Auto-tuning Neural Network Quantization Framework for Collaborative Inference Between the Cloud and Edge

Guangli Li[1,2], Lei Liu[1(✉)], Xueying Wang[1,2], Xiao Dong[1,2], Peng Zhao[1,2], and Xiaobing Feng[1]

[1] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{liguangli,liulei,wangxueying,dongxiao,zhaopeng,fxb}@ict.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, China

Abstract. Recently, deep neural networks (DNNs) have been widely applied in mobile intelligent applications. The inference for the DNNs is usually performed in the cloud. However, it leads to a large overhead of transmitting data via wireless network. In this paper, we demonstrate the advantages of the cloud-edge collaborative inference with quantization. By analyzing the characteristics of layers in DNNs, an auto-tuning neural network quantization framework for collaborative inference is proposed. We study the effectiveness of mixed-precision collaborative inference of state-of-the-art DNNs by using ImageNet dataset. The experimental results show that our framework can generate reasonable network partitions and reduce the storage on mobile devices with trivial loss of accuracy.

Keywords: Neural network quantization · Auto-tuning framework
Edge computing · Collaborative inference

1 Introduction

In recent years, deep neural networks (DNNs) [14] are widely used and show impressive performance in various fields including computer vision [12], speech recongnition [9], natural language processing [15], etc. As the neural network architectures become more complex and deeper—from LeNet [13] (5 layers) to ResNet [8] (152 layers), the storage and computation of the model is increasing. In other words, it leads to more resource requirements for network training and inference. The large size of DNN models limits the applicability of the network inference on mobile edge devices. Therefore, most of artificial intelligence (AI) applications on mobile devices send input data of DNN to cloud servers, and the procedure of network inference is executed in the cloud only. However, the cloud-only inference has some assignable weaknesses: (1) transmission overhead:

V. Kůrková et al. (Eds.): ICANN 2018, LNCS 11139, pp. 402–411, 2018.
https://doi.org/10.1007/978-3-030-01418-6_40

it leads to a large overhead of uploading data especially when the mobile edge devices are in the low-bandwidth wireless environments. (2) privacy disclosure: sometimes, personal data, e.g. one's photos and videos, are not allowed to send to the cloud servers directly.

Today's mobile devices, such as Apple's iPhone and NVIDIA's Jetson TX2, have more powerful computability and larger memory. In addition, many neural network quantization methods [3,4,7,18,19] have been proposed for reducing the resource consumption of DNNs. By using quantization, the data of a network can be represented by low-precision values, e.g. INT8 (8-bit integer). On the one hand, low-precision data reduces storage of DNNs and enables network models to be stored on the mobile edge device with limited resources. On the other hand, with the use of high-performance libraries for low-precision computing [1,2], the speed of the network inference will be improved. This makes it possible to perform some or all parts of neural network inference on mobile devices and leads to a new inference mode: cloud-edge collaborative inference.

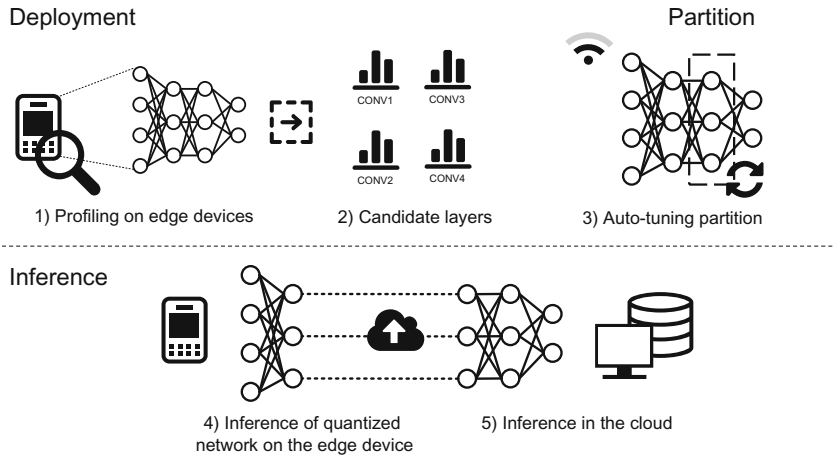

Fig. 1. Overview of auto-tuning framework

In this paper, we propose an auto-tuning neural network quantization framework as shown in Fig. 1. During deployment, the framework profiles the operators of DNNs on edge devices and generates the candidate layers as partition points. When the neural network is ready to be used, the framework starts auto-tuning for network partition. In the time of inference, the first part of the network is quantized and executed on the edge devices, and the second part of the network is executed in the cloud servers. On the edge, we use quantized neural network to reduce storage and computation. In the cloud, we use original full-precision network to achieve high accuracy.

In the collaborative inference, quantized neural networks can reduce the storage of models. Intermediate results of quantized networks are also low-precision

data, which can reduce data communication between cloud and edge. So user's mobile device could transmit less data when using AI applications. Additionally, transmitting intermediate result data, rather than the original input data, can protect personal information. In realistic scenarios, the process of analysis and testing is tedious and time-consuming. It's unfriendly for a program developer to test and decide how to partition the network. Our automatic tuning framework will help developers find the most reasonable partition of a DNN. The contributions of this paper are summarized as follows:

- *Analysis of DNN partition points* – We analyze the structures of deep neural networks and show which layers are reasonable partition points. Based on the analysis, we could generate candidate layers as partition points of a specific neural network (Sect. 2.2)
- *Auto-tuning quantization framework for collaborative inference* – We develop an auto-tuning neural network quantization framework for collaborative inference between cloud and edge. The framework quantizes neural networks according to the candidate partition points and provides an optimal mixed-precision partition for cloud-edge inference by auto-tuning (Sect. 2.3).
- *Experimental study* – We show the performance of collaborative inference of state-of-the-art DNNs by using ImageNet dataset. The framework generates reasonable network partitions and reduces the storage of inference on mobile devices with trivial loss of accuracy (Sect. 3).

2 Auto-tuning Quantization Framework

In this section, we present our auto-tuning neural network quantization framework. Firstly, we briefly introduce neural network quantization. Secondly, we analyze the structures of the state-of-the-art DNNs. Finally, we describe the auto-tuning partition algorithm.

2.1 Neural Network Quantization

In order to accelerate inference and compress the size of DNN models, many network quantization methods are proposed. Some studies focus on scalar and vector quantization [4,7], while others center on fixed-point quantization [18,19]. In this paper, we are mainly interested in scalar quantization of INT8, which is supported by many advanced computing libraries such as Google's gemmlowp [1] and NVIDIA's cuDNN [2]. In general, an operator computation of scalar quantized neural networks can be summarized as follows:

- Off-line Quantization
  Step 1. Find quantization thresholds (T_{min} and T_{max}) for calculating scale factors of $Input$, $Weights$ and $Output$;

Step 2. Quantize $Input$ and $Weights$ according to the following formula:

$$Data_Q(x) = \begin{cases} \dfrac{Data(x) - T_{min}}{|T_{max} - T_{min}|} \times Range_{LP} & x \in (T_{min}, T_{max}) \\ \|V_{low-precision}\|_{\infty} & x \geq T_{max} \\ \|V_{low-precision}\|_{-\infty} & x \leq T_{min} \end{cases} \tag{1}$$

where: $Range_{LP}$ is the range of low-precision values (e.g. 255 for INT8), $V_{low-precision}$ is the set of low-precision values, $Data(x)$ is the original value, $Data_Q(x)$ is the quantized value.

- On-device Computation
  Step 1. $Output_Q$ = Operator($Input_Q$, $Weights_Q$);
  Step 2. Dequantize $Output_Q$ according to the following formula:

$$Output = \frac{|T_{max} - T_{min}|}{Range_{LP}} \times Output_Q(x) + T_{min} \tag{2}$$

  Step 3. $Output$ = ActivationFunction($Output$);
  Step 4. Quantize $Output$ as $Input_{Next}$ according to Formula 1.

2.2 Candidate Network Partition Points

In general, a deep neural network contains many kinds of layers such as convolution layers, fully-connected layers and activation layers. We analyze the characteristics of different network layers and decide how to select candidate layers as reasonable partition points. The set of candidate layers, $Rule = \{L_1, L_2, \ldots, L_n\}$, is based on the results of the following analysis.

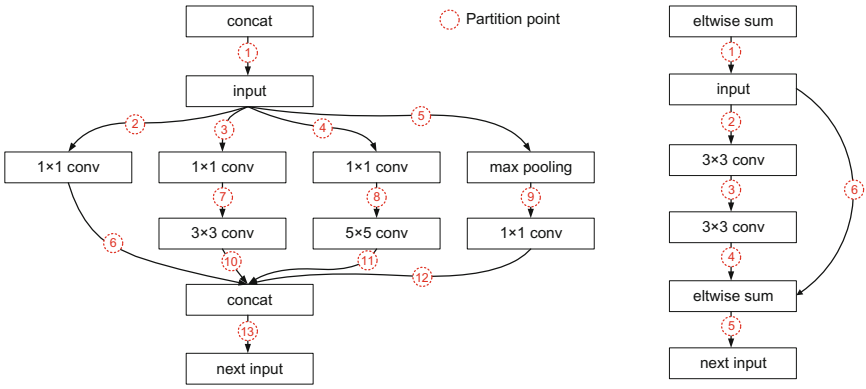

Fig. 2. Partition points of DNNs

Table 1. Analysis of inception

Partition points	Brother branch exists?	Inference mode of the brother branch	Data transmission
1, 13	No	/	INT8 × 1
2, 3, 4, 5 7, 8, 9 6, 10, 11, 12	Yes	Mobile edge	INT8 × 4
2, 3, 4, 5 7, 8, 9 6, 10, 11, 12	Yes	Cloud	INT8 × 1 + FP32 × 1

Layers in Inception Networks. Inception is a structure that contains branches, and these branches are executed in parallel and their results are merged into a network layer (e.g. concat layer). Figure 2(a) is an example of inception from GoogLeNet [17]. As shown, the inception contains 13 possible partition points. If we try all the partition points, it will take a lot of time. We divide these partition points into two groups according to whether they have at least a brother branch (separate from the same layer and merge in the same layer). The results of the analysis are shown in Table 1. When a partition point has no brother branch (e.g. 1 and 13), the output of the sub-network on edge devices contains only 1 × INT8 Blob (4D array for storing data). When a partition point has a brother branch, there are two cases: (1) its brother branch runs on the edge devices, and the sub-network output contains 4 × INT8 Blobs; (2) its brother branch runs in the cloud, and the sub-network output contains 1 × INT8 Blob and 1 × FP32 Blob. The transmission data in first group is smaller than it is in the second group. Therefore, if a network layer in inception has a brother branch, the framework will not choose it as a candidate layer.

Table 2. Analysis of residual network

Partition points	Shortcut connection exists?	Data transmission
1, 5	No	INT8 × 1
2, 3, 4, 6	Yes	INT8 × 1 + FP32 × 1

Layers in Residual Networks. There are many shortcut connections in the residual network [8]. Figure 2(b) shows an example of a residual block which contains a shortcut connection. There are 6 possible partition points in this example. According to whether the shortcut connection of a partition points exists, we divide these partition points into two groups. When a partition point has no shortcut connection (e.g. 1 and 5), the output of the sub-network on edge devices contains only 1 × INT8 Blob. Otherwise, the output of the sub-network

contains 1 × INT8 Blob and 1 × FP32 Blob. Table 2 shows the analysis result. Therefore, the network layers with shortcut connections are not reasonable candidate layers.

Non-parametric Layers. Non-parametric layers, such as ReLU and pooling, have no parameters, so they require almost no memory storage. In addition, the computation of the non-parametric layers accounts for a very small proportion of the total network computation. Therefore, our framework merges the non-parametric layers into the nearest previous parametric layers, i.e. these non-parametric layers will not be used as candidate layers.

2.3 Auto-Tuning Partition

According to the candidate rule *Rule*, the framework performs auto-tuning partition for cloud-edge collaborative inference, as described in Algorithm 1. The input of the algorithm contains candidate layer rules and a neural network. Firstly, candidate rules are used to select candidate partition points in the neural network (lines 1–2). Secondly, all candidate partition networks are tested, and the information of performance is recorded in P (lines 3–9). The function of $PredictPerformance$ can predict the performance of collaborative inference based on the results of off-line profiling. Finally, we find the best partition point in P for collaborative inference of mixed-precision neural network (lines 10–14).

Algorithm 1. Auto-Tuning Partition

Input: candidate rules *Rule*, neural network $Net = \{L_1, L_2, \ldots, L_n\}$
Output: optimize partition p_{best}

1 $P \leftarrow \Phi$; $p_{best} \leftarrow null$;
2 $Candidate \leftarrow \{L_i | L_i \in Rule\}$;
3 **for** L_i *in* $Candidate$ **do**
4 $\quad Net_{edge} \leftarrow Net.Split(First, L_i)$;
5 $\quad Net_{Cloud} \leftarrow Net.Split(L_i + 1, Last)$;
6 $\quad Engine_{Edge} \leftarrow Net_{Edge}(DataType_{<INT8>})$;
7 $\quad Engine_{Cloud} \leftarrow Net_{Cloud}(DataType_{<FP32>})$;
8 $\quad (L_i, info) \leftarrow PredictPerformance(Engine_{Edge}, Engine_{Cloud})$;
9 $\quad P \leftarrow P \cup (L_i, info)$;
10 $Env = GetEnvironment(Device_{Edge})$;
11 **for** p_i *in* P **do**
12 $\quad$ **if** $Env(p_i)$ *is better than* $Env(p_{best})$ **then**
13 $\quad\quad p_{best} \leftarrow p_i$;
14 return p_{best};

3 Experiments

In this section, we use ImageNet [6] dataset to test the collaborative inference of DNNs [8,12,16,17] and show results of our auto-tuning framework. We illustrate the most reasonable partition for each neural network. The inference of the edge performs on a mobile platform – NVIDIA Jetson TX2 (NVIDIA's latest mobile SoC) – with 4 × ARM Cortex-A57 CPUs and 2 × Denver CPUs, 8G of RAM. The inference of the cloud performs on a server with Intel Core-i7 CPU, NVIDIA TITAN Xp GPU, 16G of RAM. We use Caffe [10] with cuDNN (version 7.0.5) on the GPU of cloud servers. We use gemmlowp's [1] implementation on the CPU of the edge devices.

3.1 Experimental Results

Table 3 summarizes the results of our framework. We tested AlexNet, VGG16, ResNet-18 and GoogLeNet in different wireless network environments. For each neural network, the framework gives the best partition point and the fastest partition point. According to the inference time and the speed-up in the table, we can see that sometimes the speed of collaborative inference is faster than that of the cloud inference only. This is due to the large transmission overhead in the low-bandwidth wireless environments. In collaborative inference, we only need to download the parameters required by the edge inference, which can significantly reduce the size of download data. If users need to achieve the fastest inference speed, the fastest partition point should be selected. If users need to avoid privacy disclosure, the best partition point should be selected. In addition, quantized neural networks do not lead to a significant drop in accuracy (usually less than 1%).

Table 3. Experimental results of our framework

Neural network	AlexNet	VGG16	ResNet-18	GoogLeNet
Wireless upload (KB/s)	250	240	70	180
Best partition point	conv5	conv1_2	res4a	conv2
Inference time (s)	0.36	5.65	1.86	1.16
Speed-up	1.7×	<1×	1.13×	<1×
Model download (KB)	2278	38	1569	121
Model storage reduction	96.17%	99.97%	85.63%	98.22%
TOP-1 accuracy↓	−0.09%	0.00%	−0.19%	−0.10%

Figure 3 shows the collaborative inference time of each candidate layer in the wireless network environments. We take AlexNet as an example. Each bar represents a network partition, which consists of three parts: edge inference, data upload and cloud inference. After auto-tuning of framework, conv5 layer is

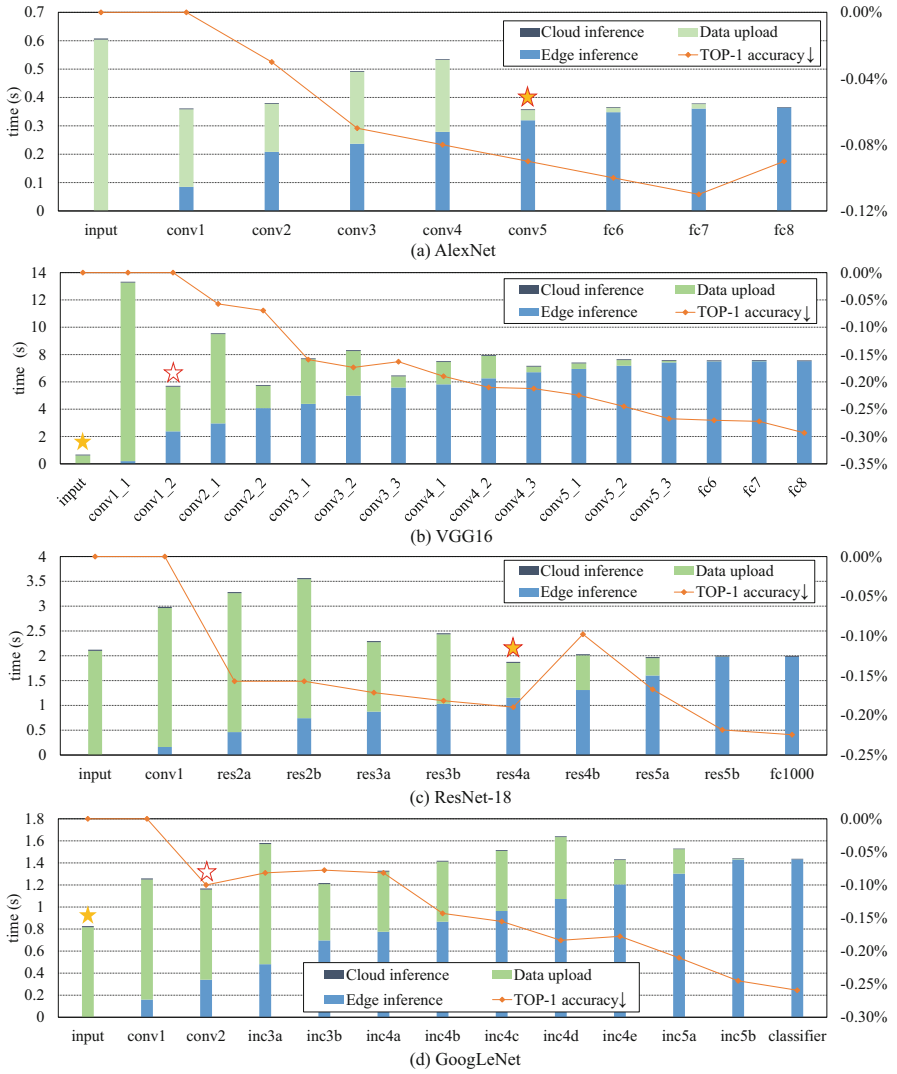

Fig. 3. Performance of each DNN partition

selected as the best partition point (marked with a hollow pentagram) and the fastest partition point (marked with a filled pentagram). On edge devices, we feed input data to the neural network and perform inference of layers from conv1 to conv5. The output data of conv5 (pool and relu are merged) is uploaded to the cloud, and then the inference of layers from fc6 to fc8 is executed in the cloud. The approach of collaborative inference achieves 1.7× speed-up. It can be seen that the accuracy drop of the network is trivial, and the largest accuracy loss in all partitions is only −0.11%.

4 Related Work

Recently, many neural network quantization methods have been proposed. Gong et al. [7] and Cheng et al. [4] explored scalar and vector quantization methods for compressing DNNs. Zhou et al. [18], Zhou et al. [19] proposed fixed-point quantization methods. Cuervo et al. [5] and Kang et al. [11] designed frameworks that support collaborative computing of mobile applications. Their frameworks perform off-line partition for full-precision neural networks, and ours performs on-line partition for mixed-precision neural networks. Overall, the application of quantization methods in cloud-edge collaborative inference has not been studied yet. To the best of our knowledge, it is the first attempt to build framework for cloud-edge collaborative inference of mixed-precision neural networks.

5 Conclusion

In this paper, we propose an auto-tuning neural network quantization framework for collaborative inference. We analyze the characteristics of network layers and provide candidate rules to choose reasonable partition points. The auto-tuning framework helps developers get the most suitable partition of a neural network. The cloud-edge mode (i.e. collaborative inference) reduces the storage of inference on mobile devices with trivial loss of accuracy and could protect personal information.

Acknowledgement. This work is supported by the National Key R&D Program of China under Grant No. 2017YFB0202002, the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under Grant No. 61521092 and the Key Program of National Natural Science Foundation of China under Grant Nos. 61432018, 61332009, U1736208.

References

1. gemmlowp: a small self-contained low-precision GEMM library. https://github.com/google/gemmlowp
2. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt
3. Cheng, J., Wang, P., Li, G., Hu, Q., Lu, H.: Recent advances in efficient computation of deep convolutional neural networks. CoRR abs/1802.00939, pp. 1–12 (2018). http://arxiv.org/abs/1802.00939
4. Cheng, J., Wu, J., Leng, C., Wang, Y., Hu, Q.: Quantized CNN: a unified approach to accelerate and compress convolutional networks. IEEE Trans. Neural Netw. Learn. Syst. **99**, 1–14 (2017)
5. Cuervo, E., et al.: MAUI: making smartphones last longer with code offload. In: International Conference on Mobile Systems, Applications, and Services, pp. 49–62 (2010)
6. Deng, J., et al.: ImageNet: a large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, pp. 248–255. IEEE Computer Society (2009)

7. Gong, Y., Liu, L., Yang, M., Bourdev, L.D.: Compressing deep convolutional networks using vector quantization. CoRR abs/1412.6115, pp. 1–10 (2014). http://arxiv.org/abs/1412.6115
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Computer Vision and Pattern Recognition, pp. 770–778 (2015)
9. Hinton, G., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
10. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. In: ACM International Conference on Multimedia, pp. 675–678 (2014)
11. Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., Tang, L.: Neurosurgeon: collaborative intelligence between the cloud and mobile edge. ACM Sigplan Not. **52**(4), 615–629 (2017)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
13. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
14. Lecun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
16. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556, pp. 1–14 (2014). http://arxiv.org/abs/1409.1556
17. Szegedy, C., et al.: Going deeper with convolutions. In: Computer Vision and Pattern Recognition, pp. 1–9 (2015)
18. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: towards lossless CNNs with low-precision weights. CoRR abs/1702.03044, pp. 1–14 (2017). http://arxiv.org/abs/1702.03044
19. Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., Zou, Y.: DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR abs/1606.06160, pp. 1–13 (2016). http://arxiv.org/abs/1606.06160