Efficient Deep Learning Inference based on Model Compression

Qing Zhang, Mengru Zhang, Mengdi Wang, Wanchen Sui, Chen Meng, Jun Yang
Alibaba Group
{sensi.zq, mengru.zmr, didou.wmd, wanchen.swc, mc119496, muzhuo.yj}@alibaba-inc.com

Weidan Kong, Xiaoyuan Cui, Wei Lin
Alibaba Group
{weidan.kong, xiaoyuan.cui, weilin.lw}@alibaba-inc.com

Abstract

Deep neural networks (DNNs) have evolved remarkably over the last decade and achieved great success in many machine learning tasks. Along the evolution of deep learning (DL) methods, computational complexity and resource consumption of DL models continue to increase, this makes efficient deployment challenging, especially in devices with low memory resources or in applications with strict latency requirements. In this paper, we will introduce a DL inference optimization pipeline, which consists of a series of model compression methods, including Tensor Decomposition (TD), Graph Adaptive Pruning (GAP), Intrinsic Sparse Structures (ISS) in Long Short-Term Memory (LSTM), Knowledge Distillation (KD) and low-bit model quantization. We use different modeling scenarios to test our inference optimization pipeline with above mentioned methods, and it shows promising results to make inference more efficient with marginal loss of model accuracy.

1. Introduction

DNN has become the main framework in various applications, such as computer vision [27, 16, 13], natural language processing [29, 19], speech recognition [1], *etc.* DNN can outperform many conventional machine learning methods, due to the increased amount of training data, more powerful computing resources as well as dramatically increased model parameters. However when we deploy DNN model in resource limited platforms, *e.g.* mobile systems, or latency-sensitive applications, *e.g.* online services, the main blocking issues are high computation complexity and huge model storage size. For example, ResNet [5] needs 25.5 MB for model weight storage and 4.1 billion float point operations (FLOPs) to classify a single image, when the image spatial size is 224×224. As a result, reduction of the model size and acceleration of the inference time gain more and more attention in recent years. Newly proposed network structures, including DenseNet [10] and ResNeXt [32], already take the resource issue into consideration, however the gain is not enough. More inference optimization procedures over existing networks are proposed based on different frameworks. From perspective of algorithm, the inference optimization methods can be categorized into two classes: 1) *reducing the number of model parameters* and 2) *reducing the model representation precision.*

The methods of reducing the number of model parameters aim to get a more compact network, they can be categorized into low rank factorization, weight pruning, KD and new network architecture designs.

TD belongs to low-rank factorization based techniques, which use tensor decomposition to estimate the informative parameters of the deep models. Convolutional kernels can be viewed as a 4D tensor, which is of significant amount of redundancy. Regarding to the fully-connected layer, it can be viewed as a 2D matrix and the low-rankness can also help. TD aims to reduce the network FLOPs by decomposing a large filter into several small tensors by Canonical Polyadic (CP) decomposition [17] or Tucker decomposition [14].

Weight pruning can reduce the model size by removing some redundant parameters. It is firstly proposed by the work optimal brain damage [18], which uses Taylor expansion to estimate the influence of each weight on the total loss, referred as weight saliency. The low-saliency weights are pruned and the remained weights are finetuned to maintain the original accuracy. For static neural networks such as Convolutional Neural Network (CNN), the work in [3] employs the magnitude of the weights to evaluate the weight importance and determine which parameters should be removed. This kind of pruning (fine-grained pruning) needs dedicated compute libraries or/and hardware design, such as EIE [3] and SCNN [24]. It will be difficult to be applied by users who do not have such accelerator support. More works explore to find structural pruning, which can get practical speed-up over existing compute libraries, such as Compute Unified Device Architecture (CUDA) and CUDA Deep Neural Network library (cuDNN). Works in [21, 6] prune the weights offline at filter-level using LASSO regression, a finetuning procedure is conducted to compensate the performance loss. Other methods like [31, 20] conduct sparsity guidance while training. [31] uses group sparsity regularization filter-wise or channel-wise, and Network Slimming (NS) [20] conducts ℓ_1-norm on the scaling factors of Batch Normalization (BN) layers. However, such sparsity-induced weight pruning methods usually ignore the network topology, so that additional post-processing layer may be needed to deal with complex network structure, e.g. cross-connections. In order to address this issue, we propose GAP, which is capable of adapting to different network structures, especially the widely used cross connections and multi-path data flow in recent novel convolutional models. The GAP can adaptively prune the models at vertex-level as well as edge-level without any post-processing and it does not need any customized computation library or hardware support. In terms of dynamic neural networks like LSTM, Narang *et al.* [23] use RNN connection pruning techniques to compress model size of Deep Speech 2 [1]. However, little work has been carried out to reduce coarse-grain structures rather than fine-grain connections in RNNs. ISS for LSTM can learn to reduce the number of basic structures within LSTM units. After learning those structures, final LSTMs are still regular LSTMs with the same connectivity, but have the reduced sizes [30].

Knowledge Distillation is proposed by Hinton *et al.* to guide the student network training by a pretrained teacher model using soft target [7]. The method aims to transfer the knowledge from a complex teacher model to the student network. FitNet [25] extends the method by distilling the knowledge not only in the output but also the intermediate representations. [33] proposed to transfer not the features in each layer but the flow between layers. In such a way, the teacher explains the solution process of a problem and the student learns the flow of solution procedure.

Some researches explore new network architecture to get the inference-efficiency at the beginning of network design, such as SqueezeNet [12] and MobileNet [9] *etc*. The main technique is to replace the large convolution filters by a stack of small filters and train the network end-to-end. Since the goal of our inference optimization pipeline is to compress the pretrained models and accelerate inference procedure, the new network architecture will not be considered in this work.

Reducing the model representation precision is equivalent to the network quantization, which compresses the bitwidth of the weights, activations or both. [2] quantizes the weights value through k-means clustering and product quantization, which achieves significant model size and FLOPs reduction with less than 1% accuracy loss. Han *et al.* further proposed to use Huffman coding to represent the quantized weights as well as the codebook to improve the compression rate [4]. [28] shows that it is possible to achieve great speed-up with no-loss in accuracy using 8-bit quantization. Extreme quantization is to binarize the network [11], using 1-bit to represent a value. Such kind of works using fixed-point or binary representation need specially designed compute acceleration library or hardware.

In this paper, we propose a pipeline that optimizes DL inference. Given a pre-trained model , we can use TD/GAP to optimize CNN parameters, followed by a self-taught KD procedure to maintain the accuracy during finetuning. ISS can be used to prune the parameters in LSTM cells. Since some NVIDIA GPU devices have been capable of executing 8-bit integer 4-element vector dot product instructions to accelerate DNN inference, we implement 8-bit quantization to reduce the model representation precision and optimize DL inference.

2. Methods

2.1. TD method

The motivation behind low-rank decomposition is to find an approximate tensor that is close to convolutional kernel but facilitates more efficient computation. Among many low-rank based methods, two key differences are in how to rearrange the four dimensions, and on which dimension the low-rank constraint is imposed. Convolutional kernel can be viewed as a 4D tensor, its shape is $D_k \times D_k \times M \times N$, where D_k is the filter size, M is the number of input channel and N is the number of output channel. We rearrange the 4D convolution kernel and obtain M 3D kernels, each of which is size of $D_k \times D_k \times N$. And then we adopt CP factorization, decomposing each 3D tensor into a summation of R rank-1 tensors as in Figure 1, that is $X_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr} + e_{ijk}$, where e_{ijk} represents reconstruction error.

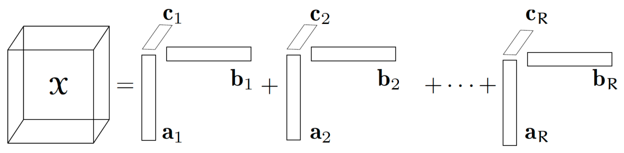

Figure 1. The CP tensor decomposition.

Based on this, we can conduct TD for convolutional kernels and replace a standard convolution operation with depthwise and pointwise convolutions as Figure 2.

When we choose to keep only one channel in the depth-

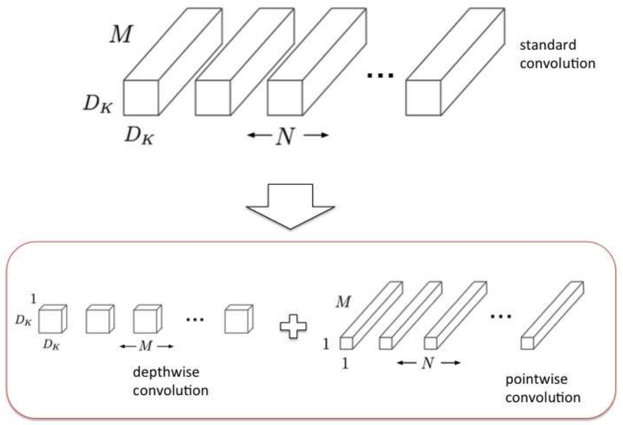

Figure 2. From standard convolution to depthwise convolution and pointwise convolution.

wise convolution, we can compute the ratio of computational cost before and after TD as (1):

$$\frac{D_k \cdot D_k \cdot M \cdot D_f \cdot D_f + M \cdot N \cdot D_f \cdot D_f}{D_k \cdot D_k \cdot M \cdot N \cdot D_f \cdot D_f} = \frac{1}{N} + \frac{1}{D_k \cdot D_k}, \tag{1}$$

where D_f is the size of input feature map.

2.2. GAP for CNN

For convolution, we use $\mathcal{X}$, $\mathcal{W}$, $\mathcal{Z}$ to denote the input feature maps, convolution kernels and output feature maps, respectively. Each channel $\mathbf{Z}_i$ in the output feature maps corresponds to a filter $\mathcal{W}_i$, and the batch normalized result is represented by $\hat{\mathbf{Z}}_i$,

$$\mathbf{Z}_i = \mathcal{X} * \mathcal{W}_i, \;\; \hat{\mathbf{Z}}_i = \frac{\mathbf{Z}_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \cdot \gamma_i + \beta_i, \tag{2}$$

where μ_i and σ_i^2 are mean and variance of the channel, γ_i and β_i are the scaling factor and bias factor, respectively.

We use a symbol W to represent all the parameters in CNN, including $\{\mathcal{W}\}$, $\{\gamma\}$, $\{\beta\}$ and also the other parameters, such as those in the FC layer. W can be learned by the following optimization,

$$\min_W f(\mathcal{I}, W) + R(W), \tag{3}$$

where, $\mathcal{I}$ denotes the input pairs including data and labels, $f(\cdot)$ is the loss function, $R(\cdot)$ is the regularization used in the training process.

We use a graph $G = \{V, E\}$ to represent a network, where vertices $\{V\}$ denote the computation operations and edges $\{E\}$ show the data flow. In CNN, the computation operations include convolution, BN, activation, concat, add and FC, *etc*. Since convolution accounts for the majority of computational load, we focus on the pruning of convolution vertices in our method. Given a pretrained model, GAP can be conducted using the following steps:

1) Re-train with sparsity regularization. The sparsity is conducted on some parameters with certain structural pattern to make some vertices or edges removable;
2) Sort all the weights and determine the pruning threshold;
3) Remove the correpsonding vertices or edges according to the threshold;
4) Finetune the pruned graph with or without self-taught KD.

2.2.1 Vertex-level pruning

For CNN with BN layers, the scaling factors in BN layers can play a role of measuring the importance of each channel, and thus can be directly used for channel selection with sparsity regularization. The channel-level pruning can be obtained as (3):

$$\min_W f(\mathcal{I}, W) + R(W) + \lambda_s R_s(\{\gamma\}), \tag{4}$$

where $R_s(\cdot)$ is the sparsity regularization, which is typical realized using ℓ_1-norm, λ_s is the balance parameter which can trade-off between the sparsity loss and the original loss.

In such way, insignificant BN vertices can be removed. However, in modern network like DenseNet and ResNet, in order to remove a certain convolution vertex, the graph topology should be taken into consideration. Based on the conception, we propose to adaptively prune network at vertex-level by a more structural way.

The BN vertices are classified into articulation points $\{V_a\}$ and non-articulation points $\{\bar{V}_a\}$. $\{\bar{V}_a\}$ is further split into to 1-to-1 connection $\{\bar{V}_a^{(1\text{-}1)}\}$, 1-to-n connection $\{\bar{V}_a^{(1\text{-}n)}\}$ and n-to-1 connection $\{\bar{V}_a^{(n\text{-}1)}\}$ BN vertices. Different constraints are conducted on different subsets,

$$\min_W f(\mathcal{I}, W) + R(W) + \lambda_s R_s(\{\boldsymbol{\gamma}_s\}) + \lambda_{gs} R_{gs}\left(\{\boldsymbol{\gamma}_{gs}\}\right)$$
$$\text{s.t.} \;\; \gamma_s \in \{\bar{V}_a^{(1\text{-}1)}\}, \;\; \gamma_{gs} \in \{\bar{V}_a^{(1\text{-}n)}\} \cup \{\bar{V}_a^{(n\text{-}1)}\} \tag{5}$$

The vertices in $\{\bar{V}_a^{(1\text{-}1)}\}$ are regularized by ℓ_1-norm $R_s(\cdot)$, and those in $\{\bar{V}_a^{(1\text{-}n)}\}$ and $\{\bar{V}_a^{(n\text{-}1)}\}$ are constrained by group sparsity, using $\ell_{2,1}$-norm $R_{gs}(\cdot)$, while each group denotes the vertices that share the same parent or child vertex.

2.2.2 Edge-level pruning

Different from vertex-level pruning, here we can treat a network as graph at a coarser level: a set of filters in a convolution layer is regarded as a single vertex. Similarly, a BN vertex represents a whole BN layer in edge-level pruning.

When there are multiple paths for data flow, the edges on such paths become non-bridge. Thus the multi-path pruning is equivalent to removing part of the non-bridge edges. And the sparsity regularization to make non-bridge edges pruning is conducted as steps below:

Firstly, the non-bridge edges are selected as candidates to be pruned. We only choose the last edge in each path to conduct pruning. Furthermore, in CNNs, multiple paths are always combined together using a "concat" operation. Therefore, we use concat-vertex to detect the edges to be pruned. The set of selected edges is denoted as $\{E_s\}$. Secondly, each selected edge is scaled by an additional parameter γ_e, acting as a measurement of the edge's importance. The edge scaling factors are therefore constrained using sparsity regularization,

$$\min_{W} f\left(\mathcal{I}, W\right) + R(W) + \lambda_{es} R_{es}\left(\{\boldsymbol{\gamma_e}\}\right), \tag{6}$$

where, $R_{es}\left(\cdot\right)$ denotes ℓ_1-norm on the scaling factors $\{\boldsymbol{\gamma_e}\}$, λ_{es} is the balance parameter.

2.3. Self-taught KD

The model compression may suffer from certain performance degradation, and it can be compensated through finetuning. In addition to naive finetuning, especially for classification task, we propose to finetune the network using a self-taught KD strategy.

For the compressed network, the original model is apparently a more complex model with better performance, and it can act as the teacher in KD for finetuning. In addition, the pretrained model is already provided, which is rather important in practice, as there is always limited resource and time to train a more complex teacher model for a specific task. As the knowledge is distilled from the original model to the pruned network, we denote it as self-taught KD. In the experiments section, we will show that the self-taught KD can work together with GAP to improve the model performance after graph pruning.

2.4. ISS in LSTM

The computation of a LSTM [8] cell is (7):

$$\begin{aligned}
i_t &= \sigma(x_t \cdot W_{xi} + h_{t-1} \cdot W_{hi} + b_i)\\
f_t &= \sigma(x_t \cdot W_{xf} + h_{t-1} \cdot W_{hf} + b_f)\\
o_t &= \sigma(x_t \cdot W_{xo} + h_{t-1} \cdot W_{ho} + b_o)\\
u_t &= tanh(x_t \cdot W_{xu} + h_{t-1} \cdot W_{hu} + b_u)\\
c_t &= f_t \odot c_{t-1} + i_t \odot u_t\\
h_t &= o_t \odot tanh(c_t)
\end{aligned} \tag{7}$$

Due to the element-wise operators ($\odot$ and $+$), removing an individual component from one or a few vectors independently can result in the violation of dimension consistency. Thus the whole network must obey the dimension consistency include input updates, all gates, hidden states, cell states, and outputs. ISS [30] can be learned to reduce the sizes of basic structures within LSTM units, including input updates, gates, hidden states, cell states and outputs. Removing a component of ISS will simultaneously decrease the sizes of all basic structures by one and thereby always maintain the dimension consistency. By learning ISS within LSTM units, the obtained LSTMs remain regular while having much smaller basic structures.

For the ISS learning method, the group Lasso regularization is added to the loss function in order to generate sparsity in ISS. Formally, the ISS regularization is:

$$R(W) = \sum_{n=1}^{N} \sum_{k=1}^{K^{(n)}} ||W_k^{(n)}||^2, \tag{8}$$

where W is the vector of all weights and $||\cdot||_2$ is ℓ_2-norm. By learning ISS, a structurally sparse LSTM can be obtained, which essentially is a regular LSTM with reduced hidden dimension. In addition, ISS can also be extended to vanilla RNNs and Gated Recurrent Unit (GRU), *etc*. One thing should be noticed is that ISS belongs to weight pruning methods, it also requires finetuning to make model performance close to the original one's.

2.5. 8-bit quantization for deep network

Fixed-point quantization is an effective approach for lowering the resource consumption of a network. This kind of work includes quantizing weight and activation, so that computation operation can be conducted with 8-bit instruction. Some other works try to also quantize gradients, which can result in acceleration during the network training. Here, we focus on 8-bit weight quantization and activation quantization for inference optimization.

INT8 has significantly lower precision and dynamic range compared to FP32, therefore quantization requires more than a simple type conversion from FP32 to INT8. We first categorized computational operation into three types: conv2d, matmul and other elementwise ops. Conv2d and matmul ops take a large amount of computing time during inference, so we conduct quantization for conv2d and matmul, the rest ops will execute with FP32. We adopt symmetric linear quantization using a linear scale factor to transform a FP32 value to range of INT8. Given a pretrained FP32 model, the scale factors of weight parameters can be obtained by sorting, then $|max|$s will be mapped to 127 using linear scale factors. Activations can be quantized either on-line or off-line with a calibration dataset. On-line quantization of activation is similar to that of weight. In term of off-line activation quantization, we first run inference in FP32 on calibration dataset, then collect statistics from activations of each layer, iteratively search for rational scale factors based on calibration algorithm. When we get quantized weights and activations, we need to use INT8 conv2d/matmul ops replace original FP32 ops, and add dequantization ops for the outputs of INT8 ops, since elementwise ops still run in FP32. Our experiments show that INT8 model encodes almost the same information as the original FP32 model, and it does not require any additional finetuning or retraining.

3. Experiments

A number of experiments were conducted to validate our DL inference optimization pipeline including TD, GAP with self-taught KD, ISS in LSTM and 8-bit quantization.

3.1. TD for CNN

We evaluated the effectiveness of the TD method using a classification task and a detection task. For classification task, CIFAR10 [15] was chosen as dataset and we used wide ResNet (ResNet32, k =8) [34] for TD benchmark. All units (totally 30 layers) in wide ResNet32 were selected for TD, the results are shown in Table 1. For detection task, we chose an in-house Optical Character Recognition (OCR) detection model and 500 test images from ICDAR14 challenge dataset. We analyzed this detection model and tried to decompose totally 8 middle and deep layers of its backbone ResNet50, so that its computational load can be reduced. Each of 8 layers was decomposed into a depthwise and pointwise convolution through CP decomposition. Results are shown in Table 2.

To measure the inference efficiency, we used three criteria: model Compression Ratio (CR), theoretical Speedup Ratio (SR) and practical SR. Model size and FLOPs before and after pruning were used to compute the model CR and theoretical SR. We used the practical SR as an additional indicator of inference efficiency, since the memory access and movement time are not considered in FLOPs. From the result tables, we can see TD can effectively reduce model size and inference time, the pruned model can achieve marginal loss in model performance through finetuning. Furthermore, in order to improve accelerating, we adopted depthwise operation in cuDNN and also manipulated tensor layouts to avoid overhead caused by tensor dimension transpose.

3.2. GAP with self-taught KD for CNN

We first evaluated the effectiveness of the GAP method using CIFAR10 [15]. Considering the "topology-adaptive" attribute of GAP, DenseNet and ResNeXt were chosen for the evaluation.

For experiments on CIFAR10, DenseNet-40 (k=12) and ResNeXt-29 (8 $\times$ 64d) were adopted. All the layers were pruned simultaneously based on an adaptive threshold, which was determined by the pruning proportion.

Pruning results of ResNeXt and DenseNet are shown in Table 3. The networks are pruned with the same percentage for channel-level, vertex-level and edge-level. The results suggest that the strategy of finetuning with self-taught KD performs better than naive finetuning in restoring the degradation of classification accuracy caused by pruning. We can see that structurally pruning at vertex-level can get higher model CR and theoretical SR, while vertex-level pruning can still get better performance in classification error rate. In ResNeXt, with approximately no-loss of accuracy, pruning 60% off at vertex-level can get 2.68$\times$ practical SR while only 1.69$\times$ at channel-level. In DenseNet, channel-level pruning achieves almost no speed-up as it introduces additional selection operations, which increases the memory access time. Edge-level pruning leads to the largest remaining model size and FLOPs, because edge-level pruning can only prune part of the graph. In ResNeXt, only the edges contained in the group convolution can be pruned. Similarly, only the dense connections can be removed in DenseNet. Additionally, edge-level gets the worst error rate. This is naturally because it prunes the network at a coarse-grained level, which will do more harm on the network [22]. However, the benefit of edge-level pruning is that it has little gap between practical SR and theoretical SR.

3.3. ISS for RNN

We evaluated ISS with two in-house scenarios. The first one is a in-house Optical Character Recognition (OCR) model based on Convolutional Recurrent Neural Network (CRNN) [26], and the second one is a Neural Machine Translation (NMT) model using GRU and attention mechanism.

The in-house OCR recognition model consists of a ResNet based CNN module and LSTM module. We used model CR, theoretical SR and practical SR to evaluate the ISS method for LSTM module. The results are shown in Table 4. As we can see, with ISS for LSTM, the pruned model only gets 0.23% loss in sequence accuracy. After pruning LSTM module, the whole network model size decreases by 51.1%, the theoretical SR can achieve 2.4$\times$ for LSTM module. For CPU testing, the end2end inference is accelerated by 57% compared with baseline CRNN result. We find that we do not get practical speedup with GPU, due to the time lost to scheduling overhead.

ISS can also be extended to other RNN models such as GRU. The NMT model consists of several layers of stacked bi-directional GRU as encoder, a candidate layer of GRU and stacked GRU layers as decoder. As shown in Table 5, after learning the ISS from pretrained baseline model and conducting weight pruning, we can get a new model, which is 73.22% of original model size, with 1.17$\times$ practical SR for end to end GPU inference. Meanwhile, BLEU of the pruned model is 28.16, comparing 29.0 of baseline model, ISS makes inference more efficient with marginal loss of model accuracy.

3.4. 8-bit quantization

For CNN quantization, we tested ImageNet task using GoogleNet, Inception_V3 and ResNet50. For RNN quantization, we tested TF-NMT models released with Tensorflow. And the in-house OCR recognition model in section 3.3 is based on CRNN, which contains both CNN and LSTM, it is a scenario to validate quantization result for the combination of CNN and RNN. For each model, we only conducted INT-8 quantization for conv2d and matmul ops, the rest of elementwise ops still run in FP32. The quantization results are shown in Table 6. We can see that the INT8 quantized DNN model can be used for INT8 execution engine without finetuning, the gap between accuracies of original FP32 model and INT8 model is ignorable. INT8 quantization can compress the model size and accelerate the inference procedure. However, specific hardwares and optimized kernels are necessary for practical SR, which is our ongoing work.

4. Conclusion and Future Work

In this paper, we introduce a DL inference optimization pipeline, which consists of a series of model compression methods, including TD, GAP, ISS in LSTM, KD and low-bit model quantization. Experimental results show our inference optimization pipeline can be applied to different kinds of DL model and has potential to accelerate inference engine with marginal loss of model accuracy for a variety of modeling tasks. As future work, we are going to investigate the scheme to automatically choose a compression method or combine some of these methods, so that a more rational model compression can be conducted for a network given computation resource or latency limitation. Meanwhile, the gap between theoretical and practical acceleration needs to be minimized through the mutual adaptation of algorithm-level and system-level optimization.

Table 1. TD accelerates ResNet32 for CIFAR10.

	Top1 accuracy	Model CR (whole)	Theoretical SR)	Practical SR (conv2d)	Practical SR (end2end)
Baseline	95.5%	-	-	-	-
TD all units (30 layers)	94.9%	10.0×	10.4×	1.84×	1.41×

Table 2. TD accelerates in-house OCR detection model.

	H-Mean	Model CR (whole)	Theoretical SR)	Practical SR (end2end)
Baseline	80.6%	-	-	-
TD (8 layers)	80.43%	1.72×	1.89×	1.41×

Table 3. GAP on CIFAR10.

		Pruned /%	Error /% (w/o KD)	Error /% (w/ KD)	Model Size (MB)	Model CR	Theoretical SR	Practical SR
ResNeXt-29	Baseline (Our impl.)	-	4.00	-	34.43	-	-	-
	Channel-level [20]	60%	4.28	4.09	9.10	3.78×	4.24×	1.69×
	Vertex-level	60%	4.08	4.03	6.71	5.13×	6.02×	2.68×
	Edge-level	60%	4.55	4.11	24.61	1.40×	2.24×	2.41×
DenseNet-40	Baseline (Our impl.)	-	5.60	-	1.06	-	-	-
	Channel-level [20]	50%	6.38	5.70	0.58	1.83×	1.56×	1.03×
	Vertex-level	50%	6.14	5.67	0.52	2.04×	2.09×	1.27×
	Edge-level	50%	6.24	6.00	0.90	1.18×	1.39×	1.11×

Table 4. ISS for LSTM of in-house OCR model.

	Sequence Accuracy	CR (LSTM)	CR (whole network)	Theoretical SR (LSTM)	Practical SR (LSTM on CPU)	Practical SR (end2end on CPU)
Baseline	74.22%	-	-	-	-	-
ISS for LSTM	73.99%	2.93×	2.04×	2.40×	1.95×	1.57×

Table 5. ISS for GRU of in-house NMT model.

	BLEU	CR (GRU)	CR (whole)	Theoretical SR (encoder)	Theoretical SR (candidate GRU)	Theoretical SR (decoder)	Practical SR (end2end on GPU)
Baseline	29.0	-	-	-	-	-	-
ISS for GRU	28.16	3.02×	1.37×	4.2×	1.6×	1.47×	1.17×

Table 6. 8-bit quantization for different DNN models.

	Baseline: FP32	Quantized model: INT8 conv2d/matmul
GoogleNet: top1/top5 accuracy	68.73 / 89.06 %	68.62 / 88.92 %
Inception_V3: top1/top5 accuracy	78.54 / 94.30 %	78.28 / 94.17 %
ResNet50: top1/top5 accuracy	74.82 / 92.04 %	74.36 / 91.72 %
IWSLT-13 (English-Vietnamese): BLEU	26.2	26.0
WMT-15 (German-English): BLEU	28.8	28.4
NMT (Chinese-English): BLEU	25.0	24.98
in-house OCR recognition: seq/char accuracy	72.71 / 91.81 %	72.90 / 91.78 %

References

[1] A. Dario et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016. 1, 2

[2] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 2

[3] S. Han. *Efficient Methods and Hardware for Deep Learning*. PhD thesis, Stanford University, 9 2017. Submitted to the Department of Electrical Engineering. 1

[4] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[5] P. He, W. Huang, Y. Qiao, C. C. Loy, and X. Tang. Reading scene text in deep convolutional sequences. In *AAAI*, pages 3501–3508, 2016. 1

[6] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 2

[7] G. Hinton et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. 4

[9] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

[10] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. 1

[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, pages 4107–4115. 2016. 2

[12] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 2

[13] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015. 1

[14] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 1

[15] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Technical report*, 2009. 5

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[17] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 1

[18] Y. Lecun, J. S. Denker, and S. A. Solla. Optimal brain damage. 2:598–605, 1990. 1

[19] J. Li, W. Monroe, T. Shi, S. Jean, A. Ritter, and D. Jurafsky. Adversarial learning for neural dialogue generation, 2017. cite arxiv:1701.06547. 1

[20] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763. IEEE, 2017. 2, 7

[21] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017. 2

[22] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, 2017. 5

[23] S. Narang, G. F. Diamos, S. Sengupta, and E. Elsen. Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119, 2017. 2

[24] A. Parashar et al. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 27–40. ACM, 2017. 1

[25] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 2

[26] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *CoRR*, abs/1507.05717, 2015. 5

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[28] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, volume 1, page 4, 2011. 2

[29] S. Wang and J. Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016. 1

[30] W. Wen, Y. He, S. Rajbhandari, M. Zhang, W. Wang, F. Liu, B. Hu, Y. Chen, and H. Li. Learning intrinsic sparse structures within long short-term memory. In *International Conference on Learning Representations*, 2018. 2, 4

[31] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016. 2

[32] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995. IEEE, 2017. 1

[33] J. Yim, D. Joo, J. Bae, and J. Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[34] S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. 5