

Mastering Transformers: From Building Blocks to Real-World Applications

Optimizations

Prof. Alptekin Temizel

Graduate School of Informatics, Middle East Technical University

13 Sept 2023



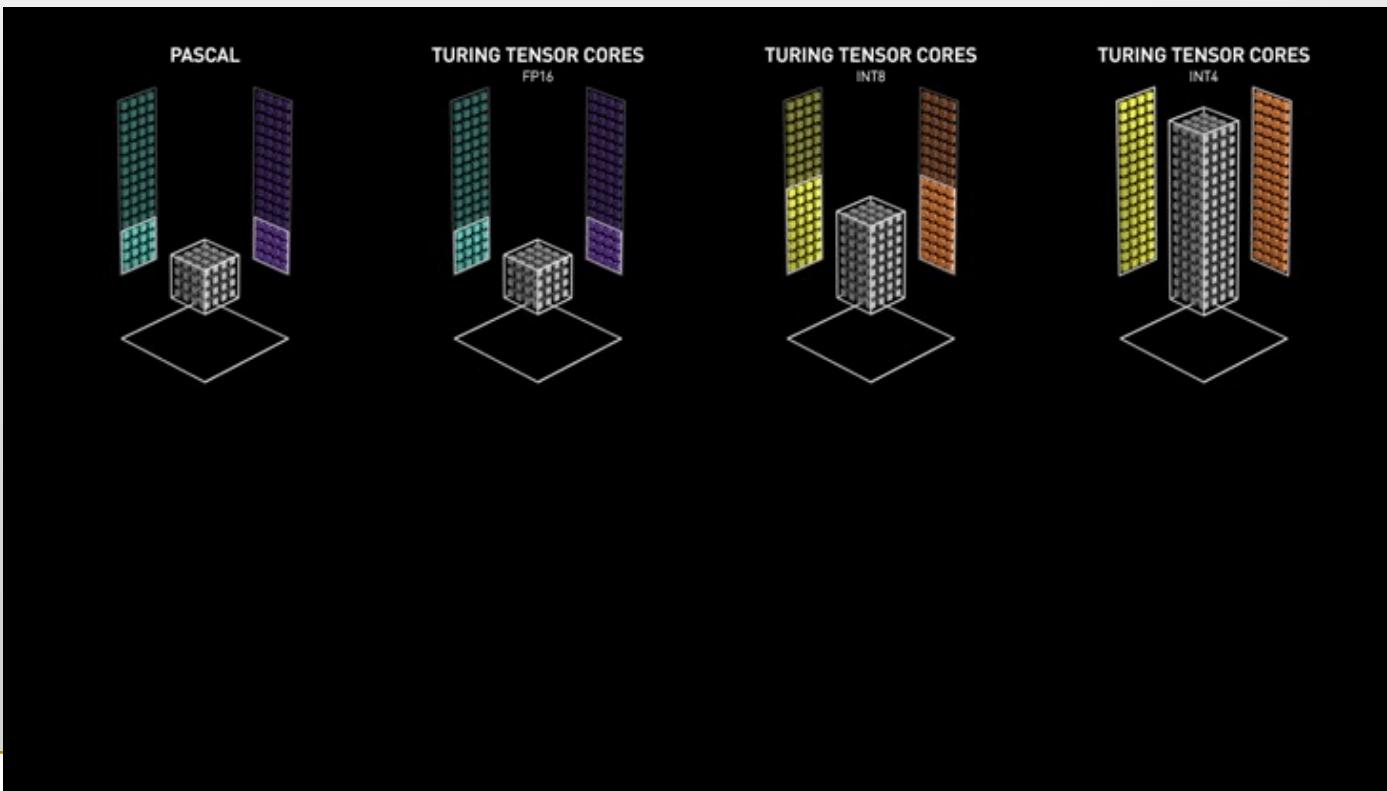
Fast inference

Use more powerful hardware and/or optimize your models:

- Make models smaller
- Make models run faster

NVIDIA Tensor Cores

- Tensor Cores are programmable fused matrix-multiply-and-accumulate units that execute concurrently alongside the CUDA cores.
- Tensor Cores implement floating point HMMA (Half Precision Matrix Multiply and Accumulate) and IMMA (Integer Matrix Multiple and Accumulate) instructions for accelerating dense linear algebra computations.



NVIDIA Tensor Cores

- ❑ Tensor Cores are already supported for deep learning frameworks: TensorFlow, PyTorch, MXNet, and Caffe2.
- ❑ Mixed-Precision Training Guide gives more information about enabling Tensor Cores when using these frameworks
- ❑ TensorRT 3.0 release also supports Tensor Cores for deep learning inference.
- ❑ cuBLAS uses Tensor Cores to speed up GEMM computations (GEMM is the BLAS term for a matrix-matrix multiplication)
- ❑ cuDNN uses Tensor Cores to speed up both convolutions and recurrent neural networks (RNNs).

NVIDIA Tensor Cores – 3rd Gen

- The NVIDIA Ampere brings support for new precisions—Tensor Float 32 (TF32) and floating point 64 (FP64).
- TF32 works just like FP32 while delivering speedups of up to 20X for AI without requiring any code change.
- Automatic Mixed Precision can provide an additional 2X performance with automatic mixed precision and FP16 by adding just a couple of lines of code.
- And with support for bfloat16, INT8, and INT4, Tensor Cores in NVIDIA Ampere architecture Tensor Core GPUs create an incredibly versatile accelerator for both AI training and inference.
- HPC grade devices A100 and A30 GPUs also enable matrix operations in full FP64 precision.

Model Optimization

- Quantization: Accelerating by reducing the precision of the data types.
- Pruning/truncation: Removing the unused parts of the model.
- Knowledge Distillation: Transferring knowledge from a large model to a smaller one.

Model Compression

- Reduces the size of a model by using fewer bits to represent parameter values.
 - E.g. half-precision floating point (16-bit), or integer (8-bit) representations
- Pros:
 - Reduces memory footprint and computation speed
- Cons:
 - Rounding errors can have downstream effects.
 - Rounding can lead to over/under-flow, or zeroing.
 - Non-trivial to implement efficient rounding/scaling.

Quantization

Speed-Up

	Batch size 1			Batch size 8			Batch size 128		
	FP32	FP16	Int8	FP32	FP16	Int8	FP32	FP16	Int8
MobileNet v1	1	1.91	2.49	1	3.03	5.50	1	3.03	6.21
MobileNet v2	1	1.50	1.90	1	2.34	3.98	1	2.33	4.58
ResNet50 (v1.5)	1	2.07	3.52	1	4.09	7.25	1	4.27	7.95
VGG-16	1	2.63	2.71	1	4.14	6.44	1	3.88	8.00
VGG-19	1	2.88	3.09	1	4.25	6.95	1	4.01	8.30
Inception v3	1	2.38	3.95	1	3.76	6.36	1	3.91	6.65
Inception v4	1	2.99	4.42	1	4.44	7.05	1	4.59	7.20
ResNext101	1	2.49	3.55	1	3.58	6.26	1	3.85	7.39

TensorRT optimized models executed on Tesla T4, input size 224x224 for all apart from the Inception networks for which the input size was 299x299

The Advantages of Lower Precision Formats

- Higher Throughput
 - Faster math operations such as convolution
- Reduced Memory Bandwidth Requirement
- Reduced Memory Size Need
 - Better cache utilization

Input Data type	Accumulation Data type	Math Throughput	Bandwidth Reduction
FP32	FP32	1x	1x
FP16	FP16	8x	2x
INT8	INT32	16x	4x
INT4	INT32	32x	8x
INT1	INT32	128x	32x

Table 1: Benefits of lower precision data types for tensor operations on the NVIDIA Turing GPU architecture

The Advantages of Lower Precision Formats



Raymond Lo 🎓🌐 · Following

AI Software Evangelist at Intel (Global Lead)

2d · Edited · 🌐

...

We have Intel Innovation 2023 coming up in less than 2 weeks! I will be there in person at San Jose Convention Center to show you how the latest Gen AI works on the latest Intel hardware. We are brewing lots and lots of demos including Stable Diffusion XL, Llama2 and other LLM chatbots, Code Completion (DeciCoder), Music Generation (Riffusion) and much more can run locally on your laptops (CPU, iGPU, dGPU, and even NPUs (coming up!) thanks to OpenVINO's optimizations! ;) Hint: Low precision is the key/king! :) For sure join us as we will explain how and why they work so well in these use cases.

Quantization Basics

Let $[\beta, \alpha]$ be the range of representable real values chosen for quantization and b be the bit-width of the signed integer representation.

Uniform quantization transforms the input value $x \in [\beta, \alpha]$ to lie within $[-2^{b-1}, 2^{b-1} - 1]$, where inputs outside the range are clipped to the nearest bound.

Considering only uniform transformations, there are only two choices for the transformation function:

$$f(x) = s \cdot x + z$$

and its special case

$$f(x) = s \cdot x$$

Quantization Basics

- Ten

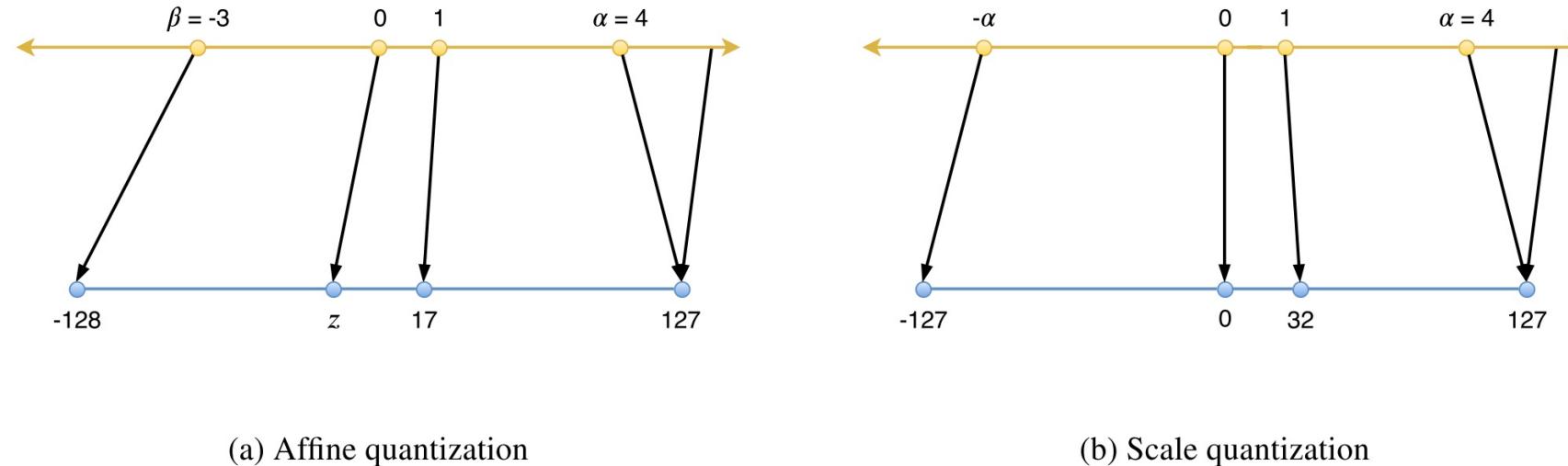


Figure 1: Quantization mapping of real values to int8

Quantization Basics

$$\text{clip}(x, l, u) \begin{cases} l, & x < l \\ x, & l \leq x \leq u \\ u, & x > u \end{cases}$$

$$x_q = \text{quantize}(x, b, s, z) = \text{clip}(\text{round}(s \cdot x + z), -2^{b-1}, 2^{b-1} - 1)$$

$$\hat{x} = \text{dequantize}(x_q, s, z) = \frac{1}{s}(x_q - z)$$

Cost of Affine Transform

- The cost increases when affine is used in weight quantization
 - Otherwise, online calculation head is added.
 - The scale quantization is enough for activation

Calibration

- The clipping range should be determined from the data
- Forwardpass some sample images and record an histogram for every activation(input) tensors
- 100-1000 samples can be enough to determine the limits. Generally 512 and 1024 in the paper.
- Selecting minumum maximum is the best way?

Calibration Types

- Rounding Error vs Clipping Error Tradeoff
- Max Calibration
- Percentile (99.9, 99.99, 99.999)
- Entropy Calibration

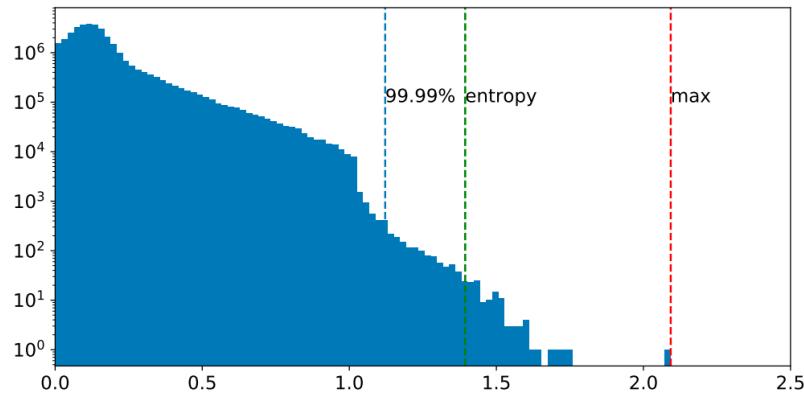


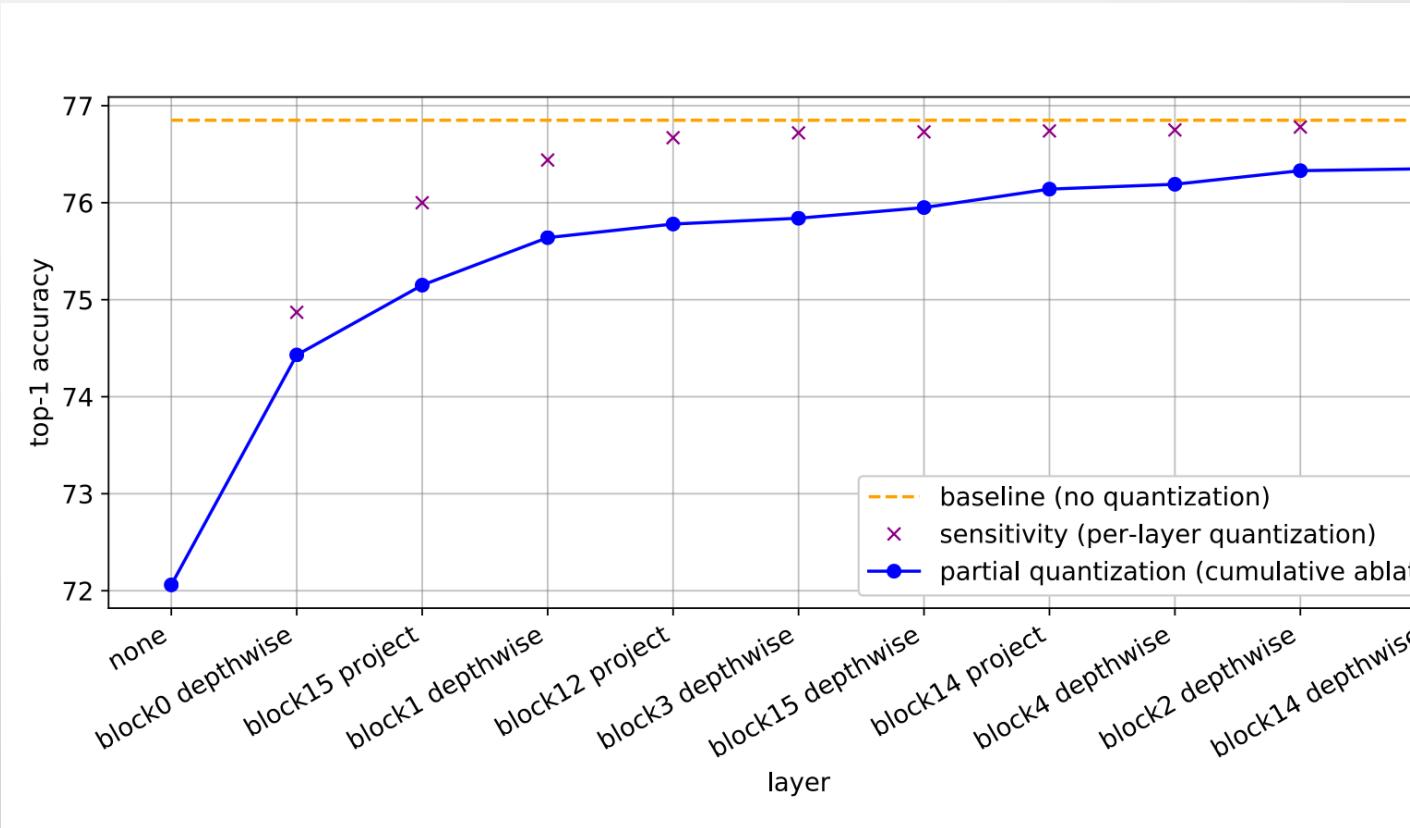
Figure 2: Histogram of input activations to layer 3 in ResNet50 and calibrated ranges

Partial Quantization

- The single scale per tensor
- The scale for every element in the tensor
- Per row/column for 2D tensors, and per-channel for 3D tensors

Partial Quantization

- Sensitive Layer
 - Quantize only a single layer and measure the performance drop



Weight Quantization

In this case, only the weights are quantized and activations remain full-precision

Tensor Quantization Granularity:

- Coarsest: per-tensor, the same quantization parameters are shared by all elements in the tensor.
- Finest: individual quantization parameters per element.
- Intermediate granularities reuse parameters over various dimensions of the tensor - per row or per column for 2D matrices, per channel for 3D (image-like) tensors, etc.

Model	fp32	Per-channel	Per-channel fold BN	Per-tensor	Per-tensor fold BN
MobileNet v1	71.88	71.59	71.59	69.58	66.88
MobileNet v2	71.88	71.61	71.61	71.12	70.21
ResNet50 v1.5	76.16	76.14	76.14	75.83	75.84
ResNeXt50	77.61	77.62	77.62	77.48	77.45
EfficientNet b0	76.85	76.72	76.72	76.68	12.93

Table 3: Accuracy with int8 quantization of weights only: per-tensor vs per-channel granularity. Fold BN indicates batch norms were folded into the preceding convolution before quantization

Activation Quantization

- In addition to the weights, activations are also quantized
- Different calibrations may be needed for activations

Post training quantization

Models	fp32	Max	Entropy	99.9%	99.99%	99.999%	99.9999%
MobileNet v1	71.88	69.51	70.19	70.39	70.29	69.97	69.57
MobileNet v2	71.88	69.41	70.28	70.68	71.14	70.72	70.23
ResNet50 v1.5	76.16	75.82	76.05	75.68	75.98	75.97	76.00
ResNet152 v1.5	78.32	77.93	78.21	77.62	78.17	78.17	78.19
Inception v3	77.34	72.53	77.54	76.21	77.52	77.43	77.37
Inception v4	79.71	0.12	79.60	78.16	79.63	79.12	71.19
ResNeXt50	77.61	77.31	77.46	77.04	77.39	77.45	77.39
ResNeXt101	79.30	78.74	79.09	78.77	79.15	79.17	79.05
EfficientNet b0	76.85	22.3	72.06	70.87	68.33	51.88	42.49
EfficientNet b3	81.61	54.27	76.96	77.80	80.28	80.06	77.13
Faster R-CNN	36.95	36.38	36.82	35.22	36.69	36.76	36.78
Mask R-CNN	37.89	37.51	37.75	36.17	37.55	37.72	37.80
Retinanet	39.30	38.90	38.97	35.34	38.55	39.19	39.19
FCN	63.70	63.40	64.00	62.20	64.00	63.90	63.60
DeepLabV3	67.40	67.20	67.40	66.40	67.40	67.50	67.40
GNMT	24.27	24.31	24.53	24.34	24.36	24.38	24.33
Transformer	28.27	21.23	21.88	24.49	27.71	20.22	20.44
Jasper	96.09	95.99	96.11	95.77	96.09	96.09	96.03
BERT Large	91.01	85.92	37.40	26.18	89.59	90.20	90.10

Partial Quantization

Model	fp32	Calibration	Total quantized layers	Full int8	Accuracy	Partial int8	
	Accuracy			Accuracy		Skipped layers	Accuracy
MobileNet v1	71.88	max	28	69.51	69.51	2	71.50
EfficientNet b0	76.85	entropy	82	72.06	72.06	10	76.35
EfficientNet b3	81.61	99.99%	131	76.96	76.96	3	81.27
Transformer	28.27	max	121	21.23	21.23	5	28.20
BERT large	91.01	max	244	85.92	85.92	141	90.41

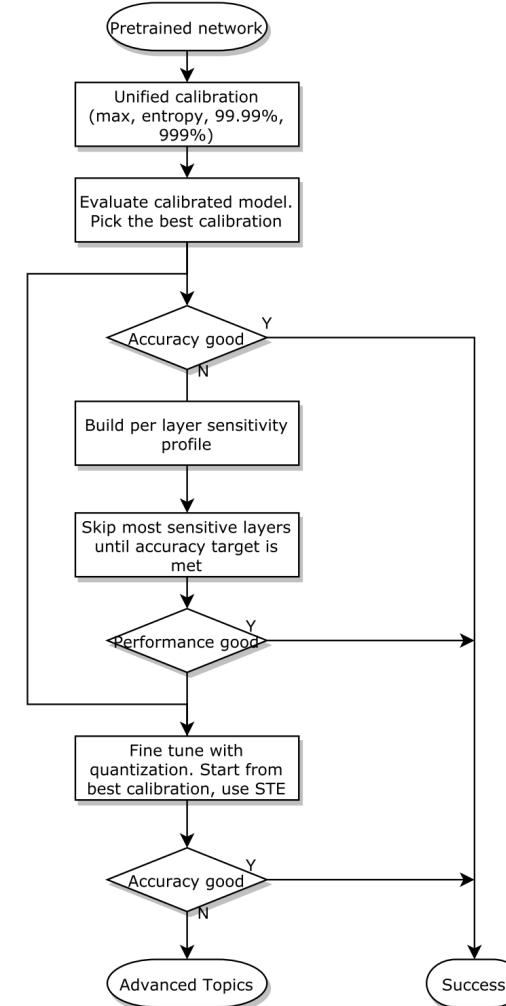
Table 6: Partial post training quantization

Quantization Aware Training

- Quantization Aware Training (QAT)
 - What if we fine tune over quantized activations and weights
 - The idea is to add fake quantize and dequantize functions
 - Operations is still on fp32 performance
 - Does not require architecture change
- Learns also scales during fine-tuning (PACT)
 - But not a difference between PACT and PTQ+QAT

Quantization Workflow

- Weights
 - Use scale quantization with per-column/per-channel granularity
 - Use a symmetric integer range for quantization [-127, 127]) and max calibration
- Activations
 - Use scale quantization with with per-tensor granularity



Inference Results on CuLane Dataset

TensorRT Inference						
Performance inference of R-18-DeeplabV3						
Category	Torch	TensorRT fp32	TensorRT fp16	TensorRT int8	TensorRT ptq int8 100 samples calibration	TensorRT qat wo ptq
Normal	90.2	90.2	90.2	89.0	89.5	90.3
Crowded	71.9	71.9	71.9	71.0	71.3	71.2
Night	66.4	66.4	66.3	63.5	65.0	63.9
No Line	43.2	43.2	43.2	42.3	42.9	39.9
Shadow	72.3	72.3	72.5	69.7	71.7	68.0
Arrow	85.5	85.5	85.4	83.6	84.3	85.7
Dazzle Light	63.8	63.8	63.8	59.2	62.6	60.5
Curve	66.8	66.8	66.7	63.4	65.1	64.0
Overall	72.9	72.9	72.9	71.3	72.1	71.6
Crossroad	2131	2132	2129	1821	1991	1371
Total Inference Time	07:00	06:47	04:52	04:28	04:37	04:37
Total Inference Ratio	1	0.97	0.7	0.64		

Calibration Results

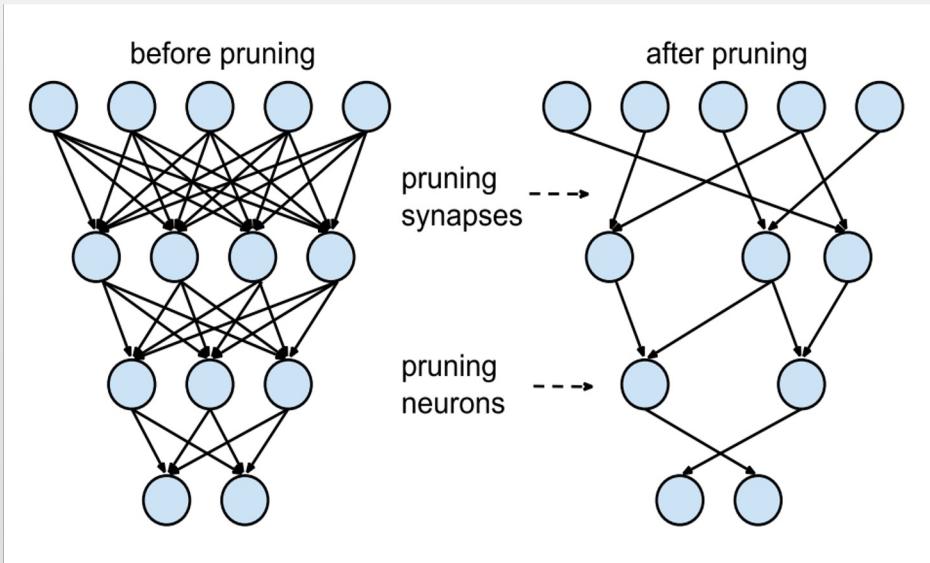
Baseline	Max	Entropy	Mse	99.9	99.99	99.999	99.9999
60.0	59.8	59.72	59.78	53.1	58.92	59.67	59.82

mIoU results with respect to calibration samples

32	64	128	256	512
59.90	59.82	59.84	59.86	59.82

Pruning

- Remove the less useful parameters.
 - Efficient storage schemes for sparse data.
- Can prune either weights or neurons:
 - Weights: set weights which are almost zero to zero.
 - Neurons: neurons with invariant activations can be removed.

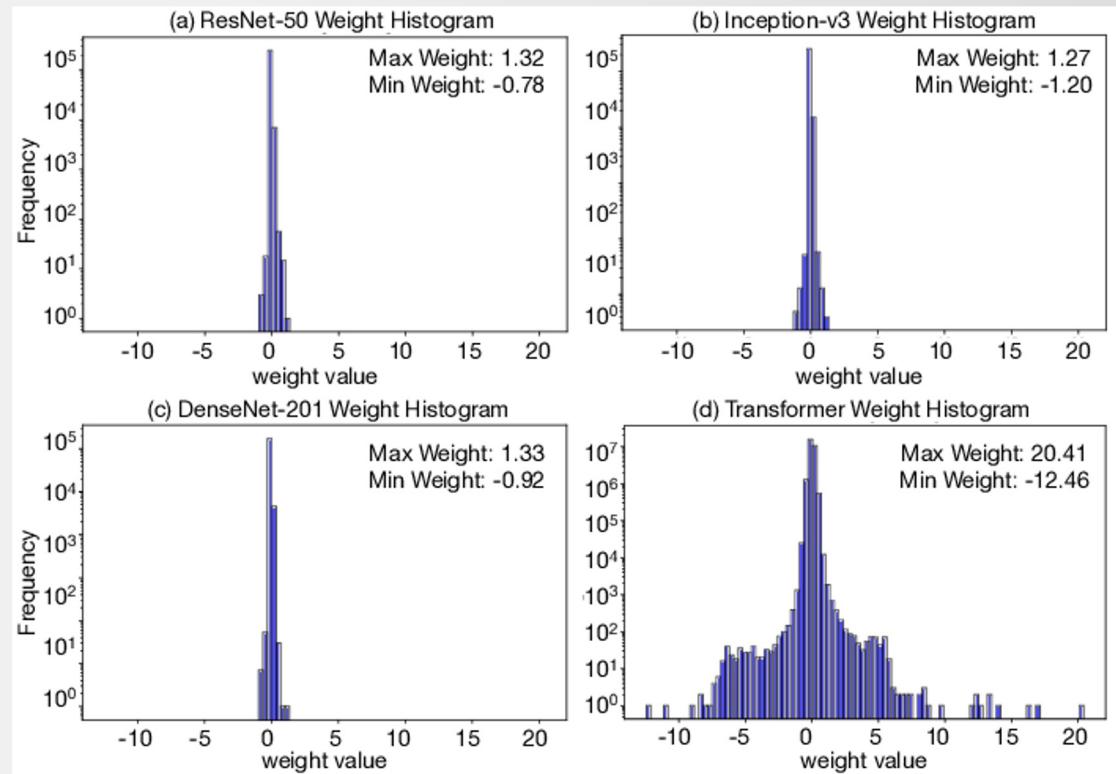
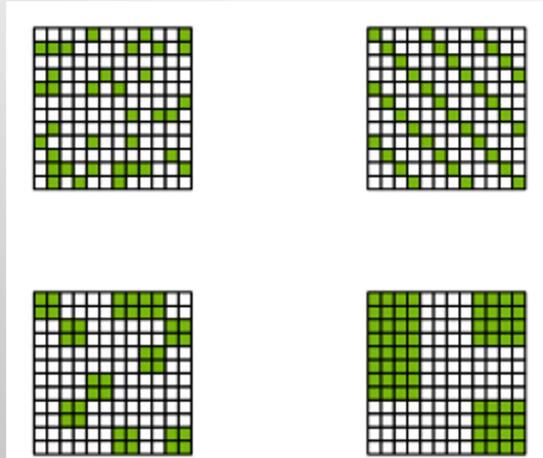


Pruning

- Mainstream pruning approaches can be roughly categorized into two schemes, *structural pruning* and *unstructural pruning*.
- Structural pruning changes the structure of neural networks by physically removing grouped parameters
- Unstructural pruning conducts zeroing on partial weights without modification to the network structure.
- Compared to unstructural ones, structural pruning does not rely on specific AI accelerators or software to reduce memory consumption and computational costs, thereby finding a wider domain of applications in practice [1]

Pruning

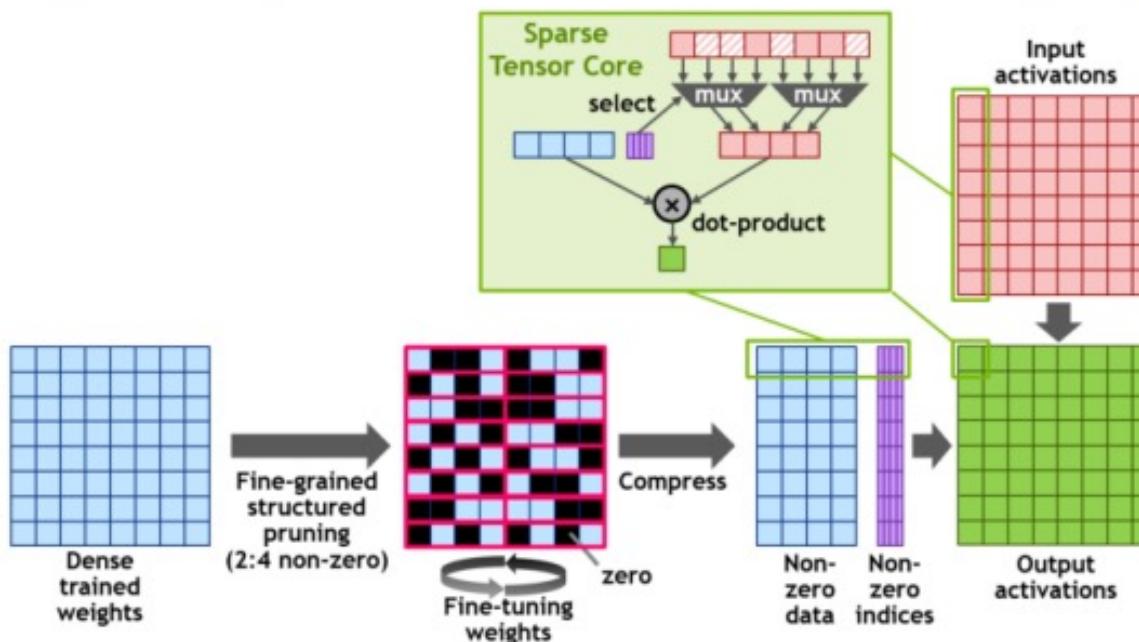
- Reduced memory bandwidth
- Reduced memory footprint
- Acceleration (especially in presence of hardware acceleration)



NVIDIA Tensor Cores – 3rd Gen

- Ampere architecture brings support for sparsity feature.
- Sparsity feature takes advantage of the fine-grained structured sparsity in deep learning networks to double the throughput of the Tensor Core operations.
- Sparsity is constrained to 2 out of every 4 weights being nonzero. It enables the tensor core to skip zero values, doubles the throughput, and reduces the memory storage significantly.
- Networks can be trained first on dense weights, and then pruned, and later fine-tuned on sparse weights.

Figure 5: Ampere GPU 3rd Generation Tensor Core Sparsity



Structured Sparsity in NVIDIA GPUs

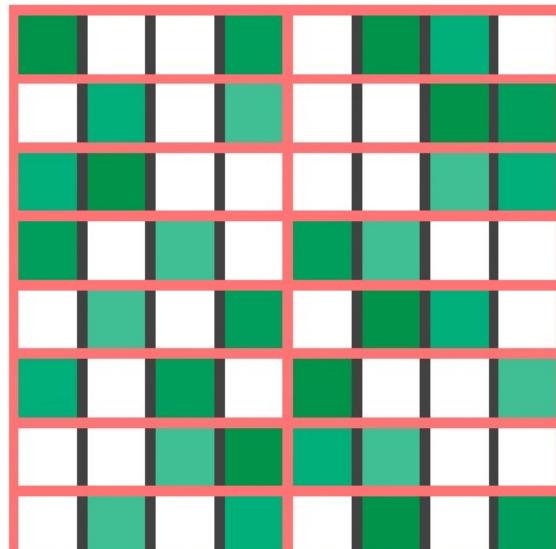
Fine-grained structured sparsity for Tensor Cores

- 50% fine-grained sparsity
 - 2:4 pattern: 2 values out of each contiguous block of 4 must be 0
 -

Addresses the 3 challenges:

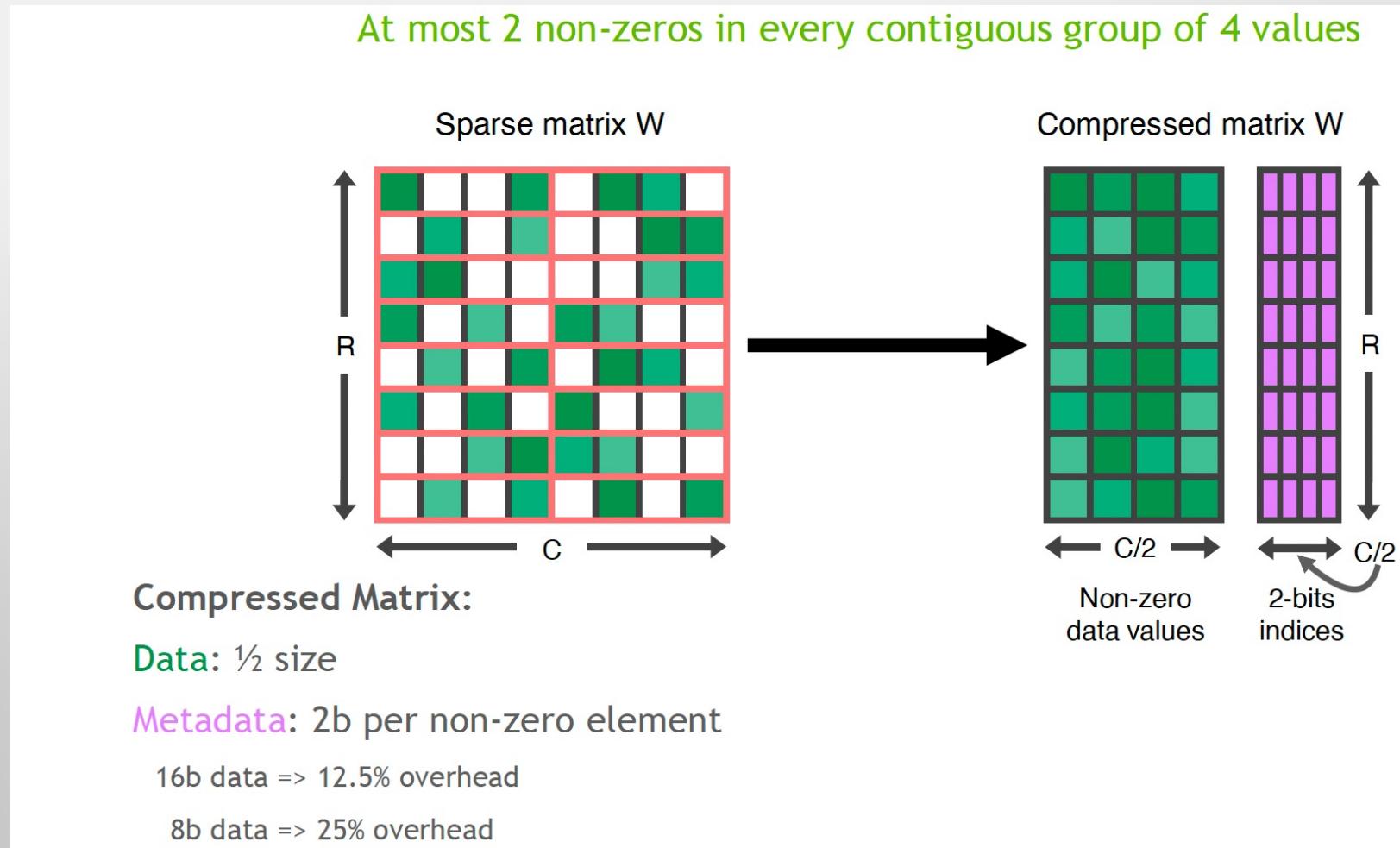
- Accuracy: maintains accuracy of the original, unpruned network
 - Medium sparsity level (50%), fine-grained
 - Training: a recipe shown to work across tasks and networks
 - Speedup:
 - Specialized Tensor Core support for sparse math
 - Structured: lends itself to efficient memory utilization

2:4 structured-sparse matrix

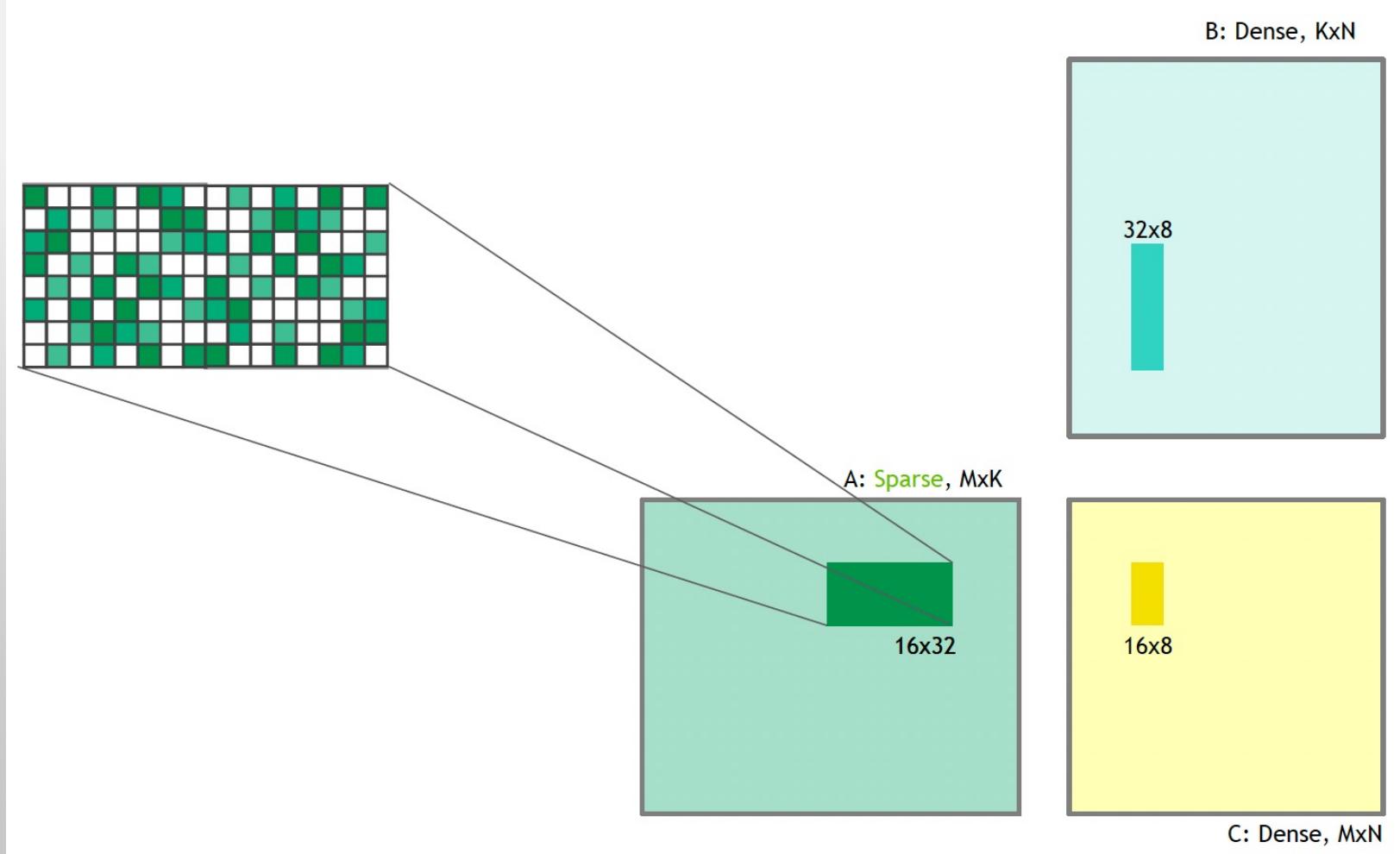


 = zero value

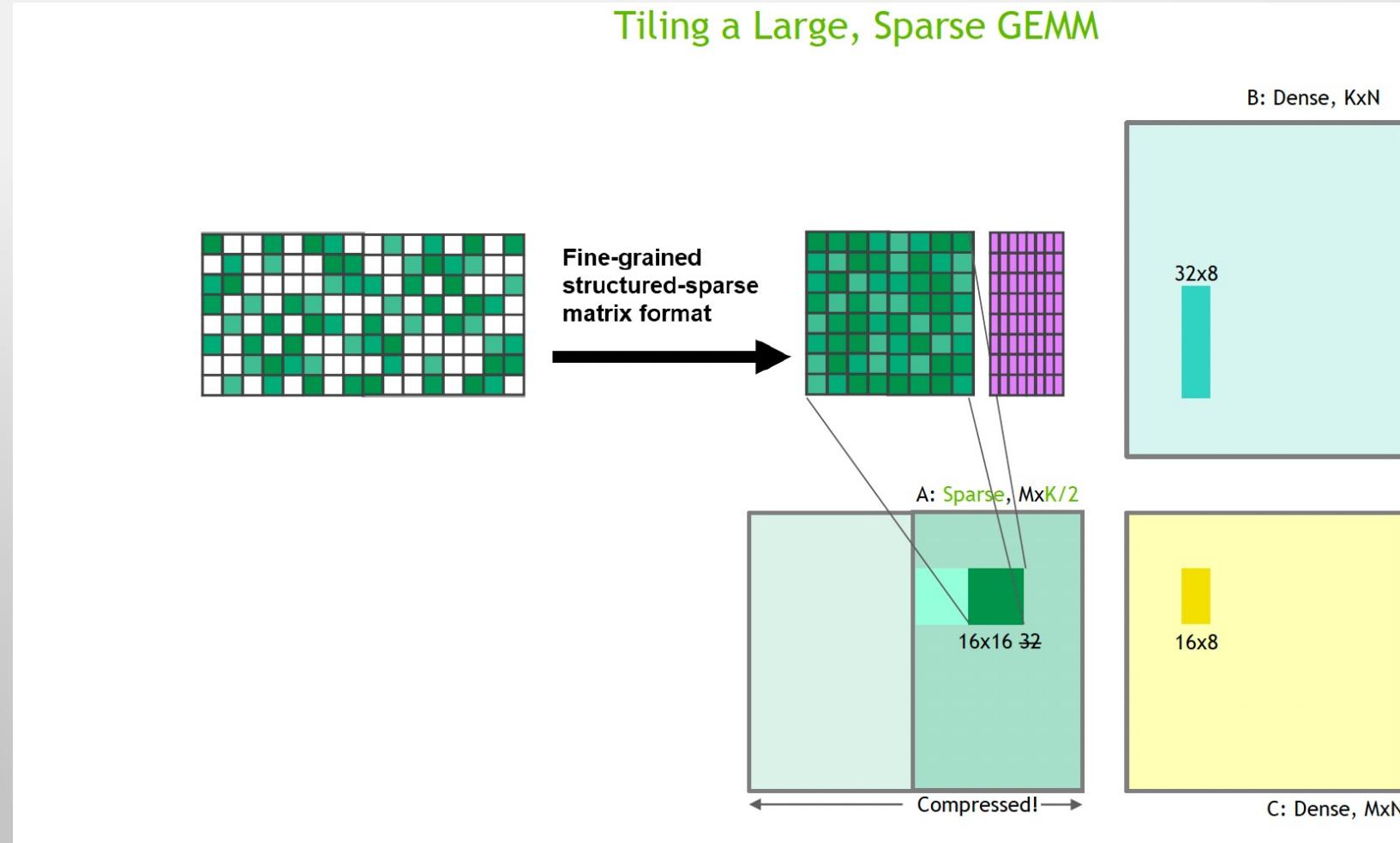
Sparsity in NVIDIA GPUs



Sparsity in NVIDIA GPUs



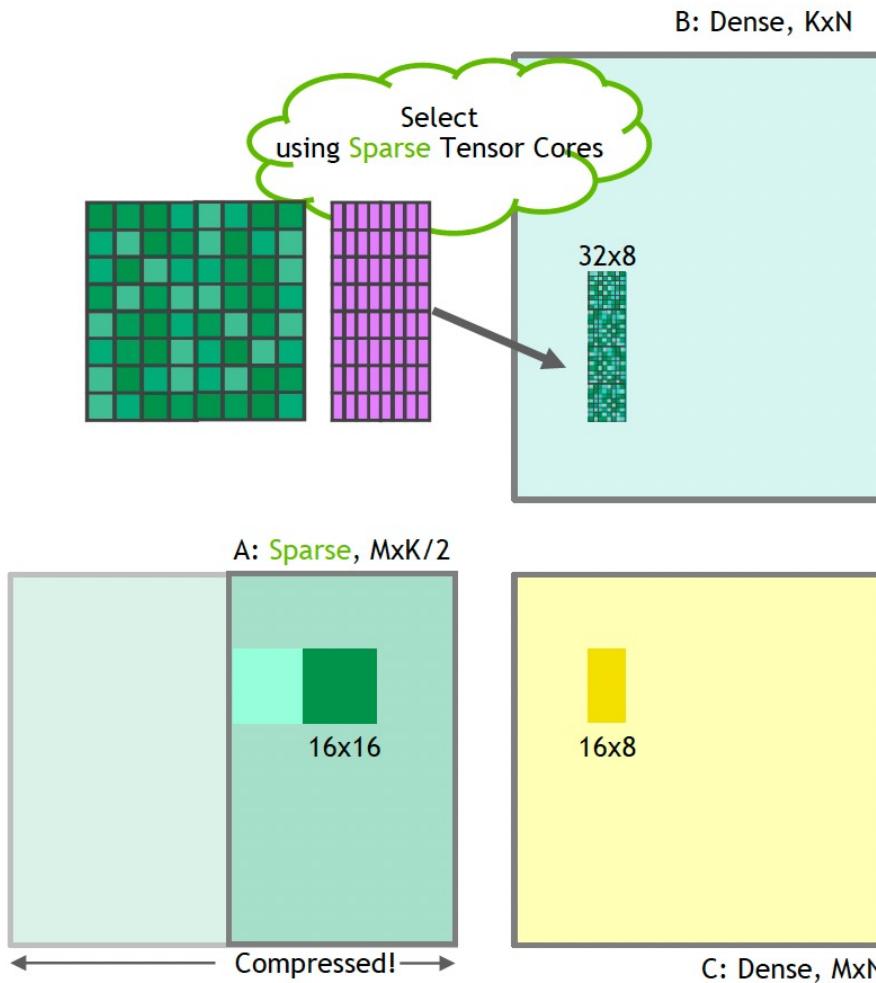
Sparsity in NVIDIA GPUs



Sparsity in NVIDIA GPUs

Sparse Tensor Cores - Hardware Magic

SparseTensor Cores (FP16)
16x32* 32x8 effective matrix multiplication -1 cycle
2x the work with the same instruction throughput



DepGraph: Towards Any Structural Pruning

“Structural pruning enables model acceleration by removing structurally-grouped parameters from neural networks. However, the parameter-grouping patterns vary widely across different models, making architecture -specific pruners, which rely on manually-designed grouping schemes, non-generalizable to new architectures.

In this work, we study a highly-challenging yet barely-explored task, any structural pruning, to tackle general structural pruning of arbitrary architecture like CNNs, RNNs, GNNs and Transformers.”

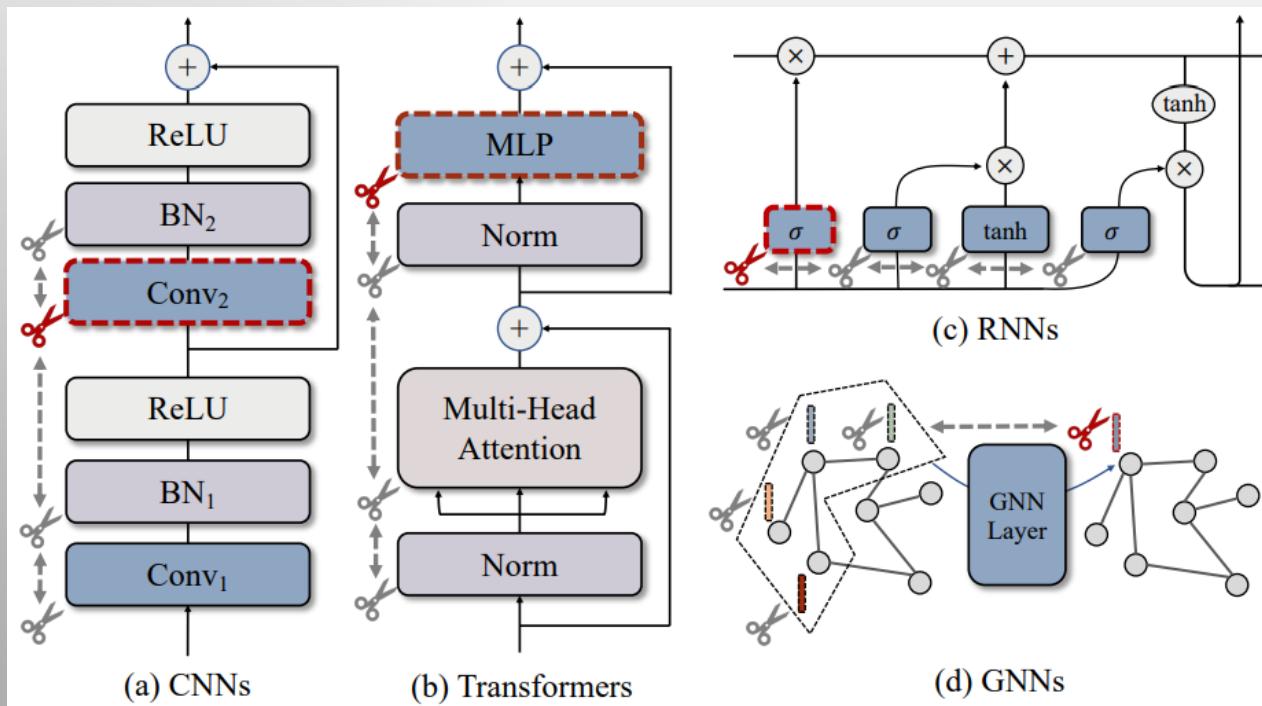
ViT-B/16	ViT-B/16	81.07	-	-	17.6
	CP-ViT [52]	77.91	77.36	-0.55	11.7
	Ours+EMA	81.07	79.58	-1.39	10.4
	Ours	81.07	79.17	-1.90	10.4

Table 3. Pruning results on ImageNet.

DepGraph: Towards Any Structural Pruning Torch-Pruning

General-purpose Pruning Toolkit: TP enables structural pruning for a wide range of deep neural networks, including Large Language Models (LLMs), Diffusion Models, Yolov7, yolov8, Vision Transformers, Swin Transformers, BERT, FasterRCNN, SSD, ResNe(X)t, ConvNext, DenseNet, ConvNext, RegNet, DeepLab, etc.

Different from `torch.nn.utils.prune` that zeroizes parameters through masking, Torch-Pruning deploys an algorithm called DepGraph to remove parameters physically.

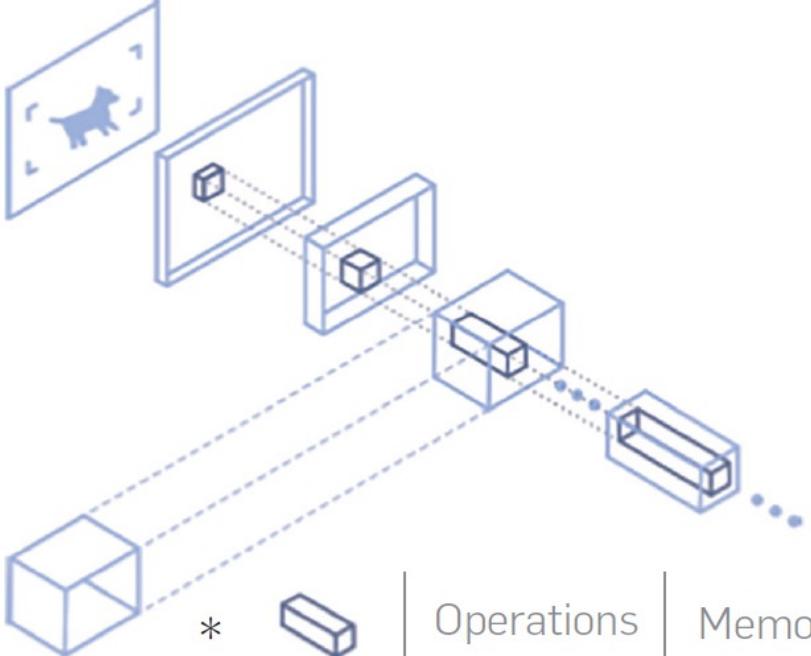


<https://github.com/VainF/Torch-Pruning>

Binary Networks

- The major computational bottleneck is in the convolutional operations
- GPUs can parallelize these huge amount of floating point operations. But GPUs are expensive and consume extensive power to run.
- With binary networks, it is possible to reduce the precision of the parameters and the activation values for the neurons from 32 bits all the way down to a single bit.
- By reducing the precision there are savings in both memory and computation.
- Single bit precision enables using logical operations instead of floating point operations.

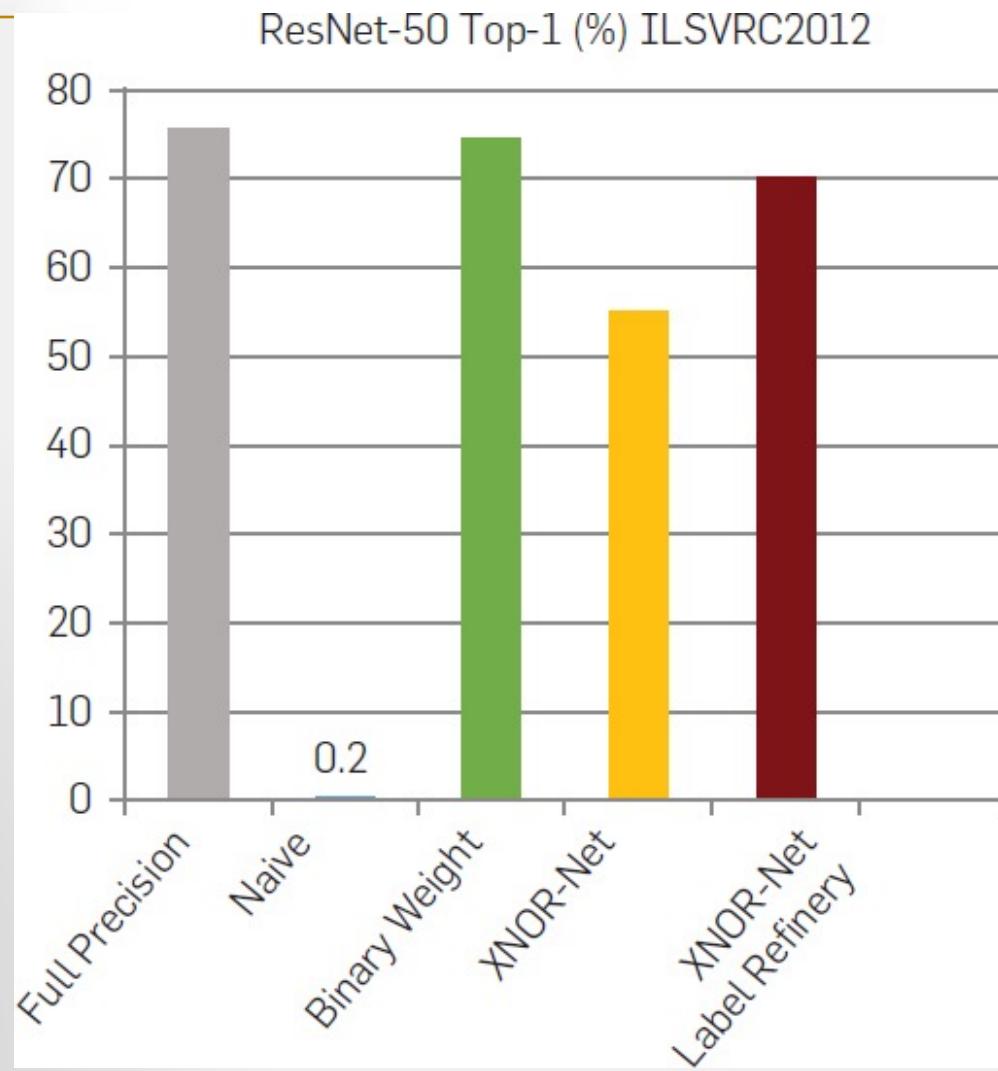
Binary Networks



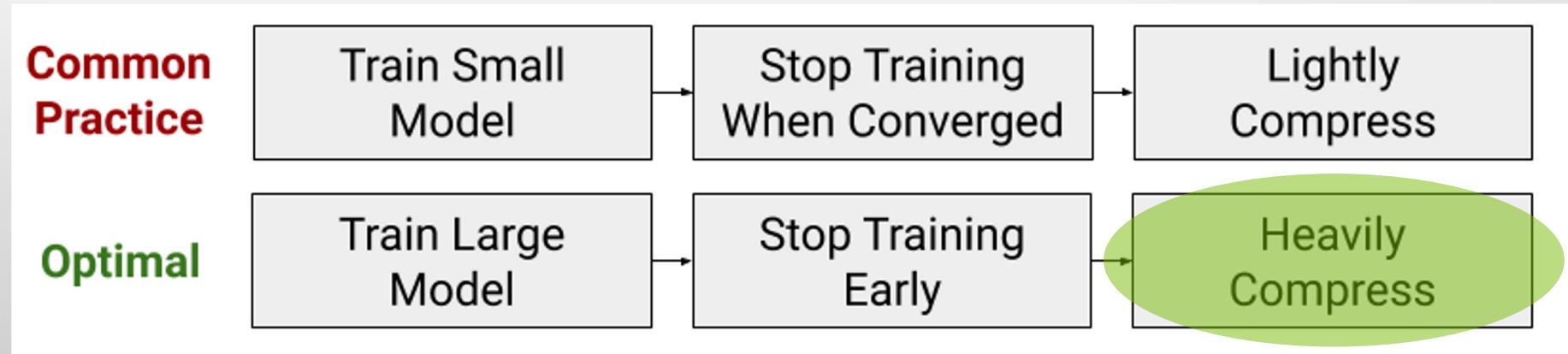
$\{-1,+1\}$	$\{0,1\}$
MUL	XNOR
ADD, SUB	Bit-Count (popcount)

	$*$	Operations	Memory	Computation	Accuracy Res-Net-50 (top-1)	
Full precision	R	$* R$	+ - x	1x	1x	75.7%
Binary weight	R	$* B$	+ -	$\sim 32x$	$\sim 2x$	75.1%
XNOR-Networks	B	$* B$	XNOR Bit-count	$\sim 32x$	$\sim 58x$	70.3%

Binary Networks



Model Compression



Knowledge distillation

- Train a small model “student” to mimic the results of a larger model “teacher”.
 - Fast to train student network if teacher is pre-trained.
 - Teacher and student can be completely different architectures.
 - E.g. DistillBERT, reduces size of BERT by 40%, and increases inference speed by 60%, while retaining 97% language understanding.
- If teacher is not pre-trained, may require larger dataset and training time, because you first have to train the more complex, teacher model, then train the student model.