Introduction to Model Compression and Acceleration for DNNs

Jinbo wang

July 10,2019

Intro

Edge Computing

- ➢ Why
- computationally expensive and memory intensive(VGG-16,500MB,15B FLOPs) hindering their deployment in embedded and mobile device with low memory resources and constrained download bandwidth or in applications with strict latency requirements(RS,50ms)
- energy consumption is dominated by memory access

- ➢ Goal
- perform model compression and acceleration in deep networks without significantly decreasing the model performance (accuracy)
- fitting the model into on-chip SRAM cache rather than off-chip DRAM memory

- ➢ Application
- real-time applications such as online learning and incremental learning, virtual reality, augmented reality, and smart wearable devices ,auto drive,

Categories

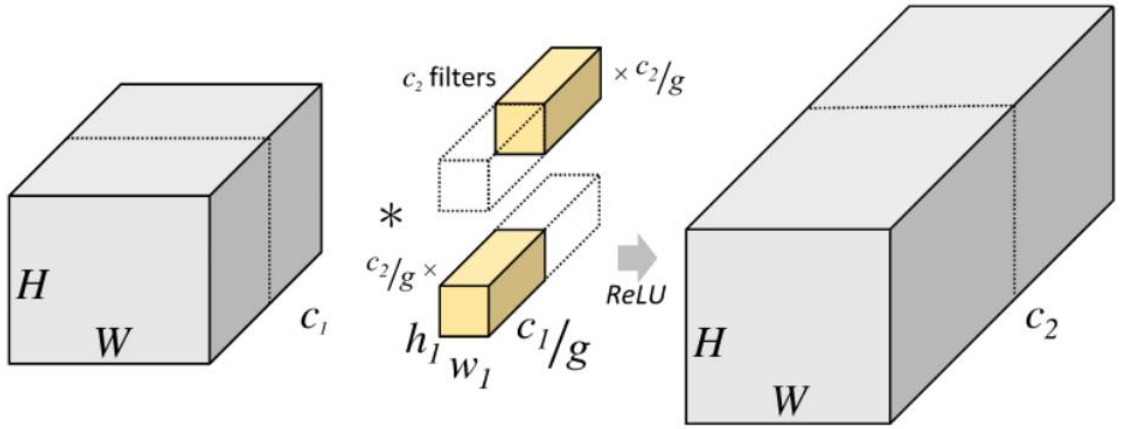
- low-rank approximation:Using matrix/tensor decomposition to estimate the informative parameters
- network quantization(filter num):make the parameters tend to be sparse by adding items in the loss function
- knowledge distillation:Training a compact neural network with distilled knowledge of a large model
- hardware accelerator:FPGA/ASIC
- compact network design:MobileNet,ShuffleNet
- network pruning and sharing(filter weight):Reducing redundant parameters which are not sensitive to the performance
- software optimization(HPC):Tensorrt(FP16,INT8 activation value fixed-pointer quantization),TVM

Topic

- compact network design:MobileNet,ShuffleNet
- network pruning and sharing:Reducing redundant parameters which are not sensitive to the performance(模型压缩：参数稀疏、剪裁、量化、分解)
- software optimization(HPC):Tensorrt(FP16,INT8)

Compact network design

- Group conv

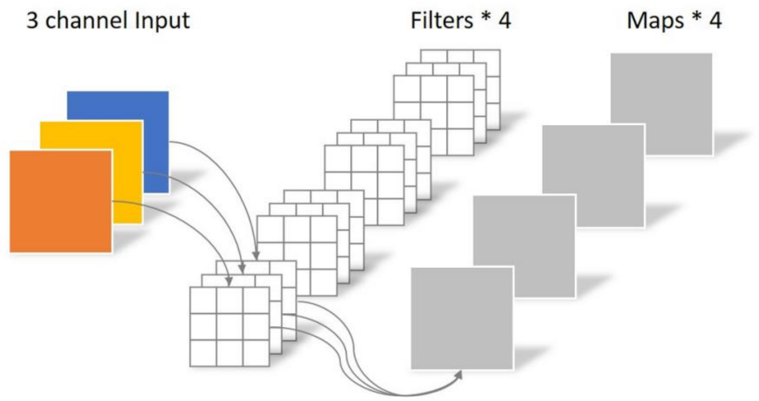


A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convolutional layer is now exactly half the depth, i.e. half the parameters and half the compute as the original filter.

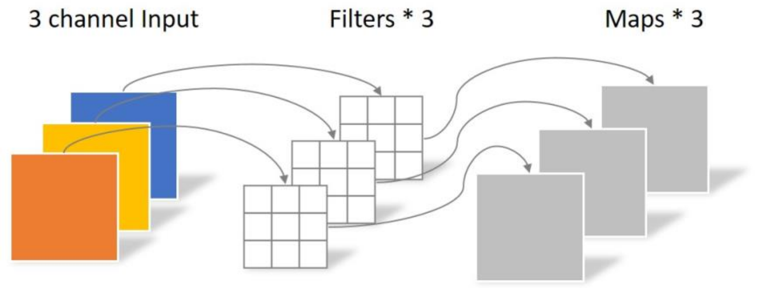
Conv:256×3×3×256

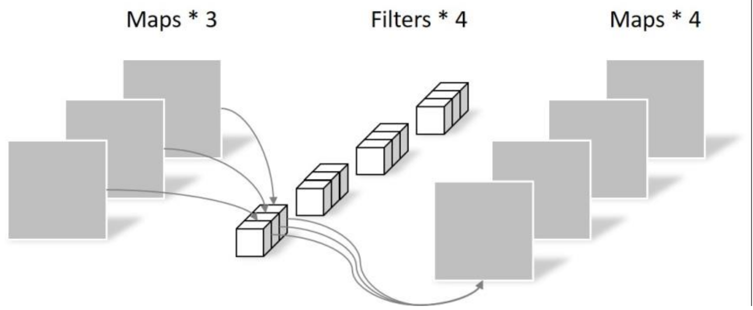
Group conv:8×32×3×3×32

Compact network design



Conv: 4 × 3 × 3 × 3 = 108





Depthwise Conv : 3 × 3 × 3 = 27

Pointwise Conv : 1 × 1 × 3 × 4 = 12

Compact network design:ShuffleNet

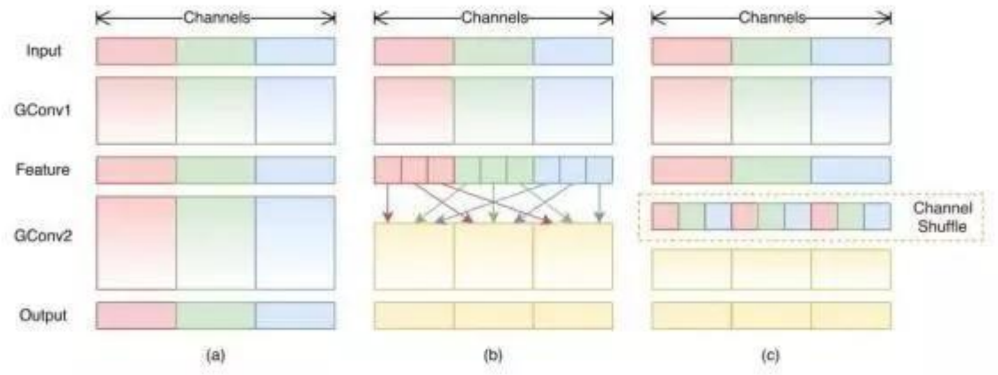


Figure 1: Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle.

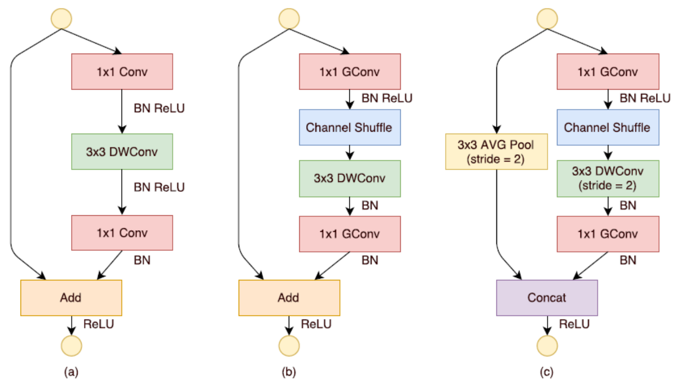


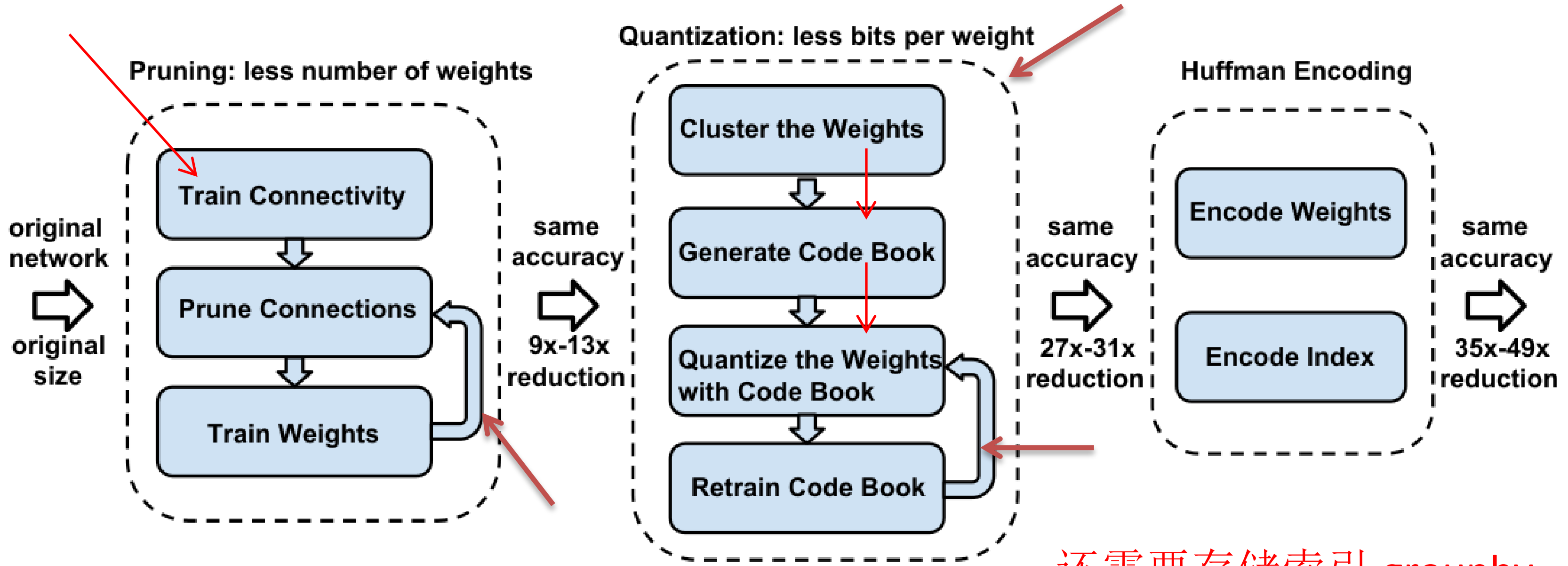
Figure 2: ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

Network pruning and sharing

- ➢ DEEP COMPRESSION:COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING,TRAINED QUANTIZATION AND HUFFMAN CODING(best paper 2016)
- ➢ Result：
- AlexNet by 35X, from 240MB to 6.9MB, VGG-16 by 49X from 552MB to 11.3MB without loss of accuracy.
- 3X to 4X layerwise speedup and 3X to 7X better energy efficiency

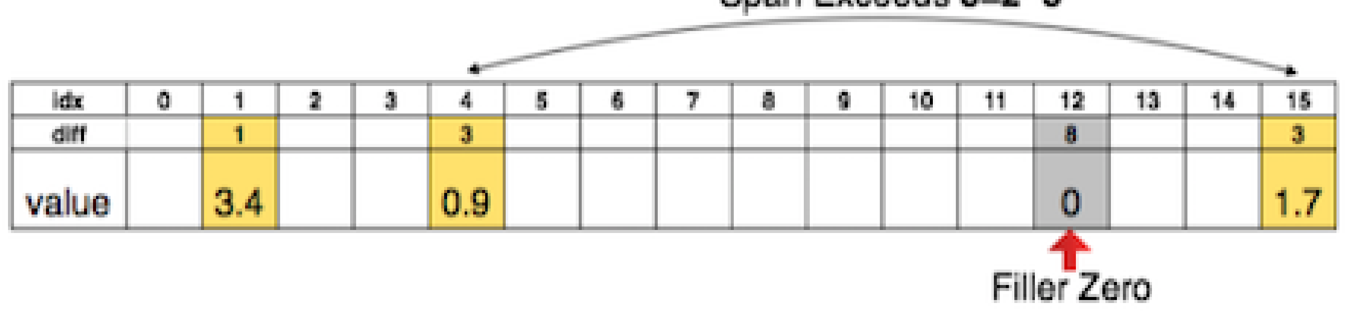
The three stage compression pipeline

Weights->codebook(effective weights after k-means)->quantize wights



还需要存储索引 groupby

Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by $10\times$, while quantization further improves the compression rate: between $27\times$ and $31\times$. Huffman coding gives more compression: between $35\times$ and $49\times$. The compression rate already included the meta-data for sparse representation. The compression scheme doesn't incur any accuracy loss.



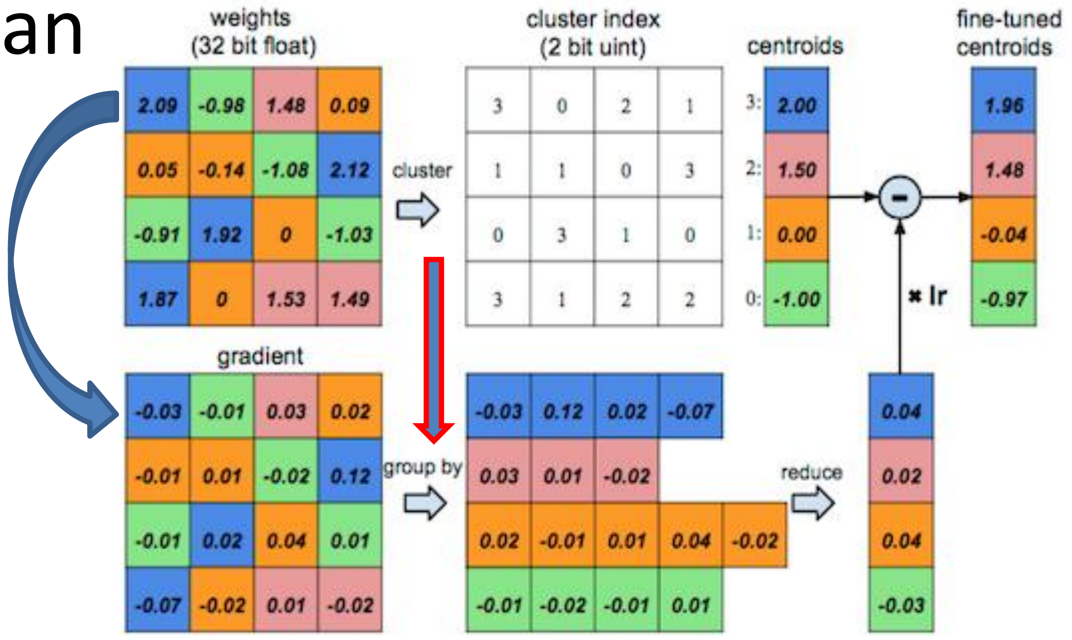
idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
diff		1			3								8			3
value		3.4			0.9								0			1.7

- ➢ Network Pruning
- proved to be a valid way to reduce the network complexity and over-fitting
- prune the small-weight connections below a threshold
- store the sparse structure using CSR or CSC format
- store the index difference instead of the absolute position, and encode this difference in 8 bits for conv layer and 5 bits for fc layer.
- When we need an index difference larger than the bound, we the zero padding solution shown in Figure 2: in case when the difference exceeds 8, the largest 3-bit (as an example) unsigned number, we add a filler zero

$16 * 32/(4 * 32 + 2 * 16) = 3.2$ (4*4filter每个位置属于4 (2bits)个聚类（codebook，effctive weight）中的哪一组

- Network quantization and weight sharing

- reducing the number of bits required to represent each weight and limit the number of effective weights(and fine-tune) : share the same weight(k-mean



➢K-means:centroid initialization

- larger weights play a more important role, but there are fewer large weights:more centroids have large absolute value
- Linear initialization :尽量包含所有的范围，密度大的地方压缩效率越高，越是我们想要的

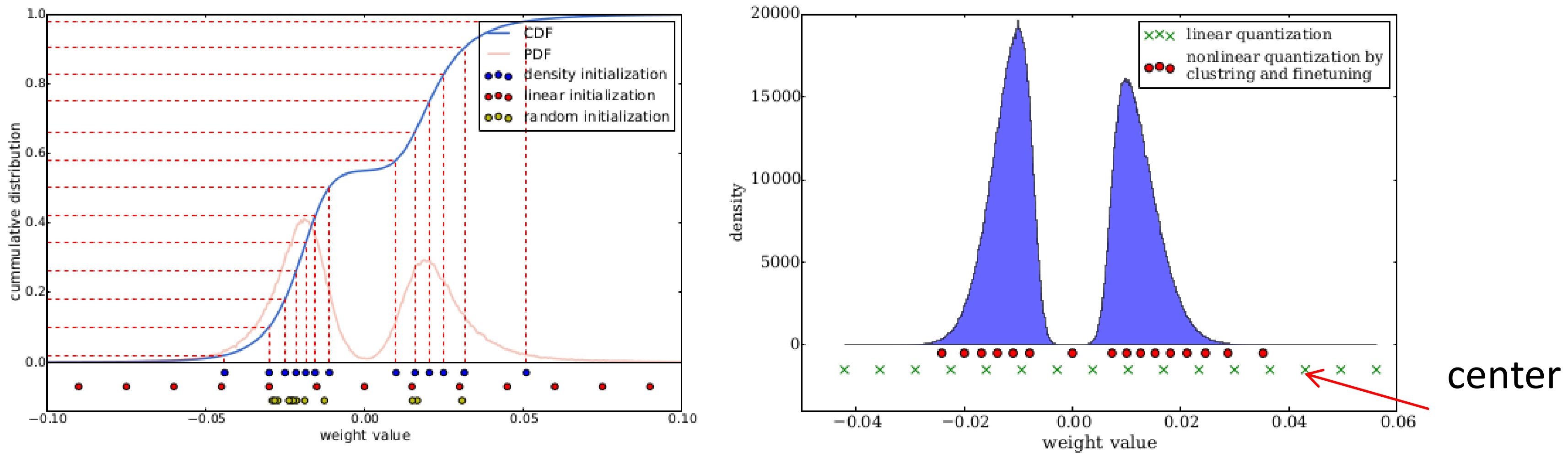
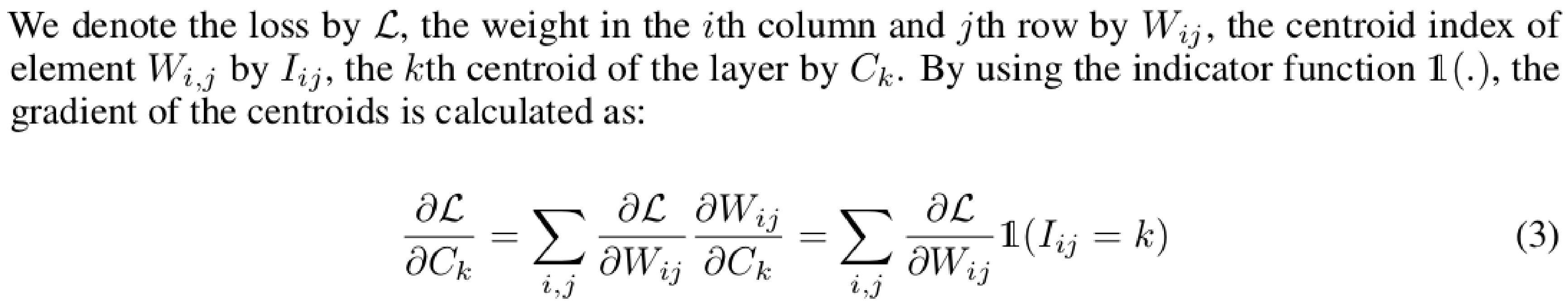


Figure 4: Left: Three different methods for centroids initialization. Right: Distribution of weights (blue) and distribution of codebook before (green cross) and after fine-tuning (red dot).

- FORWARD AND B ACK - PROPAGATION
- There is one level of indirection during feed forward phase and back-propagation phase looking up the weight table.An index into the shared weight table is stored for each connection

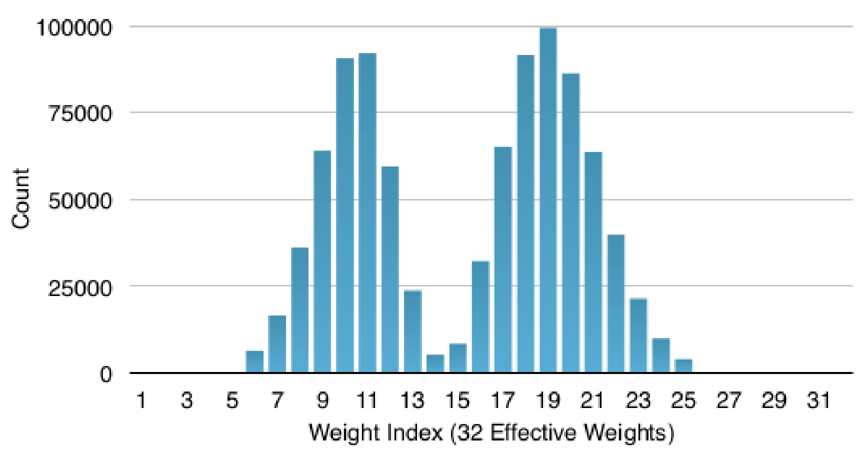


We denote the loss by $\mathcal{L}$, the weight in the ith column and jth row by W_{ij}, the centroid index of element $W_{i,j}$ by I_{ij}, the kth centroid of the layer by C_k. By using the indicator function $\mathbb{1}(.)$, the gradient of the centroids is calculated as:

$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \frac{\partial W_{ij}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \mathbb{1}(I_{ij} = k) \quad (3)$$

➢Huffman(lossless data compress) Coding

- The table is derived from the occurrence probability for each symbol. More common symbols are represented with fewer bits(variable-length codewords)



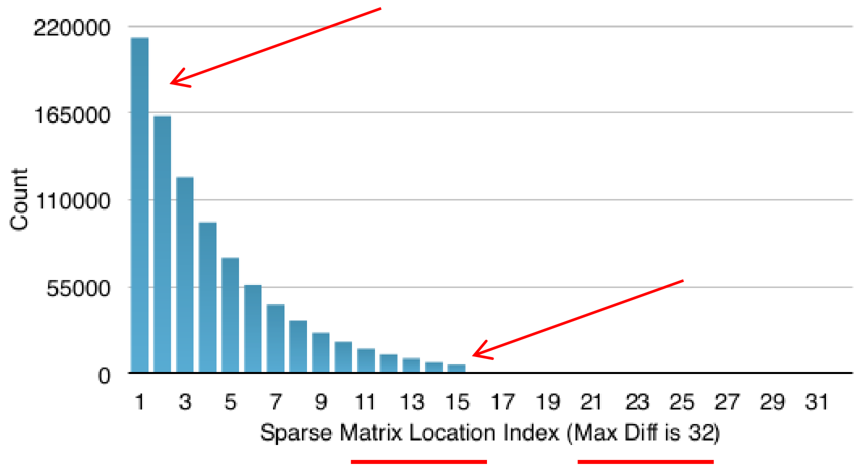


Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased.

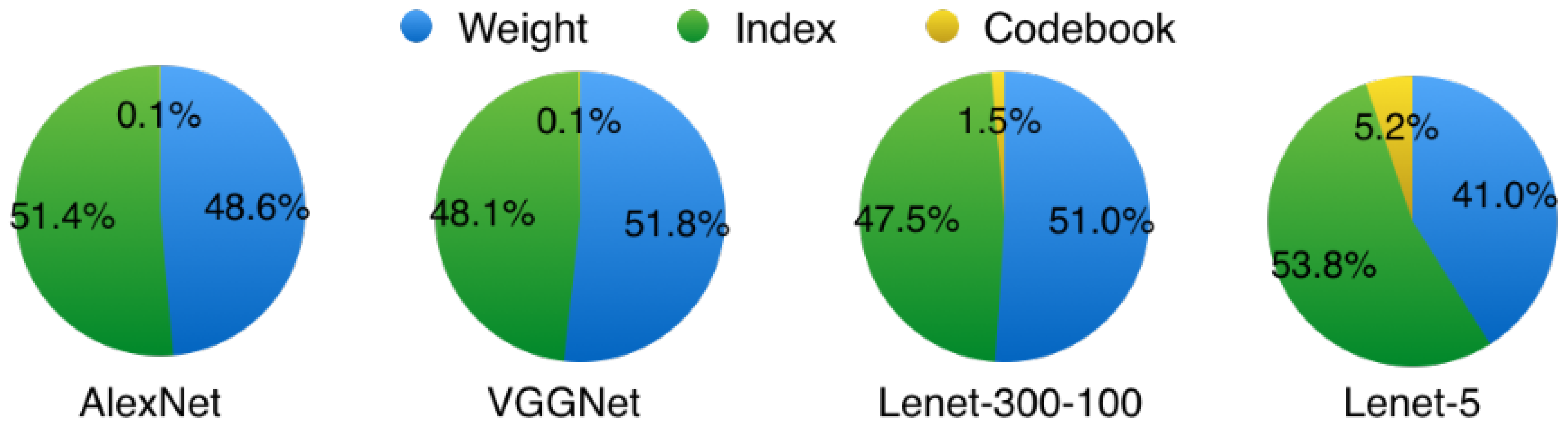
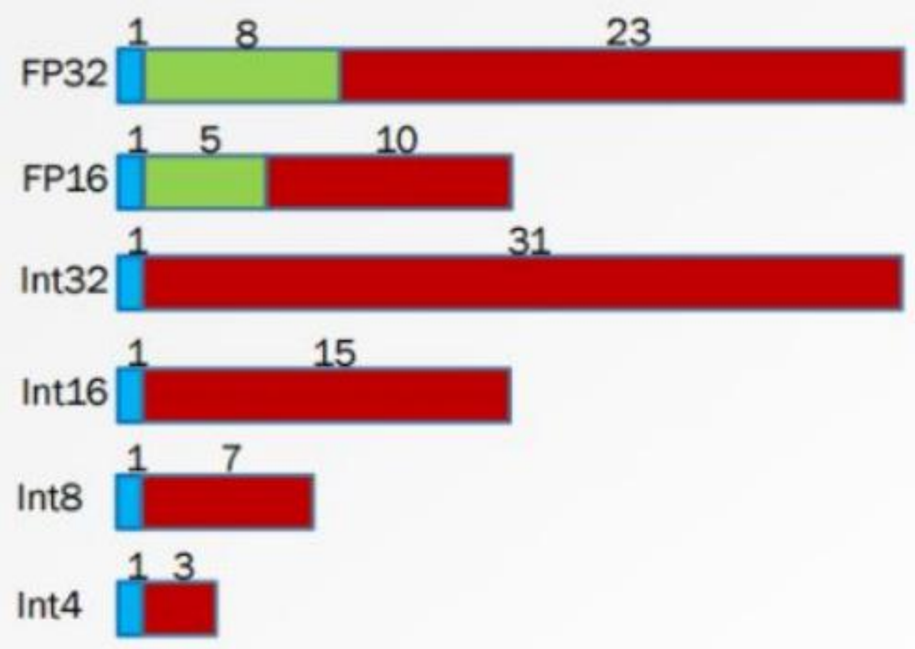
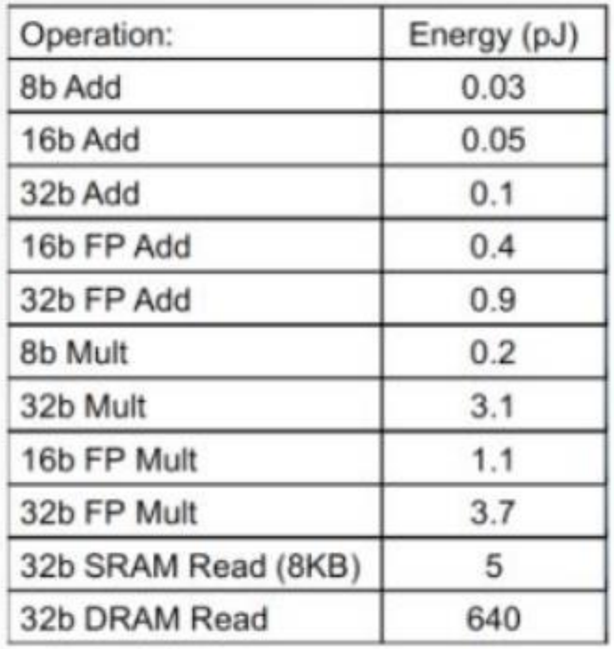


Figure 11: Storage ratio of weight, index and codebook.

Software optimization

- ➢Goal: Convert into FP32,FP16,INT8
- higher throughput, lower memory and Low energy consumption requirements.

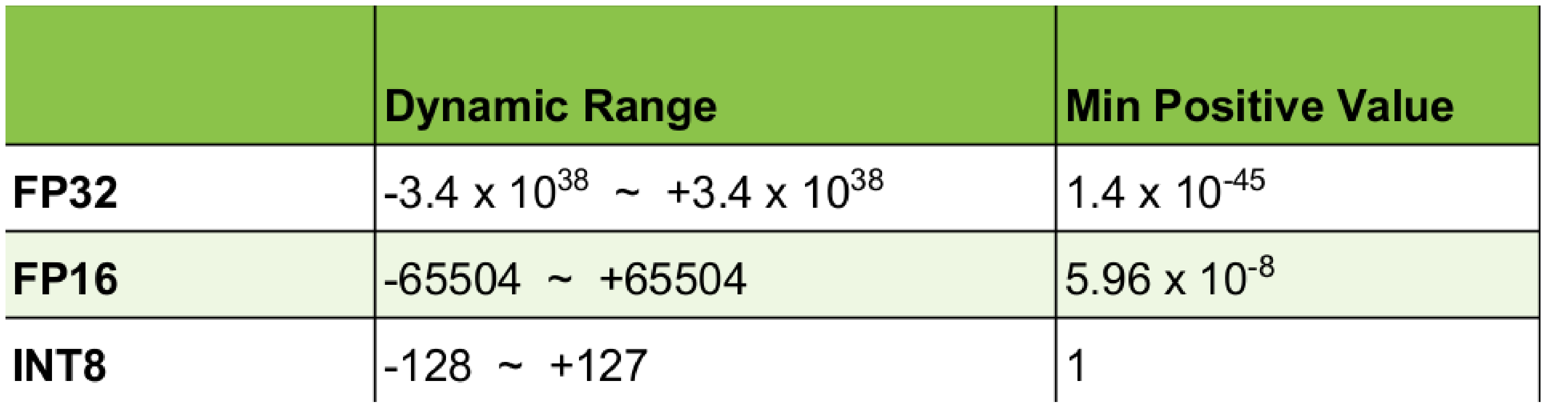




Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

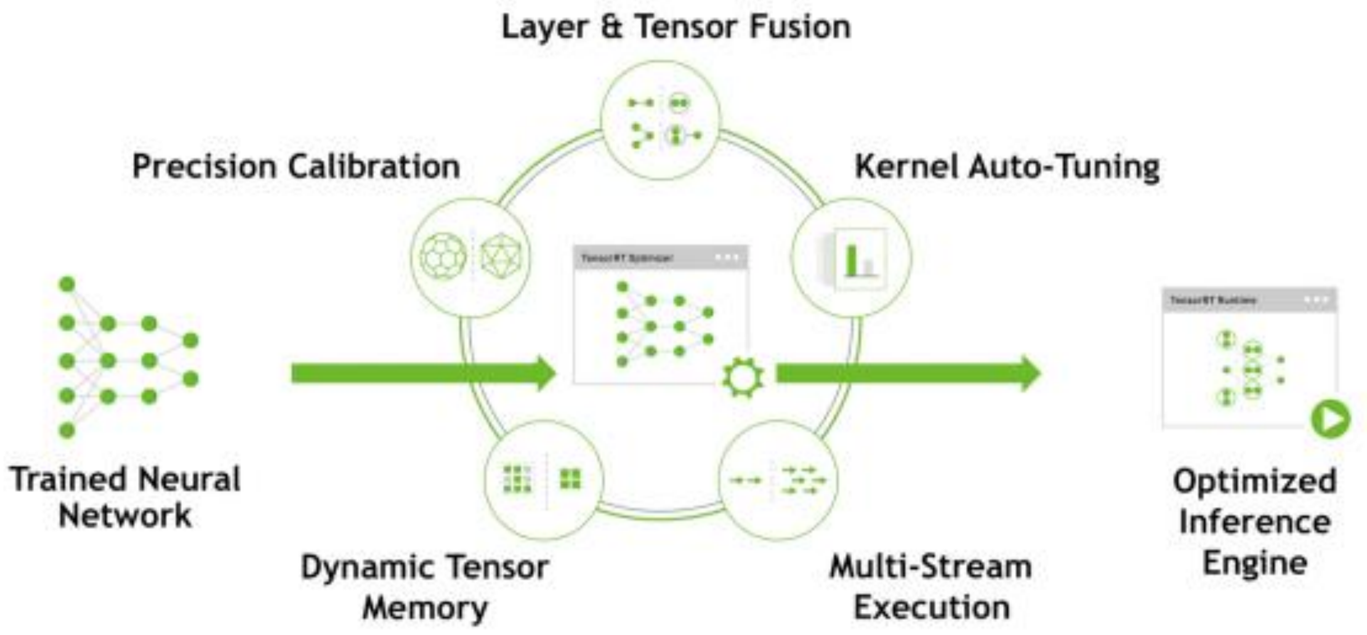
Software optimization

- Challenge: significantly lower precision and dynamic range



	Dynamic Range	Min Positive Value
FP32	-3.4×10^{38} ~ $+3.4 \times 10^{38}$	1.4×10^{-45}
FP16	-65504 ~ +65504	5.96×10^{-8}
INT8	-128 ~ +127	1

- TensorRT optimizes trained DNNs to produce adeployment-ready runtime inference engine



- Layer and tensor fusion and elimination of unused layers

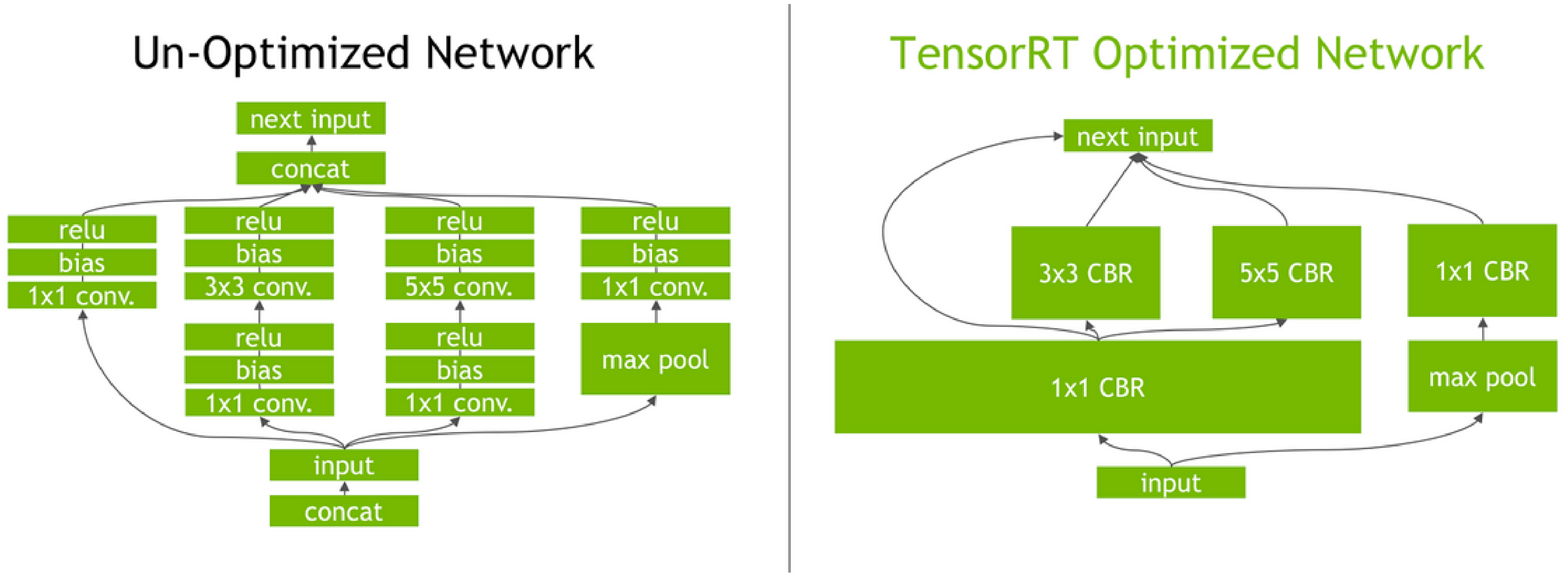


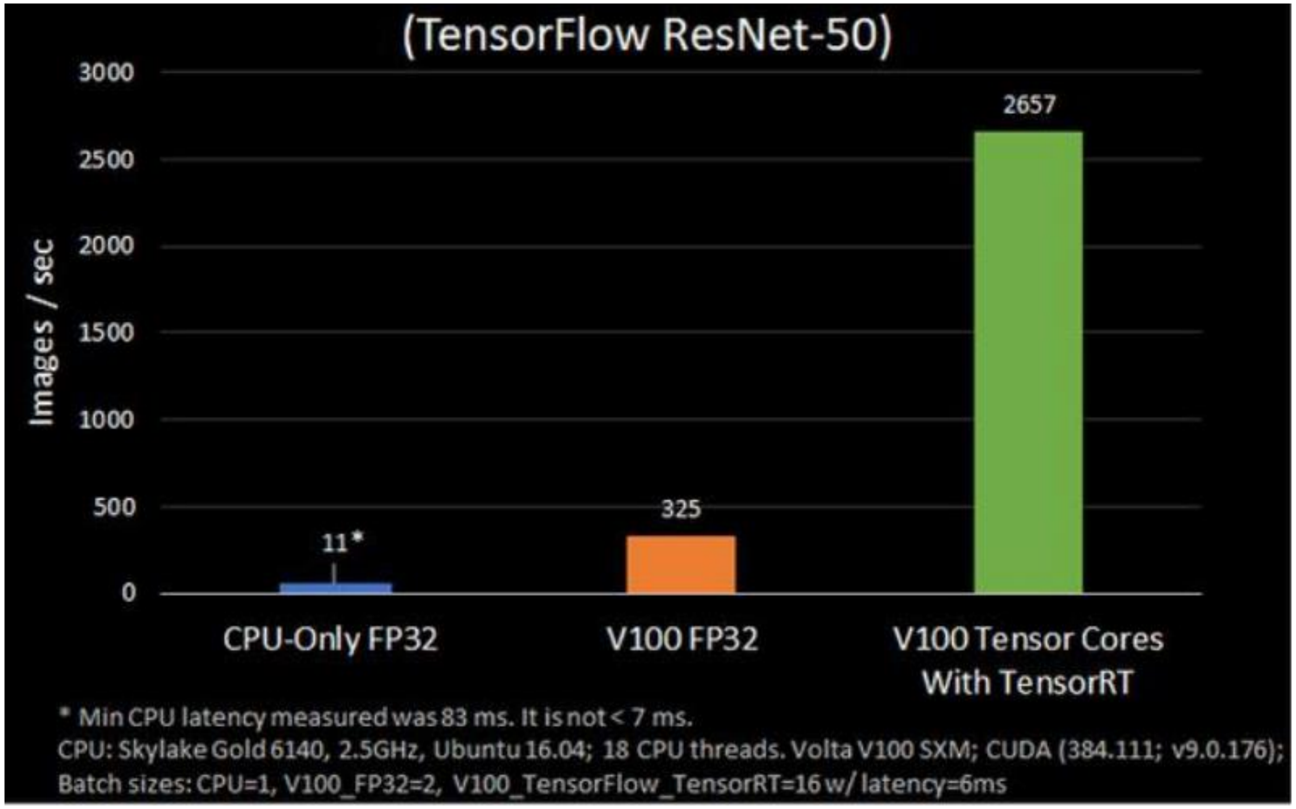
Figure 4. TensorRT's vertical and horizontal layer fusion and layer elimination optimizations simplify the GoogLeNet Inception module graph, reducing computation and memory overhead.

- FP16 and INT8 Precision Calibration:Tensor Values = FP32 scale factor * int8 array + FP32 bias
- Kernel Auto-Tuning: convolutions, pick the implementation from a library of kernels that delivers the best performance for the target GPU
- Dynamic Tensor Memory :improves memory reuse by designating memory for each tensor only for the duration of its usage, avoiding memory allocation overhead for fast and efficient execution.

- Symmetric linear quantization
- Do we really need bias? No!
- Q: How to optimize threshold selection? Saturate above |threshold| to 127

- During the optimization phase TensorRT also chooses from hundreds of specialized kernels, many of them hand-tuned and optimized for a range of parameters and target platforms. As an example, there are several different algorithms to do convolutions. TensorRT will pick the implementation from a library of kernels that delivers the best performance for the target GPU, input data size, filter size, tensor layout, batch size and other parameters.

Result



(TensorFlow ResNet-50)
Images / sec
3000
2500
2000
1500
1000
500
0
11*
325
2657
CPU-Only FP32
V100 FP32
V100 Tensor Cores With TensorRT
* Min CPU latency measured was 83 ms. It is not < 7 ms.
CPU: Skylake Gold 6140, 2.5GHz, Ubuntu 16.04; 18 CPU threads. Volta V100 SXM; CUDA (384.111; v9.0.176);
Batch sizes: CPU=1, V100_FP32=2, V100_TensorFlow_TensorRT=16 w/ latency=6ms

Discussion

- While the pruned network has been benchmarked on various hardware, the quantized network with weight sharing has not, because off-the-shelf cuSPARSE or MKL SPBLAS library does not support indirect matrix entry lookup, nor is the relative index in CSC or CSR format supported. So the full advantage of Deep Compression that fit the model in cache is not fully unveiled. A software solution is to write customized GPU kernels that support this. A hardware solution is to build custom ASIC architecture specialized to traverse the sparse and quantized network structure, which also supports customized quantization bit width. We expect this architecture to have energy dominated by on-chip SRAM access instead of off-chip DRAM access.

suggestions

- Pruning and sharing :get reasonable compression rate with no accurancy loss???
- pre-trained models, you can choose either pruning & sharing,low rank;low rank and transferred conv filters is end-to-end solution
- Special domain:medical images classification transterred filter(have rotation transformation property)
- KD
- Object detection:conv layer :low rank;fc :pruning

challenges

- Channel pruning may change the input of following layer;
- structural matrix and transferred convolutional filters impose prior human knowledge to the model, which could significantly affect the performance and stability.
- KD-based approaches and exploring how to improve their performances.

Trends

- None-fine-tuning or unsupervised compression
- Hyperparameters(pruning,low rank,bitwidth of fixed-point quantization)
- Object detection
- Hardware-software co-design

Reference

- Recent Advances in Efficient Computation of Deep Convolutional Neural Networks
- DEEP COMPRESSION : COMPRESSING DEEP N EURALN ETWORKS WITH PRUNING , TAINED Q UANTIZATIONAND HUFFMAN CODING
- https://github.com/Ewenwan/Caffe-Python-Tutorial/blob/master/quantize.py
- https://github.com/Ewenwan/quantized-cnn
- https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide

Thanks