INVITED: Bandwidth-Efficient Deep Learning

Song Han
Massachusetts Institute of Technology, Google
Cambridge, MA
songhan@mit.edu

William J. Dally
Stanford University, NVIDIA
Stanford, CA
dally@stanford.edu

ABSTRACT

Deep learning algorithms are achieving increasingly higher prediction accuracy on many machine learning tasks. However, applying brute-force programming to data demands a huge amount of machine power to perform training and inference, and a huge amount of manpower to design the neural network models, which is inefficient. In this paper, we provide techniques to solve these bottlenecks: saving memory bandwidth for inference by model compression, saving networking bandwidth for training by gradient compression, and saving engineer bandwidth for model design by using AI to automate the design of models.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Hardware** → **Hardware accelerators**;

KEYWORDS

Neural Networks, Accelerator, Inference, Training, Model Compression, Gradient Compression

ACM Reference Format:
Song Han and William J. Dally. 2018. INVITED: Bandwidth-Efficient Deep Learning. In *DAC '18: The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA.* ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3195970.3199847

1 INTRODUCTION

In the post-ImageNet era, computer vision and machine learning researchers are solving more complicated AI problems using larger data sets driving the demand for more computation. However, we are in the post-Moore's Law world where the amount of computation per unit cost and power is no longer increasing at its historic rate. This mismatch between supply and demand for computation highlights the need for efficient algorithms and hardware for deep learning.

Conventional deep learning algorithms use a brute-force approach: with a large amount of training data [2], simply design a large model [15] and apply a large amount of computation power. There is little consideration about efficiency. As a result, many problems begin to emerge: deep neural nets are slow to train — taking days or weeks, slow to deploy — especially on edge devices with limited computation resource, and consume large amounts of power — which exceeds the power budgets of edge devices and increases the total cost of ownership (TCO) of a data center.

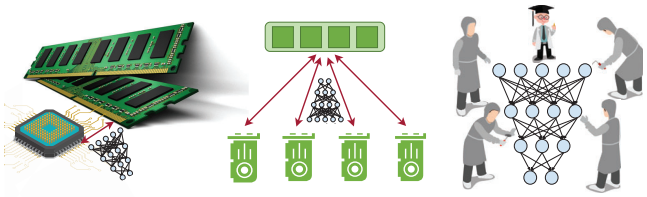

Figure 1: Bottlenecks for efficient deep learning computing: memory bandwidth, network bandwidth, and engineer's bandwidth. We provide techniques to save these bandwidth.

To solve these problems, we must overcome three bottlenecks: memory bandwidth, network bandwidth, and engineer bandwidth (Figure 1). These bottlenecks limit efficiency for both training and inference. First, large models take a long time to train, and the large gradient communication occupies a significant amount of **network bandwidth** making it difficult to scale. Second, large models take large **memory bandwidth** at inference time which results in high power consumption when deploying deep neural networks on edge devices. Finally, the huge design space of deep neural networks makes finding the optimal network a difficult task requiring large amounts of **engineer bandwidth** to carefully design and compress the models to fit the latency and accuracy requirements. Skilled machine learning engineers are expensive in tech companies, thus engineer bandwidth is the most critical of the three bottlenecks.

"Less is more"
— Robert Browning, 1855

In this paper, we discuss three techniques toward "bandwidth-efficient deep learning": use the smallest memory bandwidth, network bandwidth, and engineer's bandwidth to achieve efficient deep learning computing. So given the same computation resource and engineer headcount, we can produce more productive models.

The paper is organized as follows. In section 2, we discuss model compression techniques that save memory bandwidth for efficient inference. In section 3, we discuss gradient compression techniques that save network bandwidth for efficient training. In section 4, we discuss how to use AI to improve AI, as a design automation that saves engineer's bandwidth and can perform better than human expertise.

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3199847

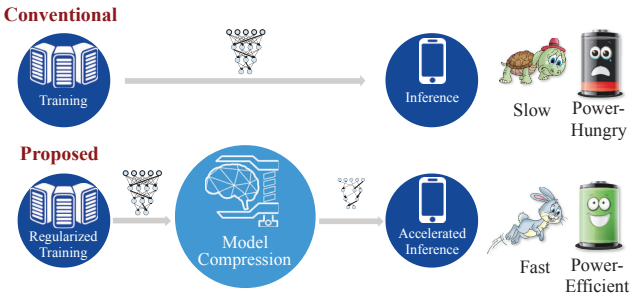

Figure 2: Save the memory bandwidth by model compression for efficient inference.

Table 1: Pruning deep neural networks by 3× to 13× without losing accuracy.

Network	Top-1 Error	Top-5 Error	Params	Pruning Rate
AlexNet	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9×**
VGG-16	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13×**
GoogleNet	31.14%	10.96%	7.0M	
GoogleNet Pruned	31.04%	10.88%	**2.0M**	**3.5×**
SqueezeNet	42.56%	19.52%	1.2M	
SqueezeNet Pruned	42.26%	19.34%	**0.38M**	**3.2×**
ResNet-50	23.85%	7.13%	25.5M	
ResNet-50 Pruned	23.65%	6.85%	**7.47M**	**3.4×**

2 SAVE MEMORY BANDWIDTH

The peak FLOPS of a deep learning processor is often advertised. However, the utilization of these FLOPS is often constrained by memory bandwidth. For example, TPU-1 had 92 TOPs/second peak throughput, but was badly constrained by memory bandwidth (2.8-3.7 TOPs/second real performance on LSTM benchmarks) [14]; A Tesla P40 with fewer peak OPs/second but more memory bandwidth was faster. We propose model compression techniques that can bring down the model size and reduce the memory bandwidth required for inference.

Accuracy used to be the sole figure-of-merit for computer vision challenges such as ImageNet. However, without constraints on model size, such criteria limits the practicality of deploying the models in production, especially on mobile scenarios such as in Caffe2Go [3] and Tensorflow Lite [5]. For mobile applications we need to achieve the highest accuracy at a given model size (Figure 2).

There are four distinct approaches to model compression: Pruning [10], Quantization [9], Decomposition [28] and Distillation [12]. We focus on pruning and quantization.

Pruning reduces the number of connections. DNNs are over-parameterized. We use SGD, a convex optimization method, to optimize a DNN, which is a highly non-convex problem. Redundancy is needed to avoid getting stuck at a local minima. However, once the model converges, we can remove those redundant connections. This is similar to human brain development [24][25], where excess synapses formed in the first few months of life are gradually

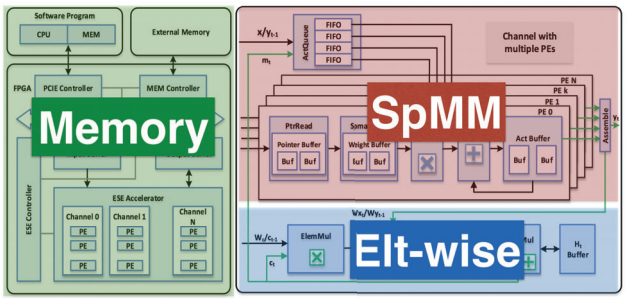

Figure 3: The ESE hardware architecture for accelerating sparse LSTM. Available on AWS Market Place[1].

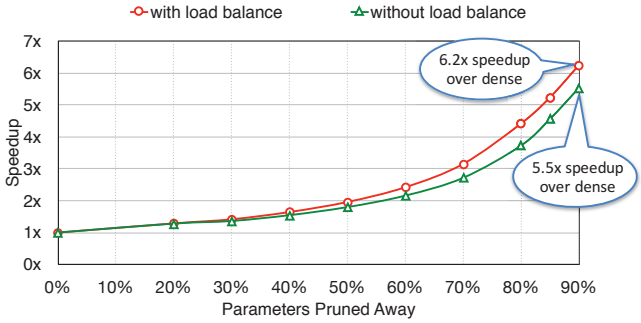

Figure 4: Speedup with sparsity on specialized hardware.

"pruned", with neurons losing little-used connections while preserving the functionally important connections. Failing to prune may even cause diseases [13].

The results for pruning modern CNNs on ImageNet are shown in Figure 1. We achieve pruning rates of 3× to 13× without losing accuracy. However, sparsity introduces irregularity, which is inefficient on general purpose hardware. We built specialized accelerators that efficiently traverse the sparse matrix [7, 8, 22]. The system architecture of the ESE accelerator is shown in Figure 3, which is accessible on AWS [1]. With proper load balancing, we achieve 6.2× speedup under 90% sparsity compared with a dense model.

However, we don't always have access to specialized hardware. In such cases, we can prune at coarse granularity [19] to achieve speedup on dense architectures (CPU/GPU/TPU). Figure 5 shows four different granularity levels of structured pruning. In (d), filter-level structured sparsity removes entire filters and the resulting CNN is still dense, which can directly run on CPU/GPU. We showed the acceleration result on NVIDIA TX1 in Figure 6. Much previous work on pruning focused on classification; here we demonstrate that pruning also works well on detection. In this case, we pruned Yolo+GoogleNet from 17.5GOPs per image to 8.9GOPs per image. The baseline detection mAP for car/pedestrian/bike is car/pedestrian/bike are 59.47/28.48/45.43. After pruning we achieved 2× operation reduction, and the mAP became 59.30/28.33/47.72, negligible change. The mAP for pedestrian even increased; we suspect is due to less overfitting. On NVIDIA TX1, we observed 1.7×

[1]https://aws.amazon.com/marketplace/pp/B079N2J42R

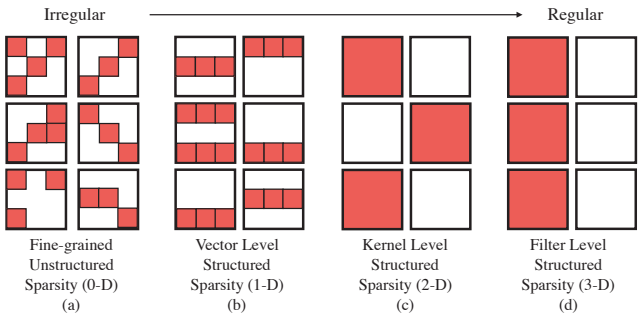

Figure 5: Structured pruning at different granularity levels.

Figure 6: Speedup with sparsity on general-purpose hardware (NVIDIA Jetson TX1) for object detection.

speedup using of-the-shelf cuDNN library under different batch sizes (Figure 6). There are more design spaces between (a) and (d) in Figure 5 that may lead to more efficient implementation.

In contrast to pruning, quantization reduces the number of bits per connection. Quantization may be uniform or non-uniform. Uniform quantization places an equal distance between adjacent centroids, while non-uniform quantization allows a different distance between adjacent centroids. Uniform quantization is widely used in Tensorflow and TensorRT, however, we found non-uniform quantization with K-means clustering can achieve 2× better pruning ratio, as shown in Figure 7. At four bits, Resnet50 loses 1.5% Top-1 accuracy with uniform quantization. With non-uniform quantization (Kmeans), it doesn't lose any accuracy. From the implementation perspective, non-uniform quantization requires extra logic for codebook lookup, which we implemented in EIE [8]. It adds a 16-entry lookup table in the data-path, which has negligible area.

Pruning and Quantization can be combined together to boost the compression ratio. The combined compression results for modern CNNs are shown in Figure 2. It's not surprising to see that fully convolutional neural networks (such as SqueezeNet, Inception-V3 and ResNet-50) can be compressed less because convolutional layers are parameter-efficient, but they can still be compressed by an order of magnitude.

3 SAVE NETWORKING BANDWIDTH

Large-scale distributed training requires significant communication bandwidth for gradient exchange that limits the scalability of multi-node training and requires expensive high-bandwidth network

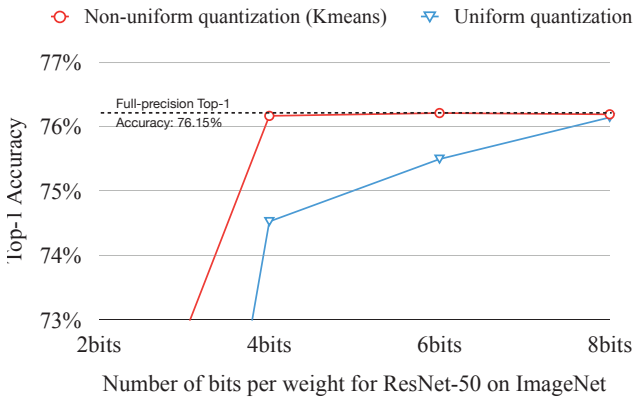

Figure 7: Non-uniform quantization outperforms uniform quantization widely used in Tensorflow and TensorRT.

Table 2: Deep Compression saves the model size by 10× to 49× with no loss of accuracy.

Network	Top-1 Error	Top-5 Error	Params	Compress Rate
AlexNet	42.78%	19.73%	240 MB	
AlexNet DC	42.78%	19.70%	**6.9 MB**	**35×**
VGG-16	31.50%	11.32%	552 MB	
VGG-16 DC	31.17%	10.91%	**11.3 MB**	**49×**
SqueezeNet	42.5%	19.7%	4.8 MB	
SqueezeNet DC	42.5%	19.7%	**0.47MB**	**10×**
Inception-V3	22.55%	6.44%	91 MB	
Inception-V3 DC	22.34%	6.33%	**4.2 MB**	**22×**
ResNet-50	23.85%	7.13%	97 MB	
ResNet-50 DC	23.85%	6.96%	**5.8 MB**	**17×**

infrastructure. According to Rent's rule[16], interconnect scales slower than logic. To improve the scalability of distributed training, we propose Deep Gradient Compression (DGC) [18] to greatly reduce the communication bandwidth.

The motivation is illustrated in Figure 8. Distributed training improves the productivity of training deeper and larger models [1, 20, 27, 29]. There are two components of iteration time: computation time (to perform forward-backward) and communication time (to exchanged the gradients). By increasing the number of GPUs and taking advantage of data parallelism, the **computation** time can be reduced (Figure 8, middle). However, the **communication** time can't be reduced, but even *increased* since there are more gradients to be exchanged with more nodes. This dwarfs the savings of computation time, especially for recurrent neural networks (RNN) where the computation-to-communication ratio is low, which has poorer scalability. Therefore, the network bandwidth bottlenecks the scalability of distributed training.

Deep Gradient Compression (DGC) solves the communication bandwidth problem by compressing the gradients (Figure 8, right). We send only the important gradients (sparse update). We use the gradient magnitude as a simple heuristics for importance: only gradients larger than a threshold are transmitted. To avoid losing information, we accumulate the rest of the gradients locally. Eventually, these gradients become large enough to be transmitted.

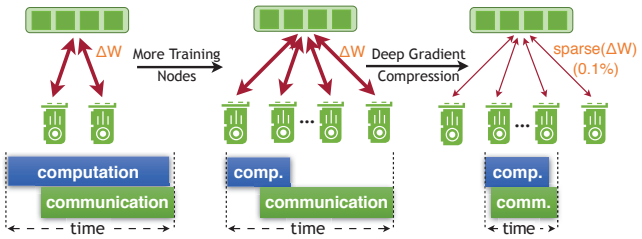

Figure 8: Save network bandwidth using Deep Gradient Compression (DGC) for efficient training.

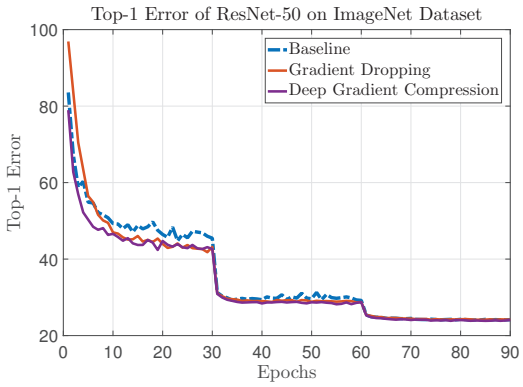

Figure 9: Learning curve of ResNet-50 on ImageNet with deep gradient compression.

Momentum SGD is widely used in place of vanilla SGD. However, gradient dropping and local accumulation don't directly apply to the momentum term, since it ignores the discounting factor between the sparse update intervals.

Distributed training with momentum SGD on N training nodes follows [23]:

$$u_t = mu_{t-1} + \sum_{k=1}^{N} \left(\nabla_{k,t}\right), \quad w_{t+1} = w_t - \eta u_t \tag{1}$$

where m is the momentum, u_t is the velocity, N is the number of training nodes, and $\nabla_{k,t} = \frac{1}{Nb}\sum_{x\in\mathcal{B}_{k,t}} \nabla f(x, w_t)$. The velocity, rather than the gradient, is finally multiplied by the learning rate and gets added to the weight. Thus, considering the momentum term, **we should accumulate the velocity, not the gradient**:

$$u_{k,t} = mu_{k,t-1} + \nabla_{k,t}, \quad v_{k,t} = v_{k,t-1} + u_{k,t} \tag{2}$$

$$w_{t+1} = w_t - \eta \sum_{k=1}^{N} sparse\left(v_{k,t}\right) \tag{3}$$

where the first two terms are accumulating the velocity, and the result $v_{k,t}$ is pruned and sent out to other nodes. When training starts, the gradients has large dynamic range. We found warm-up training is needed: rather than directly use 99.9% sparsity, starting with 75% and exponentially increase the sparsity every epoch will benefit the accuracy. By these simple changes in the local accumulation, we can fully preserve the accuracy with high pruning ratio.

Task		Baseline	Deep Gradient Compression
ResNet-50 On ImageNet	Top-1 Accuracy	75.96%	**76.15% (+0.19%)**
	Top-5 Accuracy	92.91%	**92.97% (+0.06%)**
	Gradient Compression Ratio	1 ×	**277 ×**
5-Layer GRU On LibriSpeech	Word Error Rate (WER)	9.45%	**9.06% (-0.39%)**
	Word Error Rate (WER)	27.07%	**27.04% (-0.03%)**
	Gradient Compression Ratio	1 ×	**608 ×**
2-Layer LSTM Language Model On Penn Treebank	Perplexity	72.30	**72.24 (-0.06)**
	Gradient Compression Ratio	1 ×	**462 ×**

Figure 10: Deep Gradient Compression can compress the gradient exchange by 277× to 608× without losing any accuracy

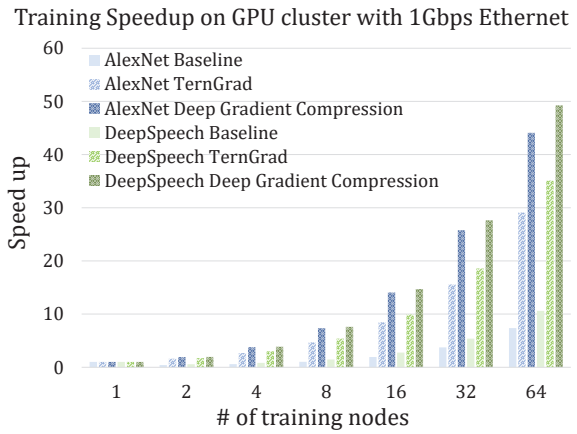

Figure 11: Deep Gradient Compression improves the speedup and scalability of distributed training.

We have applied Deep Gradient Compression to a wide range of machine learning tasks including image classification, speech recognition, and language modeling with multiple datasets including Cifar10, ImageNet, Penn Treebank, and Librispeech Corpus. On these scenarios, Deep Gradient Compression achieves a compression ratio from 270× to 600× without losing accuracy, cutting the gradient size of ResNet-50 from 97MB to 0.35MB, and for DeepSpeech from 488MB to 0.74MB. Deep gradient compression enables large-scale distributed training on inexpensive commodity 1Gbps Ethernet. The detailed accuracy and compression ratio numbers are shown in Figure 10.

We use the performance model proposed in Wen et al. [26] to perform the analytical modeling for scalability. Figure 11 shows the speedup of multi-node training compared with single-node training. Conventional training achieves very poor scalability with the low-cost commodity 1Gbps Ethernet, while Deep Gradient Compression greatly improved the scalability without needing the expensive 20Gbps ethernet. With the same budget, buy GPUs, not networking.

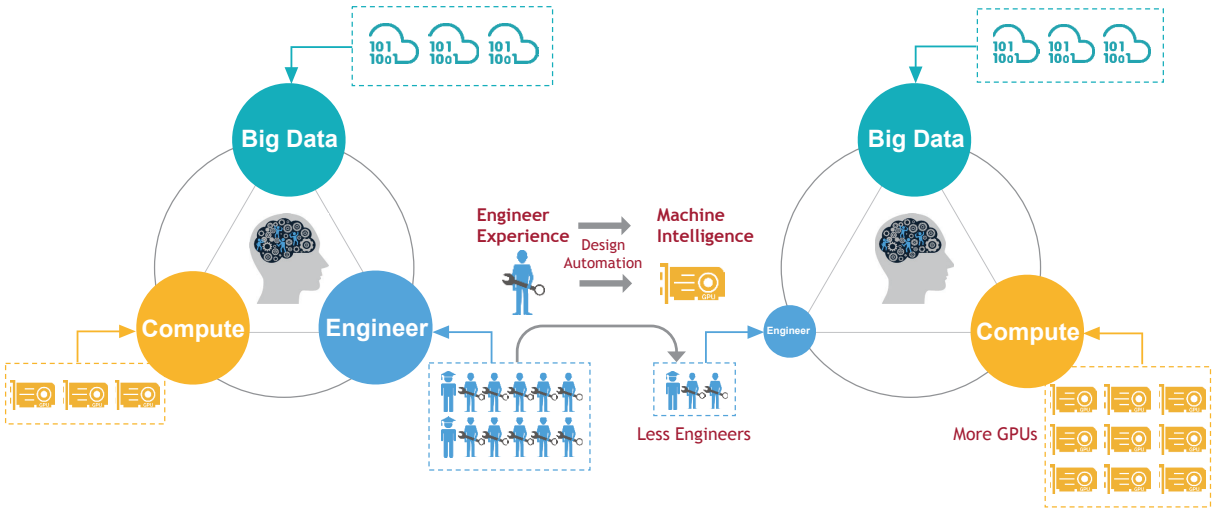

Figure 12: Save engineer bandwidth through design automation: using AI to improve AI.

4 SAVE ENGINEER BANDWIDTH

Training deep neural networks used to be a "dark art" that required many engineer hours to carefully adjust model architectures and tune hyper-parameters. Machine learning engineer time (bandwidth) is valuable, so it's desirable to reduce the manual labor required to optimize neural network designs through design automation, saving engineer bandwidth. As illustrated in Figure 12, three types of resources are required to train and optimize a deep neural network: a large training dataset, a powerful computation engine, and experienced engineers (in this cartoon, two PhDs with ten engineers). By applying AI to optimize the network, we can trade computing time for engineer time. We can achieve a network of equal or higher quality with fewer engineers (in this cartoon, only one Ph.D. with two engineers) by applying more computing resources.

Saving engineer bandwidth makes the business more scalable, especially with the recent advancement of cloud machine learning services attracting an increasing number of customers [4, 21]. Customers can, and we want them to, grow exponentially; but it is hard to find an exponentially increasing number of engineers to service these customers. Therefore, we need design automation. With design automation, machine learning businesses can attract more customers. The one Ph.D. and eight engineers in Figure 12 don't loose their job. Rather, they are freed to work on a new, more meaningful job. Everyone moves up the ladder, leaving the tedious labor to a farm of GPUs.

We provide a case study of neural network design automation with the automatic model compression (AMC) engine [11]. Conventional neural network model compression techniques [9, 10] require *hand-crafted* heuristics and domain experts to explore the large design space trading off model size, speed, and accuracy, which is usually sub-optimal and time-consuming. Instead, we propose Automated Model Compression (AMC) that leverages reinforcement learning to efficiently sample the design space and greatly improve the model compression quality.

Figure 13 illustrates our AMC engine. We aim to automatically find the sparsity ratio to prune each layer of a network. We train

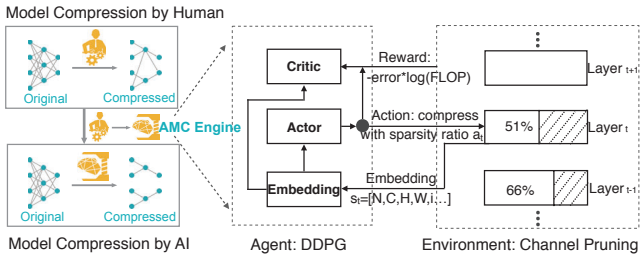

Figure 13: Automated Model Compression (AMC) engine. Left: AMC replaces human and makes model compression fully automated, performing better than human. Right: form AMC as a reinforcement learning problem.

a DDPG [17] agent to learn through trial and error: penalizing accuracy loss while encouraging model shrinking. Each time the DDPG agent receives a layer embedding s_t and outputs a sparsity ratio a_t for each layer. With all layers compressed, the validation accuracy of the pruned model is evaluated without fine-tuning, which is an efficient proxy for fine-tuned accuracy: it took only four GPUs five hours to search the best pruning policy for ResNet-50 on Imagenet, rather than a human took a day of to search, and worse.

AMC not only saves engineers' bandwidth but also performs better than human. We are able to push the expert-tuned compression ratio of ResNet-50 on ImageNet from 3.4× to 5×, see Figure 15, without loss of performance on ImageNet (original ResNet50's [top-1, top-5] accuracy=[76.13%, 92.86%]; AMC pruned model's accuracy=[76.11%, 92.89%]). The density of each layer during each stage is displayed in Figure 14. We found very salient sparsity pattern given by our AMC agent: peaks are 1×1 convolution, crests are 3×3 convolution. The reinforcement learning agent automatically learns that 3×3 convolution has more redundancy than 1×1 convolution and can be pruned more. We can also find that the density distribution of AMC is quite different human expert's result shown in Table 3.8 of [6], which suggests that AMC can fully explore the design space and allocates sparsity in a better way.

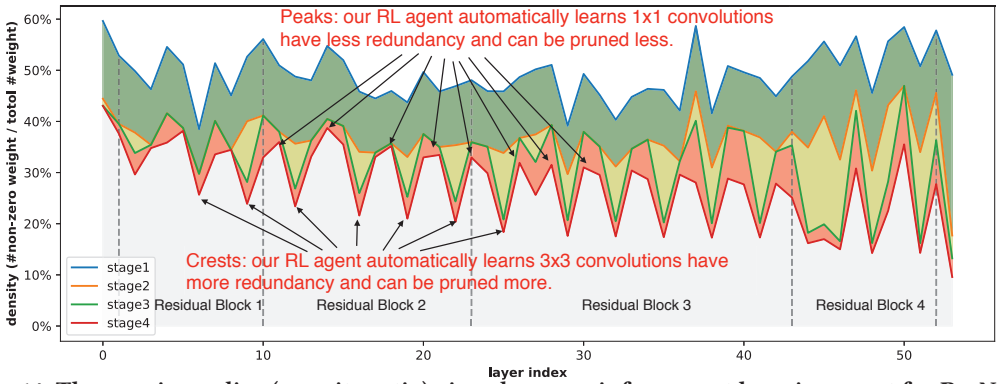

Figure 14: The pruning policy (sparsity ratio) given by our reinforcement learning agent for ResNet-50.

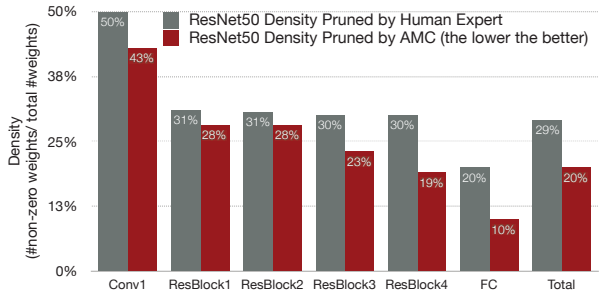

Figure 15: Our reinforcement learning agent (AMC) can prune the model to a lower density than achieved by human experts without loss of accuracy. (Human expert: 3.4× compression on ResNet50. AMC : 5× compression on ResNet50.)

5 CONCLUSION

We discussed three bottlenecks toward efficient deep learning computing: memory bandwidth, network bandwidth, and engineer bandwidth. For each bottleneck, we provided an efficient algorithm to reduce the required bandwidth: Deep Compression reduces model size by 10× to 50×, Deep Gradient Compression reduces communication bandwidth by 270× to 600×, and AMC engine that uses AI to automate the model compression process, reducing a day of human time to 5 hours of GPU time. These bandwidth-efficient algorithms lead to faster, more energy efficient and more scalable deep learning computing.

REFERENCES

[1] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System.. In *OSDI*, Vol. 14. 571–582.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 248–255.

[3] Facebook. 2017. Delivering real-time AI in the palm of your hand. (2017). https://code.facebook.com/posts/196146247499076/delivering-real-time-ai-in-the-palm-of-your-hand

[4] Google. 2017. Google Cloud AI. (2017). https://cloud.google.com/products/machine-learning/

[5] Google. 2017. Introduction to TensorFlow Lite. (2017). https://www.tensorflow.org/mobile/tflite/

[6] Song Han. 2017. *Efficient Methods and Hardware for Deep Learning*. Ph.D. Dissertation. Stanford University.

[7] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, and William Dally. 2017. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[8] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 243–254.

[9] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[10] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*. 1135–1143.

[11] Yihui He and Song Han. 2018. AMC: Automated Model Compression and Acceleration with Reinforcement Learning. *arXiv preprint arXiv:1802.03494* (2018).

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[13] Matthew B Johnson and Beth Stevens. 2018. Pruning hypothesis comes of age. (2018).

[14] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, and et al. . 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

[16] Mary Yvonne Lanzerotti, Giovanni Fiorenza, and Rick A Rand. 2005. Microminiature packaging and integrated circuitry: The work of EF Rent, with an application to on-chip interconnection requirements. *IBM journal of research and development* 49, 4.5 (2005), 777–803.

[17] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[18] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2018. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. *ICLR'2018* (2018).

[19] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. 2017. Exploring the Regularity of Sparse Structure in Convolutional Neural Networks. *arXiv preprint arXiv:1705.08922* (2017).

[20] Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael I Jordan. 2015. Sparknet: Training deep networks in spark. *arXiv preprint arXiv:1511.06051* (2015).

[21] NVIDIA. 2017. NVIDIA GPU Cloud. (2017). https://www.nvidia.com/en-sg/gpu-cloud/

[22] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joe Emer, Stephen Keckler, and William J. Dally. 2017. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *44th International Symposium on Computer Architecture*.

[23] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks* 12, 1 (1999), 145–151.

[24] JP Rauschecker. 1983. Neuronal mechanisms of developmental plasticity in the cat's visual system. *Human neurobiology* 3, 2 (1983), 109–114.

[25] Christopher A Walsh. 2013. Peter Huttenlocher (1931-2013). *Nature* 502, 7470 (2013), 172–172.

[26] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. In *Advances in Neural Information Processing Systems*.

[27] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data* 1, 2 (2015), 49–67.

[28] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. 2016. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence* 38, 10 (2016), 1943–1955.

[29] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. 2010. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*. 2595–2603.