Matrik: Jurnal Manajemen, Teknik Informatika, dan Rekayasa Komputer
Vol. 23, No. 3, July 2024, pp. 567∼580
ISSN: 2476-9843, accredited by Kemenristekdikti, Decree No: 200/M/KPT/2020
DOI: 10.30812/matrik.v23i3.3961

Deep Learning Model Compression Techniques Performance on Edge Devices

Rakandhiya Rachmanto , Ahmad Nabhaan , Arief Setyanto
Universitas AMIKOM, Yogyakarta, Indonesia

Article Info

Article history:

Received March 30, 2024
Revised June 11, 2024
Accepted june 24, 2024

Keywords:

Deep Learning
Edge Devices
Model compression

ABSTRACT

Artificial intelligence at the edge can help solve complex tasks faced by various sec-tors, such as automotive, healthcare, and surveillance. However, challenged by the lack of computational power from the edge devices, artificial intelligence models are forced to adapt. Many have developed and quantified model compression approaches to tackle this problem over the years. However, not many have considered the overhead of on-device model compression, even though model compression can take a considerable amount of time. With the added metric, we provided a more complete view of the efficiency of model compression on the edge. **This research aimed** to identify the benefit of compression methods and their tradeoff between size and latency reduction versus accuracy loss and compression time in edge devices. **In this work, the quantitative method** was used to analyze and rank three common ways of model compression: post-training quantization, unstructured pruning, and knowledge distillation based on accuracy, latency, model size, and time to compress overhead. **We concluded that knowledge** distillation is the best, with a potential of up to 11.4x model size reduction and 78.67% latency speed up, with moderate accuracy and compression time loss.

Corresponding Author:

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

How to Cite:
R. Rachmanto, A. Nabhaan, and A. Setyanto, "Deep Learning Model Compression Techniques Performance on Edge Devices ", *MATRIK: Jurnal Manajemen, Teknik Informatika, dan Rekayasa Komputer*, Vol. 23, No. 3, pp. 567-580, July, 2024.

1. INTRODUCTION

Deep learning (DL) advancements have fueled its adoption across diverse sectors [1]. Artificial intelligence can be em-bedded in wearable devices to monitor or diagnoseients [2, 3]. In terms of surveillance, smart home, and autonomous driving, real-time data from cameras and sensors can be utilized to predict behaviors to prevent crime or avoid accidents [4–6]. Manufacturing plants benefit from intelligent monitoring systems capable of predicting anomalies and providing optimizations to the production process. Many tasks within these sectors are characterized by the demand for high reliability, responsiveness, and security [7]. To lessen reliance on dedicated cloud servers due to the risk of latency instability, these tasks are executed by small, low-powered boards such as the Jetson modules [8, 9]. These boards are positioned within the same network as the input source (e.g., sensors) to ensure swift feedback. This paradigm, known as edge artificial intelligence (edge AI), is challenged by the constrained specifications of edge devices [4, 10]. A more complex task might demand larger models or simultaneous model executions [11], potentially impeding the inference performance and highlighting the need to adapt to this condition. Runtime optimization toolkits such as TensorRT [12], OpenVINO [13, 14], or Apache TVM [14], and accelerators like EdgeTPU [15] serve to enhance existing hardwares capability. Moreover, AI models can also adapt through compression. Model compression aims to lessen computational burden by reducing a models complexity while minimizing the risk of accuracy loss. Over the years, numerous model compression methodologies, e.g., quantization, pruning, and distillation, have been proposed and improved [16, 17], followed by pipelines combining multiple compressions [18–20].

Many have tried to quantify the performance of model compressions. Deep learning models such as Densenet -121 and MobileNet are useful in many cases, such as the classification of coral reefs [21]. However, they are usually large and work well in higher-end processors with huge memory. Ahn et al. tested the performance of quantized MobileNetV2, Dense-Net121, and VGG16 on Raspberry Pi 4B and two HPC platforms, TACC Frontera and RI2, from The Ohio State University. They noted that quantization could reduce the model size to half or even a quarter of the original model and that accuracy suffers a dip when choosing INT8-SQ (called INT8 in this paper), most noticeably in DenseNet121. They reported that hardware support is another important factor in latency speed-up, as the two HPC platforms report different speed-up levels due to different Intel CPU generations. DeepEdgeBench [22] analyzed the performance of MobileNetV1 and MobileNetV2 configurations on accuracy, latency, and power consumption. Minimizing power consumption is also an important factor reported on edge machine learning [23]. Using a multitude of edge devices, such as Jetson Nano, Coral Dev Board, and Arduino, they noted a 0.4% - 0.6% accuracy drop on quantized models. They hinted that latency affects device specifications and runtime (TensorFlow vs. TF Lite vs. TensorRT). Pruning experiments conducted by Jin et al. in 2022 [24] suggest that fine-grained pruning (in this paper as unstructured pruning) can be effective in reducing a models size and number of operations (FLOPS) without affecting accuracy, with ResNet50 and DenseNet121 still capable of more than 90% Top-5 accuracy after pruning. Knowledge distillation is first discussed by Geoffrey Hinton, who achieved great results in MNIST and speech recognition cases. In one MNIST experiment, the distilled model can predict 98.6% of 3s correctly, even though the number 3s was omitted in the training data. Over the years, improvements have been made, such as Contrastive Representation Distillation [25] or distilling through function matching and consistent teaching [26].

In this work, we studied the effects of three common compression techniques: 1) post-training quantization (PTQ), 2) unstructured pruning, and 3) knowledge distillation (KD). Each methods accuracy, latency, model size, and compression time are quantified and analyzed. Two edge devices, Jetson Nano and Jetson Xavier NX, are used to commence the model compression and inference. Our novelty is on-device compression execution, which the mentioned papers have not addressed. On-device compression is vital for many reasons. Without sending models and data to the cloud, there are fewer data breach risks for use cases such as surveillance and healthcare [4]. Frequent model updates can also be done without relying on the cloud, ensuring quicker and more customizable model adaptability when needed. With the added metric of compression time, this research studies each methods advantages and hindrances and ranks which of the three compression mechanisms is worth the time cost, given its other effects in model size reduction, latency improvement, and loss of accuracy. These will add to the discussions of model compressions in general and showcase the capabilities of current common ways of optimizing edge AI.

The rest of this paper is organized as follows: Section 2 describes the research methodology. This section provides information about the metrics, edge devices, open-source libraries, and DL models used in this work. Furthermore, this section ex-plains the three tested compression methods and experiment setups. The results of the experiments are presented in Section 3, with each metric having its own subsections. At the end of section 3, there are comparisons to previous studies. Finally, Section 4 concludes the work, offering future research opportunities and areas of improvement.

2. RESEARCH METHOD

This section details our approach, metrics, and experiment setup. **The approach and metrics** subsection explains the four parameters measured in this research, followed by edge device specifications in the next subsection. The Subsection 2.3 outlines a discussion about the TensorFlow framework and other libraries, along with Deep Learning models being used in this research. The three compression techniques are discussed in Subsection 2.4, followed by the experiment setup and scenario in the SubSection **Benchmark Procedure**.

2.1. Approach and Metrics

We used the quantitative method and took four metrics: Top-1 accuracy, latency, model size, and compression time. For top-1 accuracy and compression time, measurement is done five times, and the average value is displayed for each model-compression method combination. Latency measurement is done eleven times, with the first measurement considered warm-up and, thus, discarded, the remaining ten to be averaged and displayed.

2.2. Edge Devices

As depicted in Table 1, it shows detailed specifications of edge devices. The NVIDIA-developed Jetson Nano (Nano) [8] and Jetson Xavier NX (Xavier) [27] are utilized in this research. Both edge devices are equipped with GPU and software sup-port from NVIDIA, allowing users to easily run artificial intelligence (AI) tasks on such small form factor and power draw. Jetson Nano has a quad-core ARM Cortex-A57 CPU, a 128-core NVIDIA Maxwell GPU, and 4GB LPDDR4 Shared memory. Jetson Xavier is more powerful, with a six-core Armel CPU capable of a 1.9 GHz max frequency, 384-core NVIDIA Volta GPU, and a shared 8GB LPDDR4 memory. Both devices used their best power modes, drawing 10W and 20W on Nano and Xavier, respectively, and the jetson_clocks utility is turned on to ensure all devices operate at their maximum capability dur-ing testing.

Table 1. Edge device specifications

	Jetson Nano	**Jetson Xavier NX**
CPU	4C ARM Cortex-A57 CPU @ 1.5 GHz	6C NVIDIA Armel @ 1.9 GHz
GPU	128-core NVIDIA Maxwell @ 921 MHz	384-core NVIDIA Volta, 48 Tensor cores @ 1100 MHz
Memory	4GB LPDDR4 Shared	8GB LPDDR4X Shared
OS	Ubuntu 18.04	Ubuntu 20.04
JetPack	4.6.1	5.1
Python	3.6.9	3.8

At the OS level, both devices use Ubuntu, albeit with different versions. Jetson Xavier NX, as the more powerful device, gets to run Ubuntu 20.04 instead of the Ubuntu 18.04 of Nano. On top of the OS, these devices are configured to use JetPack SDK, a Jetson-specific SDK configuring AI libraries such as cuDNN, CUDA, and TensorRT. The latest SDK version for Jetson Nano is v4.6.4, with cuDNN v8.2.1 and CUDA 10.2. JetPack v5.1, utilized by Xavier, has cuDNN v8.6 and CUDA 11.4. Not only that, the Python versions of these devices are also different, with Jetson Xavier NX using the newer Python 3.8.

2.3. Deep Learning (DL) Framework, Libraries and Models

Numerous popular open-source frameworks and libraries were utilized in this work. TensorFlow, a DL framework developed by Google, has been chosen as the base. TensorFlow trains the base models doing inference, pruning, and distillation tasks. TensorFlow Lite, a lighter version of TensorFlow, focused on execution on mobile and edge devices (TF Lite), was used during the quantization process and also for inference of the quantized model. Lastly, the TensorFlow Model Optimization Toolkit (tfmot) was used to assist with pruning.

Image classification is the backbone of many computer vision tasks running at the edge. Thus, three image classification families: 1) MobileNetV3, 2) EfficientNet [25], and 3) DenseNet [28], are chosen as the base models. The models speci-fications are shown in Table 2. We examined these models on the oxford_flowers102 (later referred to as just flower) da-taset [29], which has 102 data classes of flowers, ranging from 40 to 258 images in each class. It has a total of 1020 training images, 1020 validation images, and 6149 test images.

Table 2. Pembagian data untuk Training dan Testing

	Input Size	Model Size (MB)
MobileNetV3Small (Mob-S)	224 × 224	7.5
MobileNetV3Large	224 × 224	17
EfficientNetB1(Eff-B1)	224 × 224	34
EfficientNetB3(Eff-B3)	224 × 224	51
DenseNet169 (D169)	224 × 224	57
DenseNet201 (D201)	224 × 224	80

Released in 2019, models in the MobileNetV3 family, such as MobileNetV3Small and MobileNetV3Large, are small, efficient models perfect for inference tasks at the edge. EfficientNet offers many options, from B0 to B7 configuration, with increasing parameters and layers, ensuring availability across all needs. EfficientNetB1 and EfficientNetB3 are chosen as the medium-sized models. Then, with a model size of 57 MB and 80 MB, DenseNet169 and DenseNet201 are the two heaviest models being tested in this paper.

2.4. Compression Techniques

We assessed three model compression techniques: 1) post-training quantization (PTQ), 2) unstructured pruning, and 3) knowledge distillation (KD). These popular and widely supported compression methods offer various advantages and tradeoffs.

PTQ is a widely supported compression method that is a subclass of compression through quantization and quantization-aware training (QAT) [16]. Quantization achieves compression by lowering the bit representation of a DL models weights and activation values. Typical examples include lowering a model from the original 32-bit floating points (FP32) representation to 16-bit floating points or 8-bit integers. PTQ differs from QAT in terms of when the quantization is executed. PTQ quantizes a model after training, whereas QAT is a quantization method that runs during training. PTQ is generally easier to implement because it does not require an additional training process and only requires three parameters: the model itself, target precision mode (e.g., FP16, INT8), and a small subset of training or validation data as calibration data if choosing the INT8 option. Depending on the chosen precision mode, the quantization process can vary. A 16-bit floating point structure consists of 1 sign bit, 5 exponent bits, and 10 significand (mantissa) bits. During quantization, each FP32 parameter is adjusted to FP16 by retaining the sign bit, and then the exponent and mantissa of FP32 parameters are either truncated or rounded to comply with FP16s structure. This way, the overall FP16 representation can be as close as possible to the FP32 values. INT8 quantization is more complicated as it has an even narrower value range, only -128 to +127. To quantize to 8-bit integers, an additional step called calibration is done to accurately map the vast range of FP32 values to the constraints of 8-bit integers. The mapping process depends on the scaling factor and zero point. The value of the zero point is fixed to the value 0 in FP32, ensuring symmetrical negative and positive number representation for the mapping process. The calibration data values determine the scaling factor; with better and more varied calibration data, the scaling factor can be more accurate. After determining the zero point and scaling factor, each FP32 value is mapped to INT8. The zero point and scaling factor are also used during inference to match the INT8 representation with the FP32 original value.

Pruning refers to a compression method where a models complexity is reduced by zeroing the insignificant parts [16]. A threshold value is utilized to identify these insignificant parts, which are subsequently zeroed out. Unstructured pruning, in particular, targets individual connections, allowing greater control over the degree of pruning in a model. This technique intro-duces a parameter known as target sparsity, representing the proportion of a models zeroed connections (e.g., 50% sparsity means 50% of the models connections are zeroes). In TensorFlow, pruning happens iteratively within a training stage. After preparing the pruning parameters (algorithm, target sparsity), the training process begins with the pruning parameters acting as callbacks. Following the forward and backward pass, the model weights are updated. Each of these weights is then evaluated by the pruning algorithm, zeroing those not meeting the criteria. Lastly, the weights are updated a second time to reflect the results from the pruning algorithm before moving on to the next forward pass.

Knowledge distillation, often referred to as the Student-Teacher network, is a technique introduced by Geoffrey Hinton [30] to compress large, complex deep learning models. In this process, a smaller model, known as the student, is trained to mimic the predictions and learned representations of a larger model, the teacher. The teacher model generates soft targets or predictions from the input data during distillation. The student model then undergoes a forward pass to make its predictions. The discrepancy between the students predictions and the softened predictions of the teacher is quantified through three loss functions: the student loss, the teacher loss, and the distillation loss. The student loss measures the difference between the stu-dents predictions and the ground truth labels, while the teacher loss quantifies the difference between the students predictions and the softened predictions of the teacher. These two

losses are combined to form the distillation loss, which balances the contributions of the students own predictions and the knowledge transferred from the teacher. The distillation loss will guide the students backward pass to adjust its parameters. Two hyperparameters can be adjusted to refine the distillation process: alpha (α) and temperature (T). Alpha determines the balance between the student loss and the teacher loss, affecting how specified or generalized the student model will be. Temperature determines the softening of the teacher models prediction, impacting how closely the student model is following its teachers output. Careful balancing of these two parameters is key to a good knowledge distillation process.

2.5. Benchmark Procedure

We first trained every listed DL model to the oxford_flowers102 dataset using transfer learning as a preliminary step. The oxford_flowers102 dataset is split into training, validation, and testing. These images are resized into 224 $\times$ 224 pixels and normalized accordingly to the tested models requirement. Lastly, these images are then split into batches of 4. Models are trained for 10 epochs and saved to the disk. This process is repeated five times, and each saved model is labeled, e.g., MobileNetV3Large-1, MobileNetV3Large-2, etc.

Benchmarking is separated into two parts. First is compression benchmarking, where we test compression configuration and model combinations. Each previously listed DL model will be compressed into: FP16 PTQ, INT8 PTQ, 25% - 50% - 75% sparsity rate for unstructured pruning, and lastly, KD, of which Mob-S as the smallest model is chosen to be the student model. The metric of compression time (overhead) is measured in this part, and measurement for every DL model compression configuration is repeated five times and averaged to ensure result similarity and reproducibility. Similar to the preliminary step, every compressed model is saved to the disk and labeled accordingly, e.g., INT8-DenseNet169-1.tflite.

Each compression method has its own parameters. Hence, the compression benchmarking is separated into three Python scripts, catering to distinct requirements, as shown in Figure 1. For PTQ with TensorFlow Lite, the benchmarker accepts the location of the original model, chosen precision, and calibration data if needed. Then, the quantization process is started utiliz-ing those parameters. To do pruning benchmarking, we first provide the model with the location of the base model, with train-ing and validation set to use during the iterative pruning process and the chosen pruning algorithm and sparsity level. We chose the algorithm ConstantSparsity that ensures the target sparsity of 25%, 50%, or 75% is enforced throughout the pruning stage. After loading the model and training data, the model goes through the pruning process for five epochs. Knowledge distillation requires the location of the teacher model, with training data requirements similar to pruning, where the flower datasets training set and validation set are used. We decided to set the temperature value to 10 and alpha to 0.1 to simplify execution. When the distillation benchmarker is first started, the teacher model will be loaded into memory, and then the student model (MobileNetV3Small) will be set up. Distillation will then commence for five epochs.

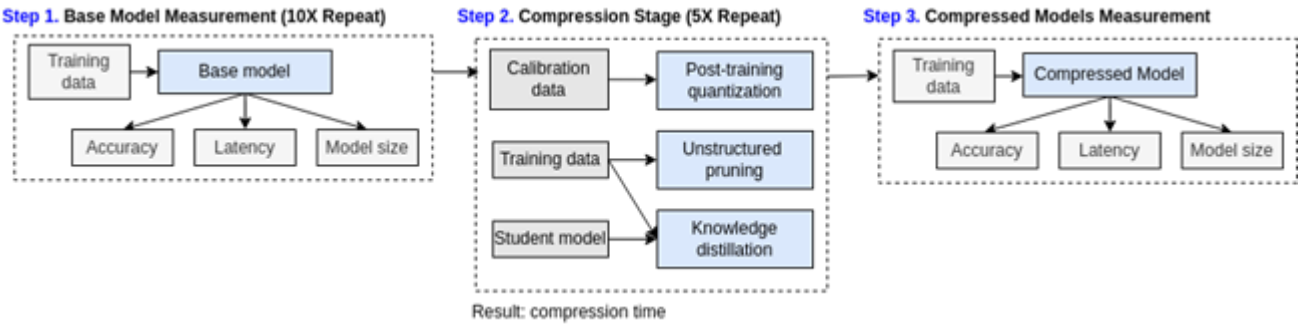


Figure 1. The general flow of model compression

The second part is model benchmarking, where the accuracy, latency, and model size of the base (original) and com-pressed models are measured. Accuracy measurement is done by running each model against the test set of the flower dataset. This step is repeated five times until every labeled model on the disk is tested. Latency is tested by taking each configurations number-1 labeled model, for instance, EfficientNetB1-1 for base or 25%-EfficientNetB1-1 for the pruned, and giving a task of 1000 image inference. The latency formula is depicted in Equation (1). It is calculated by taking the whole duration of the inference task in seconds, and this duration is then converted into milliseconds (ms) by multiplying it by 1000, then divided by the number of images, in this case, also 1000. This measurement is done 11 times, of which the first one is considered warm-up and, thus, discarded. The remaining ten

will be noted and averaged. The model size of the base and compressed models across runs (i.e., same configuration, different labels) are identical; therefore, they are only noted once.

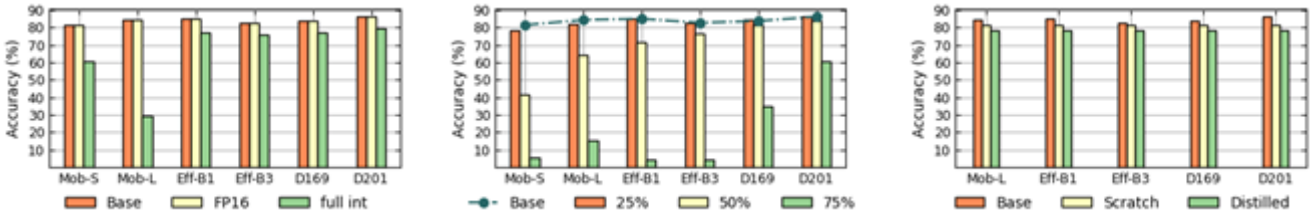
$$Latency = \frac{Totalexecutiontime * 1000}{numberofimages} \quad (1)$$

3. RESULT AND ANALYSIS

This section showcases the results of our experiments. In the Model Accuracy subsection, we analyze the model accura-cies of the three compression techniques. Then, the Latency subsection discusses a comparison of execution time followed by the size of the model prior to and after compression analysis in the Model Size subsection. We compare the base models in the three previously mentioned sections. Next, we showcased the overhead (compression time) results in the next subsection. We discuss these results about previous works in the Discussion subsection.

3.1. Model Accuracy

As shown in Figure 2, with only 0.003% accuracy loss on average, FP16 conversion has the least accuracy drop between all methods. INT8 quantized models have varying levels of accuracy loss, with the two smallest models, Mob-S and Mob-L, having the largest drops. INT8 Mob-S is 20.93% less accurate, while Mob-L has 29.16% accuracy. EfficientNet and DenseNet INT8 quantized models drop moderately between 6.44% - 7.55% accuracy loss. The models precision mode causes this accu-racy discrepancy. With a limited value range, INT8 quantization risks erasing important information from the base FP32 model. INT8 quantization also depends on model robustness. As stated in Section 2.5, we only provide these networks with non-augmented images for 10 epochs to train our base models. Aside from that, 128 images are prepared as calibration data for quantization. The model could be more robust if the training and calibration images were better prepared. Therefore, the risk of accuracy loss can be lessened.



PTQ accuracy Pruned models accuracy Distilled models accuracy

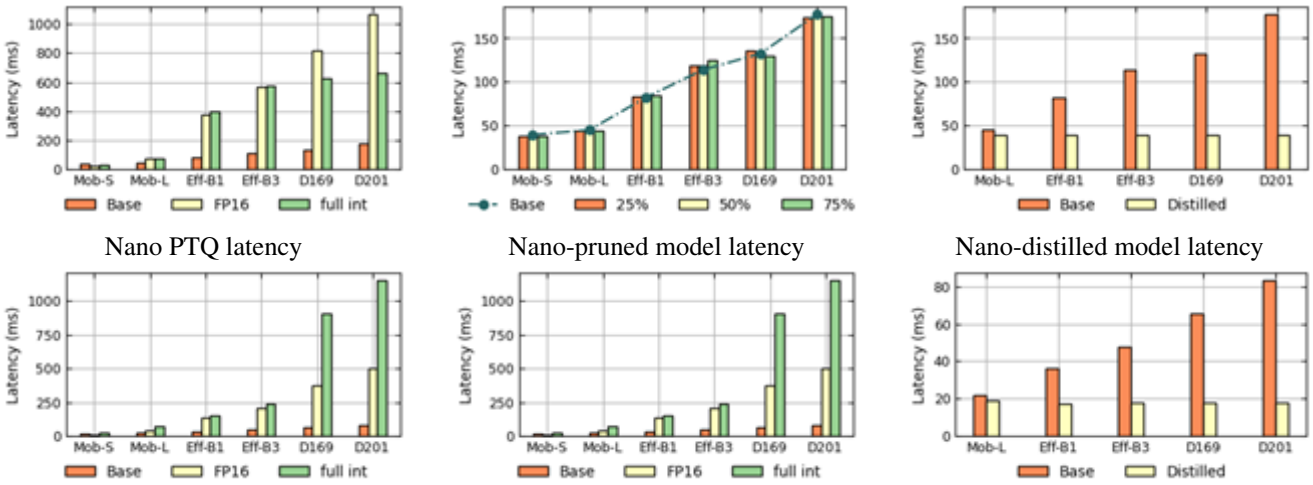
Figure 2. Model accuracies

Unstructured pruning achieves compression by zeroing connections, risking accuracy drop, as seen in Figure 2. While 25% sparse models are still generally accurate at 3.15% maximum accuracy loss, the same cannot be said for higher sparsity levels. On 50% sparsity, only the three biggest models, D201, D169, and Eff-B3, can handle the sparsity gracefully, with 2.06% - 6.26% accuracy loss. Eff-B1 suffers a 13.44% accuracy drop, followed by Mob-L and Mob-S at 20.3% and 39.5% accuracy loss, respectively. On 75% sparsity, all models took a major accuracy drop. Eff-B1 and Eff-B3 fall below 5% accuracy, at only 4.14% and 4.56%. Mob-S and Mob-L are slightly better, with 5.41% and 15.4% accuracy. D169 is only 35.08 accurate, while D201 still manages 60.47% accuracy. We observed that the models number of parameters might correlate with sparsity resilience. In 25% and 50% sparsity, the bigger the model is (in turn, the models size is also bigger), the better they respond to the sparsity level. However, model architecture might also affect this. Eff-B1 and Eff-B3, the medium-sized models, have worse accuracy than the small ones, Mob-S and Mob-L, on 75% sparsity.

For distillation, we have two comparison points: base and scratch. Base refers to the uncompressed model, while scratch refers to the student model (Mob-S) but trained from scratch instead of distilled. After 5 epochs of distillation, we found that distilled models are consistent across the board, with accuracy ranging from 78.21% to 78.5%. The scratch model is 81.36% accurate, making the difference between scratch and distilled model 2.99%. These distilled models are 4.2% to 7.9% less accu-rate than their base equivalent.

3.2. Latency

Figure 3 shows the recorded latencies of compressed models. Base latency is measured on GPU, which is also shown in each graph. Except for Mob-S, TF Lite Quantization affects latency negatively, with every quantized model being slower. On Nano, quantized Mob-S models are 32.15% - 45.04% faster than its base. On Xavier, the FP16 quantized model is 45.45% faster, while the INT8 quantized model is 9.5% slower than the base. This latency increase is very obvious, especially in larger models such as D169 and D201. On Nano, quantized D169 and D201 are 3.73x - 6.2x slower. On Xavier, these models are 5.75x - 13.82x slower than the base. This latency increase is because TF Lite can only utilize CPU, unlike the pruned or dis-tilled models that run on GPU. Interestingly, on Nano, FP16 is slower for larger models, but on Xavier, INT8 is slower. Indi-cating hardware differences that can be further investigated.



Nano PTQ latency | Nano-pruned model latency | Nano-distilled model latency

Xavier PTQ latency | Xavier pruned model latency | Xavier distilled model latency

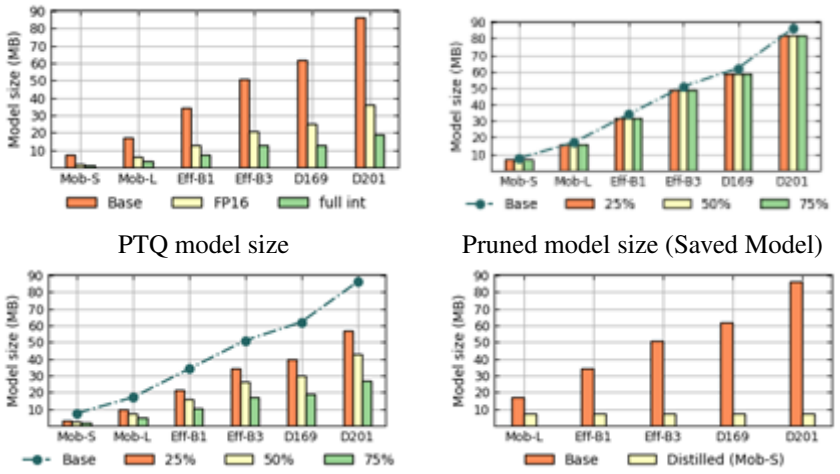
Figure 3. Model latencies

Generally, unstructured pruning does not improve latency because it happens on connections instead of specific parts such as convolution channels. Pruning on connections causes irregularities that make it harder to speed up latency. As seen in Figure 3, there are insignificant latency differences and inconsistencies between sparsity levels and which sparsity level is fast-er. On Nano, the inference is the fastest, with 75% sparsity for pruned Mob-S, at 38.3 ms; other sparsity levels are 38.4 ms and 38.5 ms. However, for Eff-B1, pruning with 25% sparsity has the best latency at 83.7 ms. Compared to the original model, some pruned models are even slightly slower. Pruned Eff-B3 models make inferences at the speed of 118.5 ms - 125.1 ms, whereas the base models inference speed is 113.9 ms. The only instance of pruned models having a noticeable speed-up is Eff-B3 on Xavier. The sparse models infer images at the speed of 34.7 - 34.9 ms, compared to 47.9 ms if unpruned, resulting in a close to 28% reduction in latency. This uncommon speed-up might be because of hardware differences, as Xavier has a newer GPU generation (See Subsection 2.2) equipped with Tensor cores or differences in library versions. As newer hardware, Xavier is also running on newer Ubuntu and AI-specific libraries such as CUDA, cuDNN, and can utilize TensorFlow v2.12 (instead of the older v2.7 used in Nano). Nevertheless, it is unclear why this speed-up only happens to Eff-B3 and no other models.

For knowledge distillation, use Mob-S as the student or distilled model. As such, the distilled model latency is identical to Mob-Ss latency on the respective devices, 39.34 ms on Nano and 17.91 ms on Xavier. We noted that the latency speed is more apparent as the teacher gets larger. Mob-S is approximately 12.7% - 13.64% faster than the smallest teacher, Mob-L. In contrast to Eff-B1, the student is 52% faster. The distilled model is faster than D169 by 69.9% and 72.87%. D201, as the big-gest teacher tested, has 177.3 ms latency on Nano and 83.2 ms on Xavier. For D201, the latency improvement is even higher, as the student is 77.67% faster on Nano and 78.67% faster on Xavier.

3.3. Model Size

Figure 4 denotes the model sizes of compressed models. Both quantizations have a large effect on model size. On aver-age, FP16 managed to reduce the model size by 2.75x. Mob-S and Mob-L reap the most benefits; with FP16, their model size is reduced to 2MB and 6MB, compared to 7.5MB and 17 MB. Eff-B1 to D201 experienced a reduction of 2.38x to 2.61x in model size. 8-bit integers only occupy 1 byte of memory, compared to the 2-4 bytes consumed by FP16 and FP32. Hence, INT8 also has more impact than FP16 regarding model size reduction. On average, their model size is 4.72x smaller than the base. Mob-S and Mob-L are again the biggest benefactors for this compression, as their INT8 models are 5.77x and 4.86x smaller than the original. Eff-B3 has the lowest INT8 model reduction rate at only 3.92x, and its model size decreased from 51MB to 13MB. At the same time, other models achieved 4.47x - 4.76x size reduction.



PTQ model size

Pruned model size (Saved Model)

Pruned model size (gzipped)

Distilled model size

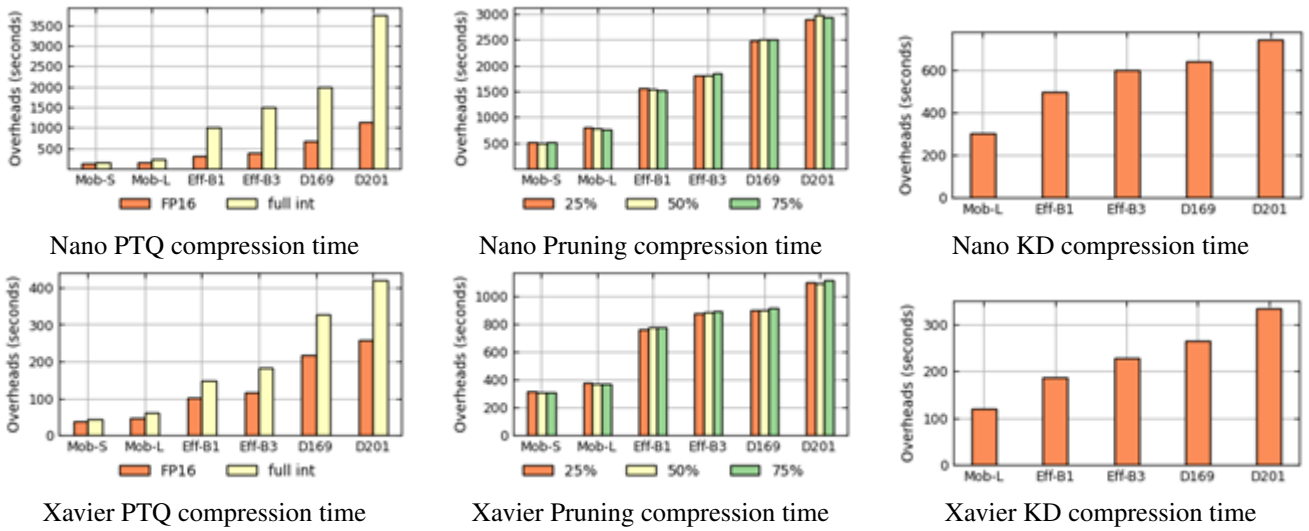
Figure 4. Model size comparison

The effects of pruning on model size are not immediately apparent. Figure 4 depicts the SavedModel size comparison be-tween pruned models and their base. When stored as SavedModel, Mob-S only experienced a 0.9MB reduction, while D201s sparse models are merely 4MB smaller than its original. Another observation is that the model sizes of sparse models are iden-tical, unaffected by the sparsity level. Unstructured pruning only zeroized the weight instead of removing it entirely or lowering its precision as PTQ. The zeroized (pruned) and the non-zero weights still use the FP32 precision, making the model size reduction marginal. However, the reduction is seen after gzip compression. After gzip, the 25% sparse models are 1.68x small-er than the base, 50% and 75% sparse models are even smaller, their size reduced to be 2.2x and 3.36x smaller. Gzip can re-duce the weights of pruned models by identifying the redundant zeroes and representing them more efficiently, thus reducing file size further.

We designated Mob-S (7.5MB), the smallest model in this work, the student distillation model. As such, each teachers model reduction rate differs (Mob-L to D201). Mob-L, the smallest teacher, benefitted the least with only a 2.27x size reduc-tion. Unlike Eff-B1, Eff-B3, and D169, distilling with Mob-S can achieve a 4.5x - 8.27x size reduction. D201 benefitted the most as the largest teacher, with an 11.4x size reduction.

3.4. Compression Time (Overhead)

While compression time goes proportionally with the model size, FP16 quantization is the fastest to do in PTQ compared to INT8 for both devices, as pictured in Figure 5. For Mob-S on Nano, FP16 quantization is 19.8% faster than INT8. For D201 quantization on Nano, FP16 is 69.87% faster. Due to better hardware specifications, Mob-S FP16 quantization on Xavier is faster by 8.6% compared to Nano. Meanwhile, the D201 FP16 quantization is faster by 38.3%. This is due to the calibration process that INT8 needs to do to properly map the 8-bit values to the range of the original weights.



Nano PTQ compression time | Nano Pruning compression time | Nano KD compression time

Xavier PTQ compression time | Xavier Pruning compression time | Xavier KD compression time

Figure 5. Compression time for all methods and device

Except for D201 INT8 quantization, pruning is generally the most time-consuming compression method on both devices. In contrast to quantization, pruning is much slower to do, especially compared to FP16 quantization, by 2.57x - 5.36x on Nano and by 4.15x - 7.95x on Xavier. The gap is more apparent in smaller models such as Mob-S, where FP16 quantization took 121.74 seconds, 4.24x faster than pruning for 25% sparsity, which took 516.32 seconds. Pruning is more time-consuming as they have a little computation overhead compared to regular training or distillation. TensorFlow, through the TensorFlow Model Optimization Toolkit, does pruning iteratively. As such, there are computation overheads such as determining which weights to prune and restructuring the matrices to be sparse matrices, aside from regular forward and backward propagation. The larger the model and the more complicated the pruning algorithm, the more time it takes to prune it.

Knowledge distillation is faster than pruning, even though both are executed for 5 epochs. The distillation algorithm uti-lized in this study is completed in just 24.62% - 37.97% of the time required for pruning. In contrast to FP16 PTQ on Nano, distillation is slower to complete by 34.17% - 50.86% for Mob-L to Eff-B3. However, for D169 and D201, distillation is com-pleted faster by 4% and 51%, respectively. Versus INT8 quantization, the difference is much more pronounced, especially on D169, D201, of which distillation on Nano is 3.1x and 5.02x faster to do. On Xavier, quantizing all teacher models to FP16 is faster than distilling them. Mob-L, Eff-B1, and Eff-B3 INT8 quantization is also faster than distilling. D169 and D201, on the other hand, are faster by 23.3% and 25.9%.

3.5. Discussion

PTQ proved effective in reducing a models size, with FP16 and INT8 able to compress models to 2.75x and 4.72x small-er than originally, on average. Previous research by Ahn et al. through different benchmark suites (MLPerf) reported simi-lar findings for model size reduction, in which their INT8-SQ (INT8 in this paper) is smaller than FP32 and FP16. We observed that TF Lite PTQ affects latency negatively, with larger models such as D201 on Xavier having their inference 13x slower. noted differently, as they experienced speed-ups in their edge device, but inference slowed down on their TACC HPC platform. We found that FP16 can keep up with the original model, with less than 1% loss of accuracy on average. INT8 quantized models do not have viable accuracy, particularly for the small Mob-S and Mob-L models. Overhead-wise, PTQ is one of the fastest methods to compress a model because it does not need a training stage.

Similar to the work [24], we noted that unstructured pruning can reduce a models size. However, we found that the re-duction happens only after gzip compression. As the zeroized weights and normal weights are still stored as 32-bit floating points on memory, the difference between pruned and unpruned model size on the Saved Model format is negligible. Unlike [24], we found that higher sparsity levels, such as 75%, affect accuracy negatively, with models such as Eff-B1 and Eff-B3 having less than 5% accuracy. We also found the latency of pruned models to be inconsistent across sparsities, with some 25% of sparsities being faster than bigger ones or even sparse models being slower than their originals. The compression time of unstructured pruning on TensorFlow is considerably larger than the other methods due to it happening iteratively via a training stage and doing more intensive

operations, such as determining which connections to prune and restructuring the matrices to be sparse matrices, other than forward and backward propagation. With our accuracy results, the time commitment for this method can be longer, as fine-tuning might be required to restore the models accuracy.

Distillation is the most effective in reducing a models size. 2.27x size reduction is achieved on Mob-L, our smallest teacher model, not very far from FP16 compressions model size reduction rate. However, on larger teachers such as D201, distillation can reduce the model size to roughly 11x smaller, going from 83.2 MB originally to only 17.75 MB. Similarly, with early distillation works by Geoffrey Hinton, we found distillation to be great at retaining accuracy from larger models. The smaller student model (Mob-S) is only 4.2% - 7.9% less accurate than the original models, even at a few epochs. It also brings benefits in terms of latency speed up, as the student model is much more suited for edge environments with latencies of 49 ms and 19.5 ms on Nano and Xavier, respectively. Distillation overheads sit between PTQ (the fastest) and unstructured pruning (the slowest).

Therefore, we ranked distillation as the best method for on-device compression out of the three tested in this paper. KD has a great model size reduction rate latency speed up, with rather moderate accuracy loss and overhead as tradeoffs. PTQ is second, with consistent model size reduction and being less time-consuming. However, quantized models are slower as they are run on the CPU. FP16 has the least amount of accuracy loss, but the same cannot be said for INT8, where the models risk a considerable decrease if not trained or calibrated properly. Unstructured pruning in TensorFlow is not recommended, as model size reduction is not felt immediately but through additional gzip compression. The higher degree of sparsity continually decreases accuracy with inconsistent or no latency benefit while being the most time-consuming method.

4. CONCLUSION

This work studied the effects of PTQ, unstructured pruning, and KD, three popular DL model compression strategies. These techniques achieved compression through different means, with distinct advantages and tradeoffs. Compressions were conducted in Jetson Nano and Jetson Xavier NX, two low-powered AI-capable edge devices. We examined the effects of compression on six image classification models of varying sizes, from small models like MobileNetV3Small and MobileNetV3Large to larger ones such as DenseNet201. Further, we also explored configurations available in PTQ and un-structured pruning in the form of precision modes (FP16, INT8) and sparsity levels, e.g., 25%, 50%, and 75%, which can im-pact the resulting models accuracy, latency, model size, and compression time. These measurements identified compression benefits, such as size and latency reduction. We also identified challenges, such as accuracy loss and long compression time, for KD and unstructured pruning, two of the more complex compression strategies tested in this paper. We concluded that KD is the best out of the three tested compression methods, with a balance of size and latency reduction, without sacrificing too much accuracy and being too time-consuming.

Although this studys results corroborate with previous works findings regarding model size reduction, our study reveals discrepancies in latency and accuracy results. Accuracy differences may be the result of the base models configuration, i.e., the training process or specific code implementations for the compression process. Similarly, variations in latency results could be influenced by hardware capability, of which newer or more complex hardware might contain sets of instructions that can better leverage the compression. Additionally, the DL framework of choice (i.e., TensorFlow, PyTorch) might utilize the hardware differently, resulting in different levels of latency benefits during inference. These nuances are important considera-tions for devising a more optimal model compression strategy for model size and other aspects. Hence, our study confirms the efficacy of model compression in answering the problem of adapting AI models to edge devices and complements the discus-sion of model compressions efficiency by introducing another key factor to consider for a tradeoff: the overhead (compression time). More importantly, this work highlights the need for further research into different approaches considering the diverse interactions between hardware, software, and algorithms, ensuring more optimal compressed model performance across multiple deployment configurations.

5. ACKNOWLEDGEMENTS

The authors would like to thank Dr. In Kee Kim and Ting Jiang of The University of Georgia for their assistance, provid-ing assets and valuable opinions over the course of this research.

6. DECLARATIONS

AUTHOR CONTIBUTION

Rakandhiya Rachmanto carried out the experiments and analysis and took the lead in writing the manuscript. Ahmad Nabhaan helped set up and maintain the Jetson Nano Edge device. Arief Setyanto, the correspondent author, supervised the project. All

authors contributed feedback and ideas during the writing and planning of this project.

FUNDING STATEMENT

This research is funded by the Garuda Academic of Excellence (Garuda ACE) program.

COMPETING INTEREST

The authors declare no conflict of interest.

REFERENCES

[1] D. Liu, H. Kong, X. Luo, W. Liu, and R. Subramaniam, "Bringing AI to edge: From deep learnings perspective," *Neurocomputing*, vol. 485, no. 2, pp. 297–320, May 2022, https://doi.org/10.1016/j.neucom.2021.04.141. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231221016428

[2] S. A. Alowais, S. S. Alghamdi, N. Alsuhebany, T. Alqahtani, A. I. Alshaya, S. N. Almohareb, A. Aldairem, M. Alrashed, K. Bin Saleh, H. A. Badreldin, M. S. Al Yami, S. Al Harbi, and A. M. Albekairy, "Revolutionizing healthcare: the role of artificial intelligence in clinical practice," *BMC Medical Education*, vol. 23, no. 1, pp. 689–699, Sep. 2023, https://doi.org/10.1186/s12909-023-04698-z. [Online]. Available: https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-023-04698-z

[3] D. R. Pulimamidi and G. P. Buddha, "The Future of Healthcare: Artificial Intelligence 's Role In Smart Hospitals And Wearable Health Devices," *Tui Jishu/Journal of Propulsion Technology*, vol. 44, no. 5, pp. 2498–2504, Dec. 2023, https://doi.org/10.52783/tjjpt.v44.i5.2990. [Online]. Available: https://www.propulsiontechjournal.com/index.php/journal/article/view/2990

[4] J. Mendez, K. Bierzynski, M. P. Cullar, and D. P. Morales, "Edge Intelligence: Concepts, Architectures, Applications, and Future Directions," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 5, pp. 1–41, Sep. 2022, https://doi.org/10.1145/3486674. [Online]. Available: https://dl.acm.org/doi/10.1145/3486674

[5] "Enabling automation and edge intelligence over resource constraint IoT devices for smart home," vol. 491, no. 2.

[6] K. Muhammad, A. Ullah, J. Lloret, J. D. Ser, and V. H. C. De Albuquerque, "Deep Learning for Safe Autonomous Driving: Current Challenges and Future Directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4316–4336, Jul. 2021, https://doi.org/10.1109/TITS.2020.3032227. [Online]. Available: https://ieeexplore.ieee.org/document/9284628/

[7] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen, "Distributed Artificial Intelligence Empowered by End-Edge-Cloud Computing: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 591–624, 2023, https://doi.org/10.1109/COMST.2022.3218527. [Online]. Available: https://ieeexplore.ieee.org/document/9933792/

[8] A. A. Suzen, B. Duman, and B. Sen, "Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. Ankara, Turkey: IEEE, Jun., https://doi.org/10.1109/HORA49412.2020.9152915. [Online]. Available: https://ieeexplore.ieee.org/document/9152915/

[9] S. K. Prashanthi, 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. [Online]. Available: https://dl.acm.org/doi/10.1145/3570604

[10] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, Aug. 2020, https://doi.org/10.1109/JIOT.2020.2984887. [Online]. Available: https://ieeexplore.ieee.org/document/9052677/

[11] H. Zhang, G. Wang, Z. Lei, and J.-N. Hwang, "Eye in the Sky: Drone-Based Object Tracking and 3D Localization," in *Proceedings of the 27th ACM International Conference on Multimedia*, vol. 10, no. 9. Nice France: ACM, Oct. 2020, pp. 899–907, https://doi.org/10.1145/3343031.3350933. [Online]. Available: https://dl.acm.org/doi/10.1145/3343031.3350933

[12] 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)*, vol. 11, no. 2. Storrs, CT, USA: IEEE, Nov. 2021, pp. 226–237, https://doi.org/10.1109/IISWC53511.2021.00030. [Online]. Available: https://ieeexplore.ieee.org/document/9668285/

[13] N. A. Andriyanov, "Analysis of the Acceleration of Neural Networks Inference on Intel Processors Based on OpenVINO Toolkit," in *2020 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*. Svetlogorsk, Russia: IEEE, Jul. 2020, pp. 1–5, https://doi.org/10.1109/SYNCHROINFO49631.2020.9166067. [Online]. Available: https://ieeexplore.ieee.org/document/9166067/

[14] K. T. Madathil, A. Dugar, N. Patil, and U. Cheramangalath, "Optimizing Machine Learning Operators and Models for Specific Hardware Using Apache-TVM," in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, vol. 7, no. 1. Delhi, India: IEEE, Jul. 2023, pp. 1–7, https://doi.org/10.1109/ICCCNT56998.2023.10306572. [Online]. Available: https://ieeexplore.ieee.org/document/10306572/

[15] K. Seshadri, B. Akin, J. Laudon, R. Narayanaswami, and A. Yazdanbakhsh, "An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks," in *2022 IEEE International Symposium on Workload Characterization (IISWC)*, vol. 11, no. 1. Austin, TX, USA: IEEE, Nov. 2022, pp. 79–91, https://doi.org/10.1109/IISWC55918.2022.00017. [Online]. Available: https://ieeexplore.ieee.org/document/9975395/

[16] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, no. 7, pp. 5113–5155, Oct. 2020, https://doi.org/10.1007/s10462-020-09816-7. [Online]. Available: http://link.springer.com/10.1007/s10462-020-09816-7

[17] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A Comprehensive Survey on Model Quantization for Deep Neural Networks in Image Classification," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 6, pp. 1–50, Dec. 2023, https://doi.org/10.1145/3623402. [Online]. Available: https://dl.acm.org/doi/10.1145/3623402

[18] P. Hu, X. Peng, H. Zhu, M. M. S. Aly, and J. Lin, "OPQ: Compressing Deep Neural Networks with One-shot Pruning-Quantization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, pp. 7780–7788, May 2021, https://doi.org/10.1609/aaai.v35i9.16950. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16950

[19] J. Kim, S. Chang, and N. Kwak, "PQK: Model Compression via Pruning, Quantization, and Knowledge Distillation," in *Interspeech 2021*. ISCA, Aug. 2021, pp. 4568–4572, https://doi.org/10.21437/Interspeech.2021-248. [Online]. Available: https://www.isca-archive.org/interspeech_2021/kim21m_interspeech.html

[20] N. Aghli and E. Ribeiro, "Combining Weight Pruning and Knowledge Distillation For CNN Compression," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Nashville, TN, USA: IEEE, Jun. 2021, pp. 3185–3192, https://doi.org/10.1109/CVPRW53098.2021.00356. [Online]. Available: https://ieeexplore.ieee.org/document/9523139/

[21] 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. [Online]. Available: https://journal.universitasbumigora.ac.id/index.php/matrik/article/view/3683

[22] S. P. Baller, A. Jindal, M. Chadha, and M. Gerndt, "DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*. San Francisco, CA, USA: IEEE, Oct. 2021, pp. 20–30, https://doi.org/10.1109/IC2E52221.2021.00016. [Online]. Available: https://ieeexplore.ieee.org/document/9610432/

[23] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, "An energy efficient IoT data compression approach for edge machine learning," *Future Generation Computer Systems*, vol. 96, pp. 168–175, Jul. 2020, https://doi.org/10.1016/j.future.2019.02.005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X18331716

[24] X. Jin, X. Ma, and L. Tao, "Research and analysis of pruning algorithm," in *International Conference on Algorithms, Microchips and Network Applications*, F. Cen and N. Sun, Eds. Zhuhai, China: SPIE, May 2022, pp. 70–78, https://doi.org/10.1117/12.2636499. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/12176/2636499/Research-and-analysis-of-pruning-algorithm/10.1117/12.2636499.full

[25] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *36th International Conference on Machine Learning*, 2020, pp. 10 691–10 700, https://doi.org/10.48550/ARXIV.1905.11946. [Online]. Available: https://arxiv.org/abs/1905.11946

[26] L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil, and A. Kolesnikov, "Knowledge distillation: A good teacher is patient and consistent," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, Jun. 2022, pp. 10 915–10 924, https://doi.org/10.1109/CVPR52688.2022.01065. [Online]. Available: https://ieeexplore.ieee.org/document/9879513/

[27] A. Setyanto, T. B. Sasongko, M. A. Fikri, and I. K. Kim, "Near-Edge Computing Aware Object Detection: A Review," *IEEE Access*, vol. 12, pp. 2989–3011, 2024, https://doi.org/10.1109/ACCESS.2023.3347548. [Online]. Available: https://ieeexplore.ieee.org/document/10374363/

[28] G. Huang, Z. Liu, G. Pleiss, L. V. D. Maaten, and K. Q. Weinberger, "Convolutional Networks with Dense Connectivity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 12, pp. 8704–8716, Dec. 2022, https://doi.org/10.1109/TPAMI.2019.2918284. [Online]. Available: https://ieeexplore.ieee.org/document/8721151/

[29] M. Zhang, H. Su, and J. Wen, "Classification of flower image based on attention mechanism and multi-loss attention network," *Computer Communications*, vol. 179, pp. 307–317, Nov. 2021, https://doi.org/10.1016/j.comcom.2021.09.001. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0140366421003303

[30] M. Ganaie, M. Hu, A. Malik, M. Tanveer, and P. Suganthan, "Ensemble deep learning: A review," *Engineering Applications of Artificial Intelligence*, vol. 115, pp. 105–151, Oct. 2022, https://doi.org/10.1016/j.engappai.2022.105151. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S095219762200269X

[This page intentionally left blank.]