A Study of the Computation Amount and Computation Time of Classical Deep Neural Networks

Yihui Cheng, Baiyi Liu
School of Information Engineering and Business Administration, Guangdong Nanhua Vocational College of Industry and Commerce, Guangzhou, Guangdong, China

Abstract: Deep convolutional neural networks have made great progress in a variety of computer vision tasks. With the gradual improvement of its performance, the layers of neural networks become deeper, and the training-validation time and computational complexity increase dramatically. How to find the relationship between characteristics of deep neural networks and their training-validation time is of great significance for accelerating convolutional neural networks. This paper analyzes the computation of several classical deep convolutional neural networks (DNNs) proposed in the field of image recognition, and compares the computation with the total time of actual training and verification. It is found that the computation of a deep neural network is not linearly related to running time of training and verification.

Keywords: Deep Neural Network; Image Classification; Computation; CNN; Computational Complexity

1. Introduction

In 1959, Hubel and Wiesellt[1] was found that animal visual cortex cells were able to respond to bright stimuli in the receptive field. Inspired by this, Fukushima proposed a model of visual cognitive mechanisms called the New Cognitive Machine in 1980[2], which is combined with biological vision theory to arrive at a neural network model.

At the end of the 1980s, the maturity of backpropagation algorithms provided key support for the practical application of CNNs[3]. LeCun et al combine backpropagation algorithms with convolution operations[4]. It was proposed that LetNet-5 network can be well used for handwritten number classification tasks. This model achieves an accuracy rate of more than 99% in the handwritten digit recognition task of bank checks, which verifies the effectiveness of CNN in real-world scenarios for the first time. LetNet-5 network includes convolution layers, pooling layers and full connection layers. The size of the parameters is only about 60,000. The role of convolution is to model the local structure of the features of the previous layer, whereas the role of pooling is to spatially merge similar and adjacent features. The input data at different locations share all the convolution kernels, and each convolution kernel corresponds to a feature map, so the number of feature maps is the same as the number of convolution kernels. The pooling operation only retains the maximum value of each local block in the feature map, which gives the model some translational invariance. The convolution and pooling operations are followed by multiple fully joined operations in order to fuse the extracted data feature map to obtain the final result.

After LeNet-5 was proposed, in the decades, CNN research has fallen into a slump. Since the beginning of the 21st century, with the rise of big data and high-performance computing, the difficulty of CNN training has been gradually solved. The most influential work was AlexNet, designed by Alex Krizhevsky et al. in 2012[5], won the ILSVRC. AlexNet introduced the ReLU activation function to solve the vanishing gradient problem and applied Dropout to prevent overfitting. In addition, AlexNet used GPU parallel training for the first time to break through the computing bottleneck of the CPU and augment the dataset. These improvements once again validated the potential of deep CNN.

The success of AlexNet had sparked an upsurge of exploration of deep CNNs in the academic and industry community and then more CNN network with a deeper network structure were proposed. In 2014, VGGNet constructs a deeper network (16-19 layers) by stacking small-size convolutional kernels (3×3), which proves the effect of network depth on the improvement of feature expression ability[6].

However, as the number of layers increases, the problems of gradient vanishing and training instability become more and more significant. In 2015, ResNet solved this problem by introducing residual connections[7]. The residual structure allows the gradient to bypass some layers and directly backpropagate, making it possible for the network to break through a thousand-layer depth for the first time. Subsequently, GoogleNet[8] proposed the Inception module, which improved the efficiency of feature extraction through multi-scale convolutional kernel parallel computation, and used 1×1 convolutional dimensionality reduction to control the sum of parameter quantities and allow gradients to be directly transmitted back to the shallow layer.

CNN initially focused on image classification, but its application has penetrated into many fields. In computer vision, object detection and image generation, both rely on CNNs as feature extractors. In the medical field, CNN is used for tasks such as breast cancer pathological section analysis and lung CT module detection[9]. In autonomous driving, CNN processes camera and lidar data in real time to complete lane line recognition, pedestrian detection, and path planning[10].

By increasing the depth of the network, CNNs can approximate arbitrarily complex nonlinear mappings from input data to output. At the same time, the deep network structure significantly reduces the computational efficiency of the model and makes it very difficult to train. CNNs are also moving towards lightweighting, explainability, and integration with other models.

2. Compression and Acceleration of CNNs

CNNs have become one of the core models in the field of deep learning due to their excellent performance in computer vision tasks. However, with the increasing complexity of the model, for example, the number of parameters of VGG-16 is as high as 138 million[6]. Its huge storage requirements and computing costs seriously restrict its practical application in mobile terminals, embedded devices, and scenarios with high real-time requirements. This contradiction has given rise to a research boom in model compression and acceleration technology. The goal is to significantly reduce the size, computational complexity and dependence on hardware resources of the model through various means such as algorithm optimization, structural reconstruction and hardware adaptation. On the premise of maintaining the accuracy of the model as much as possible, the implementation of technology was promoted in resource-constrained environments.

2.1 The Technical Methods of Compression and Acceleration

The core idea of model compression and acceleration is to eliminate redundant information in neural networks. This redundancy can exist at multiple levels, such as parameters, computational structures, or data representations. At present, the mainstream compression techniques include pruning, quantization, knowledge distillation, low-rank approximation, and compact architecture design, while the acceleration methods cover two directions: structural optimization and hardware adaptation.

Pruning techniques enable model lightweighting by removing redundant weights or structures from the network. Unstructured pruning filters on a single weight, such as sorting pruning based on the absolute value of the weight[11]. However, due to its sparsity, it is difficult to be effectively used by hardware, and the actual acceleration effect is limited. In contrast, structured pruning is done on a filter or channel basis, or the LASSO regression is used to iteratively select the optimal subset[12]. However, the challenge of pruning is how to balance precision with compression. Excessive pruning may lead to a sharp drop in model performance, and layer-by-layer pruning requires repeated fine-tuning, increasing training costs. In recent years, automated pruning frameworks e.g., AMC have dynamically adjusted pruning rates through reinforcement learning[12], which significantly reduces the complexity of manual parameter tuning.

At the heart of the quantization technique is the conversion of floating-point weights and activation values to low-bit integers, which reduces memory footprint and accelerates computation. For example, 8-bit quantization can reduce the size of the model to 1/4 of the original size, while achieving a 3x increase in inference speed on hardware that supports fixed-point operations, such as FPGAs[13]. Quantization is divided into Post-training Quantization and Quantization Perception Training (QAT). The former directly transforms the pre-trained model, which is suitable for rapid deployment but sensitive to accuracy loss. The

latter simulates the quantization error during training and optimizes the weight distribution through backpropagation, thus maintaining higher accuracy at low bits. Extreme quantization, such as a 1-bit binary network, can greatly compress the model[14], but this results in a significant drop in precision, so it is often necessary to dynamically allocate the bit widths of different layers in conjunction with a mixed-precision strategy. For example, ResNet-50 achieves a 2.7x speedup with a 1.2% loss of accuracy by mixing 4/8-bit quantization[15].

Knowledge distillation transfers knowledge from a large teacher model to a small student model through transfer learning. Traditional distillation methods focus on the probability distribution of the output layer[16], while the advanced method introduces the matching of intermediate feature maps, such as the activation map that uses the attention mechanism to align the teacher and student models[17]. For example, TinyBERT compresses the number of parameters of the BERT model from 110 million to 14 million through multi-layer feature distillation, improves the inference speed by 9.4 times, and maintains 92% of the original performance in the GLUE benchmark task[18]. However, the effect of knowledge distillation is highly dependent on the quality of the teacher model, and the training process needs to load both large and small models at the same time, which has high requirements for computing resources.

Low-rank decomposition is based on the theory of matrix factorization, which disassembles the high-dimensional convolution kernel into the product of multiple low-dimensional matrices[19], which can be approximated as the product of two small matrices, reducing the number of parameters. Although low-rank decomposition can effectively compress the model theoretically, it has poor adaptability to nonlinear activation functions such as ReLU and may introduce large reconstruction errors after decomposition. As a result, this method is often used in combination with other techniques, such as quantification, to compensate for the limitations of a single method.

The compact network design enables efficient computation at the model architecture level. For example, the MobileNet series uses Depth-wise Separable Convolution, which splits the standard convolution into two steps: channel-by-channel convolution and 1x1 point convolution, reducing the amount of computation[20]. ShuffleNet breaks the information isolation of group convolution through channel shuffle, which further reduces the computational cost while ensuring accuracy[21]. This type of design often requires a combination of a large amount of experimental and domain knowledge, and the emergence of neural architecture search (NAS) provides a new way of thinking about the automated design of efficient models. For example, EfficientNet achieves a significant increase in accuracy with the same compute budget by using a composite scaling strategy that adjusts depth, width, and resolution simultaneously.

2.2 Acceleration Strategy and Hardware Co-Optimization

On the basis of algorithm optimization, hardware adaptation has become the key to accelerated inference. Structural optimization includes methods such as cascaded pruning and dynamic calculation path selection. For example, the inference latency of ResNet-50 can be reduced by 15% by removing redundant residual connections or branching structures. Mixed-precision quantization dynamically allocates the number of bits according to the sensitivity of each layer to the quantization error. For example, 8-bit quantization is used for the underlying feature map, while 16-bit precision is reserved for the classification layer, so as to balance speed and accuracy overall.

Hardware acceleration relies on the extreme use of computing resources. GPUs accelerate convolution operations through parallel computing, while TPU is customized optimized for matrix multiplication. On mobile, the ARM NEON instruction set accelerates 8-bit integer operations with framework-level optimization and implement real-time inference on the device side. The dedicated AI chip, Google Edge TPU, further unleashes the potential of quantization and pruning through hardware-algorithm co-design. For example, deploying a 4-bit quantized MobileNet-V2 on an FPGA can achieve inference speeds up to 20 times faster than CPUs while reducing power consumption to 1/10.

3. The Amount of Computation of the Neural Network

Besides the accuracy of a model, the spatial complexity and temporal complexity of the model also need to be considered to evaluate the performance of deep neural networks. The

spatial complexity corresponds to the amount of the parameter, the time complexity corresponds to the amount of computation. This study only focus on the amount of computation.

The amount of computation can be measured by two performance indicators, i.e. floating-point operations (FLOPs) and multiply–accumulate operations(MACs)[21]. One MAC has one multiplication operation and one addition operation. Since addition can be performed in parallel, here we only consider the number of multiplications and ignore addition. FLOPs include addition, subtraction, multiplication etc. FLOPs can be approximated by multiplying MACs by 2.

K is the size of the convolution kernel, C_{out} is the number of output channels, C_{in} is the number of input channels, W_{out} is the width of the output feature map, H_{out} is the height of the output feature map, and g is the number of groupings of the group convolution.

(1) For the fully connected layer, its MAC is calculated as follows:

$$MACs = C_{in}C_{out} \quad (1)$$

(2) For a traditional convolutional layer, its MAC is calculated as follows:

$$MACs = H_{out}W_{out}C_{in}K^2C_{out} \quad (2)$$

(3) For a group convolutional layer, its MAC is calculated as follows:

$$MACs = \frac{H_{out}W_{out}C_{in}K^2C_{out}}{g} \quad (3)$$

(4) For a deeply separable convolutional layer, its MAC is calculated as follows:

$$MACs = H_{out}W_{out}K^2C_{out} \quad (4)$$

According to the above calculation methods, the FLOPs of the five common neural network models are summarized in Table 1.

Table 1. FLOPs of Five Common Neural Network Models

model	FLOPs (G)
Alex	1.42
VGG16	30.94
Googlenet	3.02
ResNet18	3.64
DenseNet	15.7

4. Experiments

4.1 Experimental Environment

All experiments were run on a workstation equipped with an Intel Core Processor Core(TM) i7-13700K @3.40 GHz, 96G DDR5 RAM, NVIDIA GeForce RTX 4080. The operating system is Windows 11 Professional Edition, and the software runtime environment is Python 3.9, Pytorch 2.1.0, and cuda12.2.

4.2 Datasets

FashionMNIST was used in this experimental which has a total of 70,000 images, including 10,000 test images and 60,000 training images. Each image is a 28x28 grayscale image. It covers 10 different categories of products.

4.3 Experimental Data

Five common classical neural networks were used in the experiment: AlexNet, VGG, GoogleNet, ResNet, and DenseNet. During training, all parameters of several models, including num_epochs, batch_size, and lr parameters, are the same. The total duration of training and validation is recorded. Table 2 shows the total duration of the actual training verification of the five classical neural networks.

Table 2. Running Time

model	Running time(s)
Alex	230.77
VGG16	454.72
GoogleNet	587.38
ResNet18	522.82
DenseNet	571.19

4.4 Experimental Result

As shown in Figure 1, the running time and FLOPs are compared in the experiment. We can see from Figure 1, for the same computing platform, the same dataset, and the same hyperparameter settings, the total time of training and validation does not increase linearly with the increase of FLOPs.

Figure 1. Running Time and FLOPs

5. Conclusions

The more computationally intensive a neural network, the longer the total time spent on training and validation. What is the minimum amount of computation required for a neural network for a particular task? This question is worthy of further research and exploration.

Acknowledgments

This paper is supported by Qingyuan City Industry-Education Integration Social Science Special Project. Project Name (No. ZJCYJY202451): Research on the cultivation of employment ability and industry adaptation for students in Qingyuan Vocational College under the background of industry education integration - taking information technology majors as an example.

References

[1] Hubel, D.H. and Wiesel, T.N., 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology, 160(1), p.106.

[2] Fukushima, K., 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological cybernetics, 36(4), pp.193-202.

[3] Rumelhart, D.E., Hinton, G.E. and Williams, R.J., 1986. Learning representations by back-propagating errors. nature, 323(6088), pp.533-536.

[4] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324.

[5] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[6] Simonyan, K., and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” 3rd International Conference on Learning Representations (ICLR 2015), Computational and Biological Learning Society, 2015, pp. 1–14.

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

[8] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[9] Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M. and Thrun, S., 2017. Dermatologist-level classification of skin cancer with deep neural networks. nature, 542(7639), pp.115-118.

[10] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J. and Zhang, X., 2016. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[11] Han, S., Mao, H. and Dally, W.J., 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

[12] He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J. and Han, S., 2018. Amc: Automl for model compression and acceleration on mobile devices. In Proceedings of the European conference on computer vision (ECCV) (pp. 784-800).

[13] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H. and Kalenichenko, D., 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

[14] Rastegari, M., Ordonez, V., Redmon, J. and Farhadi, A., 2016, September. Xnor-net: Imagenet classification using binary convolutional neural networks. In European conference on computer vision (pp. 525-542). Cham: Springer International Publishing.

[15] Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P. and Keutzer, K., 2018. Mixed precision quantization of convnets via differentiable neural architecture search. arXiv preprint arXiv:1812.00090.

[16] Hinton, G., Vinyals, O. and Dean, J., 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

[17] Zagoruyko, S., 2018. Weight parameterizations in deep neural networks (Doctoral dissertation, Université Paris-Est).

[18] Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F. and Liu, Q., 2019. Tinybert: Distilling bert for natural language understanding. arXiv preprint arXiv:1909.10351.

[19]Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y. and Fergus, R., 2014. Exploiting linear structure within convolutional networks for efficient evaluation. Advances in neural information processing systems, 27.

[20]Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[21]Zhang, X., Zhou, X., Lin, M. and Sun, J., 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 6848-6856).

