Master of Science in Artificial Intelligence
STAT 7008 Programming for Data Science
2023-24 Semester 1 Group Project

Topic 06 - Model Compression

A Study on Model Pruning and Quantization Algorithms

Lin Xiaotao †	**Teng Yiduo**	**Wang Congyi**	**Zhang Ruijie**	**Yao Yang**
3036213847	3036216227	3036215261	3036212348	3036212520

† Leader of the group

Links

Presentation Video

7008_GP_Presentation.mp4

Presentation Slides

7008_GP_Slides.pptx

December 4, 2023

Contents

1 Introduction

1.1 Description

This topic is about model compression. In recent years, deep learning networks (DNNs) have shown remarkable performance. However, most of the neural network models are computationally expensive and memory intensive, hindering their deployment in devices with low memory resources or in applications with strict latency requirements. Particularly, past few years have witnessed the significant progress in virtual reality, augmented reality and smart wearable devices, bring fundamental challenges in deploying deep learning systems to portable devices with limited resources.

In case of compression without significantly decreasing model performance, the network compression technologies such as network pruning and quantization, low-rank factorization, knowledge distillation and transferred convolutional filters make it possible to deploy DNNs on unmanned drones, smart phones and intelligent glasses[1, 2].

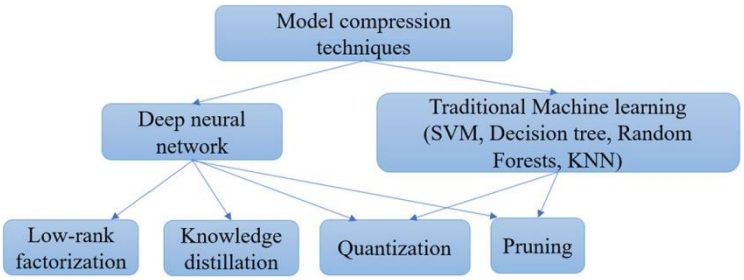


Figure 1.1 Different methods of model compression

The advantages of model compression include low communication bandwidth, small cloud computing resource cost, quick response time, and improved data privacy[3, 4].

1.2 Ideas

In our project, we will demonstrate the powerful capabilities of the most mainstream methods, model pruning and model quantization.

Our experiment with CIFAR-10 dataset as benchmark and VGG network as baseline will qualitatively and quantitatively evaluate the impact of the two methods on model size and model performance.

2 Methodology

2.1 Pruning algorithm

Pruning is widely used in the optimization of neural networks, which improves the efficiency and inference speed of the model and reduces the risk of overfitting by pruning the parameters in the neural network and setting some unimportant parameters to zero.

2.1.1 Overview of the pruning algorithm

It is generally presented as a three-stage process：

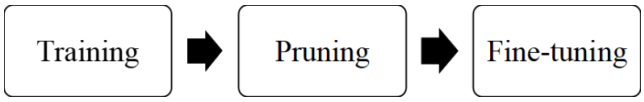


Figure 2.1 Three-stage process of pruning

The first step is to train the large model to get the best network performance, based on which it is pruned to get a simplified network, and finally fine-tuned to gain some of the lost performance, which is generally done by initializing the pruned network with the important parameters retained in the pruned large network, and then fine-tuning it for a few rounds on the training set.

2.1.2 Classification of pruning methods

Classification based on pruning direction

Weight Pruning, Activation Pruning and Gradient Pruning are the three common directions in pruning methods.

We can get the effect of weights, activation and gradient sparsity of the neural network on the speedup potential from the table below:

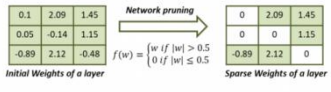
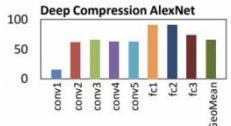
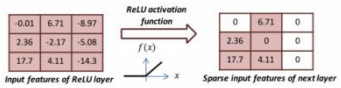
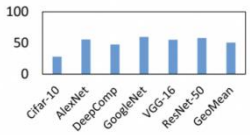
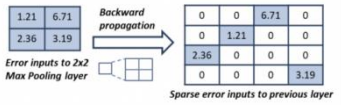
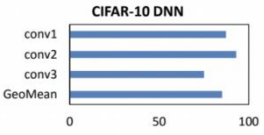
Sparsity Types	Data Structure	Sources	Percentage of Sparsity	Speedup Potential
Static Sparsity	Weight			1x~50x
Dynamic Sparsity	Activation			1x~10x
	Gradient			1x~10x

Figure 2.2 model_sparsification_summary

Classification based on pruning strategy

Pruning is mainly divided into structured pruning and unstructured pruning, the main difference between the two is whether it will destroy the model structure. The sparse model was obtained after unstructured pruning. Although the calculation amount of the model is greatly reduced, due to the destruction of the original model structure, it is necessary to design a specific hardware structure to accelerate. Structured pruning usually takes the filter or the whole network layer as the basic unit of pruning. When a filter is pruned, the overall structure of the model remains the same and can still be accelerated using existing hardware. In summary, unstructured pruning is more consistent with the random distribution of weights of different sizes in neural networks, with higher model compression and accuracy, but with higher hardware requirements, so a trade-off should be made between model validity and computational efficiency when selecting a method.

Classification based on pruned objects

We can prune at different levels and different positions of neural network, mainly weight pruning, neuron pruning, filter pruning, channel pruning.

1. Weight Pruning

The values in the weight matrix whose values are close to 0 in absolute value are set

to 0 and then the network is fine-tuned.

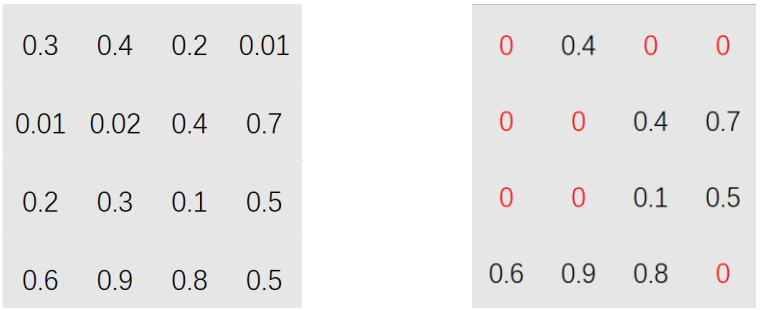


Figure 2.3 Weight pruning before pruning (left one) and after pruning (right one)

2. Neuron Pruning

Neurons are pruned, but the operation is different from weight pruning, because the neuron contains a set of weights, so we need to set a row or column in the weight matrix to 0.

Figure 2.4 Neuron pruning before pruning (left one) and after pruning (right one)

3. Filter pruning

There are different methods by which we can determine whether a filter is important or not and thus decide whether to prune it or not. This paper[5] uses Taylor Expansion as a measure of the pruning criterion to pruning filters. This paper[6] utilizes Geometric Median for model pruning, pulling absolute importance down to a relative level, deeming filters that are too similar to other filters unimportant.

4. Channel Pruning

Channel pruning is the most commonly used and easier to implement pruning strategy, which prunes several groups of channels at a time, e.g. this paper[7] introduces the γ of

the scaling factor in the BN layer to sort the channels and then prunes them.

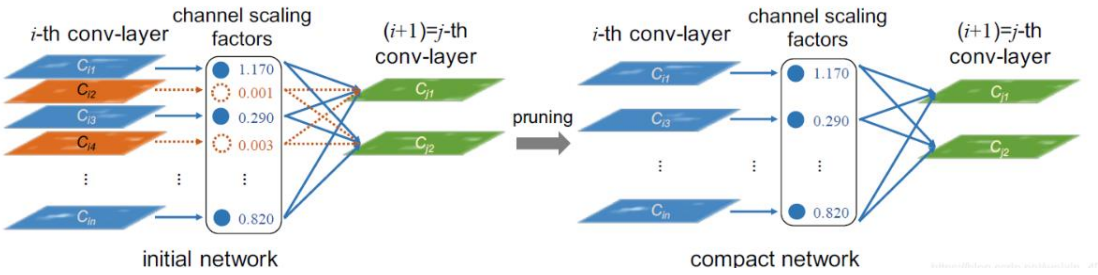


Figure 2.5 Channel pruning

2.1.3 Methods used in this experiment

The method chosen this time is the classical L1-norm unstructured pruning method, which, as mentioned above, performs much better, so we adopt this method and apply it to VGG-16 and ResNet-18. Here we use VGG-16 as an example:

(1) Iterate through the weight matrices of all convolutional and fully connected layers in the VGG16 network.

(2) Calculate the L1 parameter of each weight matrix.

(3) Determine the number of weights to be pruned based on the pre-determined pruning ratio and set the weight with the smaller L1 -norm to zero.

(4) Perform fine-tuning.

A more intuitive example is given to illustrate:

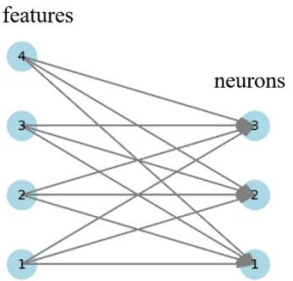


Figure 2.6 Fully connected layer

Weight matrix

1	-2	-0.1
-0.2	3	0.2
-2	3	0.2
-2	0.1	3

Pruning matrix

0	-2	0
0	3	0
-2	3	0
-2	0	3

Figure 2.7 Matrix after pruning

Then, a more intuitive visualization is given by setting the lighter-colored values in

the matrix, i.e., values closer to zero.

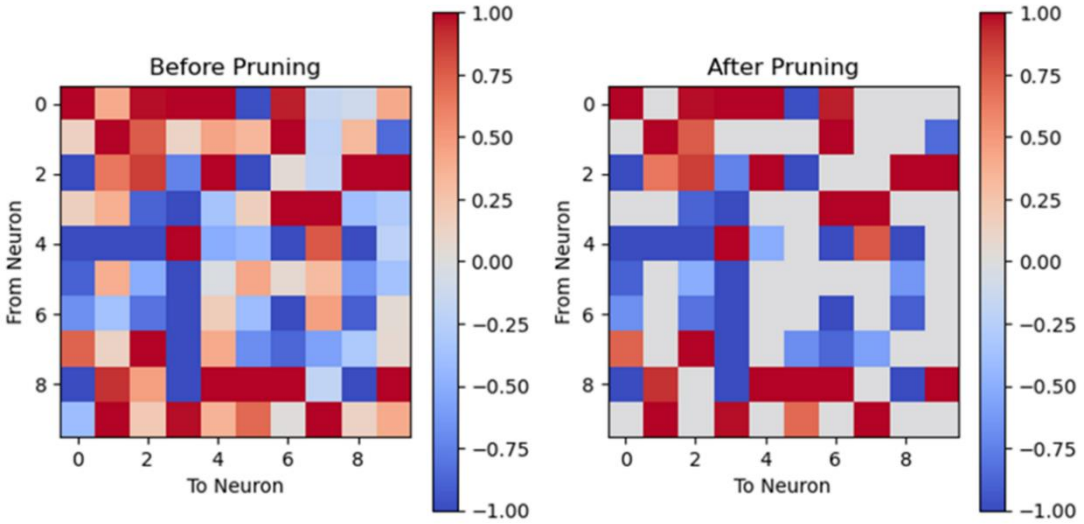


Figure 2.8 L1-norm unstructured pruning

2.2 Quantization algorithm

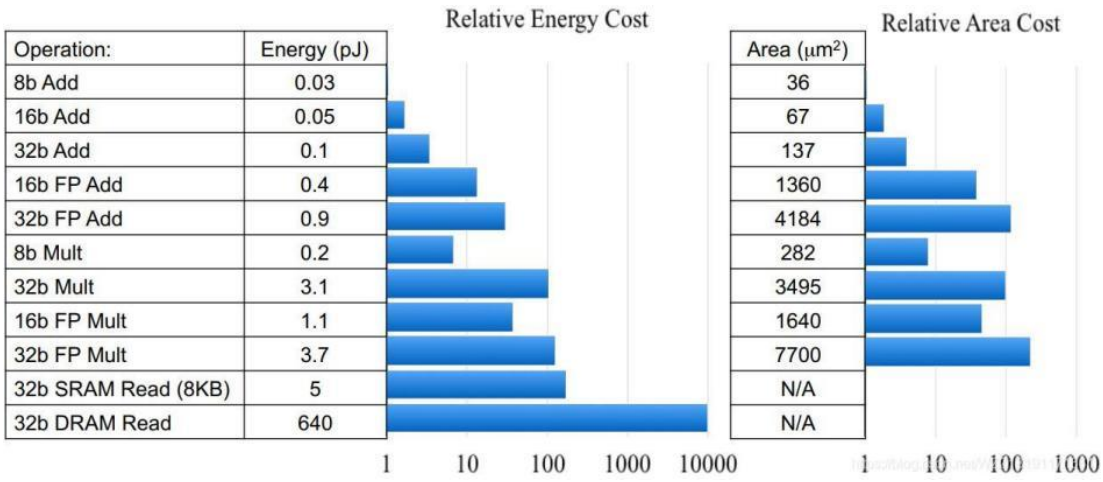
Quantization is a method to reduce the number of bits used to represent parameters, thereby decreasing the model's size.

2.2.1 Advantages

Parameter Compression: Using low-precision integer data to represent originally high-precision floating-point data reduces the parameters size.

Speed Enhancement: When weights and activations are quantized to integer values, floating-point operations can be converted into efficient integer operations, improving inference speed.

Memory Reduction: Quantizing activation values reduces the memory footprint during inference.



Operation:	Energy (pJ)	Area (μm²)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

Figure 2.9 Energy cost and memory cost

2.2.2 Disadvantages

Model Accuracy Decline: Quantization operations introduce errors to the network's values, leading to a decline in model accuracy.

2.2.3 Categories of quantization

Linear and Non-linear Quantization

Linear Quantization (uniform quantization): it maintains a fixed difference between adjacent quantization values. The quantization process involves dividing the floating-point value by a scaling factor and then rounding and clamping.

$$Q = \text{clamp}(\text{round}\left(\frac{r}{s}\right)) \qquad \begin{array}{l} r: float \\ Q: \text{int} \\ dequantization: r = S * Q \end{array}$$

Non-linear Quantization: The intervals between quantization values are not uniform. In areas with abundant data, the quantization intervals are small, leading to high precision. In areas with scarce data, the quantization intervals are large, resulting in lower precision.

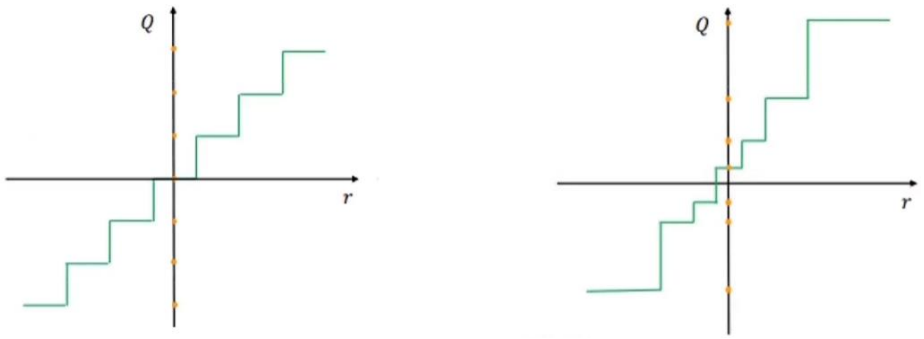


Figure 2.10 Linear (left one) and non-linear (right one) quantization

Symmetric and Asymmetric Quantization

Quantization can be categorized into symmetric quantization and asymmetric quantization based on whether the zero point of float values is mapped to the zero point of quantized values.

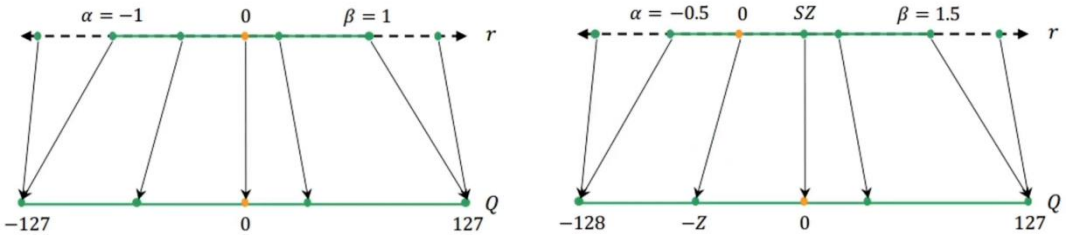


Figure 2.11 Symmetric (left one) and asymmetric (right one) quantization

Symmetric Quantization: The zero point of floating-point values directly maps to the zero point of quantized values. It does not require additional parameters for zero-point adjustment. The only parameter associated with quantization is the scaling factor.

Asymmetric Quantization: Asymmetric quantization has an additional parameter 'Z' to adjust the mapping of the zero point, which is commonly referred to as the zero point.

$$Q = \text{round}(S(r - Z)) \qquad \begin{matrix} Z : \text{zero point} \\ S : \text{Scale} \end{matrix}$$

Granularity of Quantization

Quantization per-layer: All filters within each network layer share the same quantization parameters. The selection of the range to represent the floating-point values requires consideration of all filters within the current layer. Applying the same

range to all filters within a layer implies identical scaling factors and zero points. While this implementation is straightforward, its effectiveness is limited.

Quantization per-channel: The required range and quantization parameters are individually calculated for each filter within each layer. This approach better preserves the information of each filter, resulting in superior quantization performance.

Precision Selection

Uniform Precision: All layers in the network use the same bit-width for quantization. It simplifies implementation but may result in significant accuracy loss, especially when quantizing to lower bit-widths.

Mixed Precision: Different network layers use different bit-widths for quantization, which aims to balance the trade-off between model size and performance.

2.2.4 Quantization techniques

Post-training quantization directly applies quantization to a pre-trained model without further training or fine-tuning. It offers a low overhead for deployment but may result in higher accuracy degradation. Post-training quantization can be divided into weight quantization and full quantization.

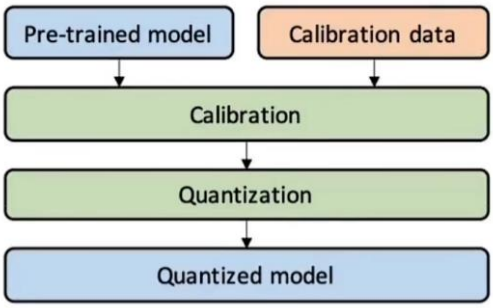


Figure 2.12 Post-training quantization

Weight quantization involves converting model parameters from floating-point numbers to integers with lower bit-width. In weight quantization, only the model's weights undergo quantization, and they are stored in integer form, effectively compressing the model size.

Full quantization encompasses quantization both model weights and activation values. This not only compresses the model size, reducing the memory footprint during inference but also allows the use of efficient integer arithmetic units, speeding up the inference process. Full quantization includes static quantization and dynamic quantization.

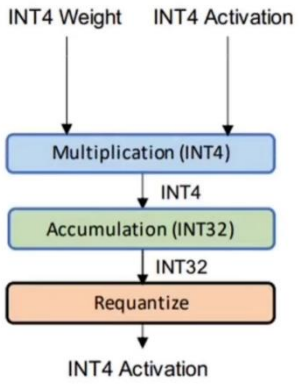


Figure 2.13 quantization process

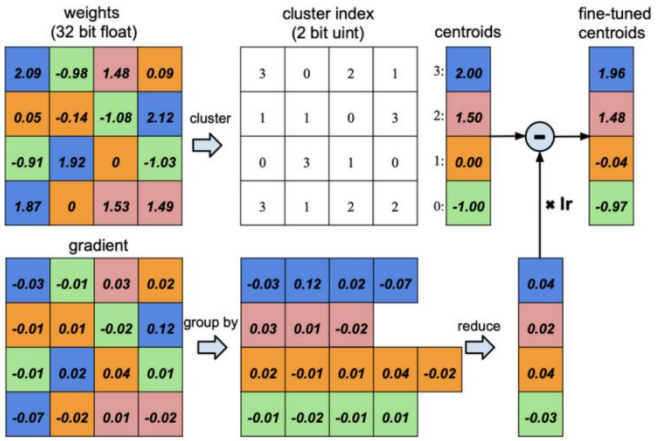


Figure 2.14 Examples

Static Quantization: In static quantization, the quantization parameters for model weights and activations are precomputed offline and remain fixed during inference. Activation values quantization requires information about the distribution of activation values.

Dynamic Quantization: In dynamic quantization, the quantization parameters related to activation values are computed in real-time during the inference phase. While this approach may yield better results, it introduces additional overhead during inference.

2.2.5 Examples

As shown in the diagram (Figure 2.15), we represent the original 32-bit float values using 2-bit integer types.

3 Implementation

3.1 CIFAR-10 dataset

The CIFAR-10 is a labeled dataset of tiny images[11], which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The CIFAR-10 was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Here we divided dataset into 5 training batches and 1 testing batch with 10,000 images in each batch.

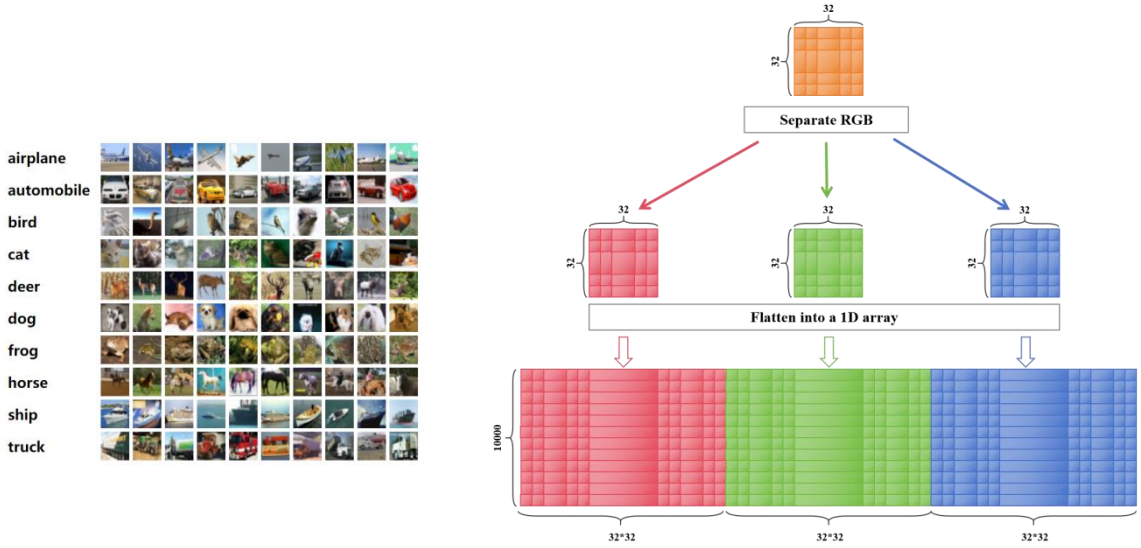


Figure 3.1 Outlook of CIFAR-10 (left one) and data structure of images (right one)

3.2 VGG network base model

The architecture of VGG-16 is very simple, it consists of 16 convolutional layers and 3 fully connected layers. The main feature of VGG-16 is that it has a relatively deep network structure and a small-sized convolutional kernel, which allows it to learn richer image features. However, due to the large number of parameters, VGG-16 can be time-consuming with limited computational resources.

3.3 ResNet network base model

What we use in this experiment is ResNet18, comes from the 18 Shortcut connection convolutional layers contained in its neural network. It can eliminating the difficulty of training neural networks with too much depth.

3.3 Model Pruning

Firstly, we pre-trained a VGG-16 model (self-contained in pytorch used) and a ResNet18 model using the CIFAR-10 dataset. Then, pruning was performed using the L1_unstructured method. The pruning percentages are: 10%, 30%, 50%, 70%, 90% and the accuracies are compared separately.

3.4 Model Quantization

For model quantization, we apply Pytorch for completing static quantization as well as the dynamic quantization, both of them arepost-training quantization. We make modifications after we load the pre-trained models.

For static quantization, there consist of four steps. Firstly, we merge the layers together for acceleration and the improvement of accuracy. Then we set the histogram_observer and min_max_observer as the observer of activation and weight separately. After that, we feed some dataset to the transferred model for obtaining the distributional characteristics of the data.

For Dynamic quantization, we make the weight parameter of the linear part of the network converted from float32 to int8. And we use the min max observer which can transfer the range of output to the int8 range output. These two processes are belonging to the Tensor-wise operation.

4 Experiments

4.1 Qualitative analysis and scenarios discussion

In terms of qualitative analysis, model pruning can trim the redundant and non-critical weights. Thus, model pruning may be more appropriate for models with many redundant variable. In specific scenarios, the approach is more suitable for increasing the interpretability of the model, applications that require an understanding of the model's decision-making process (e.g., medical diagnosis and financial risk assessment).

Model quantization refers to compressing a model by reducing the number of bits required for weight representation or activation. This means sacrificing computational accuracy in exchange for reducing the size of the model, which also indicates that model quantization is more generalizable. This method is more suitable for model use with resource-constrained devices, Scenarios that require real-time application of the model are also well suited and large-scale deployment.

4.2 Quantitative analysis

In this experiment, we tested the performance of the baseline model and the processed model separately. In this process, we used VGG-16, Resnet-18 as mentioned above for our baseline model, while for the dataset, we used CIFAR-10 for training and testing, with a ratio of about 4:1. We used both pruning and quantization methods to process the model, and the following are the details.

4.2.1 Benchmark

In our project, we use the following three key metrics to evaluate the performance of the compressed model.

Accuracy: Accuracy is an important measure of the compressed model’s predictive

ability. We calculate the accuracy rate by following formula.

$$Accuarcy = \frac{TP + TN}{TP + FN + FP + TN}$$

Model File Size: Smaller the file size, the simpler the model, which can lower the cost of storage and deployment. We evaluate the effectiveness of our compression by checking the size of file.

Model Runtime on CPU: Shorter runtime means that our compression technique can reduces the size of the model and improve efficiency. We evaluate the effectiveness of our compression by measuring the time required for the compressed model to process the same amount of data.

4.2.2 Pruning

VGG16		ResNet18	
Proportion	l1_unstructured	Proportion	l1_unstructured
0	0.9087	0	0.9412
0.1	0.909	0.1	0.9419
0.3	0.9066	0.3	0.9403
0.5	0.9022	0.5	0.93
0.7	0.8551	0.7	0.8265
0.9	0.1	0.9	0.0996

Table 4.1 VGG16_and_ResNet18_accuracy

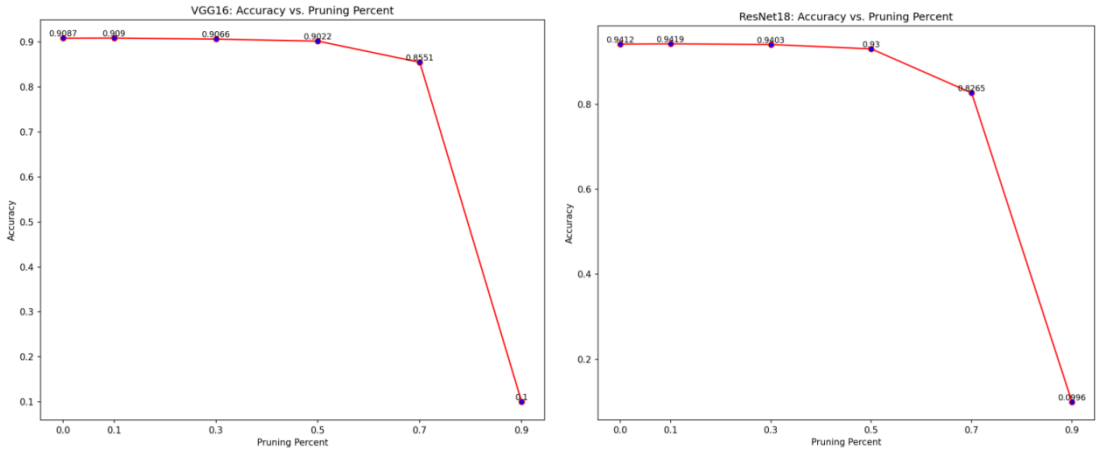


Figure 4.1 VGG16_accuracy (left one) and ResNet18_accuracy (right one)

From Table 4.2 and Figure 4.1, we can find that for VGG16 and ResNet18 using the L1_unstructured method, for pruning 10%, 30% and 50% there is no significant drop

in accuracy compared to the original pre-trained model until pruning 90%, where the accuracy drops to 0.1.

Pros

(1) Simple and easy to implement: It is based on the L1-paradigm of each parameter, which does not need to consider the structural information of the network in the pruning process, so it is suitable for various network structures.

(2) Sparsity: L1-unstructured pruning can achieve a high degree of sparsity, i.e., most of the parameters are set to zero. This can significantly reduce the number of parameters and computation of the model, thus speeding up the inference process.

(3) Interpretability: L1-unstructured pruning preserves the sparse structural information after pruning, which makes the model have better interpretability.

Cons

(1) The L1-unstructured pruning method only prunes based on the L1-paradigm of each parameter and does not consider the correlations between the parameters.

(2) Since L1-unstructured just sets some of the parameters to 0, the size of the model is unchanged.

Overall, L1-unstructured pruning is a simple and effective pruning method to achieve parameter sparsity and model compression.

4.3.3 Quantization

After making dynamic and static quantization, we can obtain the following result:

	VGG16			Resnet18		
Type	**Before**	**Static**	**Dynamic**	**Before**	**Static**	**Dynamic**
Size (mb)	527.81	132.01	174.09	42.73	10.79	42.76
Accuracy (%)	91.22	90.99	91.25	94.15	93.94	94.63
CPU time (%)	100.0	43.62	74.98	100.00	34.58	71.48

Table 4.2 VGG16_and_ResNet18_result

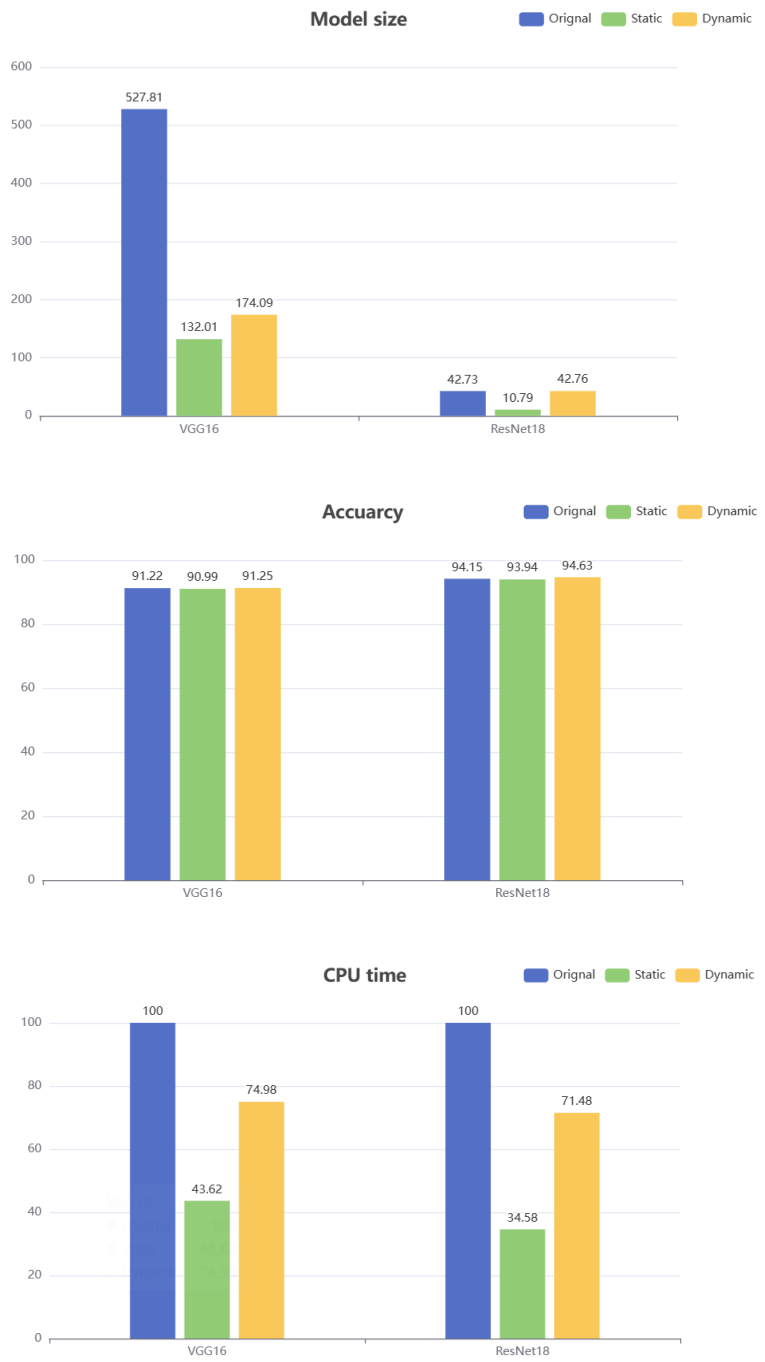


Figure 4.3 Model_size (top one), accuracy (middle one) and CPU_time (bottom one)

The result can be shown as the graph. We can notice that in both model, size have significant decrease comparing to the model without quantization except dynamic quantization for resnet18, we speculate that it may cause by a smaller percentage change of the model weights. The accuracy has slightly change comparing to the original model, but the CPU time has significant decrease for both quantization method. However, the CPU decreasing time for Dynamic quantization method is

relatively low compared to the static quantization method.

4.3.4 Overall Analysis

In terms of quantitative analysis, pruning has the highest retention of model correctness, and we can see that compared to the 90.87% correctness of the baseline VGG, the model with 30% pruning can achieve 90.66% (the correctness for ResNet is only reduced from 94.12% to 94.03%), which is even better than the quantized one. However, in terms of the compression size of the model, the quantized model performs better because its quantized content is more extensive compared to the pruned one. Its dynamic compression and static compression are compressed to 43.62% and 74.98% of the original model respectively, which is relatively consistent with the theoretical reasoning.

5 Conclusion

5.1 Summary

This project focuses on the model pruning and quantization. We first introduced the working principles of the algorithms. In the experiments, we selected CIFAR-10 as the dataset and VGG as the baseline and compared the two algorithms. Furthermore, we qualitatively and quantitatively analyzed the performance of them, and gave their respective applicable scenarios. The results show that pruning compressed the model size by 50.01% and the accuracy dropped by 0.25%, while quantization compressed the model size by 74.99% with the accuracy dropped by 1.10%. That means, in this case, quantization performed better than pruning. At last, we discussed the limitations and the development directions.

5.2 Limitations and improvements

Self-adaptive compression Existing pruning technology requires manual setting of pruning percentage, which makes the algorithm have poor generalization ability and the algorithm is complex. We can achieve automatic adaptation and decision-making by adjusting some hyperparameters.

Unsupervised compression Existing compression methods including pruning and quantization both need labeled data. However, labeled data is sometimes unavailable. Besides, training humans to label requires considerable human effort and professional knowledge[2]. So we can consider to design unsupervised methods.

On-demand compression When given resource constraints (storage, computational power and energy) and user-specified performance goals (accuracy and latency), the model should perform targeted on-demand compression to achieve maximum resource utilization[12].

References

[1] Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282.

[2] Cheng, J., Wang, P. S., Li, G., Hu, Q. H., & Lu, H. Q. (2018). Recent advances in efficient computation of deep convolutional neural networks. Frontiers of Information Technology & Electronic Engineering, 19, 64-77.

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[4] Deng, Y. (2019, May). Deep learning on mobile devices: a review. In Mobile Multimedia/Image Processing, Security, and Applications 2019 (Vol. 10993, pp. 52-66). SPIE.

[5] Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440.

[6] He, Y., Liu, P., Wang, Z., Hu, Z., & Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 4340-4349).

[7] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE international conference on computer vision (pp. 2736-2744).

[8] Meng, F., Cheng, H., Li, K., Luo, H., Guo, X., Lu, G., & Sun, X. (2020). Pruning filter in filter. Advances in Neural Information Processing Systems, 33, 17629-17640.

[9] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from

tiny images.

[10] Choudhary, T., Mishra, V., Goswami, A., & Sarangapani, J. (2020). A comprehensive survey on model compression and acceleration. Artificial Intelligence Review, 53, 5113-5155.

[11] Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., & Ramabhadran, B. (2013, May). Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6655-6659). IEEE.

[12] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

Contributions

Lin Xiaotao (3036213847): Evaluation and Implementation of Quantitative Models, ResNet Network Training;

Teng Yiduo (3036216227): Literature Survey, Quantization Methods Analysis, Comparison of Quantization and Pruning;

Wang Congyi (3036215261): Literature Survey, Pruning Theory Analysis, Baseline Training, Comparison of Quantization and Pruning;

Zhang Ruijie (3036212348): Evaluation and Implementation of Pruning Models, VGG-16 Network Training;

Yao Yang (3036212520): Literature Survey, Compression Theory Research, Introduction, Weakness Analysis, Conclusion and Summary.