Matrik: Jurnal Manajemen, Teknik Informatika, dan Rekayasa Komputer
Vol. 24, No. 1, November 2024, pp. 25∼38
ISSN: 2476-9843, accredited by Kemenristekdikti, Decree No: 200/M/KPT/2020
DOI: 10.30812/matrik.v24i1.3938

Characterizing Hardware Utilization on Edge Devices when Inferring Compressed Deep Learning Models

Ahmad Nabhaan[1], **Rakandhiya Rachmanto**[2], **Arief Setyanto**[1]
[1]Universitas AMIKOM, Yogyakarta, Indonesia
[2]University of Georgia, Athens, United State

Article Info

Article history:

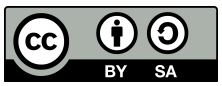
Received March 30, 2024
Revised September 09, 2024
Accepted October 08, 2024

Keywords:

Deep Learning
Edge Devices
Hardware Utilization
Memory Allocation
Post-training Quantization

ABSTRACT

Implementing edge AI involves running AI algorithms near the sensors. Deep Learning (DL) Model has successfully tackled image classification tasks with remarkable performance. However, their requirements for huge computing resources hinder the implementation of edge devices. Compressing the model is an essential task to allow the implementation of the DL model on edge devices. Post-training quantization (PTQ) is a compression technique that reduces the bit representation of the model weight parameters. This study looks at the impact of memory allocation on the latency of compressed DL models on Raspberry Pi 4 Model B (RPi4B) and NVIDIA Jetson Nano (J. Nano). This research aims to understand hardware utilization in central processing units (CPU), graphics processing units (GPU), and memory. This study focused on the quantitative method, which controls memory allocation and measures warm-up time, latency, CPU, and GPU utilization. Speed comparison among inference of DL models on RPi4B and J. Nano. This paper observes the correlation between hardware utilization versus the various DL inference latencies. According to our experiment, we concluded that smaller memory allocation led to high latency on both RPi4B and J. Nano. CPU utilization on RPi4B. CPU utilization in RPi4B increases along with the memory allocation; however, the opposite is shown on J. Nano since the GPU carries out the main computation on the device. Regarding computation, the smaller DL Size and smaller bit representation lead to faster inference (low latency), while bigger bit representation on the same DL model leads to higher latency.



Corresponding Author:

Arief Setyanto, +6281316024569
Faculty of Computer Science,
Universitas AMIKOM, Yogyakarta, Indonesia,
Email: arief_s@amikom.ac.id

How to Cite:
A. Nabhaan, R. Rachmanto, and A. Setyanto, "Characterizing Hardware Utilization on Edge Devices when Inferring Compressed Deep Learning Models", *MATRIK: Jurnal Manajemen, Teknik Informatika, dan Rekayasa Komputer*, Vol. 24, No.1, pp. 25-38, November, 2024.

1. INTRODUCTION

Deep learning (DL) has promising results in solving many computation problems, such as image processing, natural language processing, and time series prediction [1, 2]. Convolutional neural networks (CNN) based on techniques such as VGG16, EfficientNet, DenseNet, MobileNetV3, and many more have achieved great success in image classification applications [3–6]. Yet the high hardware utilization requirements of the well-known successful DL model led to hard implementation in several edge devices [7, 8]. Edge devices such as Raspberry Pi 4 model B (RPi4B) and NVIDIA Jetson Nano (J. Nano) have been designed to work in a constrained environment. Edge computing improves resource use for low-latency applications by relocating processes closer to clients, ensuring network-aware services are continuously accessible [9, 10]. It allows for compact design and is embedded in many use cases, such as robotics, smart cars, agricultural sector, and drone applications [11–14]. However, the native DL model cannot be implemented directly on those edge devices. A native DL model could suffer performance bottlenecks due to limited hardware utilization [15, 16]. This approach has a low processing level and might be performed as a real-time classification in a low-cost system, similar to one of the research approaches of Labib et al. [17], we need a smaller DL model. Post-training quantization (PTQ) is applied to this work to make the DL model smaller. In concept, The PTQ method to directly quantize neural network models without fine-tuning, with symmetrical power-of-two thresholds and uniform quantizers, is a method that minimizes processing costs by allowing integer arithmetic without cross-terms linked to zero-point and floating-point scaling [18]. Eventually, PTQ reduces latency and power consumption by replacing floating-point computations with efficient low-bit operations [19]. To apply PTQ in our models, TensorFlow Lite (TFLite) for RPi4B and NVIDIA TensorRT (TensorRT) for J. Nano are employed in this study. Memory allocation inevitably occurs when loading a DL model. DL models have tensors in them to calculate inputs that will be released as outputs. Intermediate tensors in DNNs have the potential to use quite a lot of memory, even with modest input sizes [20]. Then, memory allocation management plays a role in hardware performance and makes its characteristics appear when DL models are run with limited memory. Memory limitations could make the DL models run out of memory or increase computing time. Overall, we believe that various data characteristics will appear with various devices and DL models, and memory limitation can lead to this research, such as Hasan et al., which was characterized by various metrics. In addition, different hardware platforms can result in different hardware performances because of different computation frequencies, memory access speeds, and I/O communication latency [21].

The research conducted by Shafi et al. [22] investigates the impact of software optimization frameworks, specifically the TensorRT inference framework, on the accuracy and performance of neural network models when deployed on edge devices. Their analysis focuses on evaluating the performance gains and non-deterministic behaviors resulting from these software optimizations, particularly on real-world applications such as intelligent traffic intersection control and Advanced Driving Assistance Systems (ADAS)—additionally, the study by Wisultschew et al. [23] compares the hardware performance and the validation of DNN accuracy on various edge devices while executing object classification. Prashanthi SK et al. [24] delved into the influence of hardware and platform configurations on the training performance of DNNs on edge devices, aiming to provide insights for optimizing training time and energy consumption. Jing et al. [25] conducted research exploring system characteristics while running general AI applications on Intel Software Guard Extensions (SGX). Lastly, Hao et al. [26] evaluated edge devices' performance and resource heterogeneity for deep learning tasks, compared DNN models on various devices and assessed the performance of popular deep learning frameworks in edge computing scenarios. These studies collectively contribute to advancing knowledge in edge computing, deep learning, and their practical applications.

Overall, researchers [22–26] have studied and characterized the performance of hardware and the DL model when executing the DL model on edge devices to determine the suitability and effectiveness of the DL applications they deploy on their specified edge devices. However, several researchers neglected the lower limit of hardware performance, which the DL model can still run. Therefore, hardware performance characteristics could appear in a limited resource or a normal condition. This research is aware that memory limitations could influence computing time. Memory limitations are a characteristic of hardware performance when the DL model is executed on a device. It will be beneficial if DL models are resistant to edge devices with limited memory. Moreover, this research applies DL to favor memory-constrained devices and maximize available resources.

Our research objectives focus on understanding the characteristics of DL models under stringent memory limitations to identify the performance of edge devices while executing DNN tasks. The distinction between our study and the studies performed by researchers [22–26] lies in our detailed examination of the RPi4B CPU, J. Nano GPU, and memory utilization. We introduce three cutting-edge DL models that have emerged in recent years. DL converter frameworks (such as TFLite and TensorRT) are utilized in our quantization work. This research measures warm-up time to evaluate hardware performance during the initial execution of the DL application. We have implemented memory allocation in our study to address memory limitations. Thus, the contributions of this research include assessing hardware performance when various DL models are executed on the RPi4B CPU and J. Nano GPU, applying memory allocation with specified constraints while benchmarking DL models to evaluate their robustness on memory-limited edge devices, listing these models in a table, plotting benchmark outcomes to uncover hardware performance traits during DL

model execution, and drawing correlation between DL latency and its determinants (such as hardware utilization during inference). Based on the experiment, we conclude that MobileNetV3Small in integer-8 quantization on RPi4B and DenseNet121 in integer-8 quantization on J. Nano are the most adaptable to memory limitations. CPU utilization and latency on RPi4B are significantly related, and GPU utilization affects latency on J. Nano. Meanwhile, the correlation of memory allocation on RPi4B hasn't had a significant relationship with latency. Finally, we implicate that future studies can utilize optimized DL models to enhance real-time decision-making processes, ultimately benefiting fields like healthcare, transportation, and smart cities by facilitating quicker and more dependable AI applications. Furthermore, exploring hybrid approaches that combine other DL compression techniques with other optimization techniques may yield even greater efficiency, allowing for broader deployment of AI solutions in resource-constrained environments.

2. RESEARCH METHOD

This research employs a quantitative method to explore variations in hardware performance when executing the DL model with constrained memory allocation. To support our research, we utilize popular edge devices as hardware platforms, various state-of-the-art DL models, DL frameworks that enable PTQ features, and classification for DL applications using a pre-trained DL model with the 102 Flower dataset. Additionally, we elaborate on the benchmark method, outlining the scenario of memory allocation and the design of the hardware utilization benchmark.

2.1. Choosing Hardware Platform

We determine edge devices based on recent years' two most popular edge devices. Enhancing RPi4B is the more effective approach for increasing deep learning performance [27], as it is applicable for DL models with TFLite. Then, J. Nano can evaluate image classification faster with embedded GPU, but low power consumption [28], as it is applicable for DL models with TensorRT. We compile an overview of these devices as shown in Table 1.

Table 1. Edge Devices List

	Raspberry Pi 4 Model B	Jetson Nano
CPU	4-core Cortex-A72	4-core Cortex-A57
GPU	-	128-core Nvidia Maxwell
Memory	8GB LPDDR4	4GB LPDDR4
Power Supply	5V DC 2.5A	5V DC 4A

2.2. Choosing Deep Learning Models

MobileNetV3, DenseNet, and EfficientNet are all state-of-the-art convolutional neural network (CNN) architectures that excel in computer vision tasks, each with unique features and benefits. We employ MobileNetV3 because its design is effective for mobile and edge devices. It uses a combination of hardware-aware network architecture search (NAS) and a NetAdapt algorithm to fine-tune each layer [5]. Aside from its popularity, we selected DenseNet because of a compact and deep CNN with various depth and complexity, such as DenseNet-121, DenseNet-169, and DenseNet-201 [29]. In addition, we selected EfficientNet, which provided a spectrum of speed and accuracy trade-offs [6], making it highly scalable and efficient across a range of resource constraints.

Small models are required to support resource-constrained devices to run those DL models on edge devices. However, initially, the DL model was large. Fortunately, PTQ supports the rapid execution of CNN models on resource-constrained devices with considerable accuracy degradation, particularly in low-precision representations [30, 31].

In our research, we choose the several models mentioned in Table 2, which the selected model supports with our DL application. We selected image classification for the DL application, which is trained with Oxford Flowers 102 since it is a common object detection application and may be leveraged as a key component in many AI applications (e.g., face detection) on edge devices. We perform quantization with post-quantization with two types, namely 16-bit floating point (FP16) and 8-bit integer (INT8), to aim for a model that can be smaller than the baseline and more efficient in CPU, GPU, and memory workloads. TFLite assists in the generation of DL models for RPi4B, while TensorRT aids in the generation of DL models for J. Nano.

Table 2. Overview of Deep Learning Models

	Model Size (MB)				
	Baseline	TFLite		TensorRT	
		FP16	INT8	FP16	INT8
MobileNetV3Small	7.6	3.15	1.85	5.04	7.08
MobileNetV3Large	15	5.90	3.42	9.37	17.58
DenseNet121	37	13.58	7.10	19.73	34.61
DenseNet169	62	24.30	12.58	31.37	49.79
DenseNet201	86	35.07	18.05	44.66	72.89
EfficientNetB0	22	7.95	4.80	9.76	16.74
EfficientNetB1	34	12.76	7.60	15.09	26.79
EfficientNetB2	39	15.04	8.84	17.33	31.13
EfficientNetB3	51	20.78	12.09	23.46	42.61
EfficientNetB4	80	33.90	19.39	37.05	68.82
EfficientNetB5	123	54.52	30.62	58.45	110.36

2.3. The Scenario of Memory Allocation

The benchmark under DL models only evaluates latency, CPU, and memory utilization. In that circumstance, the Tegrastats command is not used. In the RPi4B scenario, memory is allocated by increasing physical memory from 10 MB to 510 MB and decreasing swap memory from 500 MB to 0 MB in 20 stages that are rounded three times. The physical and swap memory ranges are reversed in the second round. The DL model benchmarks latency, CPU, GPU, and memory utilization. Regarding model observations, this scenario is identical to the TFLite scenario. TensorRT is not compatible with Raspberry Pi. Then, TensorRT is specific for J. Nano. The J. Nano scenario has increased memory allocation from 250 MB to 1500 MB and decreased swap capacity from 1250 MB to 0 MB with 50 stages. It is rounded in the same sequence as the TFLite scenario. The scenario described in Table 3.

Table 3. Algorithm 1. The Pseudo-code of Scenario

```
Data: start: starting memory allocation;
end: ending memory allocation;
stage: memory allocation stages;
model: model path;
Benchmark: Benchmark workflow function;
Result: Benchmarking the given model with different memory allocations
foreach g in range(start, end+step, step) do
    for _ from 1 to 5 do
        Benchmark(model path=model, iteration=10,
        memalloc=g);
foreach k in reverse(range(start, end, step)) do
    for _ from 1 to 5 do
        Benchmark(model path=model, iteration=10,
        memalloc=k);
foreach l in range(start+step, end+step, step) do
    for _ from 1 to 5 do
        Benchmark(model path=model, iteration=10,
        memalloc=l);
```

The CGroup has a role in allocating memory. Physical memory is dynamically configured to suit the scenario. Swap memory is statically configured from the rest of all available memory on the device minus the applied physical memory. We set the swap memory level at 10, which is at the level when swap memory is not really prioritized.

2.4. Designing the Benchmark of Hardware Utilization

We develop a benchmark that measures DL latency and hardware utilization and deploys it alongside an image classification application. Figure 1 depicts the benchmark measurement flow. The benchmark is invoked from a Python script that takes parameters in a command line. The command of the scenario script is followed by the number of iterations and the DL model path. Before running the benchmark script, we installed a 10-second delay after the configuration of memory allocation to ensure memory alloca-

tion was implemented correctly. In the first run of the benchmark script, an image classification application acquired one input and one output. Even though the output is not saved, we only save the latency from that application. After the load model succeeds, the hardware profilers (e.g., tegrastats and pidstat by sysstat) run in the other thread. Before the DL model runs, we installed a delay of 2 seconds to ensure the hardware profilers are ready to measure the hardware utilization. If the latency is under 1 second, we install a delay of 2 seconds minus the latency to ensure the hardware profiler correctly records hardware utilization while the model runs. After the DL model succeeds in inference, we get the memory utilization by using psutil tool, and we evaluate the memory in 3 types of memory: Resident Set Size (RSS), Proportional Set Size (PSS), and Unique Set Size (USS). We measure these three types of memory to monitor physical memory utilization while physical memory is constrained. RSS identifies non-swapped physical memory in a running process. USS counts a single memory process, which would be released if the process had been properly terminated. Proportional Set Size (PSS) calculates memory shared with other processes so that it is split equitably across the methods that share it. Because of the difference in profile for each memory utilization type, we intend to analyze those utilizations while executing DL inference.

At the end of the benchmark script, the hardware profiler stops assesses the recording, stores it in a data frame with the latency, and finally cleans the cache. In our study, we benchmarked five times to ensure consistency in the results. On the first inference, we additionally ran a warm-up execution. We then ran it ten times in inference. In the final data set that will be used, we collected 1672 data points in 14 columns.

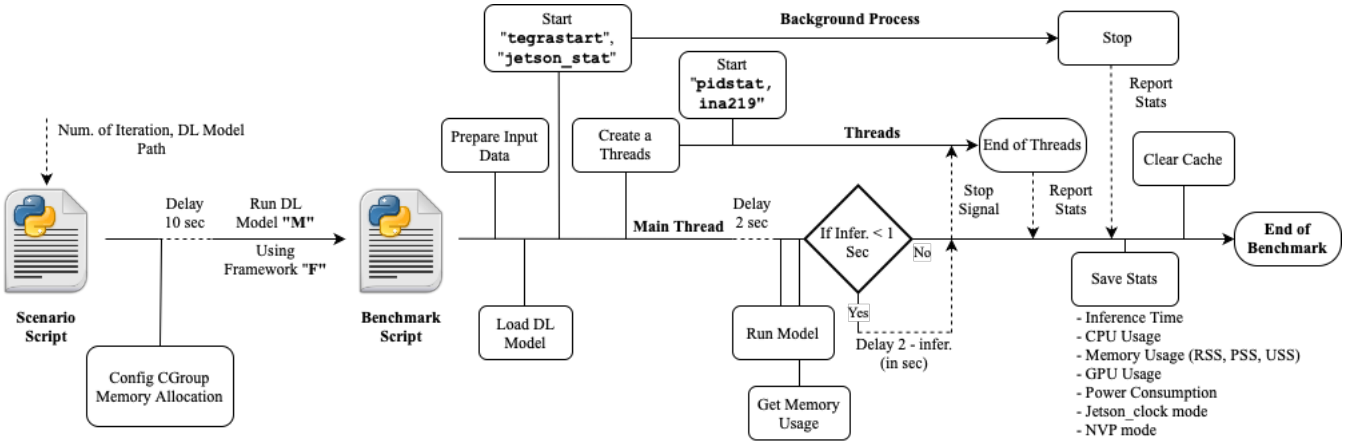


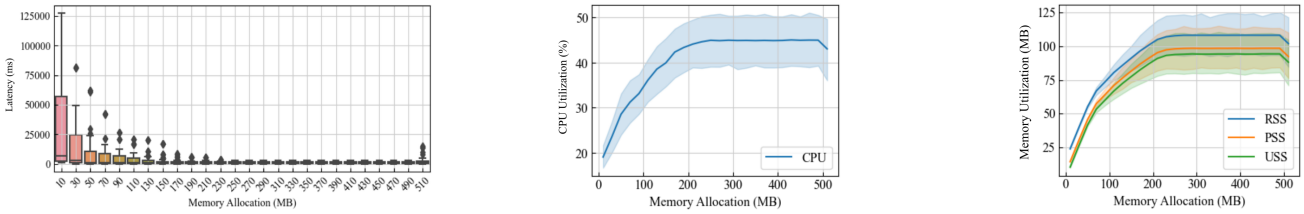
Figure 1. The Diagram of Benchmark Workflow

3. RESULT AND ANALYSIS

This research's findings include understanding the factors of latency, memory allocation, and hardware utilization. Additionally, we have other findings, including speed comparisons among DL model inferences and a correlation between latency and hardware utilization. At the end of the result and analysis point, we also compare hardware characteristic results in related research.

3.1. Memory Allocation Factor

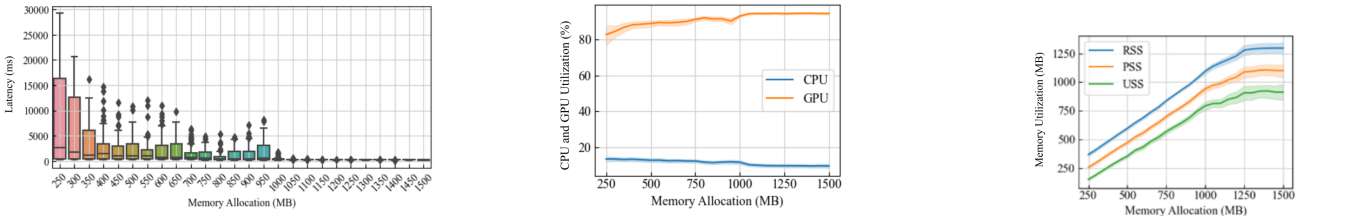
Memory Allocation on RPi4B at Figure 2(a) depicts the spread of latency with memory allocation. Memory allocation impacts memory sizes ranging from 10 MB to 230 MB. Dots in the range 30 MB to 230 MB and 510 MB in Figure 2(a) represents the minority value of the latency slower than the average. The stable latency in memory allocation is 250 MB until 490 MB, indicating that the DL model application is ready to execute properly with memory restrictions beginning at 250 on RPi4B. When memory allocation is 510 MB, swap memory allocation is 0 MB. Thus, some memories on swap saved before must be loaded into physical memory, and the model takes more time to load. Figure 2(b) and Figure 2(c) depict CPU and memory utilization with the same movement in the memory allocation factor. The movement tended to rise in the first allocation until the 510 MB allocation, at which point swap memory is freed.



(a) Memory Allocation vs. Latency (b) Memory Allocation vs. CPU and GPU Utilization (c) Memory Allocation vs. Memory Utilization

Figure 2. The Influence of Memory Allocation with the Latency and Hardware Utilization on Executions of DL Models on RPi4B

Memory Allocation on J. Nano—The DL model impacts memory allocation on J. Nano. However, Figure 3(a) illustrates that memory allocation and latency are not as linear as the DL model on RPi4B. The latency is lower than the DL model on J. Nano. According to hardware utilization, memory allocation impacts sizes ranging from 250 MB to 1000 MB. Dots in Error! Reference source not found.a come from models with a higher latency than the other model at the established memory allocation. The movement of CPU utilization at Figure 3(b) decreased slowly as memory allocation increased. On the other hand, GPU utilization increased slowly as memory allocation increased, although the allocation of around 1000 MB experienced a decrease in GPU utilization. Memory utilization significantly increased until around 1200 MB and steadily in the last memory allocation.



(a) Memory Allocation vs. Latency (b) Memory Allocation vs. CPU and GPU Utilization (c) Memory Allocation vs. Memory Utilization

Figure 3. The Influence of Memory Allocation with the Latency and Hardware Utilization on Executions of DL Models on J. Nano

3.2. Warm-up Time vs The Latency after Warm-Up

We are concerned about the time disparity between the first inference (warm-up) time of the DL model and the latency after warming up the inference of the DL model. The warm-up time of DL models on RPi4B is slightly the same as the latency after warming up the inference of the DL model, as seen in Figure 4. However, when using DL models, FP16 quantization models are slower to infer than INT8 quantization models. According to Figure 5 Most DL models on J. Nano have a longer warm-up time than the latency after warming up the inference of the DL model. Only DenseNet's warm-up time is slightly the same as the latency after warming up the inference of the DL model.

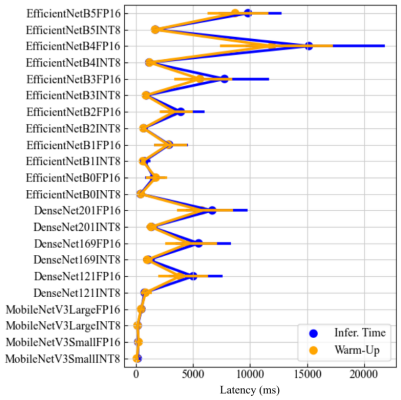


Figure 4. The latency after warming up the inference of DL model vs. warm-up time pattern for all DL models on RPi4B

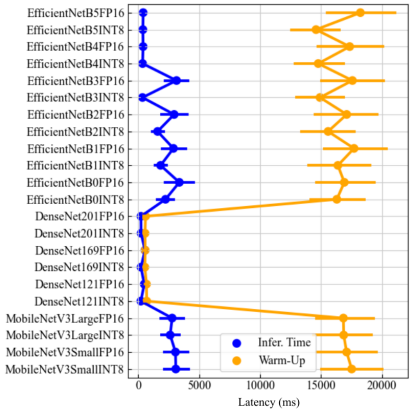
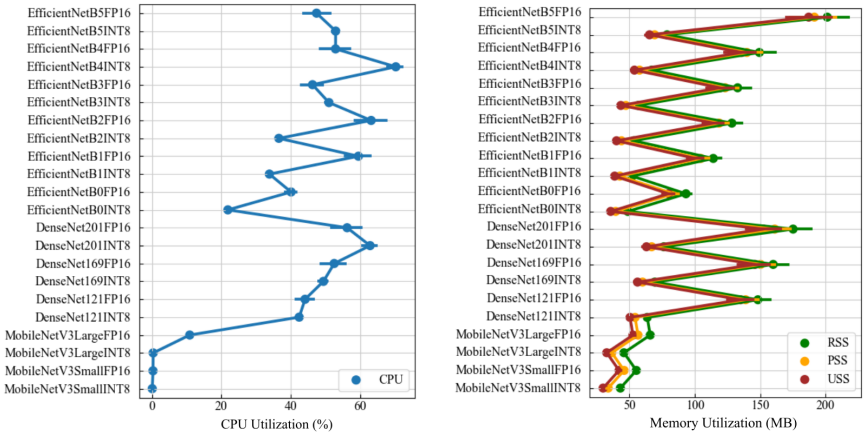


Figure 5. The latency after warming up the inference of the DL model vs. warm-up time pattern for all DL models on J. Nano

Based on the pattern of Figure 4 and Figure 5, RPi4B does not need more time at the first inference of the DL models. Otherwise, J.Nano needs more time at the first inference of the DL models, except for all DenseNet models. It denotes that the hardware and the type of DL model influence the warm-up time.

3.3. Hardware Utilization Factor

Hardware Utilization on RPi4B—According to Figure 6 DL models of type FP16 are slower than those of type INT8 when executed on RPi4B. Then, Figure 6(a) and Figure 6(b) describes CPU and memory characteristics on RPi4B with the DL models, respectively. EfficientNetB3, EfficientNetB5, and DenseNet201 with FP16 quantization require less CPU work than those models with INT8 quantization. EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB4, DenseNet121, DenseNet169, and MobileNetV3Large with FP16 quantization have higher CPU computation. However, the CPU utilization for MobileNetV3Small in all quantization types seems to be the same. The average memory utilization for the model in FP16 quantization is higher than that of the models in INT8 quantization. In memory detail, RSS, PSS, and USS use slightly the same memory. Those patterns are the same for all DL models. Thus, RSS can represent memory utilization on RPi4B while executing inference of DL models because it records the largest memory utilization.

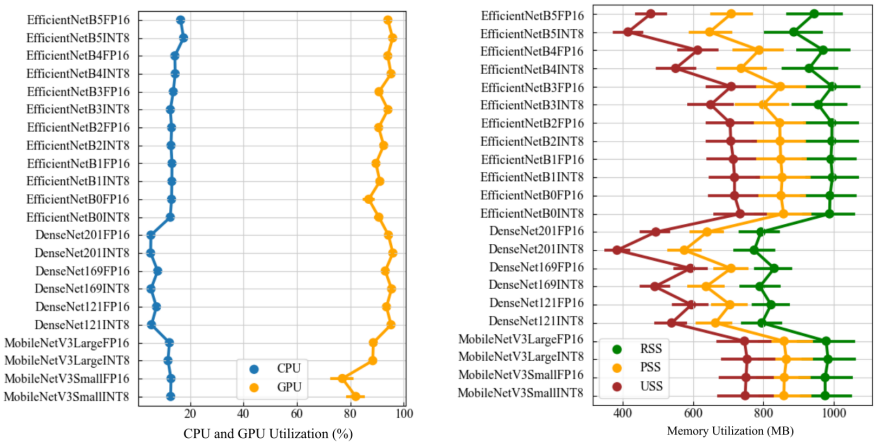


(a) CPU Utilization (b)Memory Utilization

Figure 6. The plot of Hardware Utilization While Executing Inference of DL on RPi4B

Hardware Utilization on J.Nano at Figure 7(a) and Figure7(b) describe the hardware utilization on J. Nano while executing inference of DL. Regarding CPU utilization, EfficientNetB5, DenseNet169, and DenseNet201 with INT8 quantization are inclined higher than both models with FP16 quantization. EfficientNetB0 in all quantization types are stable until EfficientNetB4,

DenseNet201, MobileNetV3Small, and MobileNetV3Large. Regarding GPU utilization, INT8 quantization models, such as MobileNetV3Small, EfficientNetB5 until EfficientNetB0, and DenseNet in all structure types, are higher than those with FP16 quantization. MobileNetV3Large in all quantization types is stabilized. In memory utilization, DenseNet in all structure types, and EfficientNetB3 until EfficientNetB5, those with INT8 quantization are inclined lower than those models with FP16 quantization. Memory tends to be stable on MobileNetV3Small, MobileNetV3Large, and EfficientNetB0 to EfficientNetB2, whose DL models use both types of quantization.



(a) CPU Utilization (b)Memory Utilization

Figure 7. Plot of Hardware Utilization While Executing Inference of DL on J. Nano

3.4. Speed Comparison among Inference of DL Models on Edge Devices

We understood that latency is influenced by memory allocation. Thus, the memory allocation makes the limitation of latency. Eventually, we will determine the fastest and slowest latency to determine the speed performance of the DL better. We list them in Table 4 for DL models that run on RPi4B and Tab le 5 for DL models that run on J. Nano.

Table 4. Pembagian data untuk Training dan Testing

Model Name	Latency (ms)
MobileNetV3Small INT8	150.17
MobileNetV3Large INT8	150.29
MobileNetV3Small FP16	193.39
EfficientNetB0 INT8	391.65
MobileNetV3Large FP16	475.16
EfficientNetB2 INT8	670.69
EfficientNetB1 INT8	730.94
DenseNet121 INT8	754.03
EfficientNetB3 INT8	905.92
DenseNet169 INT8	1087.71
EfficientNetB4 INT8	1150.71
DenseNet201 INT8	1353.67
EfficientNetB0 FP16	1566.71
EfficientNetB5 INT8	1691.64
EfficientNetB1 FP16	2903.79
EfficientNetB2 FP16	3930.29
DenseNet121 FP16	5022.87
DenseNet169 FP16	5498.97
DenseNet201 FP16	6689.44
EfficientNetB3 FP16	7768.62
EfficientNetB5 FP16	9830.10

Table 5. Pembagian data untuk Training dan Testing

Model Name	Latency (ms)
DenseNet121 INT8	223.01
DenseNet201 FP16	232.74
DenseNet169 INT8	233.85
DenseNet201 INT8	241.39
EfficientNetB3 INT8	356.83
EfficientNetB4 INT8	366.59
EfficientNetB5 INT8	390.60
EfficientNetB4 FP16	407.59
EfficientNetB5 FP16	413.81
DenseNet121 FP16	502.06
DenseNet169 FP16	562.14
EfficientNetB2 INT8	1595.63
EfficientNetB1 INT8	1843.46
EfficientNetB0 INT8	2246.86
MobileNetV3Large INT8	2630.39
MobileNetV3Large FP16	2790.28
EfficientNetB1 FP16	2896.85
EfficientNetB2 FP16	2954.30
MobileNetV3Small FP16	3065.22
MobileNetV3Small INT8	3091.38
EfficientNetB3 FP16	3150.36

Based on Table 3, MobileNetV3Small INT8 quantization is the fastest latency for RPi4B. The majority of INT8 quantization models occupy a low latency. Entire quantization types of MobileNetV3Small and MobileNetV3Large are in the top-10 faster. In the top 10, there are also low variations of EfficientNet and DenseNet, such as EfficientNetB0 until EfficientNetB3, DenseNet121, and DenseNet169, which use INT8 quantization. Under the top-10 DL models with INT8 quantization are just EfficientNetB4 and EfficientNetB5. Another FP16 quantization model is placed below the top-10, showing the lowest latency.

Table 4 states DenseNet121 INT8 quantization is the fastest latency for J. Nano. The same model in FP16 quantization is classified in the top-10. Middle-variant DL models of INT8 quantization models, such as DenseNet169 and EfficientNetB3, are in the top 10 faster. DenseNet201, EfficientNetB4, and EfficientNetB5, which all have all types of quantization, are among the top-10. The FP16 quantization models, such as DenseNet169, MobileNetV3Small, MobileNetV3Large, EfficientNetB0, and EfficientNetB3, were placed under the top 10 showing the lowest latency. MobileNetV3Small, MobileNetV3Large, EfficientNetB0, and EfficientNetB3, which are INT8 quantization models, are placed in the top-10 as well.

Eventually, we denoted the result from the order in both tables that DL inference latency on RPi4B will be fast if the DL model quantized with INT8. Otherwise, DL inference latency on J. Nano will be fast if we use a certain DL model. J. Nano does not make a neat order. This means that J. Nano needs to optimize model selection with agile algorithms for difficult-to-determine characteristics.

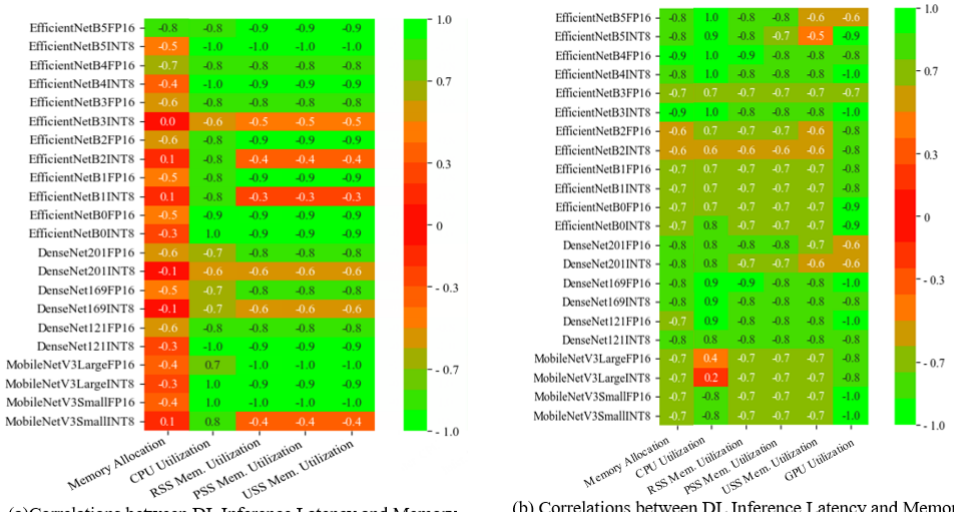
3.5. Correlation of Deep Learning Inference Latency with Memory Allocation and Hardware Utilization

Each DL model has unique hardware utilization. Hardware utilization affects the latency of DL models. However, we need to identify the factors that influence the latency of DL models specifically. We assess the influence of factors on latency when the DL model performs inference and memory allocation using Pearson correlation (in equation 1) [32]. x are dependent factors and y is an independent factor.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2(y_i - \bar{y})^2} \tag{1}$$

Pearson coefficients have values ranging from 1 to -1. The score demonstrates a linear connection between two variables. Coefficient score 1 implies that the variables are perfectly correlated. A negative score indicates the opposite of a correlation, while 0 indicates no connection between two variables. For easy classification, colors represent correlation values in the heatmap. The correlation score is good if it is closer to green. Otherwise, the correlation score is worse if it is closer to red. The color tends to be orange, which means the correlation level is moderate.

Memory Allocation—According to Figure 8(a), memory allocation has various correlations with the latency of DL models on RPi4B. EfficientNetB5 and EfficientNetB4 in FP16 quantization have good correlations in memory allocation. Otherwise, in INT8 quantization, those models have a moderate correlation. EfficientNetB0 until EfficientNetB3, DenseNet201, DenseNet169, DenseNet121, MobileNetV3Large, and MobileNetV3Small in FP16 quantization have a moderate correlation. Contrarily, those models in INT8 quantization have a low correlation. In correlations of DL model latencies with memory allocation, as shown in Figure 8(b), EfficientNetB2 has a moderate correlation in all quantization types, while the other DL models have a good correlation.



(a)Correlations between DL Inference Latency and Memory Allocation and Hardware Utilization on RPi4B

(b) Correlations between DL Inference Latency and Memory Allocation and Hardware Utilization on J. Nano

Figure 8. Heatmap of Correlations between DL Inference Latency and Memory Allocation and Hardware Utilization on a Couple of Edge Devices

Hardware Utilization—CPU utilization has a role in latency. CPU utilization in DL inference is not weakly correlated across all the DL models, as seen in Figure 8(a). It's just that EfficientNetB3 and DenseNet201 latency with INT8 quantization have a moderate correlation. The remaining latency of DL models exhibits a high correlation. EfficientNetB2, EfficientNetB3, DenseNet201, DenseNet169, and MobileNetV3Small, which use INT8 quantization, have a moderate correlation between latency and memory utilizations (e.g., RSS, PSS, and USS). EfficientNetB1 latency with INT8 quantization has the worst correlation. The remaining latency of DL models exhibits a high correlation.

Most latency of DL models on J. Nano exhibit a decent correlation with hardware utilization, as stated in Figure 8(b), notably, the correlation between CPU utilization and latency. The Latency of EfficientNetB2 with INT8 quantization and MobileNetV3Large with FP16 quantization have a moderate correlation with hardware utilization. A weak relationship between latency and hardware utilization exists only for MobileNetV3Large with INT8 quantization. The other DL models have a good correlation. In memory correlation, EfficientNetB2 in INT8 quantization has a moderate correlation with all memory utilization details. There is a moderate correlation between latency and USS memory utilization, including EfficientNetB2 with FP16 quantization, EfficientNetB5 with all quantization types, and DenseNet201 with INT8 quantization. The latency of other models correlates well with the USS memory utilization. RSS and PSS memory utilization strongly correlate with every DL model latency except EfficientNetB2 with INT8 quantization. For correlation with GPU utilization, moderate correlation only occurs in DenseNet201 latency with all quantization types and EfficientNetB5 latency with FP16 quantization. Most latency of DL models has a good correlation with hardware utilization. Therefore, hardware utilization is unique to the latency of our DL models. Hao supports these results. J [26] that various DL model executions impact hardware utilization and loading DL model at the first time.

3.6. Comparing Hardware Characteristic Results in Related Research

This research results show that the hardware characteristic appears differently from related research. Because our objective research examines the inference of DL models in allocated memory to reveal the hardware characteristic, we limit the memory and search for the fastest DL model instead of optimizing the DL model to achieve optimal utilization, as Shafi et al. [22] research,

they studied to evaluate the impact of TensorRT on edge devices. However, they did not find the behaviors resulting from TensorRT optimizations. By limiting memory, we found the behaviors that TensorRT needs more times loading in warm-up time, and it has characteristics between the hardware utilization and each latency of the DL model. In addition, we measured 3 types of memory, RSS, PSS, and USS, to understand the details of memory utilization. Still, we did not measure swap memory, which we realize is a weakness of this research. One of the weaknesses of this research is that it does not explain that DenseNet in TensorRT runtime uses low memory. Still, the GPU utilization loads more than other models, with higher memory utilization. Nevertheless, we focused more on other parameters, such as warm-up time, than related works [22–26], to examine the differences in warm-up time and the resulting latency after warming up the deep learning model's inference.

4. CONCLUSION

Evaluation of DL model inference on RPi4B and J. Nano provides valuable insights into the impact of hardware utilization characteristics on their latency. Specifically, the analysis for inference of DL models on RPi4B highlights that memory allocation has a limited effect on latency, with the INT8 quantization models proving to be faster with limited memory. Additionally, the consistent relationship between CPU utilization and latency underscores the importance of efficient hardware utilization in optimizing DL inference. The comparison between model sizes of INT8 and FP16 quantization models reveals intriguing differences, with smaller models like MobileNetV3 demonstrating faster latency, while larger models like EfficientNetB5 may require more processing time. On the other hand, evaluating TensorRT on J. Nano emphasizes the significant role of GPU performance in determining latency, which is particularly evident in DenseNet models. Despite the general expectation that smaller models would perform faster, the results show that the relationship between model size, quantization type, and latency is more nuanced and depends on various factors, such as GPU performance and the specific DL model being used. This highlights the need for further research to develop predictive algorithms that accurately forecast latency based on a comprehensive analysis of all contributing factors. In addition, we suggest future research concerning swap memory to understand more details of memory utilization.

5. ACKNOWLEDGEMENTS

Thanks to the Garuda Research and Excellence (Garuda ACE) program, Universitas AMIKOM Yogyakarta, Dr. In Kee Kim, Ph.D., and Ting Jiang from the University of Georgia for supporting and endeavoring to finish this research.

6. DECLARATIONS

AUTHOR CONTIBUTION

Ahmad Nabhaan is the first author, who contributed to collecting data and organizing the analysis. Rakandhiya Rachmanto is the second author who contributed to creating DL models he was started to join this project when he was pursuing his Undergraduate in Universitas Amikom Yogyakarta. Arief Setyanto is the correspondent author who contributed to supervising this research.

FUNDING STATEMENT

The researcher would like to thank the Garuda Research and Excellence (Garuda ACE) program for funding this research.

COMPETING INTEREST

Further research can be directed to making DL model latency predictions on edge devices (not just the two devices that have been studied) by looking at hardware behavior. Those predictions will be applied to DL schedulers to select devices that meet the desired latency.

REFERENCES

[1] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," *SN Computer Science*, vol. 2, no. 6, p. 420, nov 2021, https://doi.org/10.1007/s42979-021-00815-1.

[2] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, no. 1, p. 53, mar 2021, https://doi.org/10.1186/s40537-021-00444-8.

[3] A. Susanto, C. A. Sari, E. H. Rachmawanto, I. U. W. Mulyono, and N. Mohd Yaacob, "A Comparative Study of Javanese Script

Classification with GoogleNet, DenseNet, ResNet, VGG16 and VGG19," *Scientific Journal of Informatics*, vol. 11, no. 1, pp. 31–40, jan 2024, https://doi.org/10.15294/sji.v11i1.47305.

[4] H. P. Hadi, E. H. Rachmawanto, and R. R. Ali, "Comparison of DenseNet-121 and MobileNet for Coral Reef Classification," *MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, vol. 23, no. 2, pp. 333–342, mar 2024, https://doi.org/10.30812/matrik.v23i2.3683.

[5] D. Saha, M. P. Mangukia, and A. Manickavasagan, "Real-Time Deployment of MobileNetV3 Model in Edge Computing Devices Using RGB Color Images for Varietal Classification of Chickpea," *Applied Sciences*, vol. 13, no. 13, p. 7804, jul 2023, https://doi.org/10.3390/app13137804.

[6] R. Raza, F. Zulfiqar, M. O. Khan, M. Arif, A. Alvi, M. A. Iftikhar, and T. Alam, "Lung-EffNet: Lung cancer classification using EfficientNet from CT-scan images," *Engineering Applications of Artificial Intelligence*, vol. 126, p. 106902, nov 2023, https://doi.org/10.1016/j.engappai.2023.106902.

[7] T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications," *Sensors*, vol. 21, no. 13, p. 4412, jun 2021, https://doi.org/10.3390/s21134412.

[8] J. Lee, L. Mukhanov, A. S. Molahosseini, U. Minhas, Y. Hua, J. Martinez del Rincon, K. Dichev, C.-H. Hong, and H. Vandierendonck, "Resource-Efficient Convolutional Networks: A Survey on Model-, Arithmetic-, and Implementation-Level Techniques," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–36, dec 2023, https://doi.org/10.1145/3587095.

[9] A. Abouaomar, S. Cherkaoui, Z. Mlika, and A. Kobbane, "Resource Provisioning in Edge Computing for Latency-Sensitive Applications," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11 088–11 099, jul 2021, https://doi.org/10.1109/JIOT.2021.3052082.

[10] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, apr 2022, https://doi.org/10.1016/j.jksuci.2021.11.019.

[11] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-Computing-Enabled Smart Cities: A Comprehensive Survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 200–10 232, oct 2020, https://doi.org/10.1109/JIOT.2020.2987070.

[12] A. Garcia-Perez, R. Miñón, A. I. Torre-Bastida, and E. Zulueta-Guerrero, "Analysing Edge Computing Devices for the Deployment of Embedded AI," *Sensors*, vol. 23, no. 23, p. 9495, nov 2023, https://doi.org/10.3390/s23239495.

[13] A. Carvalho, D. Riordan, and J. Walsh, "A Novel Edge Platform Streamlining Connectivity between Modern Edge Devices and the Cloud," *Future Internet*, vol. 16, no. 4, p. 111, mar 2024, https://doi.org/10.3390/fi16040111.

[14] K. Sarvajcz, L. Ari, and J. Menyhart, "AI on the Road: NVIDIA Jetson Nano-Powered Computer Vision-Based System for Real-Time Pedestrian and Priority Sign Detection," *Applied Sciences*, vol. 14, no. 4, p. 1440, feb 2024, https://doi.org/10.3390/app14041440.

[15] S. Park, J. Lee, and H. Kim, "Hardware Resource Analysis in Distributed Training with Edge Devices," *Electronics*, vol. 9, no. 1, p. 28, dec 2019, https://doi.org/10.3390/electronics9010028.

[16] H. Li, Z. Wang, X. Yue, W. Wang, H. Tomiyama, and L. Meng, "An architecture-level analysis on deep learning models for low-impact computations," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 1971–2010, mar 2023, https://doi.org/10.1007/s10462-022-10221-5.

[17] R. P. M. D. Labib, S. Hadi, and P. D. Widayaka, "Low Cost System for Face Mask Detection Based Haar Cascade Classifier Method," *MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, vol. 21, no. 1, pp. 21–30, nov 2021, https://doi.org/10.30812/matrik.v21i1.1187.

[18] J. Maly and R. Saab, "A simple approach for quantizing neural networks," *Applied and Computational Harmonic Analysis*, vol. 66, pp. 138–150, sep 2023, https://doi.org/10.1016/j.acha.2023.04.004.

[19] J. Zhang, Y. Zhou, and R. Saab, "Post-training Quantization for Neural Networks with Provable Guarantees," *SIAM Journal on Mathematics of Data Science*, vol. 5, no. 2, pp. 373–399, 2023, https://doi.org/10.1137/22M1511709.

[20] C. Ji, F. Wu, Z. Zhu, L.-P. Chang, H. Liu, and W. Zhai, "Memory-efficient deep learning inference with incremental weight loading and data layout reorganization on edge systems," *Journal of Systems Architecture*, vol. 118, p. 102183, sep 2021, https://doi.org/10.1016/j.sysarc.2021.102183.

[21] C. Chen, P. Zhang, H. Zhang, J. Dai, Y. Yi, H. Zhang, and Y. Zhang, "Deep Learning on Computational-Resource-Limited Platforms: A Survey," *Mobile Information Systems*, vol. 2020, pp. 1–19, mar 2020, https://doi.org/10.1155/2020/8454327.

[22] O. Shafi, C. Rai, R. Sen, and G. Ananthanarayanan, "Demystifying TensorRT: Characterizing Neural Network Inference Engine on Nvidia Edge Devices," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, nov 2021, pp. 226–237, https://doi.org/10.1109/IISWC53511.2021.00030.

[23] C. Wisultschew, A. Perez, A. Otero, G. Mujica, and J. Portilla, "Characterizing Deep Neural Networks on Edge Computing Systems for Object Classification in 3D Point Clouds," *IEEE Sensors Journal*, vol. 22, no. 17, pp. 17 075–17 089, sep 2022, https://doi.org/10.1109/JSEN.2022.3193060.

[24] P. S.K, S. A. Kesanapalli, and Y. Simmhan, "Characterizing the Performance of Accelerated Jetson Edge Devices for Training Deep Learning Models," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 3, pp. 1–26, dec 2022, https://doi.org/10.1145/3570604.

[25] S. Jing, Q. Bao, P. Wang, X. Tang, and D. Wu, "Characterizing AI Model Inference Applications Running in the SGX Environment," in *2021 IEEE International Conference on Networking, Architecture and Storage (NAS)*. IEEE, oct 2021, pp. 1–4, https://doi.org/10.1109/NAS51552.2021.9605445.

[26] J. Hao, P. Subedi, I. K. Kim, and L. Ramaswamy, "Characterizing Resource Heterogeneity in Edge Devices for Deep Learning Inferences," in *Proceedings of the 2021 on Systems and Network Telemetry and Analytics*. New York: ACM, jun 2020, pp. 21–24, https://doi.org/10.1145/3452411.3464446.

[27] N. James, L.-Y. Ong, and M.-C. Leow, "Exploring Distributed Deep Learning Inference Using Raspberry Pi Spark Cluster," *Future Internet*, vol. 14, no. 8, p. 220, jul 2022, https://doi.org/10.3390/fi14080220.

[28] T. Aboneh, A. Rorissa, R. Srinivasagan, and A. Gemechu, "Computer Vision Framework for Wheat Disease Identification and Classification Using Jetson GPU Infrastructure," *Technologies*, vol. 9, no. 3, p. 47, jul 2021, https://doi.org/10.3390/technologies9030047.

[29] M. A. Wakili, H. A. Shehu, M. H. Sharif, M. H. U. Sharif, A. Umar, H. Kusetogullari, I. F. Ince, and S. Uyaver, "Classification of Breast Cancer Histopathological Images Using DenseNet and Transfer Learning," *Computational Intelligence and Neuroscience*, vol. 2022, pp. 1–31, oct 2022, https://doi.org/10.1155/2022/8904768.

[30] J. Lee, M. Yu, Y. Kwon, and T. Kim, "Quantune: Post-training quantization of convolutional neural networks using extreme gradient boosting for fast deployment," *Future Generation Computer Systems*, vol. 132, pp. 124–135, jul 2022, https://doi.org/10.1016/j.future.2022.02.005.

[31] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson, "Loss aware post-training quantization," *Machine Learning*, vol. 110, no. 11-12, pp. 3245–3262, dec 2021, https://doi.org/10.1007/s10994-021-06053-z.

[32] É. T. Morais, G. A. Barberes, I. V. A. F. Souza, F. G. Leal, J. V. P. Guzzo, and A. L. D. Spigolon, "Pearson Correlation Coefficient Applied to Petroleum System Characterization: The Case Study of Potiguar and Reconcavo Basins, Brazil," *Geosciences*, vol. 13, no. 9, p. 282, sep 2023, https://doi.org/10.3390/geosciences13090282.

[This page intentionally left blank.]