Ahmed Bounekkar is an Associate Professor at the University of Lyon 1, attached to the ERIC laboratory. Since 2009, he has been in charge of the Master MIAGE in management informatics. His research focuses on modelling in complex systems for the design of decision support methodologies. They particularly concern the development of algorithms for data structuring, machine learning and multi-objective optimisation problems. The proposed models mainly concern problems in the field of health.