ART & TECHNOLOGY PUBLISHING PTE. LTD.

Information Engineering and Applied Computing

2022 Volume 1, Issue 1

ISSN: 2630-4619

Neural Network Model Compression Algorithms for Image Classification in Embedded Systems

Heejung Shin, Hyondong Oh*

Department of Mechanical Engineering, Ulsan National Institute of Science and Technology (UNIST), Ulsan, Republic of Korea

Corresponding author: Hyondong Oh, h.oh@unist.ac.kr

Abstract

This paper introduces model compression algorithms that make a deep neural network smaller and faster for embedded systems. The model compression algorithms can be largely categorized into pruning, quantization, and knowledge distillation. In this study, gradual pruning, quantization aware training, and knowledge distillation which learns the activation boundary in the hidden layer of the teacher neural network are integrated. As a large deep neural network is compressed and accelerated by these algorithms, embedded computing boards can run the deep neural network much faster with less memory usage while preserving reasonable accuracy. To evaluate the performance of the compressed neural networks, we evaluate the size, latency, and accuracy of the deep neural network, DenseNet201, for image classification with the CIFAR-10 dataset on the NVIDIA Jetson Xavier.

Keywords

Deep learning
Model compression
Pruning
Quantization
Knowledge distillation
Embedded system

1. Introduction

To perform autonomous decision-making, unmanned vehicles need to rapidly perceive their surroundings by utilizing information acquired from various sensors. In particular, visual information from cameras contains high-dimensional data and can be effectively utilized. By employing image processing algorithms, it is possible to automatically extract and analyze features from the given visual data, enabling a high-level understanding of the surrounding environment, such as object detection and classification.

Since AlexNet won the ImageNet object classification competition in 2014 [1,2], convolutional neural networks (CNNs) have demonstrated exceptional performance in the field of computational image processing. The use of residual connections has addressed the problem of gradient vanishing, which occurs as the number of hidden layers in an artificial neural network increases [3]. With the residual connection structure resolving the gradient vanishing issue, research has focused on designing neural networks with more hidden layers and weights to achieve higher accuracy. Recent developments, such as the Vision Transformer [4], involve artificial neural networks with approximately 630 million weights.

These enlarged neural networks come with a significant computational cost compared to other algorithms, making high-performance computing devices such as GPUs essential. However, embedding high-performance computing devices into low-power embedded systems presents numerous challenges. Additionally, most embedded systems have limited memory, making it impossible to deploy artificial neural networks in some cases. Therefore, for systems like small autonomous unmanned vehicles that have limited computing power but require real-time performance or drones with constrained mission durations, efficient power management, compression, and acceleration of artificial neural networks are essential [5]. As a result, recent research efforts have been actively focused on compressing and accelerating artificial neural networks for deployment in embedded systems [6-8].

The goal of artificial neural network compression and acceleration algorithms is to develop compressed and accelerated artificial neural networks while preserving the performance of existing neural networks as much as possible. There are three primary algorithms for artificial neural network optimization:

(1) Pruning: Removes weights from the neural network that do not significantly impact its accuracy [9,10].
(2) Quantization: Reduces the data precision of artificial neural network weights to match the target hardware effectively [11].
(3) Knowledge distillation: Transfers knowledge from a larger neural network to a smaller one, enhancing the accuracy of a smaller model [12].

All three types of algorithms have been reported to effectively compress and accelerate artificial neural networks while preserving their existing accuracy [8,12-14]. However, there is limited research comparing and analyzing the performance of these three optimizing algorithms when applied together in embedded systems.

Therefore, this paper summarizes artificial neural network compression and acceleration algorithms theoretically and applies these algorithms collectively to the same environment and convolutional network for image classification tasks. Additionally, the performance of the compressed artificial neural network is compared and analyzed on the CPU of the embedded system, NVIDIA Jetson Xavier, to provide insights and comparisons.

2. Trends in the development of convolutional neural networks

The AlexNet, which won the ImageNet competition in 2012, consists of 8 layers, including 5 convolutional layers and 3 fully connected layers, by combining various convolutional filter sizes of 11×11,5×5, and 3×3. AlexNet demonstrated that having a deeper structure, in contrast to the LeNet composed of only 2 convolutional layers [15], can lead to better performance [1]. Building upon the research findings from AlexNet, VGGNet systematically increased the depth of CNNs. VGGNet constructed 19 deep convolutional layers using only 3×3 convolutional filters and achieved second place in the ImageNet competition [16]. Following VGGNet, there were numerous attempts to improve the performance of artificial neural networks by staking neural network hidden layers deeper. However, this approach encountered limitations due to the problem of gradient vanishing when constructing neural networks with depths beyond a certain threshold.

Nevertheless, in 2015, ResNet was developed and used a residual connection structure to address the gradient vanishing problem. ResNet, by employing the residual connection structure, constructed a neural network that was 8 times deeper than the previous VGGNet and won the 2015 ImageNet competition [3].

DenseNet further advanced this residual connection structure by not only using it to add features from hidden layers but also concatenating the features themselves [17]. The aforementioned research trends in image classification aimed at increasing accuracy by making neural networks deeper and with more convolutional filters. However, these neural networks posed challenges for deployment on mobile devices such as smartphones or embedded systems due to their size and computational demands. Therefore, this study explored the feasibility of applying compression and acceleration algorithms to the DenseNet-based neural networks, known for their superior performance among the aforementioned convolutional artificial neural networks, in embedded systems.

3. Pruning

Artificial neural networks are composed of a large number of weights, ranging from millions to even billions. A trained artificial neural network consists of weights with small absolute values, and it is known that among these weights, there are weights with low significance for inference [18-20]. Therefore, one way to compress artificial neural networks is to selectively remove weights with low significance, and this algorithm is called pruning.

Pruning algorithms can be broadly categorized into two types: unstructured pruning and structured pruning. The former sets the values of weights that are considered unnecessary for inference to 0. As a result, sparse weight matrices are generated, and artificial neural networks can be compressed using data structures such as coordinate list (COO), compressed sparse row (CSR), and compressed sparse column (CSC). The latter applies pruning to various components of the artificial neural networks, such as convolutional filters and individual hidden layers, rather than individual weights. It proceeds by completely removing unnecessary weights that are part of these components. Therefore, with structured pruning, it is possible to compress artificial neural networks without the need for data structures such as COO, CSR, and CSC. However, this algorithm may have difficulty preserving accuracy compared to unstructured pruning, as it may also remove significant weights within convolutional filters or specific hidden layers.

When applying pruning algorithms to artificial neural networks, the most crucial consideration is to preserve the model's accuracy as much as possible. If too many weights are pruned at once, the accuracy of the artificial neural network can be severely compromised. Thus, most algorithms follow a process where a small portion of weights is pruned (**Figure 1A**) and then undergoes a retraining process. The final compressed model is created by repeatedly applying the pruning algorithm in the same manner to the retrained neural network.

However, this iterative pruning and retraining structure has the drawback of high training costs in creating compressed artificial neural networks. To address this issue, a gradual pruning algorithm (**Figure 1B**) that integrates pruning and retraining has been proposed [13]. Gradual pruning is a member of the unstructured pruning family of algorithms, and it allows the creation of compressed artificial neural networks with minimal training costs while maximizing accuracy preservation. The study proposed a pruning scheduling as shown in equation (1) [13].

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3$$

for

$$t \in \{t_0, t_0 + \Delta t, \cdots, t_0 + n\Delta t\} \quad (1)$$

Here, *t* represents the point in the training process when the pruning algorithm is applied, and the interval during training when the pruning algorithm is applied can be determined by specifying t0 and *n*. S_i is the initial sparsity ratio typically set to 0 when applied to the initial artificial neural network, while S_f is the final sparsity ratio that the artificial neural network will eventually have. Equation (1) shows that, through gradient descent, when weights are updated, S_t % of the total weights are pruned. In this study, gradual pruning was used on compressing DenseNet201 without a decrease in accuracy.

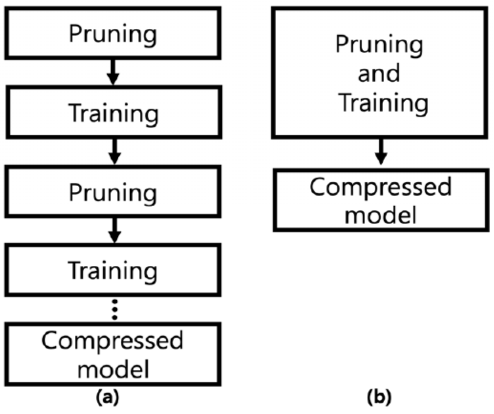

Figure 1. Flow chart of the pruning process. (a) shows a general pruning algorithm that takes a long time to get compressed neural networks; (b) combines pruning and training processes to obtain compressed neural networks in a short time.

4. Quantization

The weights of neural networks are represented in a 32-bit floating-point format, which allows for precise decimal representation, and they are updated through error backpropagation and gradient descent. While this floating-point representation provides a wider range of representable numbers compared to the fixed-point representation, it is considerably slower in computation speed than fixed-point representation, where real number operations can be replaced with integer operations. Hence, it is possible to accelerate the inference of artificial neural networks by converting the floating-point weights of trained neural networks into fixed-point representations. In addition, representing weights originally in 32 bits as 16 bits or 8 bits reduces the size of the neural network. For example, in the case of DenseNet201, which has approximately 18 million weights, if the weights are represented in 32 bits, the model size would be around 70 MB. However, by using 8-bit weights, the model size can be reduced to only 19 MB. Algorithms that leverage these characteristics of computer computation and structure to enable neural network compression and acceleration are known as quantization.

Figure 2 illustrates the process of quantizing 32-bit floating-point values into 8-bit integer values. Symmetric linear quantization is performed using equations (2) and (3).

$$x_q = clip(round\left(\frac{1}{s}x\right), \alpha_q, \beta_q)$$

where

$$s = \frac{\beta-\alpha}{\beta_q-\alpha_q} \tag{2}$$

$$clip(x,a,b) = \begin{cases} a, (x < a) \\ x, (a \leq x \leq b) \\ b, (b < x) \end{cases} \tag{3}$$

Here, x represents the input or output values of the quantized weights or activation functions of the neural networks and α,β represents the range of x. In the case of symmetric linear quantization, it has the relationship $\alpha = -\beta$. s is the scale factor used to convert values expressed in different bit widths and can be calculated using equation (2).

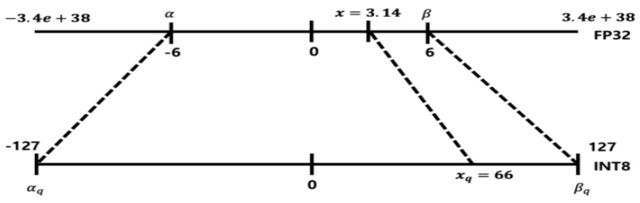

Figure 2. Symmetric linear quantization. The symmetric linear quantization algorithm maps 32-bit floating point representation to 8-bit integer representation.

When applying quantization algorithms to neural networks, just like pruning algorithms, it is essential to preserve the accuracy of the original network as much as possible. In the case of quantization from 32 bits to 16 bits, the range of representable numbers decreases but does not significantly impact the neural network's accuracy. However, when quantizing to 8 bits, the range of representable numbers decreases significantly compared to 32 bits, which can lead to a substantial reduction in the neural network's accuracy. Therefore, the most critical aspect of quantization algorithms for preserving the accuracy of the original neural

Table 1. Quantization algorithms

	Latency	Accuracy	Data requirement
Dynamic quantization	Low	Best	No
Post-static quantization	Lowest	Good	Partial
Quantization-aware training	Lowest	Best	All

network is selecting the value of for quantization. Quantization algorithms are classified into three main categories according to the method used to determine the parameter s, as shown in **Table 1**: dynamic quantization, post-static quantization, and quantization-aware training.

The first is dynamic quantization, where the given weight matrix is quantized using equation (2). Quantization of the input and output values of activation functions is performed in real-time by calculating s while the artificial neural network operates in its actual working environment. Dynamic quantization has the disadvantage of being slower compared to other quantization algorithms because it quantizes the input and output values of activation functions each time the artificial neural network runs. The second quantization algorithm is post-static quantization, which is different from dynamic quantization. In post-static quantization, a representative subset of data is input into the artificial neural network to calculate the value in advance for quantizing the input and output values of activation functions when applying the quantization algorithm. Post static quantization does not require real-time quantization during the operation of the artificial neural network, leading to faster inference speed compared to dynamic quantization. The final quantization algorithm is quantization-aware training (QAT), where the quantized artificial neural network is retrained using all the data used during the network's original training. QAT is the most accurate among the three algorithms and ensures a similar inference speed as post-static quantization. However, it requires additional training for creating compressed neural networks. The accuracy of post-static quantization and QAT are compared, and the interference acceleration performance on embedded systems is verified in this study.

5. Knowledge distillation

For systems with limited computing resources, such as embedded systems, the design of compressed artificial neural networks that are suitable for embedding and smooth operation of artificial neural networks is essential. Yet, these networks often have an insufficient number of weights, making it challenging to achieve robust performance. Hence, a learning method is required to maximize the performance of these artificial neural networks.

To address this, knowledge distillation was proposed to enhance the performance of compressed artificial neural networks using the concept of "teacher and student" [12]. The "teacher" is a high-performance artificial neural network with more weights compared to the "student", while the "student" is relatively smaller in size and has lower performance compared to the "teacher". As shown in **Figure 3**, the "student" makes different predictions from the "teacher" for the same input data, and the "student" learns by referring to the predicted values of the "teacher" using equations (4) to (7).

$$L_{CE}(y_{true}, y_s) = Crossentropy(y_{true}, y_s) \quad (4)$$

$$L_{Distill}(y_t, y_s, T) = L_{KLD}(p(y_t, T), p(y_s, T)) \quad (5)$$

$$p(y_i, T) = \sum_j \frac{\exp\left(\frac{y_i}{T}\right)}{\exp\left(\frac{y_j}{T}\right)} \quad (6)$$

$$L_{total} = \alpha L_{CE} + (1-\alpha) L_{Distill}$$

where .

$$\alpha \in [0,1], T > 0. \quad (7)$$

Here, α and T are hyperparameters, where α is an indicator of how much the "student" should reference the prediction of "teacher" during learning, and T adjusts the shape of the distribution of predictions between the "teacher" and "student" using equations (5) and (6). y_s and y_t are the outputs of the "student" and "teacher" networks, respectively, while y_{trug} is the ground truth data for the input data. L_{KLD} is the loss function calculated by using Kullback-Leibler divergence between the distributions of predictions by the "teacher" and "student".

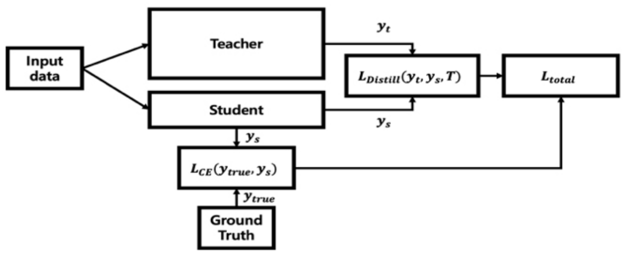

Figure 3. Knowledge distillation. The student network refers to the output of the teacher network for better training results.

Unlike traditional knowledge distillation algorithms that only use the predictions from the final layer of the "teacher", there are knowledge distillation algorithms that allow the "student" to reference the predictions from the "teacher's" hidden layers as well. This generally leads to better performance compared to traditional knowledge distillation, and it can be used not only for classification problems but also for regression problems. Heo *et al.* proposed a knowledge distillation technique where the "student" mimics the activation boundary of neurons in the hidden layer of the "teacher" artificial neural network using equations (8) and (9). This approach significantly improved the accuracy of the "student" model [14].

$$L(I) = \left\| \rho(T(I)) \odot \sigma\left(\mu_1 - r(S(I))\right) + (1 - \rho(T(I)) \odot \sigma(\mu_1 + R(S(I)))\right\|_2^2 \quad (8)$$

where

$$\rho(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; otherwise \end{cases} \quad (9)$$

Here, $T(\cdot)$ and $S(\cdot)$ are the output values of the activation functions of the "teacher" and "student", respectively, and I is the input data. $\sigma(\cdot)$ is the rectified linear unit (ReLU) activation function and $\mu 1$ is the unit margin vector. $r(\cdot)$ is a connector function proposed by Romero *et al.* [21], which serves to align the dimensions of the outputs of "teacher" and "student". $\odot$ denotes element-wise multiplication, and $\rho(\cdot)$ is a function that calculates the activation state of neurons in the neural network. This knowledge distillation algorithm is divided into two main phases: initialization and training. Initialization is the phase in which the "student" forms activation boundaries similar to those of the "teacher", and does not refer to y_{true}. After the initialization phase, the "student" model continues to learn without the assistance of the "teacher", using the entire training dataset. In this study, an algorithm from Heo *et al.* was used to improve the performance of the smaller neural network model, "student" (DenseNet121), with the guidance of a larger model, "teacher" (DenseNet201) [14].

6. Neural network compressing results

6.1. Artificial neural network training parameters and target neural network configuration

In this study, DenseNet201 was trained using TensorFlow on the CIFAR-10 dataset for image classification. The three aforementioned compressing algorithms were applied to the trained DenseNet201. The training configurations for DenseNet201 are shown in **Table 2**.

Since DenseNet is optimized for the ImageNet dataset, the CIFAR-10 was resized to 224×224, the same size as the ImageNet dataset, and the values of each channel were normalized to be within the range of 0 and 1. Data augmentation was also applied [22].

Top-N Accuracy is an evaluation metric commonly used in image classification problems. It considers a prediction to be correct if the top N values in the

Table 2. DenseNet201 training parameters and results. For training, the TensorFlow framework, CIFAR-10 dataset, and NVIDIA RTX 2070SUPER GPU are used.

Epochs (Steps)	50 (312,500)	
Batch size	8	
Optimizer	Stochastic gradient descent (Nesterov, momentum = 0.9)	
Learning rate	10^{-3}	$1 \leq$ epoch ≤ 25
	10^{-4}	$26 \leq$ epoch ≤ 38
	10^{-5}	$39 \leq$ epoch ≤ 50
Top-1 accuracy	95.15%	
Model size	70 MB	

Table 3. DenseNet201 gradual pruning results. Gradual pruning compressed the neural network, and an increase or decrease in accuracy and compression ratio was specified.

Sparsity (%)	**Top-1 accuracy (%)**	**Size (MB)**
0	95.15 (+0%)	70 (100%)
50	97.44 (+2.29%)	51 (73%)
60	97.25 (+2.10%)	41 (59%)
70	97.41 (+2.26%)	32 (46%)
80	96.91 (+1.76%)	22 (31%)
90	95.43 (+0.28%)	12 (17%)
95	92.23 (-2.92%)	6.2 (8.9%)
99	71.56 (-23.6%)	2.2 (3.1%)

discrete probability distribution output from the final layer of the neural network contain the correct label.

After training DenseNet201, the compressing algorithm was applied, and the compressed artificial neural network was mounted on the NVIDIA Jetson Xavier embedded system to assess and compare performance. In addition, the compression and acceleration performance was evaluated using TensorFlow Lite's model deployment optimization feature when deploying on the embedded system.

6.2. Application and results of pruning algorithm

Gradual pruning was applied to the pre-trained DenseNet201 with various pruning ratios ranging from 50% to 99%. S_i=t_i=0 and *n* was set to 80% of the total steps (250,000) to allow for fine-tuning in the later stages. As suggested by Zhu *et al.* [13], the learning rate used in the original training was multiplied by 1/10 (10^{-4}, 10^{-5}, 10^{-6}).

As shown in **Table 3**, it is observed that pruning the weights of DenseNet201 by up to 90% allows for approximately 6 times reduction in model size without a decrease in accuracy. In particular, by applying pruning, using a smaller learning rate, and allowing fine-tuning in the later stages of training, a slight improvement in neural network performance was observed. However, when the pruning ratio was increased to 99%, there was a dramatic 23.6% decrease in neural network performance.

6.3. Application and results of quantization algorithm

The weight quantization of DenseNet201 was performed using post-static quantization (INT8) and QAT (INT8*). In the case of QAT, the same training

Table 4. Static quantization and QAT result

	Size (MB)	Top-1 accuracy (%)	Latency (ms)
FP32	70 (100%)	95.15 (+0%)	1,274.03
INT8	19 (27%)	93.23 (-1.92%)	763.93
INT8*	19 (27%)	97.09% (+1.94%)	768.83

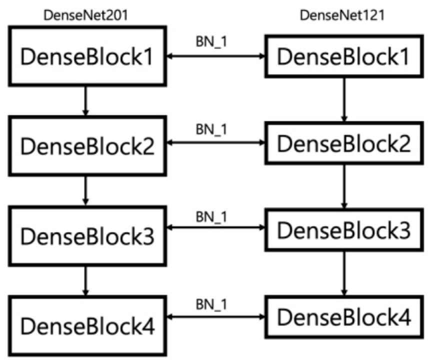

Figure 4. Hidden layers for knowledge distillation. The "student" network refers to the feature from the batch normalization layers in the dense blocks in the "teacher" network.

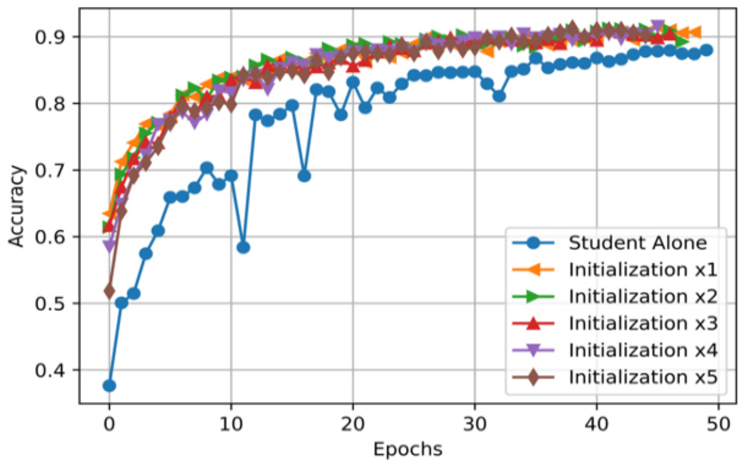

Figure 5. Knowledge distillation result. The knowledge distillation algorithm improved the accuracy of DenseNet121 using DenseNet201.

parameters of the original DenseNet201 were applied. When using INT8 for weight data precision, it was observed that the size of the artificial neural network was reduced by approximately 73% compared to the existing FP32. Also, post-static quantization resulted in a 1.92% decrease in accuracy compared to the original artificial neural network, but by applying QAT, the weights were quantized, and additional training was performed with smaller learning rates (10^{-4}, 10^{-5}, 10^{-6}) than those used in the original training. This approach led to a 1.94% increase in accuracy. Finally, taking advantage of the conversion from floating-point to fixed-point operations, the inference time on NVIDIA Jetson Xavier's CPU was reduced by approximately 40%, from 1,274.03 ms to 768.83 ms, and the results are summarized in **Table 4**. Inference time was measured as the average time taken for a total of 50 inferences with a batch size set to 1.

6.4. Application and results of knowledge distillation algorithm

For the application of the knowledge distillation technique [14], DenseNet201 was selected as the "teacher" and DenseNet121 as the "student", as shown in **Figure 4**. The first batch normalization layer of the DenseBlock, which is a component of each neural network, was configured to learn the activation boundary.

Initialization was applied from 1 to 5 times, and the accuracy was compared, with the total number of epochs set to 50, including initialization. The experimental results showed that applying the knowledge distillation algorithm more than once resulted in a similar increase in accuracy, as shown in **Figure 5**.

Furthermore, **Table 5** shows that applying knowledge distillation led to a performance increase of up to 3.5% compared to not applying it, with the Initialization ×4 case achieving the highest performance improvement. The inference time on the

Table 5. Knowledge distillation results in terms of size, accuracy, and latency. The performance of DenseNet121 has been improved using DenseNet201 by 3.5%.

	Size (MB)	Top-1 accuracy (%)	Latency (ms)
Teacher	70 (100%)	95.15	1,273.03
Student	29 (41%)	88.0 → 91.5 (+3.5%)	658.06

Table 6. DenseNet121 (knowledge distilled) pruning result

Sparsity (%)	Top-1 accuracy (%)	Size (MB)
0	91.50 (+0%)	29 (100%)
50	92.42 (+0.92%)	20 (69%)
60	91.91 (+0.41%)	16 (55%)
70	91.36 (-0.14%)	12 (41%)
80	89.93 (-1.57%)	8.0 (28%)
90	85.88 (-5.62%)	4.3 (15%)
95	78.26 (-13.2%)	2.4 (8.2%)
99	10.00 (-81.5%)	0.9 (3.0%)

Table 7. The performance of DenseNet121 on the embedded system. The network has been compressed by knowledge distillation, pruning, and QAT.

Sparsity (%)	Size (MB)	Top-1 accuracy (%)	Latency (ms)
80	4.6	93.35	499.26
90	2.8	88.60	487.65
95	1.9	85.54	497.30

NVIDIA Jetson Xavier's CPU was deduced to 658.06 ms when using knowledge distillation, which is a 48.3% reduction compared to the original DenseNet201 (1,274.03 ms).

6.5. Results of applying the integrated compressing algorithms

Each of the aforementioned compressing algorithms can be used independently to compress a neural network, and a proper integration of these algorithms can potentially yield better performance. To integrate the three aforementioned compressing algorithms, a compressed artificial neural network was implemented using knowledge distillation. Subsequently, pruning was applied to further compress the model, creating a model with sparse weights. Finally, QAT was applied to convert the model from 32-bit floating-point weights to 8-bit weights. If pruning is applied before quantization, the weights, which were originally represented as precise floating-point values, will be quantized to integers. Pruning such integer weights, starting with the lowest absolute values, would lead to a significant drop in the neural network's performance. For this reason, the TensorFlow framework used in this study does not support applying pruning algorithms to quantized artificial neural networks. Therefore, in this study, integrated compressed artificial neural networks were generated by applying pruning to the DenseNet121 model that had undergone knowledge distillation.

The pruning process for the DenseNet121 model with the knowledge distillation was conducted in the same manner as the approach applied to DenseNet201. Learning rates were reduced as pruning progressed, enabling pruning and additional training to occur

Table 8. Comparison of performance of the neural networks before and after model compression

Device	Size (MB)	Top-1 accuracy (%)	Latency (ms)
CPU	70	95.15	1,274.03
GPU	70	95.15	366.85
CPU*	4.6	93.35	499.26

simultaneously. This allowed for the preservation or improvement of the neural network's performance up to a certain pruning ratio. However, when the pruning ratio exceeded 80%, a significant drop in accuracy was observed, as shown in **Table 6**.

In QAT, pruning ratios of 80%, 90%, and 95% were applied to the compressed artificial neural networks. The results are presented in **Table 7**.

Finally, the artificial neural network resulting from applying 80% pruning and QAT to the DenseNet121 model with knowledge distillation was selected as the final compressed artificial neural network. The inference time was measured on NVIDIA Jetson Xavier's CPU, and the results are shown in **Table 8**. The integrated compressed artificial neural network, which had all three compressing algorithms applied, minimized accuracy degradation while reducing its size to 4.6 MB, which is 15.2 times smaller than the original. Inference time improved by 60.8% compared to the pre-compressing state (CPU*). It was noted that the GPU-based deployment and performance evaluation for the compressed artificial neural networks were not performed due to limited support for compressing algorithms in TensorFlow Lite and NVIDIA TensorRT. For comparison, latency results for the pre-compressing artificial neural network were added when deployed on NVIDIA Jetson Xacier's GPU in the table.

The first row (CPU) and the second row (GPU) show the results of running the uncompressed DenseNet201 on an embedded system, while the third row (CPU*) represents the results of running the compressed artificial neural network that underwent knowledge distillation using DenseNet121 ("student") and applying pruning and QAT on DenseNet201 ("teacher") on the same embedded system.

7. Conclusion

In this study, the theoretical concepts of pruning, quantization, and knowledge distillation compressing algorithms were summarized and applied to convolutional artificial neural networks to analyze their effects. The most crucial aspect of applying compressing algorithms to artificial neural networks is preserving the existing accuracy. In this study, the aforementioned algorithms were applied to a convolutional neural network with minimal accuracy loss. Firstly, in the case of pruning, the artificial neural network can be compressed by up to 5.83 times without any accuracy loss, and it was possible to achieve an 11.3 times reduction in size with only a 2.92% accuracy drop. Secondly, by applying post-static quantization and QAT, a compressed artificial neural network that is 3.7 times smaller was created and the inference time on embedded systems was reduced by 39.7%. Thirdly, in knowledge distillation, DenseNet201 was used to enhance the accuracy of DenseNet121 by 3.5% and utilized a 2.41 times smaller artificial neural network than DenseNet201, resulting in a 46.6% reduction in inference time on embedded systems. Lastly, by integrating the three compressing algorithms while preserving the performance of the artificial neural network, a compressed neural network that is 15.2 times smaller in size was created and a 60.8% reduction in inference time was achieved. In the future, the authors plan to apply the aforementioned compressing algorithms to object detection algorithms and mount them on Field Programmable GateArray (FPGA) for low-power, high-performance object detection tasks.

Disclosure statement

The authors declare no conflict of interest.

Funding

This work was supported by Theater Defense Research Center funded by Defense Acquisition Program Administration under Grant UD200043CD.

References

[1] Krizhevsky A, Sutskever I, Hinton GE, 2017, ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, 60(6): 84–90. https://doi.org/10.1145/3065386

[2] Deng J, Dong W, Socher R, et al. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 15–20, 2009: ImageNet: A Large-scale Hierarchical Image Database. 2009, Miami. https://doi.org/10.1109/CVPR.2009.5206848

[3] He K, Zhang X, Ren S, et al. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 27–30, 2016: Deep Residual Learning for Image Recognition. 2016, Las Vegas. https://doi.org/10.1109/CVPR.2016.90

[4] Dosovitskiy A, Beyer L, Kolesnikov A, et al. The 9th International Conference on Learning Representations (ICLR 2021), May 3–7, 2021: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2021, Vienna. https://openreview.net/forum?id=YicbFdNTTy.

[5] Oh S, Kim H, Cho S, et al., 2020, Development of a Compressed Deep Neural Network for Detecting Defected Electrical Substation Insulators Using a Drone. Journal of Institute of Control, Robotics and Systems, 26(11): 884–890. https://doi.org/10.5302/j.icros.2020.20.0117

[6] Howard AG, Zhu M, Chen B, et al., 2017, Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv: 1704.04861. https://arxiv.org/abs/1704.04861

[7] Uetsuki T, Okuyama Y, Shin J. 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), December 20–23, 2021: CNN-Based End-to-End Autonomous Driving on FPGA Using TVM and VTA. 2021, Singapore. https://doi.org/10.1109/MCSoC51149.2021.00028

[8] Han S, Mao H, Dally WJ, 2015, Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. arXiv: 1510.00149. https://arxiv.org/abs/ 1510.00149

[9] LeCun Y, John D, Sara S, 1989, Optimal Brain Damage. Advances in Neural Information Processing Systems. https://papers.nips.cc/paper/1989/hash/6c9882bbac1c7093bd25041881277 658-Abstract.html

[10] Hassibi B, Stork D, 1992, Second Order Derivatives for NetworkPruning: Optimal Brain Surgeon. Advances in Neural Information Processing Systems. https://proceedings.neurips.cc/paper/1992/hash/303ed4c69846ab36c2904d3ba8573050-Abstract.html

[11] Wu H, Judd P, Zhang X, et al., Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. arXiv, preprint: 2004.09602. https://arxiv.org/abs/2004.09602

[12] Hinton GE, Vinyals O, Dean J, 2014, Distilling the Knowledge in a Neural Network. Advances in Neural Information Processing Systems. https://arxiv.org/abs/1503.02531

[13] Zhu M, Gupta S. The 6th International Conference on Learning Representations (ICLR 2018), April 30–May 3, 2018:

To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. 2018, Vancouver. https://arxiv.org/abs/1710.01878.

[14] Heo B, Lee M, Yun S, et al. The 33rd AAAI Conference on Artificial Intelligence (AAAI-19), January 27–February 1, 2019: Knowledge Transfer via Distillation of Activation Boundaries Formed by Hidden Neurons. 2019, Honolulu. https://doi.org/10.1609/aaai.v33i01.33013779

[15] Lecun Y, Bottou L, Bengio Y, et al., 1998, Gradient-Based Learning Applied to Document Recognition. IEEE, 86(11): 2278–2324. https://doi.org/10.1109/5.726791

[16] Simonyan K, Zisserman A. The 3rd International Conference on Learning Representations (ICLR 2015), May 7–9, 2015: Very Deep Convolutional Networks for Large Scale Image Recognition. 2015, San Diego. https://arxiv.org/abs/1409.1556

[17] Huang G, Liu Z, van der Maaten L, et al. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 21–26, 2017: Densely Connected Convolutional Networks. 2017, Honolulu. https://doi.org/10.1109/CVPR.2017.243

[18] Han S, Pool J, Tran J, et al., 2015, Learning Both Weights and Connections for Efficient Neural Networks. arXiv: 1506.02626. https://arxiv.org/abs/1506.02626

[19] See A, Luong M-T, Manning CD. The 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL), August 11–12, 2016: Compression of Neural Machine Translation via Pruning. 2016, Berlin. https://doi.org/10.18653/v1/K16-1029

[20] Narang S, Elsen E, Diamos G, et al., 2017, Exploring Sparsity in Recurrent Neural Networks. arXiv: 1704.05119. https://arxiv.org/abs/1704.05119

[21] Romero A, Ballas N, Kahou SE, et al., 2015, Fitnets: Hints for Thin Deep Nets. arXiv: 1412.6550. https://arxiv.org/abs/1412.6550

[22] Lim S, Kim I, Kim T, et al., 2019, Fast Autoaugment. Advances in Neural Information Processing Systems, 32. https://papers.nips.cc/paper/2019/hash/6add07cf50424b14fdf649da87843d01-Abstract.html.