Efficient models: Quantization

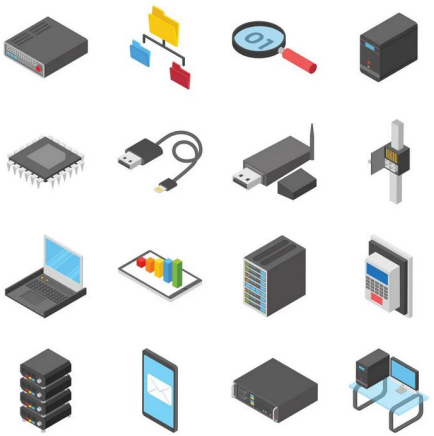
Egor Shvetsov

tg: @dalime e-mail: e.shvetsov@skoltech.ru

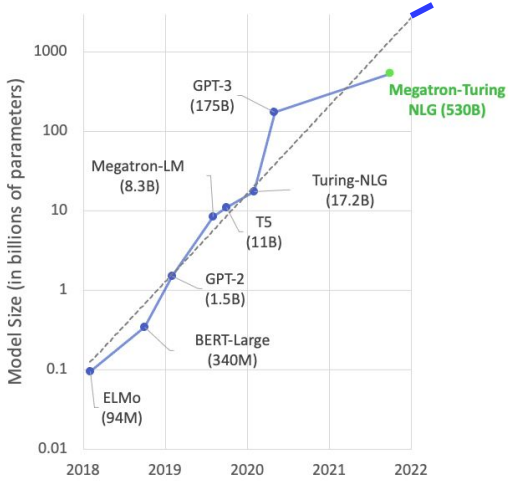
Today's plan:

1. Recap and motivation
2. Data types
3. Linear quantization
4. Quantization granularity
5. Mixed precision quantization
6. Variants of STE
7. Vector quantization - Quantization for compression
8. Binarization (TODO)

Edge devices
Autonomous devices
Robotics



Modern LLMs



- Memory
- FLOPs
- Latency
- Energy

FLOPs - number of floating point operations?

How much is it for a matrix multiplication?

$$F[(n \times p) \times (p \times m)] = ?$$

- Memory
- FLOPs
- Latency
- Energy

FLOPs - number of floating point operations?

How much is it for a matrix multiplication?

$$F[(n \times p) \times (p \times m)] = 2nmp$$

- Memory
- FLOPs
- Latency
- Energy

Input Type	Accumulation Type	Relative math throughput	Bandwidth
FP16	FP16	8x	2x
INT8	INT32	16x	4x
INT4	INT32	32x	8x
INT1	INT32	128x	32x

*Relative to FP32

model	gpu	task	energy	throughput	response_length	latency	arc	hellaswag	truthfulqa	parameters
MetaAI/Llama-7B	A100	chat	335.26	27.47	60.65	1.99	51.11	77.74	34.08	7
MetaAI/Llama-13B	A100	chat	631.57	23.2	76.47	2.97	56.31	80.86	39.9	13
tatsu-lab/alpaca-7B	A100	chat	684.17	28	132.85	4.63	52.65	76.91	39.55	7
RWKV/rwkv-raven-7b	A100	chat	735.77	61.52	214.78	3.08	39.42	66.45	38.54	7
Neutralzz/BiLLa-7B-SFT	A100	chat	881.44	33.57	161.81	4.79	27.73	26.04	49.05	7
H2OAI/H2OGPT-oasst1-7B	A100	chat	951.08	33.2	212.46	6.39	36.86	61.55	37.94	7
FreedomIntelligence/phoenix-inst-chat-7b	A100	chat	1030.89	55.84	240.34	4.12	44.97	63.22	47.08	7
StabilityAI/stablelm-tuned-alpha-7b	A100	chat	1124.4	45.42	243.88	5.21	31.91	53.59	40.22	7
databricks/dolly-v2-12B	A100	chat	1242.92	22.11	153.76	7.94	42.15	71.83	33.37	12
Salesforce/xgen-7b-8k-inst	A100	chat	1246.09	49.54	275.27	5.6	46.67	74.85	41.89	7
BAIR/koala-7b	A100	chat	1345.55	26.56	261.07	9.62	47.1	73.7	46	7
togethercomputer/RedPajama-INCITE-7B-Chat	A100	chat	1457.86	27.49	274.39	9.54	42.15	70.84	36.1	7
LMSys/vicuna-7B	A100	chat	1531.7	27.26	284.92	10.3	53.5	77.53	49	7
project-baize/baize-v2-7B	A100	chat	1588.27	31.59	326.3	10.17	48.46	75	41.66	7
LMSys/fastchat-t5-3b-v1.0	A100	chat	1802.98	19.29	314.3	20.68	35.92	46.36	48.79	3
nomic-ai/gpt4all-13b-snoozy	A100	chat	2004.73	26.57	247.8	9.28	56.06	78.69	48.36	13
OpenAssistant/oasst-sft-1-pythia-12b	A100	chat	2060.03	25.4	259.59	9.93	45.56	69.93	39.19	12
BAIR/koala-13b	A100	chat	2144.83	21.21	265.57	12.14	52.9	77.54	50.09	13
openaccess-ai-collective/manticore-13b-ch	A100	chat	2189.25	26.28	288.82	10.95	58.7	81.96	48.86	13
Camel-AI/CAMEL-13B-Combined-Data	A100	chat	2526.46	24.5	291.92	13.17	55.55	79.3	47.33	13
LMSys/vicuna-13B	A100	chat	2600.84	21.61	280.74	12.74	52.9	80.12	51.82	13

How much energy consumes LLama-7B in terms of 60W a light bulb:

- 3 seconds of work
- 5 seconds of work
- 60 seconds on work

Brain consumes 20 joules energy per second.

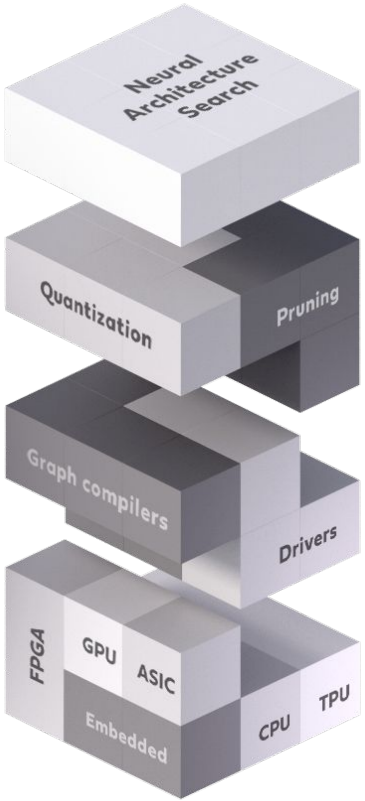
source: ml.energy

- Memory
- FLOPs
- Latency
- Energy

Arithmetic operation	ADD	MUL
8-bit Integer	0.03 pJ	0.2 pJ
16-bit Floating point	0.4 pJ	1.1 pJ
32-bit Integer	0.1 pJ	3.1 pJ
32-bit Floating point	0.9 pJ	3.7 pJ

Memory size	64-bit Memory access
8KB	10 pJ
32KB	20 pJ
1 MB	100 pJ
DRAM	1.3-2.6 nJ

paper: Energy-Efficient Model Compression and Splitting for Collaborative Inference Over Time-Varying Channels

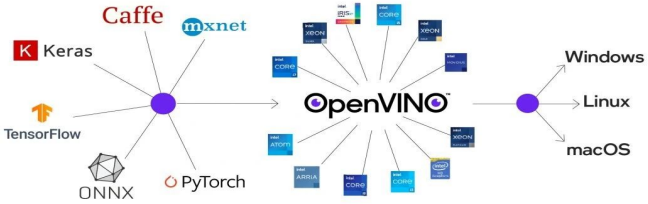


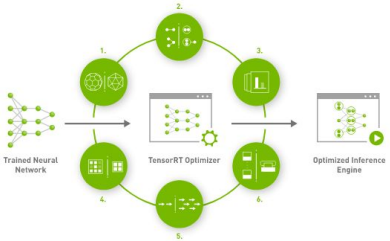
HARDWARE	SOFTWARE	ALGORITHMIC
• GPU • TPU • CPU • FPGA • Ascend • etc.	• CUDA • Graph Compilers • MKL - DNN • etc.	• PRUNING • QUANTIZATION • NAS • etc.

A majority of machine learning architectures lack explicit consideration for software or hardware properties during development, resulting in computational inefficiency.

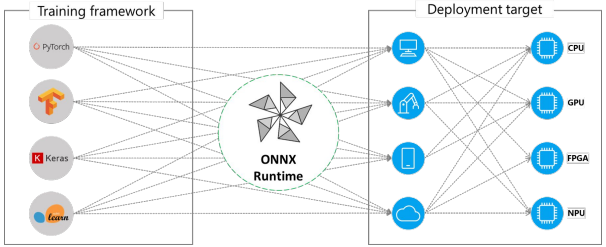
Furthermore, many models concentrate exclusively on achieving optimal results rather than accounting for practical implementation issues.

Image source: deci.ai





1. **Weight & Activation Precision Calibration**
   Maximizes throughput by quantizing models to INT8 while preserving accuracy
2. **Layer & Tensor Fusion**
   Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel
3. **Kernel Auto-Tuning**
   Selects best data layers and algorithms based on target GPU platform
4. **Dynamic Tensor Memory**
   Minimizes memory footprint and re-uses memory for tensors efficiently
5. **Multi-Stream Execution**
   Scalable design to process multiple input streams in parallel
6. **Time Fusion**
   Optimizes recurrent neural networks over time steps with dynamically generated kernels



ONNX: This tool provides a standardized format for deep learning models, enabling them to be easily shared across different frameworks and platforms. In our project, ONNX used to ensure that our ResNet-50 model can be seamlessly transitioned between various environments.

ONNX Runtime	Open source originally created by Microsoft and Facebook	Can be used as high level interface for TensorRT and OpenVino
TensorRT	Nvidia	This tool is specifically designed for NVIDIA GPUs, focusing on maximizing throughput and efficiency. It optimizes neural network models by fusing layers, selecting the most efficient data formats, and leveraging reduced precision (FP16) arithmetic where possible.
OpenVino	Intel	Developed by Intel, OpenVINO specializes in optimizing deep learning models for Intel hardware, particularly CPUs. It is a crucial tool for enhancing the performance of models where GPU resources are not available or limited.

Support of low precision computations in GPU

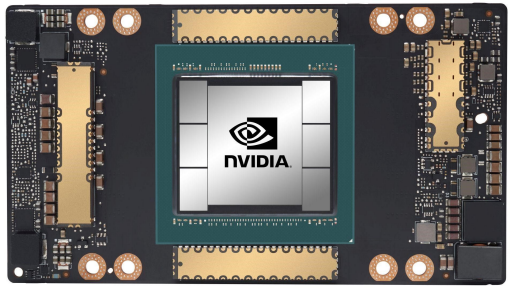
- Float16
- Float32
- Bfloat16

Will be discussed in future series …

Supported by hardware and software for most modern GPU

Not supported by hardware or software

FP 32 | FP 16 | 8 bit | 4 bit | 1 bit



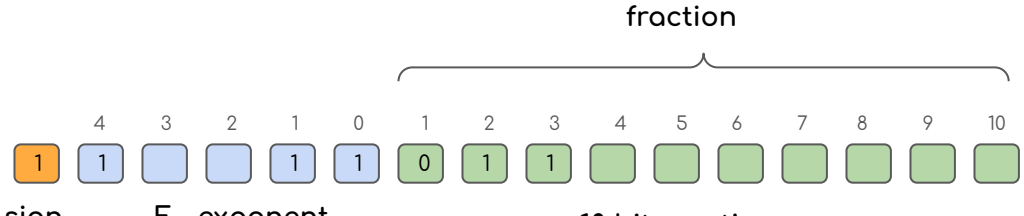
Supported by some hardware only

>>> Data formats

Int16/Int8/Int4

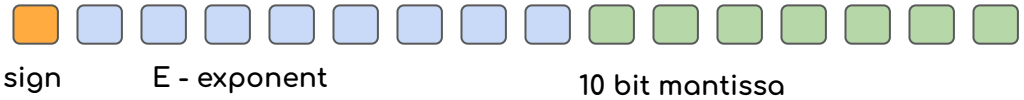
Range of values: $[-2^{n-1}, 2^{n-1} - 1]$

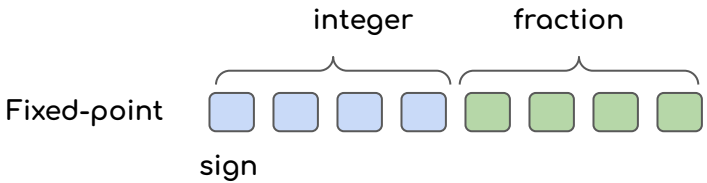
Float32 and Float16
(Floating Point)



$$\mathbf{v} = (-1)^1(1+f)2^{e-bias} \quad bias = 15$$
$$f = 2^{-2} + 2^{-3} = \frac{1}{4} + \frac{1}{8} = 0.375$$
$$e = 2^0 + 2^1 + 2^4 = 19$$
$$\mathbf{v} = (-1)^1(1+0.375)2^{19-15} = 22$$

Bfloat16





MORE:

TensorFloat-32 19 bit (supported in A100) : 1 bit sign, 8 bit exponent, 10 bit —mantissa.

E4M3 (8bit) 1 bit sign, 4 bit exponent, 3 bit mantissa
E5M2 (8bit) 1 bit sign, 5 bit exponent, 2 bit mantissa
E2M1 (4bit) 1 bit sign, 2 bit exponent, 1 bit mantissa
E3M0 (4bit) 1 bit sign, 3 bit exponent

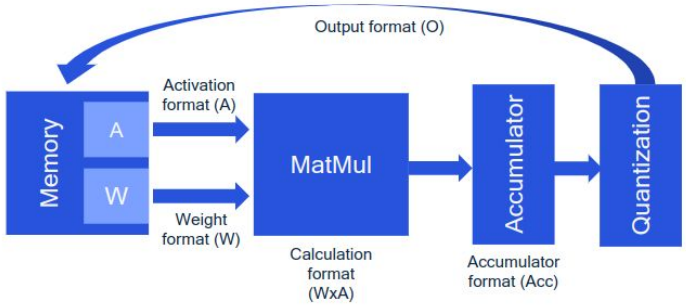
So, Float8 and int8 or Float4 and int4 use the same number of bits.
Why do we care about int4 or int8 then?

E4M3 (8bit) 1 bit sign, 4 bit exponent, 3 bit mantissa
E5M2 (8bit) 1 bit sign, 5 bit exponent, 2 bit mantissa
E2M1 (4bit) 1 bit sign, 2 bit exponent, 1 bit mantissa
E3M0 (4bit) 1 bit sign, 3 bit exponent

Int16/Int8/Int4

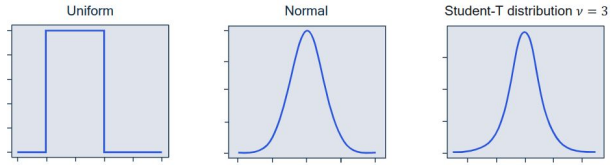
sign 3 bits

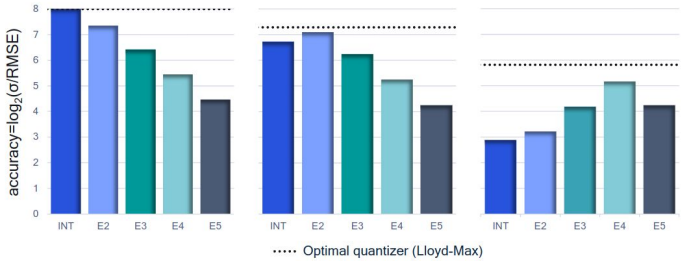
Range of values: $[-2^{n-1}, 2^{n-1} - 1]$



It is more efficient

It is not much worse





Compare int8 with FP8+E2, E3, E4 ...

It is only when outliers come into play that having more exponent bits to match the distribution makes sense!

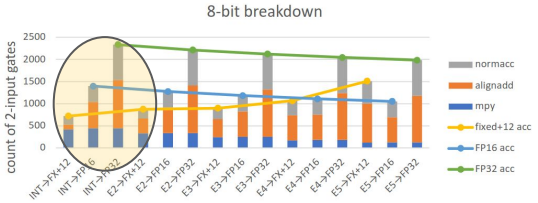
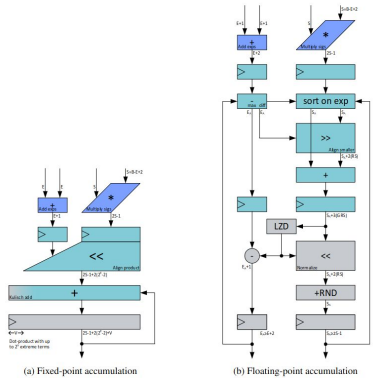


Figure 3: A count of the number of 2-input gates necessary in hardware to implement each format and accumulator combination. From left to right, INT8 with increasing exponent bits until FP8-E5. In each group of three, the first bar is for a 15+12=27-bit fixed-point accumulator. The second bar indicates the numbers for FP16 accumulation, and the third bar is the result of an FP32 accumulator. We can see that FP8-E4->16-bit requires 53% more gates, and FP8-E4->32-bit requires 183% more gates for an implementation in hardware.



(a) Fixed-point accumulation (b) Floating-point accumulation

PTQ - Post Training Quantization

Task	Model	FP32 ↑	INT8	FP8-E2	FP8-E3	FP8-E4	FP8-E5
Classification	ResNet18	69.72%	-0.08%	**-0.06%**	-0.27%	-1.15%	-4.8%
	ResNet50	76.06%	-0.07%	**-0.05%**	-0.8%	-0.99%	-3.82%
	EfficientNet B0	77.35%	-14.65%	-4.78%	**-2.8%**	-3.7%	-9.18%
	EfficientFormer	80.21%	-50.35%	-4.95%	**-0.41%**	-1.43%	-7.41 %
	DLA102	77.94%	**-0.19%**	-0.31%	-0.54%	-2.13%	-7.95%
	MobileNetV2	71.70%	-0.76%	**-0.64%**	-1.08%	-5.65%	-22.19%
	MobileNetV3	73.84%	-3.71%	**-1.48%**	-1.62%	-3.33%	-12.06%
	ViT	77.75%	-1.33%	-0.45%	**-0.04%**	-0.19%	-76.69%
Object Detection	YoloV5	56.3	-1.8	-0.5	**-0.3**	-2	-9.9
Segmentation	HRNet	81.05	-0.12	-0.02	**-0.01**	-0.28	-1.06
	CP-Pointpillar	40.94	-21.41	-14.81	**-2.86**	-2.93	-7.06
	RangeNet++	0.305	-1.67	-0.9	**-0.4**	-1.3	-3.8
	FFNet	79.16	-1.15	**-0.26**	-0.31	-0.41	-1.07
	DeeplabV3	72.91	-1.67	**-0.33**	-1.63	-34.98	-66.38
	SalsaNext	55.80	-1.58	-0.28	**-0.13**	-0.68	-2.72
NLP	BERT (GLUE)	83.06	-12.03	-2.75	-0.45	-0.26	**-0.25**
SuperResolution	QuickSRNet	32.79	-0.8	**-0.44**	-1.78	-3.24	-6.8

So, INT8 and FP8-E2 is the way to go!

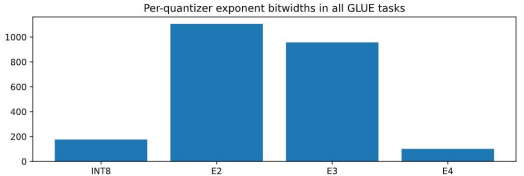


Figure 8: Optimal PTQ bit-widths for layers in the BERT model for each of the 9 different GLUE tasks. Results are generated by picking the format with the lowest MSE for each layer for that task. Only a few layers are best represented in the FP8-E4 format; similar to the results from our PTQ analysis, most layers perform better in FP8-E2 or FP8-E3.

QAT - Quantization Aware Training

Model	FP32	INT8	FP8-E2	FP8-E3	FP8-E4	W4A8
ResNet18	69.72	**70.43**	70.25	70.20	69.35	70.01
MobileNetV2	71.70	**71.82**	71.76	71.56	70.89	71.17
HRNet	81.05	**81.27**	81.20	81.14	81.06	-
DeeplabV3	72.91	**73.99**	73.67	73.74	73.22	73.01
SalsaNext (SemanticKITTI)	55.80	55.0	55.3	**55.7**	55.2	-
BERT (GLUE avg)	83.06	83.26	81.20	83.74	**83.91**	82.64

Outliers are clipped during quantization - works as regularization

Actual gain differs from theoretical and depends on algorithm and backend implementation

Latency with different batch sizes

Image/s	Batch size 1 x2			Batch size 8			Batch size 128 x8		
	FP32	FP16	Int8	FP32	FP16	Int8	FP32	FP16	Int8
MobileNet v1	1509	2889	3762	2455	7430	13493	2718	8247	16885
MobileNet v2	1082	1618	2060	2267	5307	9016	2761	6431	12652
ResNet50 (v1.5)	298	617	1051	500	2045	3625	580	2475	4609
VGG-16	153	403	415	197	816	1269	236	915	1889
VGG-19	124	358	384	158	673	1101	187	749	1552
Inception v3	156	371	616	350	1318	2228	385	1507	2560
Inception v4	76	226	335	173	768	1219	186	853	1339
ResNext101	84	208	297	200	716	1253	233	899	1724

https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9659-inference-at-reduced-precision-on-gpus.pdf

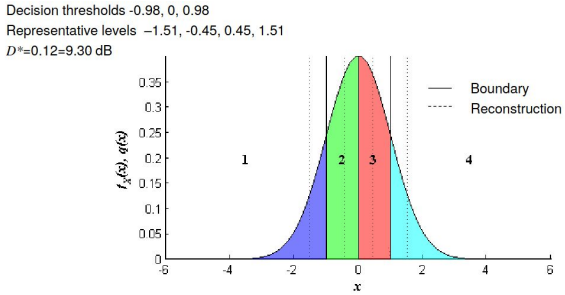
Other formats

NormalFloat4

- Normalize all weights;
- Use a quantile quantizer

Basically a dictionary with
16 values: -1,1 and 14 bins.

Lloyd-Max Quantizer



ADAPTIVE DATA TYPES

Algorithm 2: `ANT` data type selection algorithm.

Input: Tensor, T; Candidate list of numeric types, L.
Output: Quantization function, F_Q.

```
def ANT(T, L):
    minMSE = 10^9;
    foreach l ∈ L do
        F = GetQuantFunc(l); // Get the quantization method of l.
        m = ArgminMSE(T, F); // Search the minimum MSE with range clipping.
        if m < minMSE then
            F_Q = F;
    return F_Q
```

Sources:

[1] 8-BIT APPROXIMATIONS FOR PARALLELISM IN DEEP LEARNING
[2] ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network
[3] QuantizationA White Paper on Neural Network Quantization
[4] Low-Precision Arithmetic
[5] FP8 Quantization: The Power of the Exponent
[6] NF4 Isn't Information Theoretically Optimal (and that's Good)
[7] FP8 versus INT8 for efficient deep learning inference
[8] FP8 Formats for Deep Learning
[9] TensorFloat-32 in the A100 GPU

Effects of Low-Precision Computation

Pros:

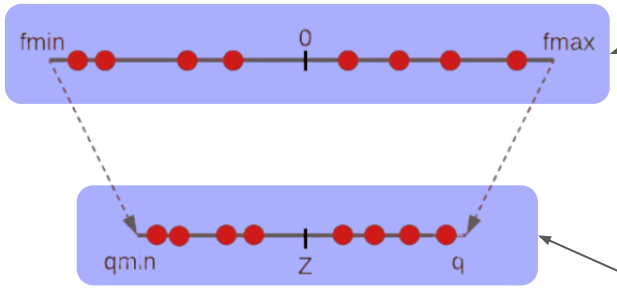
- Fewer bits allow better memory exploitation e.g.: transmit more numbers per second
- Integers have faster arithmetic
- Use less energy

Cons:

- Limits the numbers we can represent
- Introduces quantization Noise - rounding or discretisation error

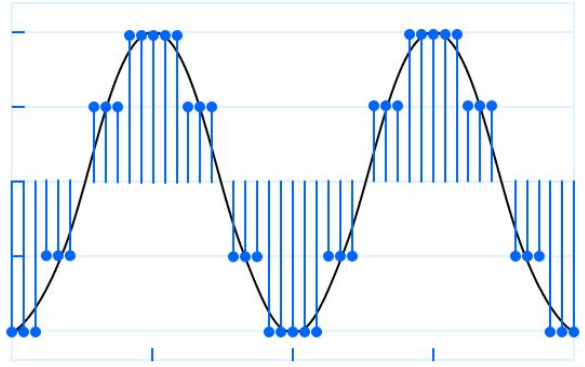
>>> Quantization

Quantization is a mapping procedure into a discrete fixed length rage.



Initial values

quantized



FP32 tensor $\boldsymbol{X} \approx s_X \boldsymbol{X}_{\text{int}} = \widehat{\boldsymbol{X}}$ scaled quantized tensor

$$\boldsymbol{W} = \begin{pmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix} \approx \frac{1}{255} \begin{pmatrix} 247 & 163 & 189 & 255 \\ 148 & 214 & 214 & 207 \\ 0 & 46 & 229 & 71 \\ 145 & 245 & 204 & 207 \end{pmatrix} = s_W \boldsymbol{W}_{\text{uint8}}$$

Scale: $s = \frac{f_{max} - f_{min}}{fq_{max} - fq_{min}}$

Quantize: $Q(x) = clip(round(\frac{1}{s}x), fq_{min}, fq_{max})$

De-Quantize: $x = sQ(x)$

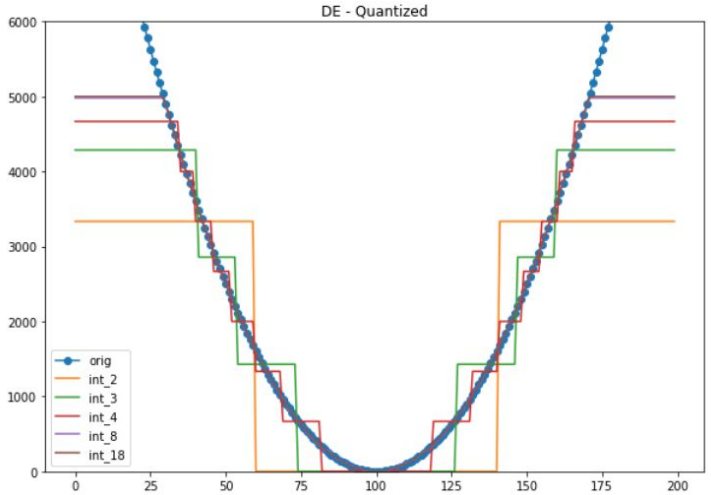
$fq_{min} = -2^{(bit-1)}$

$fq_{max} = 2^{(bit-1)} - 1$

Symmetric signed linear (uniform) quantization

Quantization is a mapping procedure into a discrete fixed length rage.

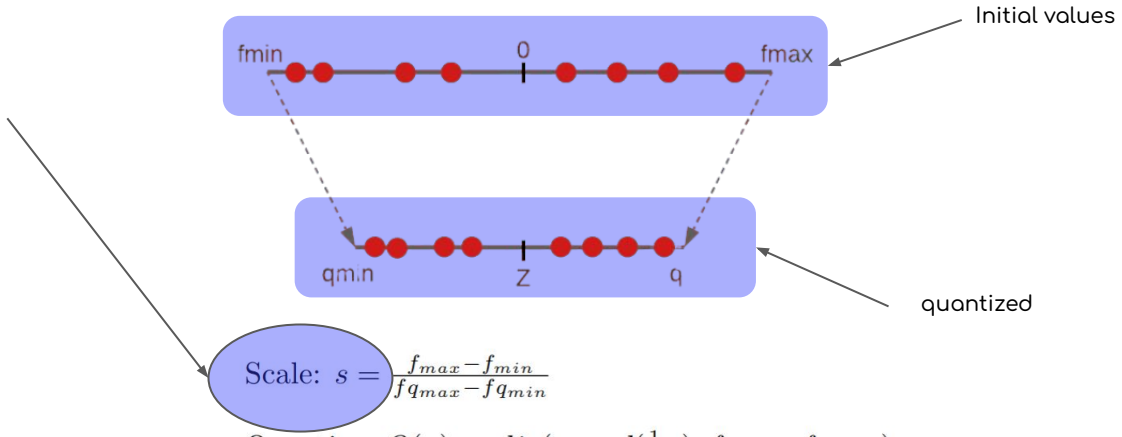
```python
def quantization(x, s, z, alpha_q, beta_q):

    x_q = np.round(1 / s * x + z, decimals=0)
    x_q = np.clip(x_q, a_min=alpha_q, a_max=beta_q)

    return x_q

def de_quantization(x_q, s, z):

    # x_q - z might go outside the quantization range.
    x_q = x_q.astype(np.int32)
    x = s * (x_q - z)
    x = x.astype(np.float32)
    return x

def generate_quantization_constants(alpha, beta, alpha_q, beta_q):

    # Affine quantization mapping
    s = (beta - alpha) / (beta_q - alpha_q)
    return s


def quantize_with_bit(x, bit):
    alpha_q = -2**(bit - 1)
    beta_q = 2**(bit - 1) - 1

    s = (x.max() - x.min()) / (beta_q - alpha_q)

    x_q = quantization(x, s, 0, alpha_q, beta_q)
    return  de_quantization(x_q, s, 0), x_q
```



Quantization is a mapping procedure into a discrete fixed length rage.

What problems may happen with choosing the scale?

Consider we need to quantize both activations and weights.



Quantize: $Q(x) = clip(round(\frac{1}{s}x), fq_{min}, fq_{max})$

De-Quantize: $x = sQ(x)$

$fq_{min} = -2^{(bit-1)}$

$fq_{max} = 2^{(bit-1)} - 1$

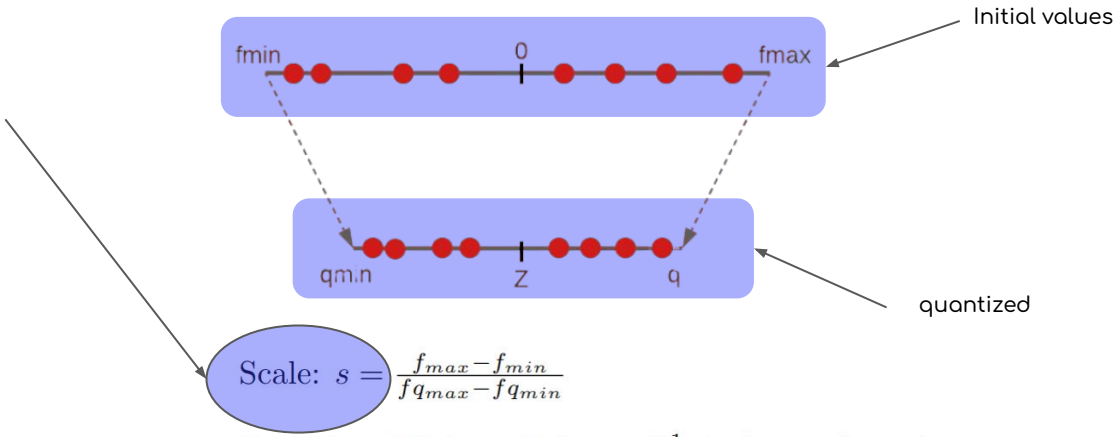
Quantization is a mapping procedure into a discrete fixed length rage.

What problems may happen with choosing the scale?

Consider we need to quantize both activations and weights.

- Collect statistics over data for activations and use some fixed value for it;
- Make scale a learnable parameter;
- Find such scale to minimize - layer-wise objective:

$$\min_{s_q, s_x} ||WX^T - Q(W, s_q)Q(X^T, s_x)||$$



Quantize: $Q(x) = clip(round(\frac{1}{s}x), fq_{min}, fq_{max})$

De-Quantize: $x = sQ(x)$

$fq_{min} = -2^{(bit-1)}$

$fq_{max} = 2^{(bit-1)} - 1$

Published as a conference paper at ICLR 2020

LEARNED STEP SIZE QUANTIZATION

Steven K. Esser *, Jeffrey L. McKinstry, Deepika Bablani,
Rathinakumar Appuswamy, Dharmendra S. Modha

IBM Research
San Jose, California, USA

ABSTRACT

Deep networks run with low precision operations at inference time offer power and space advantages over high precision alternatives, but need to overcome the challenge of maintaining high accuracy as precision decreases. Here, we present a method for training such networks, Learned Step Size Quantization, that achieves the highest accuracy to date on the ImageNet dataset when using models, from a variety of architectures, with weights and activations quantized to 2-, 3- or 4-bits of precision, and that can train 3-bit models that reach full precision baseline accuracy. Our approach builds upon existing methods for learning weights in quantized networks by improving how the quantizer itself is configured. Specifically, we introduce a novel means to estimate and scale the task loss gradient at each weight and activation layer's quantizer step size, such that it can be learned in conjunction with other network parameters. This approach works using different levels of precision as needed for a given system and requires only a simple modification of existing training code.

Network	Method	Top-1 Accuracy @ Precision 2	3	4	8	Top-5 Accuracy @ Precision 2	3	4	8
ResNet-18		*Full precision: 70.5*				*Full precision: 89.6*			
	LSQ (Ours)	**67.6**	**70.2**	**71.1**	**71.1**	**87.6**	**89.4**	**90.0**	**90.1**
	QIL	65.7	69.2	70.1					
	FAQ			69.8	70.0			89.1	89.3
	LQ-Nets	64.9	68.2	69.3		85.9	87.9	88.8	
	PACT	64.4	68.1	69.2		85.6	88.2	89.0	
	NICE		67.7	69.8			87.9	89.21	
	Regularization	61.7		67.3	68.1	84.4		87.9	88.2
ResNet-34		*Full precision: 74.1*				*Full precision: 91.8*			
	LSQ (Ours)	**71.6**	**73.4**	**74.1**	**74.1**	**90.3**	**91.4**	**91.7**	**91.8**
	QIL	70.6	73.1	73.7					
	LQ-Nets	69.8	71.9			89.1	90.2		
	NICE		71.7	73.5			90.8	91.4	
	FAQ			73.3	73.7			91.3	91.6
ResNet-50		*Full precision: 76.9*				*Full precision: 93.4*			
	LSQ (Ours)	**73.7**	**75.8**	**76.7**	**76.8**	**91.5**	**92.7**	93.2	**93.4**
	PACT	72.2	75.3	76.5		90.5	92.6	93.2	
	NICE		75.1	76.5			92.3	**93.3**	
	FAQ			76.3	76.5			92.9	93.1
	LQ-Nets	71.5	74.2	75.1		90.3	91.6	92.4	
ResNet-101		*Full precision: 78.2*				*Full precision: 94.1*			
	LSQ (Ours)	**76.1**	**77.5**	**78.3**	**78.1**	**92.8**	**93.6**	**94.0**	**94.0**
ResNet-152		*Full precision: 78.9*				*Full precision: 94.3*			
	LSQ (Ours)	**76.9**	**78.2**	**78.5**	**78.5**	**93.2**	**93.9**	**94.1**	**94.2**
	FAQ			78.4	**78.5**			**94.1**	94.1
VGG-16bn		*Full precision: 73.4*				*Full precision: 91.5*			
	LSQ (Ours)	**71.4**	**73.4**	**74.0**	73.5	**90.4**	**91.5**	**92.0**	**91.6**
	FAQ			73.9	**73.7**			91.7	**91.6**
Squeeze Next-23-2x		*Full precision: 67.3*				*Full precision: 87.8*			
	LSQ (Ours)	**53.3**	**63.7**	**67.4**	**67.0**	**77.5**	**85.4**	**87.8**	**87.7**

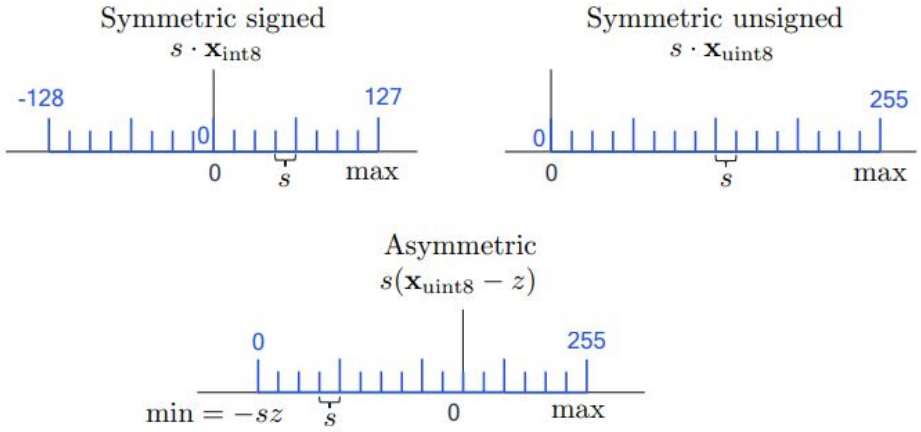
Source: Learned Step Size Quantization

Symmetric vs Asymmetric quantization

$$Quantize: Q(X) = s[clapm([\frac{1}{s}x] + z; 0, 2^b - 1]) - z]$$

Symmetric quantization: is a simplified version of the general asymmetric case. The symmetric quantizer restricts the zero-point Z to 0. This reduces the computational overhead of dealing with zero-point offset during the accumulation operation. But the lack of offset restricts the mapping between integer and floating-point domain.

$$\begin{aligned}\widehat{\mathbf{W}}\widehat{\mathbf{x}} &= s_{\mathbf{w}}(\mathbf{W}_{\text{int}} - z_{\mathbf{w}})s_{\mathbf{x}}(\mathbf{x}_{\text{int}} - z_{\mathbf{x}}) \\ &= s_{\mathbf{w}}s_{\mathbf{x}}\mathbf{W}_{\text{int}}\mathbf{x}_{\text{int}} - s_{\mathbf{w}}z_{\mathbf{w}}s_{\mathbf{x}}\mathbf{x}_{\text{int}} - s_{\mathbf{w}}s_{\mathbf{x}}z_{\mathbf{x}}\mathbf{W}_{\text{int}} + s_{\mathbf{w}}z_{\mathbf{w}}s_{\mathbf{x}}z_{\mathbf{x}}.\end{aligned}$$



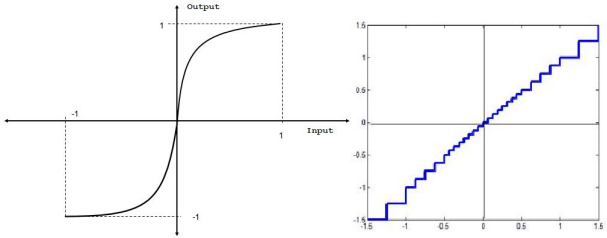
Types of quantization uniform and non-uniform

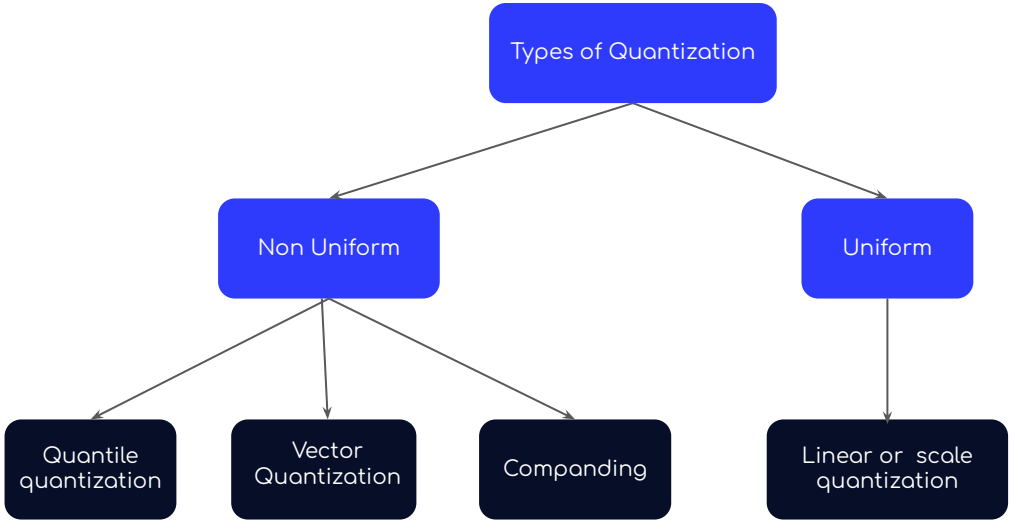
Source: https://arxiv.org/pdf/2106.08295.pdf

The goal is to perform fast computations, a uniform quantization is preferred.

Non-uniform quantization would require performing additional transformations before using the microcontroller's instructions. This overhead can be non-negligible and lead to quantization performance lower than floating-point computations. To obtain a nonconstant quantization step, a nonlinear function must be computed, either online or offline, to generate a lookup table where operands for each operation are stored. In contrast, uniform quantization is based on a constant quantization step.

Uniform affine quantization: it is the most commonly used quantization scheme because it permits efficient implementation of fixed-point arithmetic.



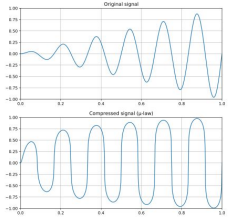


Usually requires a lookup table

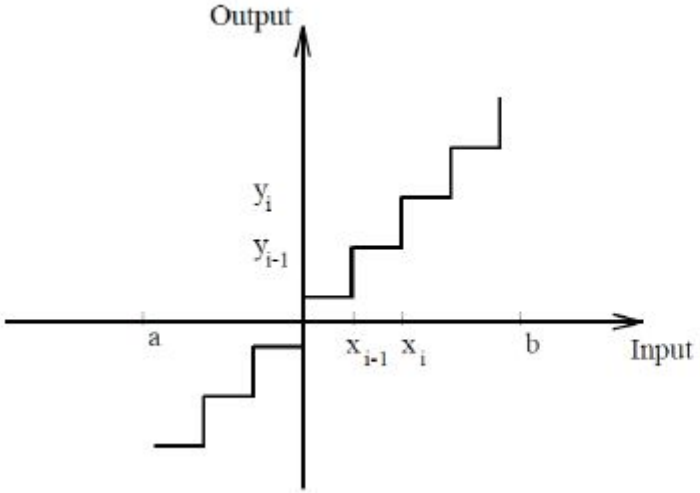
$$g(v) = (f_\Theta^{-1} \circ q_b \circ f_\Theta)(v)$$

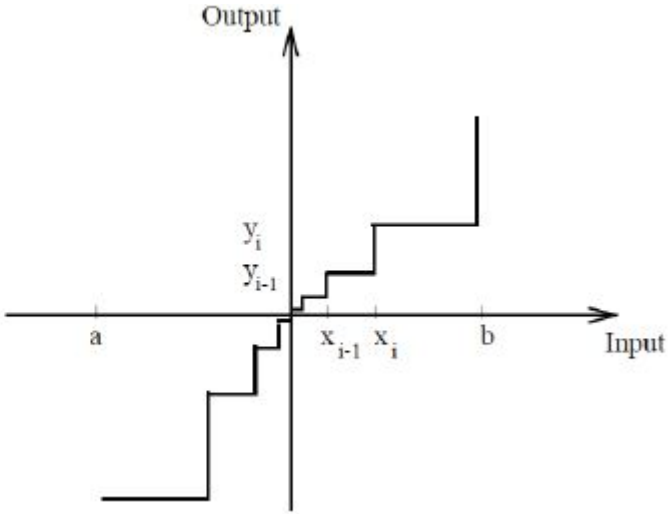
Transform data to be more uniformly distributed, use scale quantizer, apply inverse transformation.

Mu-Law is a common example from DSP.



Quantile quantization

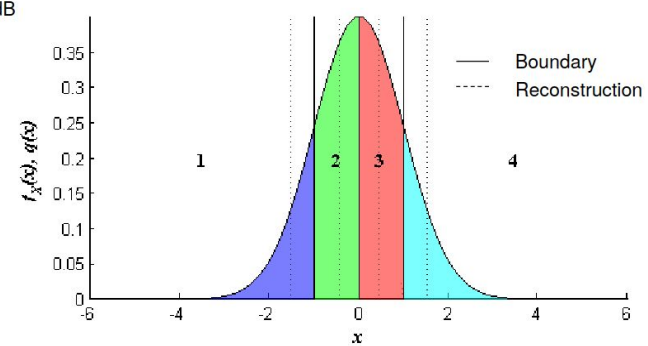




Decision thresholds -0.98, 0, 0.98

Representative levels –1.51, -0.45, 0.45, 1.51

D^*=0.12=9.30 dB

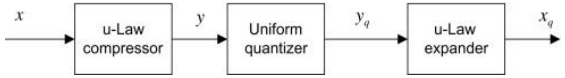


Transform data to be more uniformly distributed, use scale quantizer, apply inverse transformation.

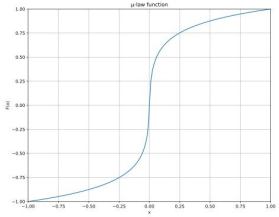
Companding

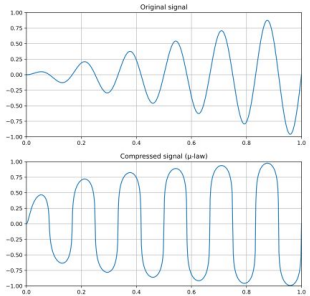
$$g(v) = (f_\Theta^{-1} \circ q_b \circ f_\Theta)(v)$$

Mu-Law is a common example from DSP.



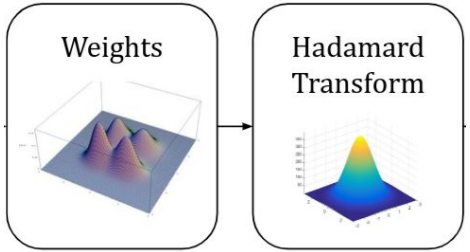
$$F(x) = \text{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)}, \quad -1 \le x \le 1,$$





More on topic : Incoherence-Optimal Matrix Completion

[QUIP #](#) - LLM Quantization



Definition 2.1 (Chee et al. (2023)). A Hessian $H \in \mathbb{R}^{n\times n}$ is μ-incoherent if its eigendecomposition $H = Q\Lambda Q^T$ has

$$\max_{i,j} |Q_{ij}| = \max_{i,j} |e_i^T Q e_j| \le \mu/\sqrt{n}.$$

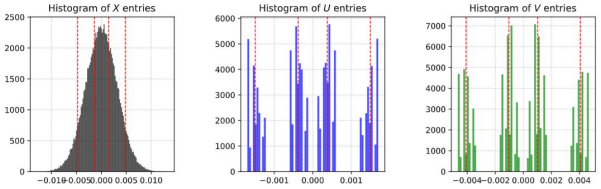
A weight matrix $W \in \mathbb{R}^{m\times n}$ is μ-incoherent if

$$\max_{i,j} |W_{ij}| = \max_{i,j} |e_i^T W e_j| \le \mu \|W\|_F/\sqrt{mn}.$$

$$U^T(\text{quantized}(\tilde{W})(Vx)) \approx U^T(\tilde{W}(Vx)) = Wx.$$

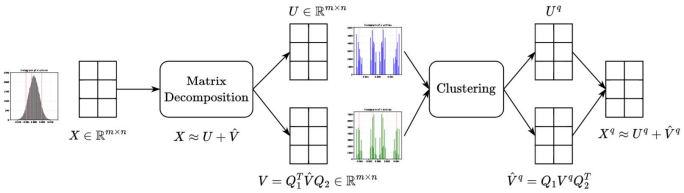
Transform weights to make W more incoherent.

[Kashin Representations](#) for LLM Quantization



$$X = U + Q_1 V Q_2^T$$

Q - to present V in some basis, so we need to Q to be easily computed and stored but it is another story.



Vector Quantization

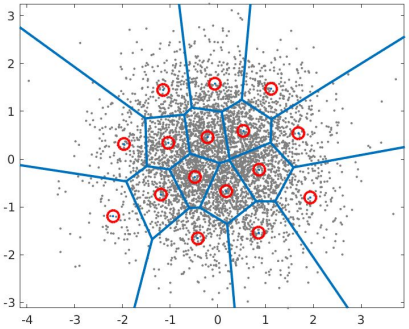
VQ developed in the early 1980s by Robert M. Gray, recommended slides by Robert M. Gray Fundamentals of Quantization

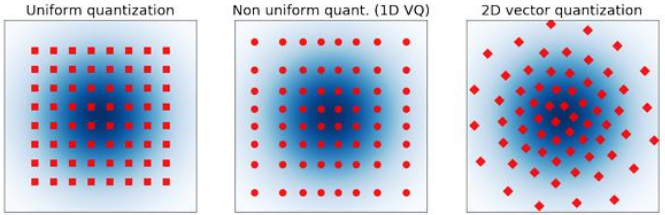
VQ is like K-means but there are some details

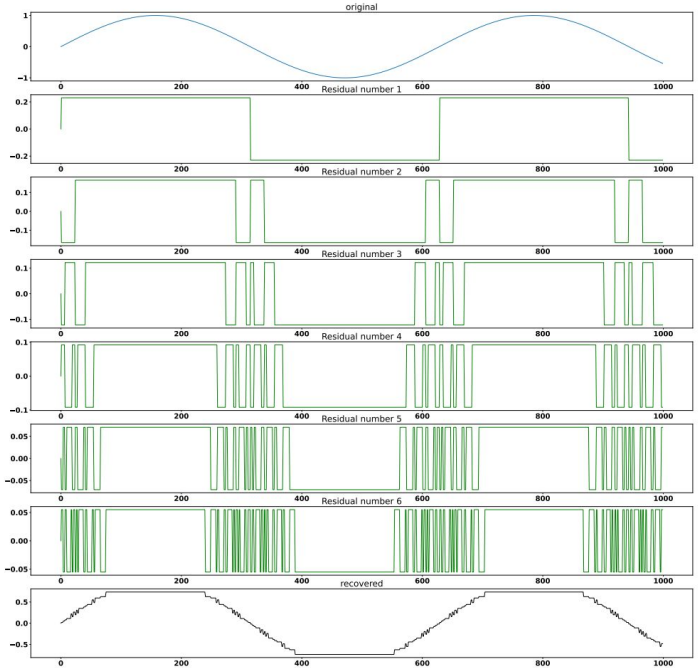
There are:

- Additive VQ
- Residual VQ
- Pyramid VQ
- Product VQ
- Tree-structured VQ,
- Lattice VQ
- Classified VQ
- Feedback VQ
- Fuzzy VQ

  etc ...







An example of residual VQ.

Why it is not optimal for this specific signal?

Algorithm 1: Residual Vector Quantization

Input: $y = \text{enc}(x)$ the output of the encoder, vector quantizers Q_i for $i = 1..N_q$
Output: the quantized $\hat{y}$
$\hat{y} \leftarrow 0.0$
residual $\leftarrow y$
for $i = 1$ *to* N_q **do**
  $\hat{y}$ += Q_i(residual)
  residual −= Q_i(residual)
return $\hat{y}$

Vector Quantization

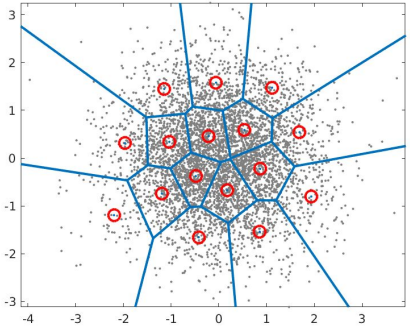
VQ developed in the early 1980s by Robert M. Gray, recommended slides by Robert M. Gray Fundamentals of Quantization

VQ is like K-means but there are some details

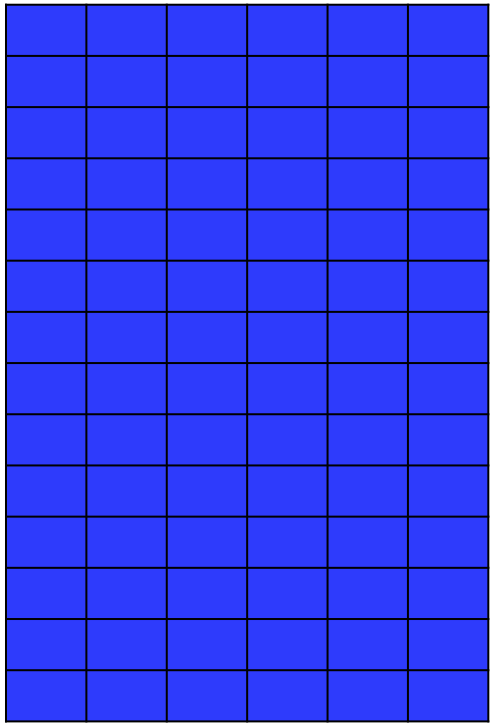
There are:

- Additive VQ
- Residual VQ
- Pyramid VQ
- Product VQ
- Tree-structured VQ,
- Lattice VQ
- Classified VQ
- Feedback VQ
- Fuzzy VQ

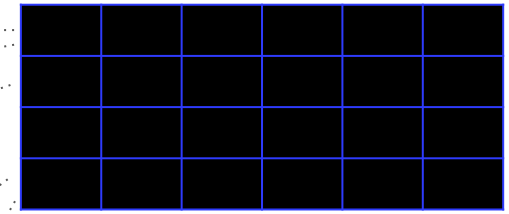
  etc ...



Matrix "W" 6 x 14



Code book "C" 6 x 4



Approximate "W" with entries from "C".

Now each row in "W" is presented as OHE index vector of 2 bit size.

Initial size: 6x14 FP
compressed size: 14 x 2 bit + 6x4 FP

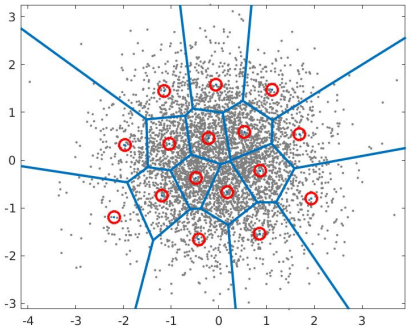
Vector Quantization

VQ developed in the early 1980s by Robert M. Gray, recommended slides by Robert M. Gray Fundamentals of Quantization

VQ is like K-means but there are some details

There are:

- Additive VQ
- Residual VQ
- Pyramid VQ
- Product VQ
- Tree search VQ etc ...



We can also combine several code books.

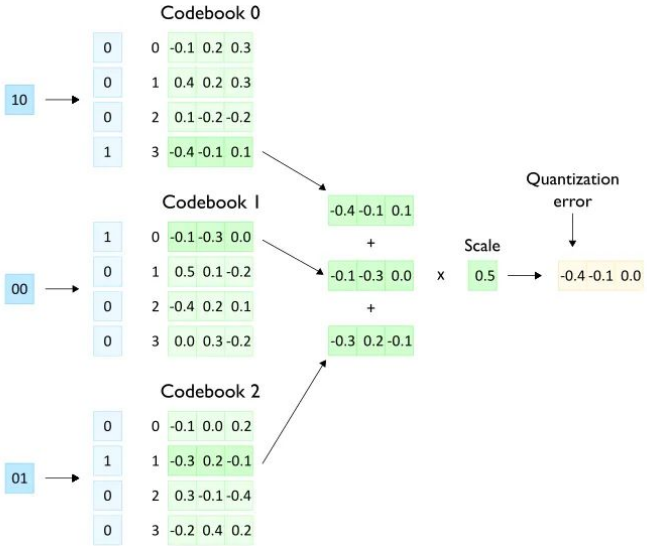


Image source: link

Vector Quantization

While weights can be indexed in a look up table for activations we will need to compute some distance metric and does not make much sense to compress activations with VQ in general, or does it?

>>> Quantization granularity

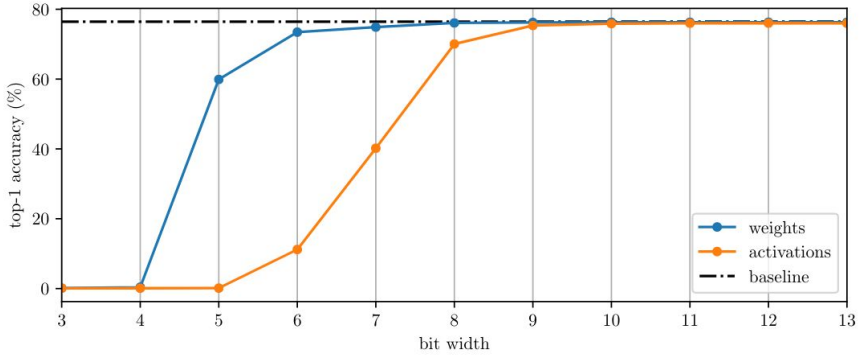
What options do we have when we quantize neural networks?

What options do we have when we quantize neural networks?

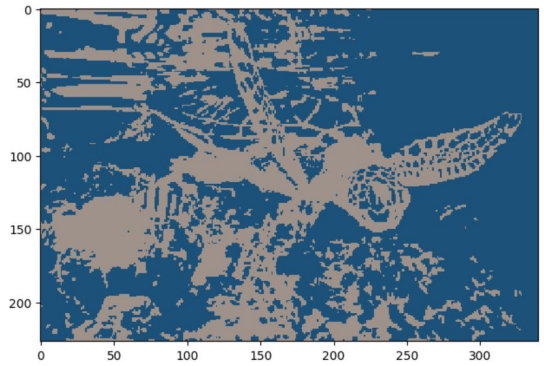
Weights and Activations, are they equally important?

both, activations and weights should have the same data type

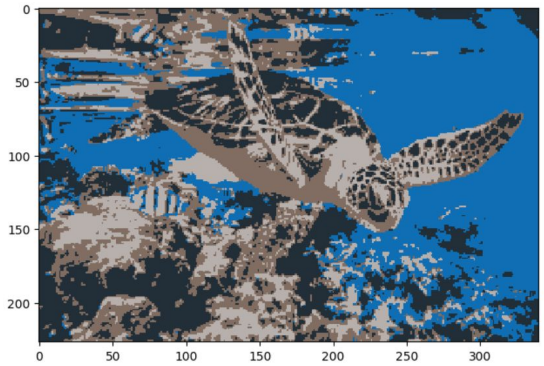
If only one type is quantized with lower precision, then we only save memory



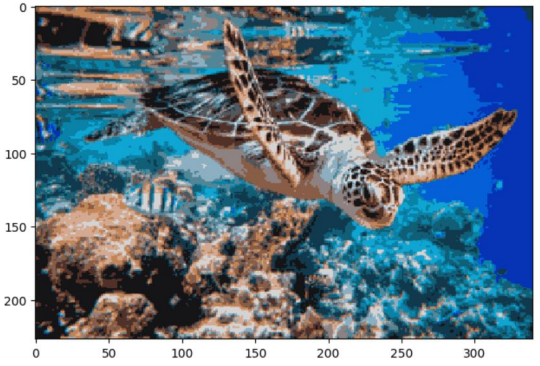
https://arxiv.org/pdf/2109.04236.pdf



1 bit - 2 possible values

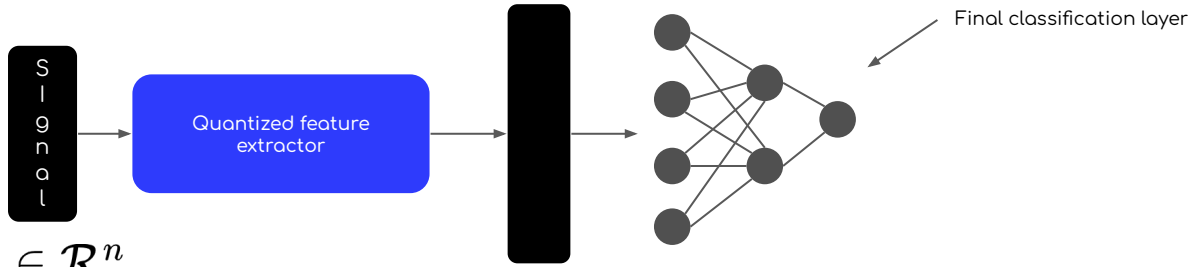


2 bits - 4 possible values



4 bits - 16 possible values

- Removing information from the signal reduces performance, especially if information is lost at the beginning of the model;
- Meanwhile, reducing weights precision does not hurt performance much;
- It is possible to map signal into a higher dimension but reduce its precision, so the overall quality is preserved **(later about it)**.



$$X_i \in \mathcal{R}^n$$

$$Q(X_{i+1}) \in \mathcal{R}^m, m >> n$$

Quantization granularity:

- Layer - wise
- Filter - wise / channel - wise
- Split a weight matrix into several blocks - what problem we may face here?

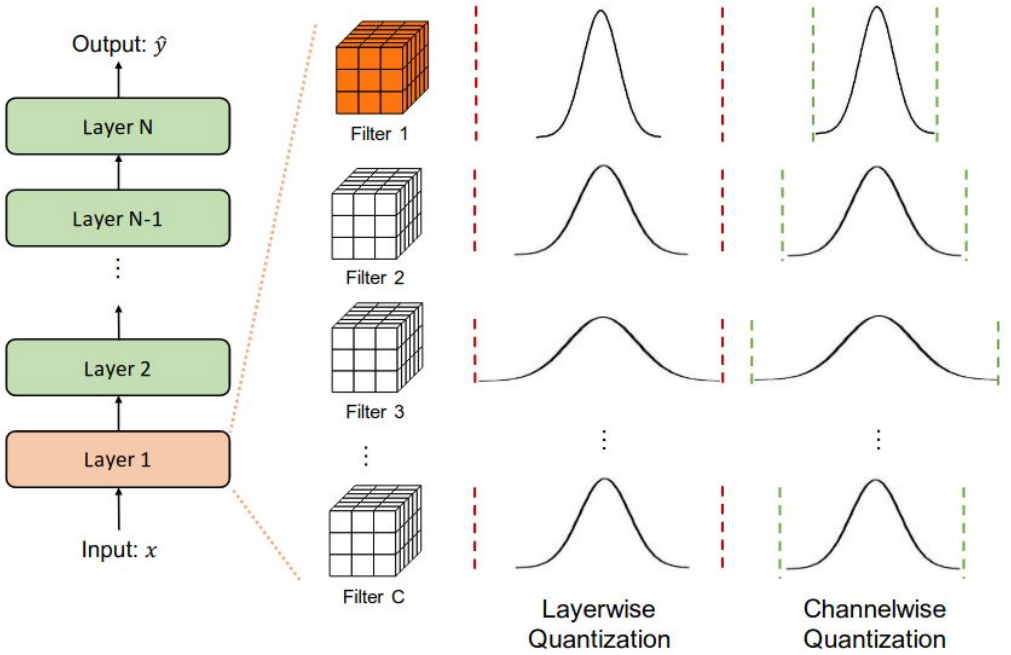


Image: A Survey of Quantization Methods for Efficient Neural Network Inference

Quantization granularity:

- Layer - wise
- Filter - wise / channel - wise
- Split a weight matrix into several blocks - what problem we may face here?

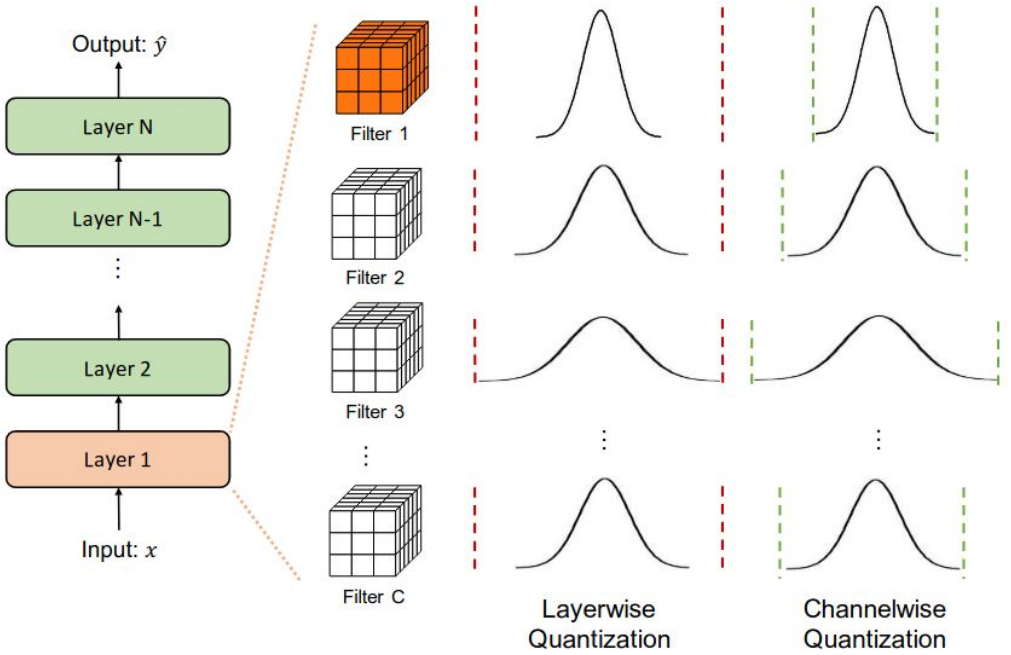
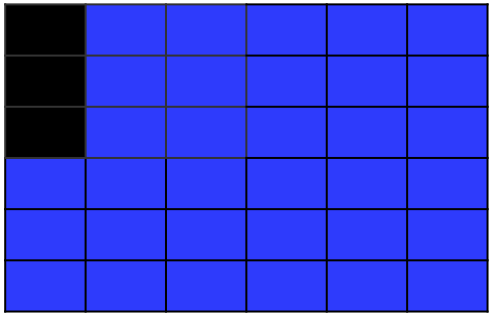


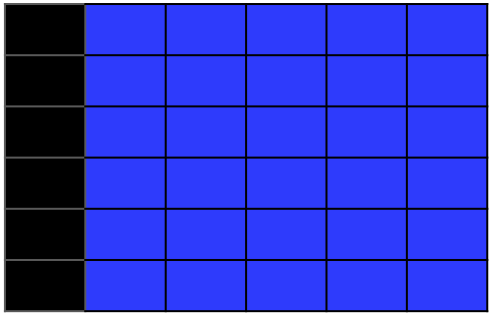
Image: A Survey of Quantization Methods for Efficient Neural Network Inference

Split a weight matrix into several blocks

Which dimension to use?

How many blocks should we have?





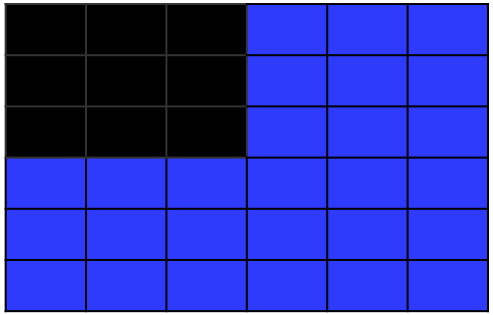


Image: A Survey of Quantization Methods for Efficient Neural Network Inference

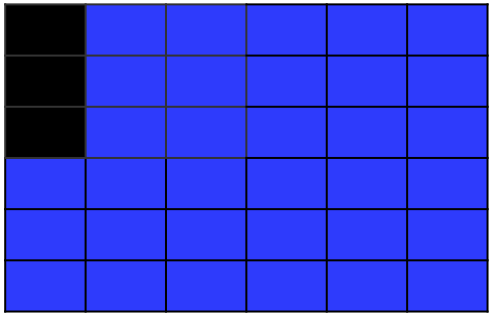
Split a weight matrix into several blocks

Which dimension to use?

- Dimension with more uniform distribution

How many blocks should we have?

- Consider that for each block you need to store quantization parameters;
- Consider L1, L2 cache size, it is usually 64.



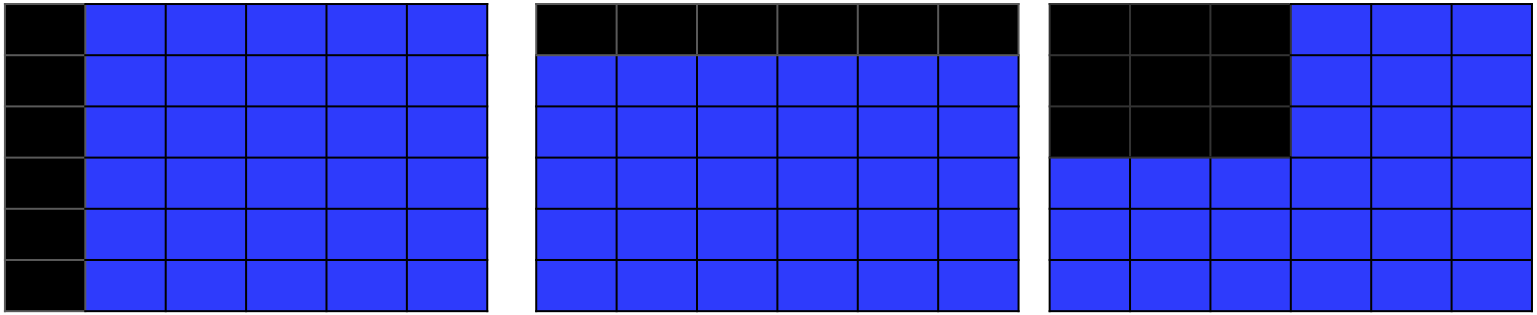
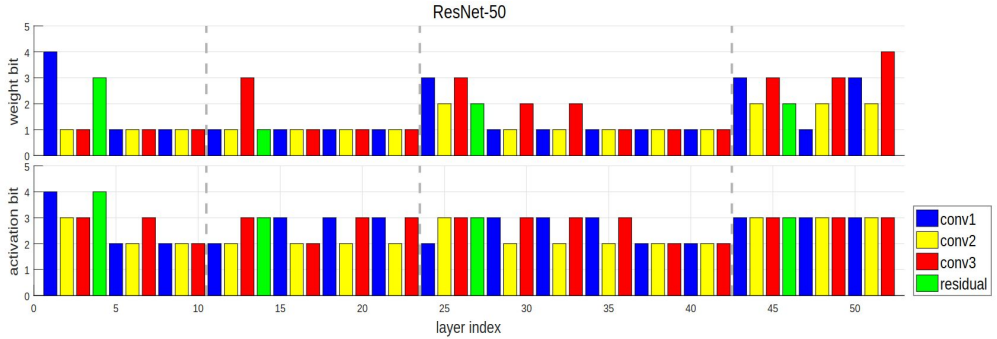
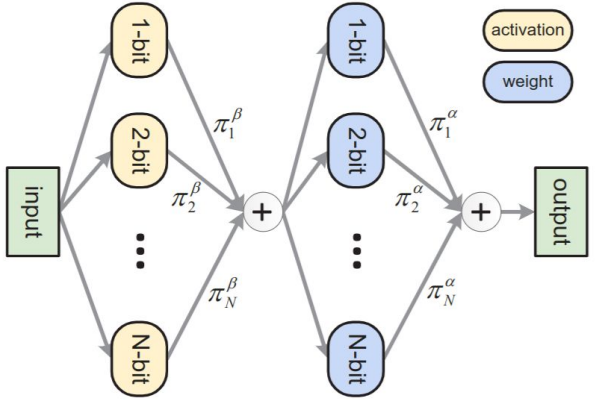
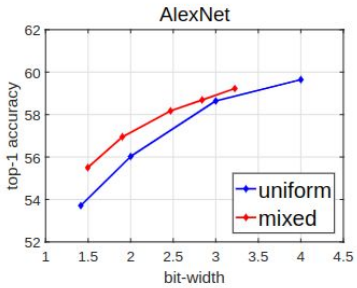


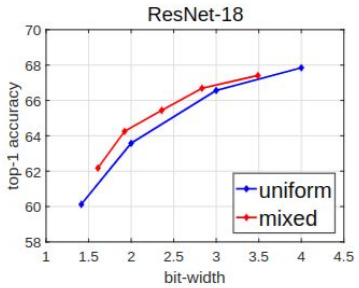
Image: A Survey of Quantization Methods for Efficient Neural Network Inference

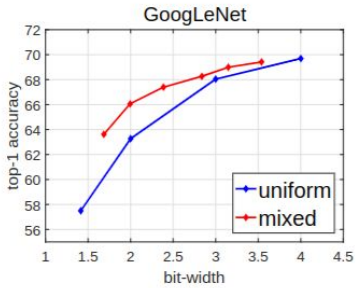
>>> Mixed precision

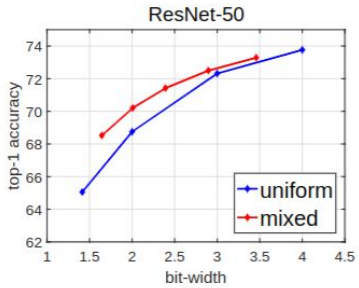


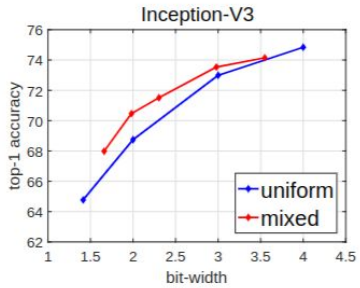






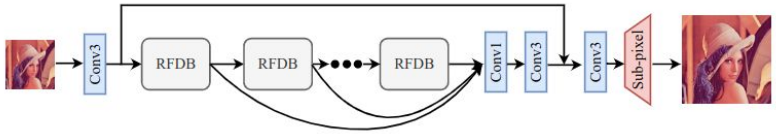




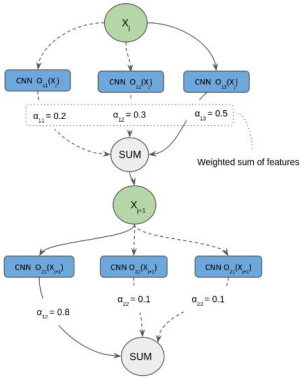


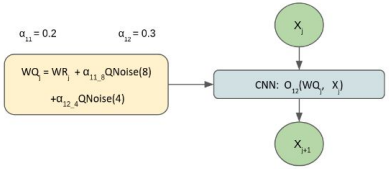
[Rethinking Differentiable Search for Mixed-Precision Neural Networks]

We take an existing Super Resolution model - RFDN and optimize it for quantization.



RFDN architecture - SOTA model in 2020 for light-weight SR.

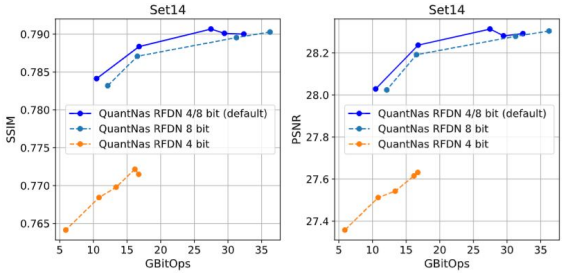




We replace each convolutional layer in the model into a searchable layer block. Options to search from are presented on the right.

Apart from that, each block has searchable quantization levels.

- **Head** 8 operations: simple 3x3, simple 5x5, growth2 5x5, growth2 3x3, simple 3x3 grouped 3, simple 5x5 grouped 3, simple 1x1 grouped 3, simple 1x1;
- **Body** 7 operations: simple 3x3, simple 5x5, simple 3x3 grouped 3, simple 5x5 grouped 3, decenc 3x3 2, decenc 5x5 2, simple 1x1 grouped 3;
- **Skip** 4 operations: decenc 3x3 2, decenc 5x5 2, simple 3x3, simple 5x5;
- **Upsample** 12 operations: conv 5x1 1x5, conv 3x1 1x3, simple 3x3, simple 5x5, growth2 5x5, growth2 3x3, decenc 3x3 2, decenc 5x5 2, simple 3x3 grouped 3, simple 5x5 grouped 3, simple 1x1 grouped 3, simple 1x1;
- **Tail** 8 operations: simple 3x3, simple 5x5, growth2 5x5, growth2 3x3, simple 3x3 grouped 3, simple 5x5 grouped 3, simple 1x1 grouped 3, simple 1x1;



Method	Model	GBitOPs	PSNR
LSQ (8-U)	ESPCN	2.3	27.26
HWGQ (8-U)	ESPCN	2.3	27.00
LSQ (4-U)	SRResNet	23.3	27.88
HWGQ (4-U)	SRResNet	23.3	27.42
EdMIPS (4,8)	SRResNet	19.4	27.92
EdMIPS (4,8)	RFDN	20.7	28.13
QuantNAS (4,8)	**RFDN**	**19.3**	**28.24**
QuantNAS (4,8)	RFDN w/o SAN	16.8	28.23
QuantNAS (4,8)	RFDN w/o ADQ	17.2	28.16
QuantNAS (4,8)	**Basic**	**2.6**	**27.81**
QuantNAS (4,8)	Basic w/o SAN	4.1	27.65
QuantNAS (4,8)	Basic w/o ADQ	4.4	27.65

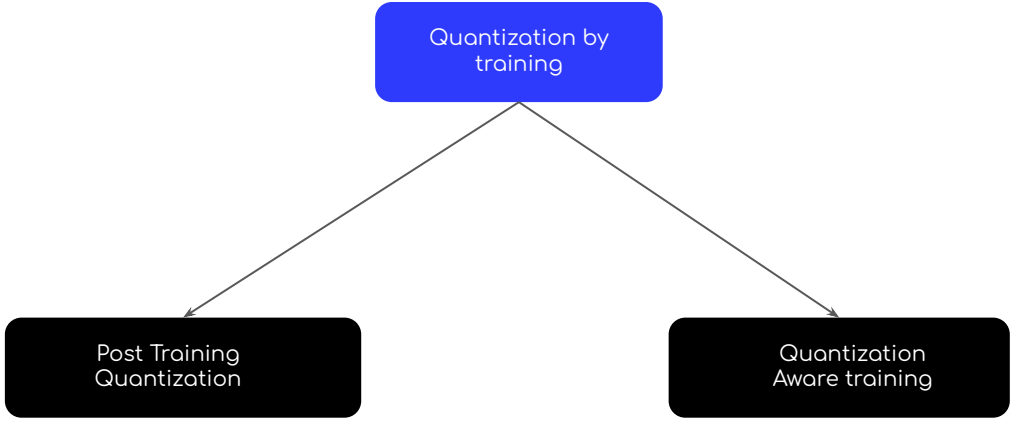
Method	GFLOPs	PSNR	Search cost
SRResNet	166.0	28.49	Manual
RFDN	27.14	28.37	Manual
AGD*	140.0	28.40	1.8 GPU days
Trilevel NAS*	33.3	28.26	6 GPU days
QuantNAS Our SP	**29.3**	28.22	1.2 GPU days
QuantNAS RFDN	**23.4**	28.3	1.6 GPU days

EDMIPS - searches only for quantization blocks, while QuantNAS searches for both, quantization levels and the quantization friendly blocks.

So, we obtain better architecture in terms of FLOPs and performance.

[QuantNAS for super resolution: searching for efficient quantization-friendly architectures against quantization noise]

- It is preferable to have activations and weights within the same data format, otherwise we will spend time to convert from one into another;
- Sometimes, converting from one format to another is not that difficult e.g. 8 bit activations and 4 bit weights;
- Most of the current LLM quantization approaches use FP16 for activations and 4 bit for weights;
- Mixed Precision can be extended to block-wise quantization, when each block in a matrix may have different levels of quantization (actually it is the case for current for LLM quantization approaches).


Quantization by training
Post Training Quantization
Quantization Aware training

Post Training Quantization (PTQ)

1. Train a model
2. Quantize weights
3. Finetune model BN statistics or other FP blocks

PROS:
- Can be applied for already trained models
- Does not necessary require access do the data or to all the data

CONS:
- Usually has lower quality than QAT

Quantization Aware Training (QAT)

1. Quantize weights
2. Train a model with quantized weights

PROS:
- Almost lossless quality for some datasets

CONS:
- Requires approximation of non-differenitable quantization function
- Usually, requires training a model from scratch
- Need access to the data

	Baseline FP32 mAP	INT8 mAP with PTQ	INT8 mAP with QAT
PeopleNet-ResNet18	78.37	59.06	78.06
PeopleNet-ResNet34	80.2	62	79.57

source

>>> STE and Alternatives

So, if we want to train a model and quantize it at the same time, what should we do?

To use QAT we need to compute gradients of a quantization function

Straight Through Estimator (STE)
2013

$$QW = Q(W)$$

$$y = F(x, QW)$$

$$W := W - \alpha \frac{\partial loss}{\partial QW}$$

Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation

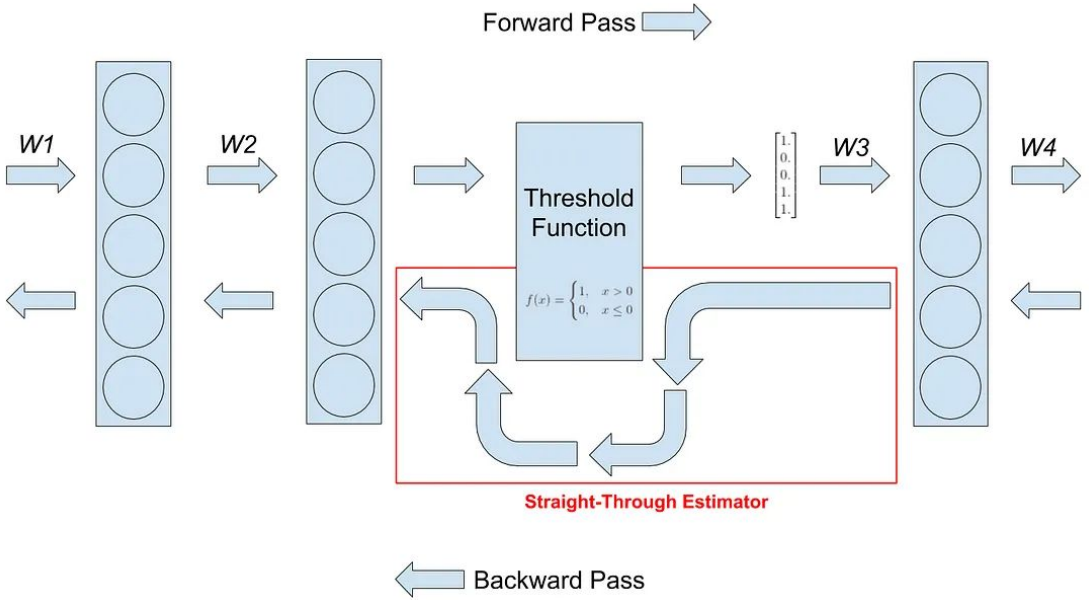


Image source

Use quantization noise to estimate quantization degradation

Differentiable!

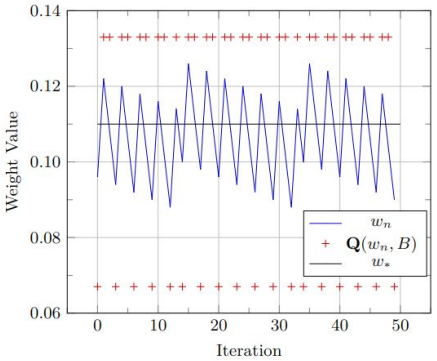
$$QW_b = Q(W, b)$$

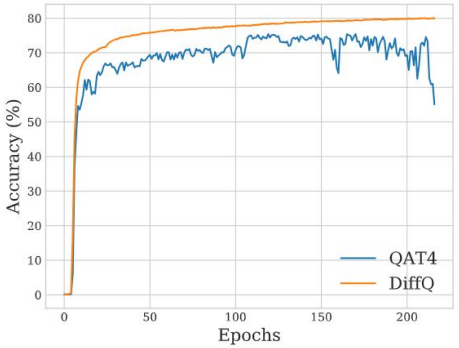
$$QN = QW_b - W \approx \frac{\Delta_b}{2}\mathbf{U}[-1, 1]$$

$$Q\hat{W}_b = W + \frac{\Delta_b}{2}\mathbf{U}[-1, 1]$$

$$y = F(x, W + \frac{\Delta_b}{2}\mathbf{U}[-1, 1])$$

Differentiable Model Compression via Pseudo Quantization Noise





Do we have any problems with that?

$$QW_b = Q(W, b)$$

$$QN = QW_b - W \approx \frac{\Delta_b}{2}\mathbf{U}[-1, 1]$$

$$Q\hat{W}_b = W + \frac{\Delta_b}{2}\mathbf{U}[-1, 1]$$

$$y = F(x, W + \frac{\Delta_b}{2}\mathbf{U}[-1, 1])$$

Differentiable Model Compression via Pseudo Quantization Noise

Assumptions:

1. QN is independent from W
2. QN is uniformly distributed
3. QN is white, that is, has a flat power spectral density

- Assumption 1 is always false
- Assumptions 2 and 3 are true only if W is uniformly distributed
- 2 and 3 may be reasonable if Δ (quantization step) is asymptotically small

$$QW_b = Q(W, b)$$

$$QN = QW_b - W \approx \frac{\Delta_b}{2}\mathbf{U}[-1, 1]$$

$$Q\hat{W}_b = W + \frac{\Delta_b}{2}\mathbf{U}[-1, 1]$$

$$y = F(x, W + \frac{\Delta_b}{2}\mathbf{U}[-1, 1])$$

Source: M. Gray Quantization introduction

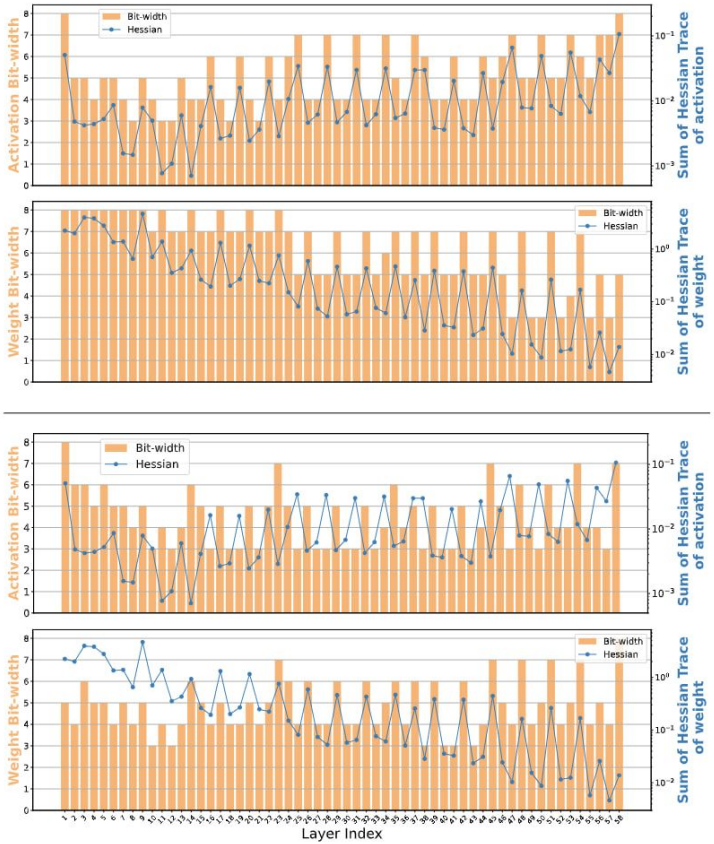
Assumptions:

1. QN is independent from W
2. QN is uniformly distributed
3. QN is white, that is, has a flat power spectral density

- Assumption 1 is always false
- Assumptions 2 and 3 are true only if W is uniformly distributed
- 2 and 3 may be reasonable if Δ (quantization step) is asymptotically small

Empirical results for model compression:

- It works to approximate 8 and 4,3 bit quantization

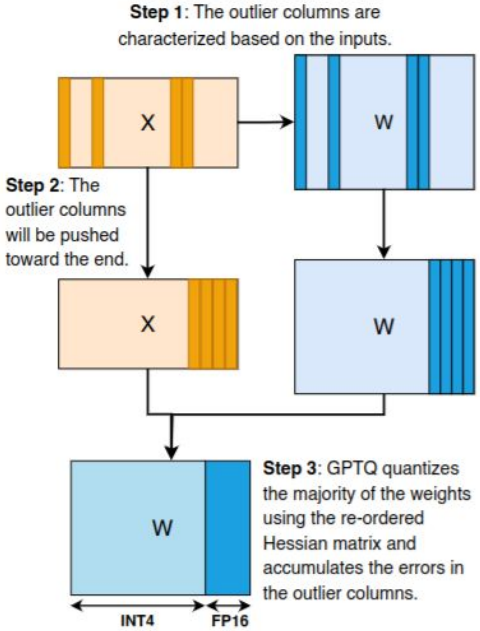


[NIPQ: Noise proxy-based Integrated Pseudo-Quantization]

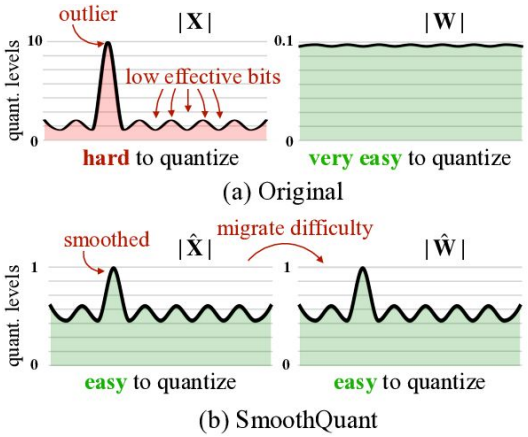
>>> Quantization and NAS

>>> LLM Quantization

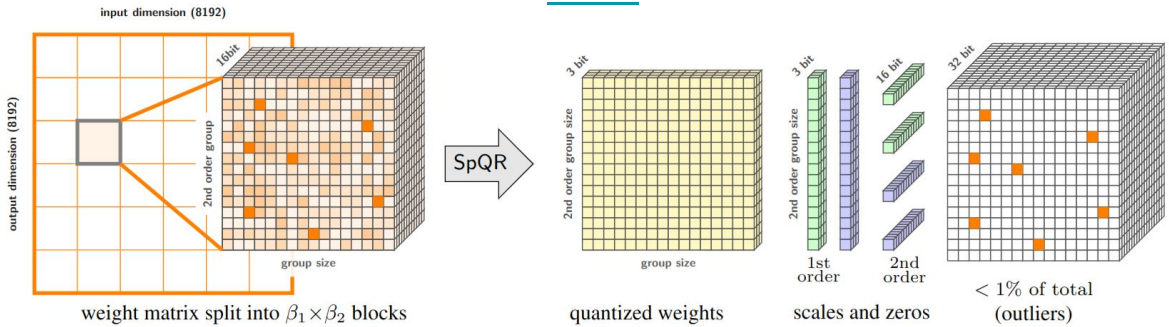
QUIK



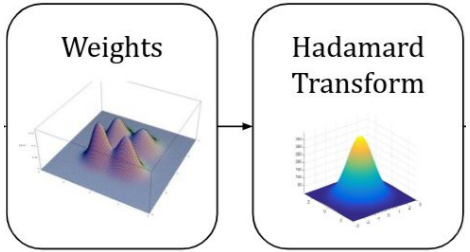
Smooth Quant



SPQR



QUIP # - LLM Quantization



Definition 2.1 (Chee et al. (2023)). A Hessian $H \in \mathbb{R}^{n\times n}$ is μ-incoherent if its eigendecomposition $H = Q\Lambda Q^T$ has

$$\max_{i,j} |Q_{ij}| = \max_{i,j} |e_i^T Q e_j| \leq \mu/\sqrt{n}.$$

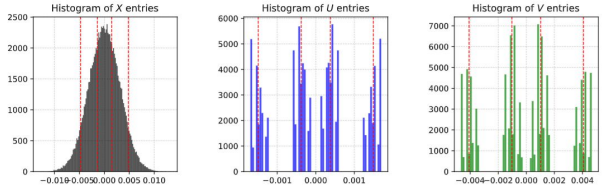
A weight matrix $W \in \mathbb{R}^{m\times n}$ is μ-incoherent if

$$\max_{i,j} |W_{ij}| = \max_{i,j} |e_i^T W e_j| \leq \mu \|W\|_F/\sqrt{mn}.$$

$$U^T(\text{quantized}(\tilde{W})(Vx)) \approx U^T(\tilde{W}(Vx)) = Wx.$$

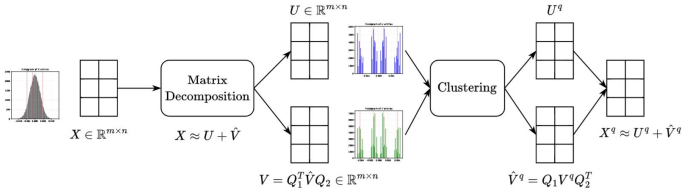
Transform weights to make W more incoherent.

Kashin Representations for LLM Quantization



$$X = U + Q_1 V Q_2^T$$

Q - to present V in some basis, so we need to Q to be easily computed and stored but it is another story.

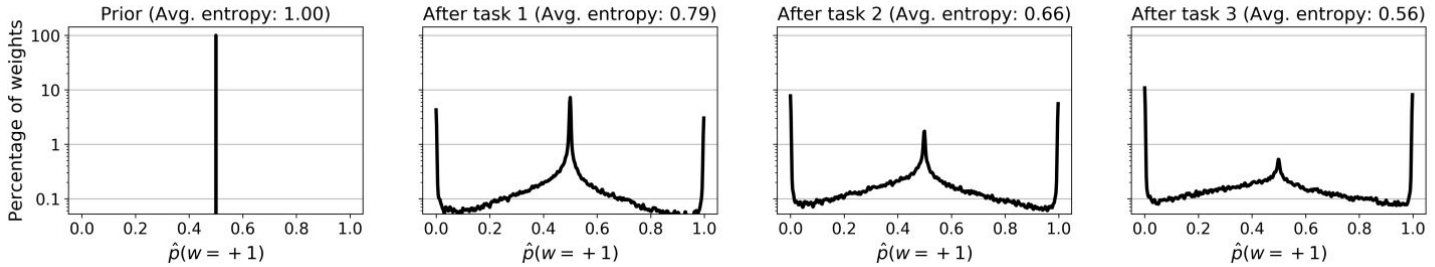


>>> Binarization

Binary weights and activations - how good can it be?

Paper	Year	ImageNet TOP1
XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks	2016	51.2
Binary Ensemble Neural Network: More Bits per Network or More Networks per Bit?	2018	61.0
Structured Binary Neural Networks for Image Recognition (GroupNet)	2019	65.2
Widening and Squeezing: Towards Accurate and Efficient QNNs	2020	69.18
ReActNet: Towards Precise Binary Neural Network with Generalized Activation Functions	2020	71.4
HIGH-CAPACITY EXPERT BINARY NETWORKS	2021	71.1

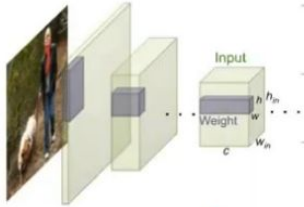
	STE	Our BayesBiNN method	Bop
Step 1: Get $\boldsymbol{w}_b$ from $\boldsymbol{w}_r$	$\boldsymbol{w}_b \leftarrow \text{sign}(\boldsymbol{w}_r)$	$\boldsymbol{w}_b \leftarrow \tanh\left((\boldsymbol{w}_r + \boldsymbol{\delta})/\tau\right)$	$\boldsymbol{w}_b \leftarrow \text{hyst}(\boldsymbol{w}_r, \boldsymbol{w}_b, \gamma)$
Step 2: Compute gradient at $\boldsymbol{w}_b$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$
Step 3: Update $\boldsymbol{w}_r$	$\boldsymbol{w}_r \leftarrow \boldsymbol{w}_r - \alpha \mathbf{g}$	$\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha \mathbf{s} \odot \mathbf{g}$	$\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha \mathbf{g}$



[1] Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization

[2] Training Binary Neural Networks using the Bayesian Learning Rule

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks 2016



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs, Real-Value Weights	+, −, ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs, Binary Weights	+, −	~32x	~2x	%56.8
BinaryWeight Binary Input (XNOR-Net)	Binary Inputs, Binary Weights	XNOR, bitcount	~32x	~58x	%44.2

– virtual reality
– augmented reality
– smart wearable devices

$I \in \mathbb{R}^{c \times w_{in} \times h_{in}}$

$W \in \mathbb{R}^{c \times w \times h}, w \leq w_{in}, h \leq h_{in}$

$W \approx \alpha B$

$B \in \{+1, -1\}^{c \times w \times h} \rightarrow$ binary filter

$\alpha > 0 \rightarrow$ scaling factor

$I * W \approx (I \oplus B)\alpha$

convolution without any multiplications

$W, B \in R^n$ where $n = cwh$

$$B^* = \arg\max_B W^T B \text{ s.t. } B \in \{+1, -1\}^n$$

$$B_i = +1 \text{ if } W_i \geq 0$$
$$B_i = -1 \text{ if } W_i < 0 \implies B^* = \text{sign}(W)$$

$$\alpha^* = \frac{W^T B}{n} = \frac{W^T \text{sign}(W)}{n} = \frac{\sum_i |W_i|}{n} = \frac{1}{n}\|W\|_1$$

Given W_t

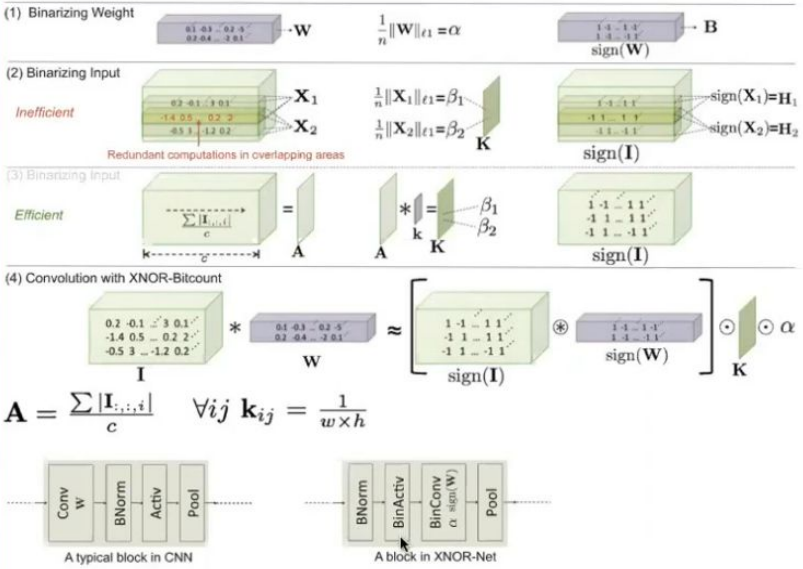
$$\alpha_t = \frac{1}{n}\|W_t\|_1$$

$$B_t = \text{sign}(W_t)$$

$$\widetilde{W}_t = \alpha_t B_t \qquad W_{t+1} = W_t - \eta \nabla_{\widetilde{W}_t} C$$

$$J(B, \alpha) = \|W - \alpha B\|^2 = \alpha^2 \underbrace{B^T B}_{=n \text{ since } B \in \{+1,-1\}^n} - 2\alpha W^T B + \underbrace{W^T W}_{:=c \text{ is a constant since } W \text{ is known}}$$

$$B^*, \alpha^* = \arg\min_{B,\alpha} J(B, \alpha)$$



$$\mathbf{A} = \frac{\sum |\mathbf{I}_{:,:,i}|}{c} \qquad \forall ij \ \mathbf{k}_{ij} = \frac{1}{w \times h}$$

Widening and Squeezing: Towards Accurate and Efficient QNNs 2020

We have a model with weights W where W has N channels and we want to find a projection matrix P so WP^T has K channels and $WP^T = sign(WP^T)$

$$min_{PB}\frac{1}{2}||WP^T - sign(WP^T)||_F^2 + \frac{\gamma}{2}||D(W) - D(sign(WP^T))||_F^2 \quad (1)$$

D - square pairwise Euclidean distance between features for all samples in the dataset, it is difficult to compute it but the expression above can be simplified. If P is orthogonal distance can be preserved.

$$min_{PB}\frac{1}{2}||WP^T - sign(WP^T)||_F^2 + \frac{\gamma}{2}||P^TP - I||_F^2 \quad (2)$$

Keep number of channles small, M is a filter wise vector

$$min_{PB}\frac{1}{2}||WP^T - M \circ sign(WP^T)||_F^2 + \frac{\gamma}{2}||P^TP - I||_F^2 + \beta||M||_1 \quad (3)$$

Layer num	orig	found	change
2	**128**	**410**	**3.2**
3	256	332	1.3
4	256	614	2.3
5	512	420	0.8
6	512	25	0.05
acc	0.93	0.92	

VGG small / CIFAR 100

1. It suggests we may need more filters at first layers
2. Usually first FP32 layers if fixed but probably we need more filters there too !?
3. The first filter is the most computationally heavy

TODO: check this assumption

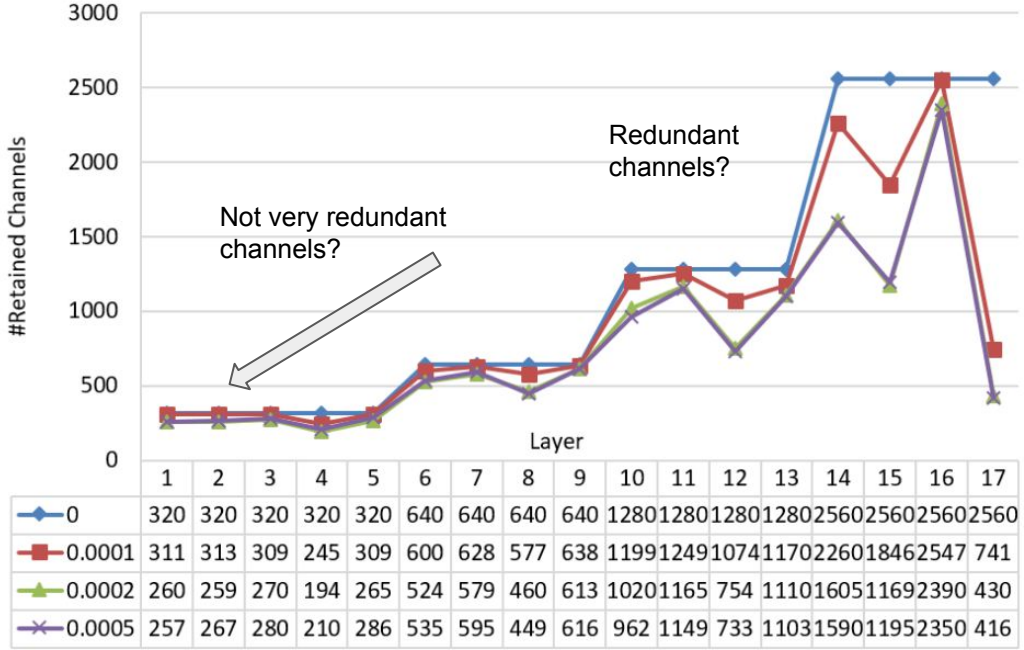
Widening and Squeezing: Towards Accurate and Efficient QNNs 2020

Increase number of channels 4 times and prune

reg	$top1_O$	$ratio$	$top1_P$	$top1_R$
0.0001	68.34	17.95	67.55	68.32
0.0002	65.27	33.06	60.84	69.19
0.0005	65.10	33.44	58.93	69.18

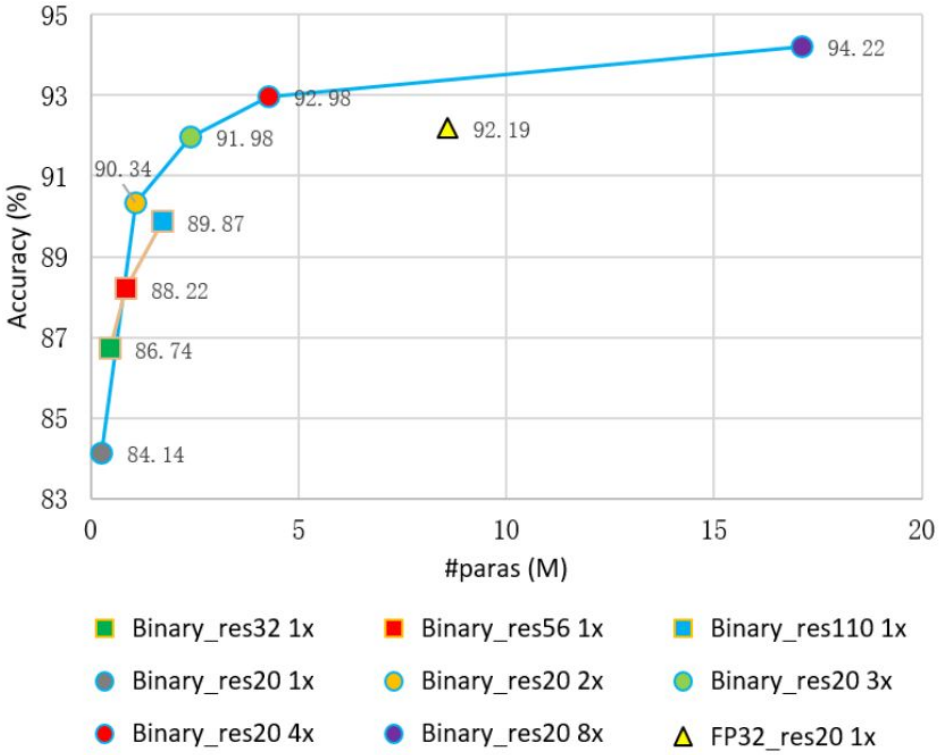
Table 9. Results of different batch norm scale regularization factors on ImageNet. reg means the value of scale regularization factor, $top1_O$ and $top1_P$ mean the accuracy before and after pruning. $ratio$ is the ratio of pruned channels, $top1_R$ is the accuracy that we retrain the pruned network from scratch.

*Authors use VGG here the network is overparameterized and it is easier to train it compared to small Resnets.

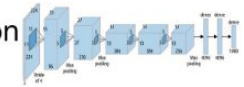
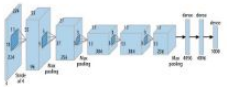
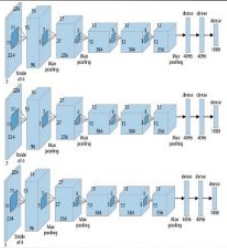


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	320	320	320	320	320	640	640	640	640	1280	1280	1280	1280	2560	2560	2560	2560
0.0001	311	313	309	245	309	600	628	577	638	1199	1249	1074	1170	2260	1846	2547	741
0.0002	260	259	270	194	265	524	579	460	613	1020	1165	754	1110	1605	1169	2390	430
0.0005	257	267	280	210	286	535	595	449	616	962	1149	733	1103	1590	1195	2350	416

Widening and Squeezing: Towards Accurate and Efficient QNNs 2020



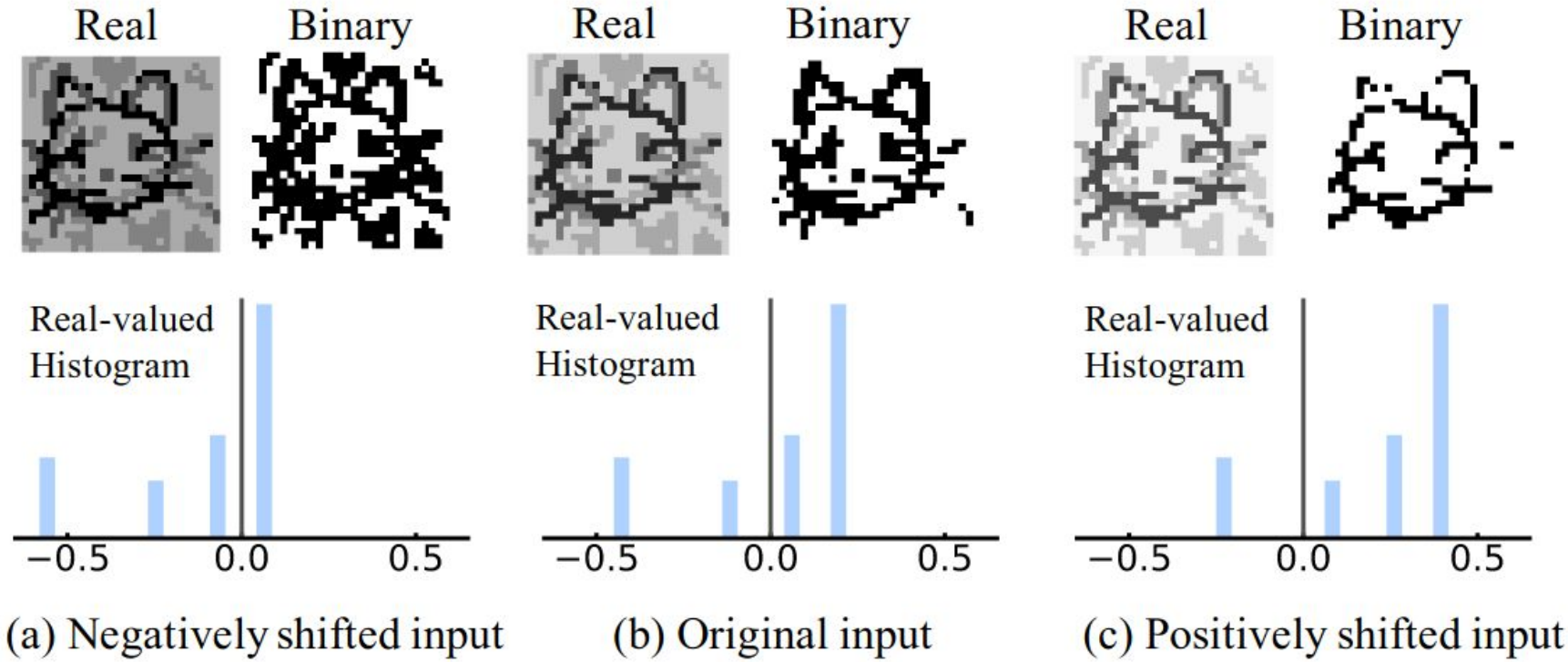
Binary Ensemble Neural Network: More Bits per Network or More Networks can Bit? 2018

Network		Numerics	Operations	Memory Speed-Up	Inference Speed-Up	Accuracy
 Input Image	Full-Precision DNN 	W: 32 bit A: 32 bit	+, -, x	1x	1x	High
	BNN 	W: 1 bit A: 1 bit	XNOR, Bitcount	~32x	(CPU) ~58x	Low
	(Ours) BENN (K ensemble) 	W: 1 bit A: 1 bit	XNOR, Bitcount	~(32/K)x	(CPU) ~58x	High

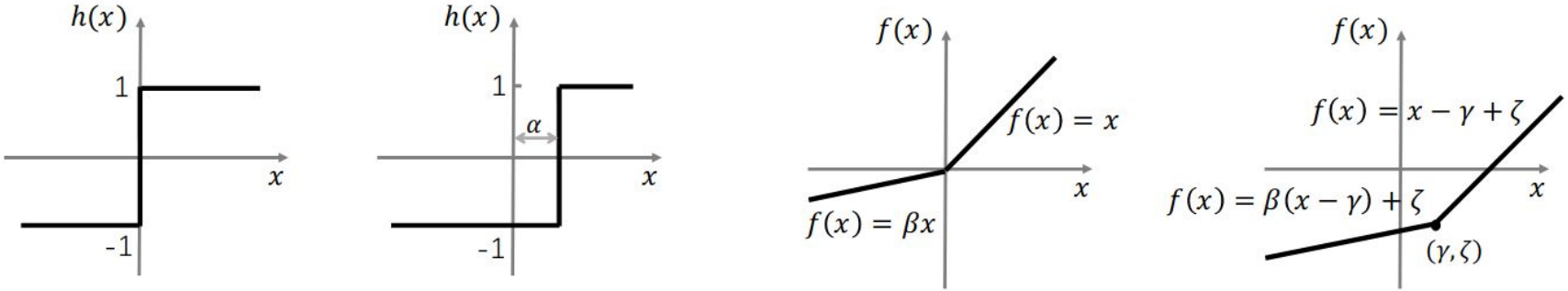
Memory savings

Inference speed up

ReActNet: Towards Precise Binary Neural Network with Generalized Activation Functions 2020



ReActNet: Towards Precise Binary Neural Network with Generalized Activation Functions 2020



(a) Sign vs. RSign

(b) PReLU vs. RPReLU

Fig. 4. Proposed activation functions, RSign and RPReLU, with learnable coefficients and the traditional activation functions, Sign and PReLU.