Accelerating Convolutional Neural Networks via Activation Map Compression

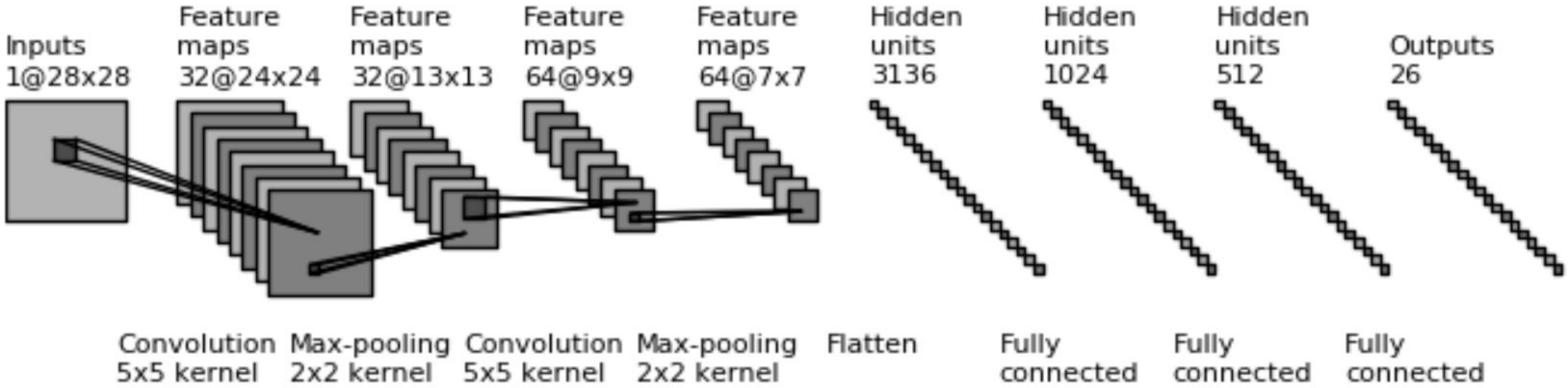
Haozhe Si, Shuen Wu, Zhongweiyang Xu

Research Problem

- Model compression (save memory)
- Model acceleration (save memory and (Multiply and ACcumulate) MACS)
    - AlexNet: 720 MMACS 60M Params [1]
    - VGG16: 15 BMACS 138M Params [1]

[1] Georgiadis, Georgios. “Accelerating Convolutional Neural Networks via Activation Map Compression.” *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018): 7078-7088.

Preliminary: ConvNet Concepts + Activation



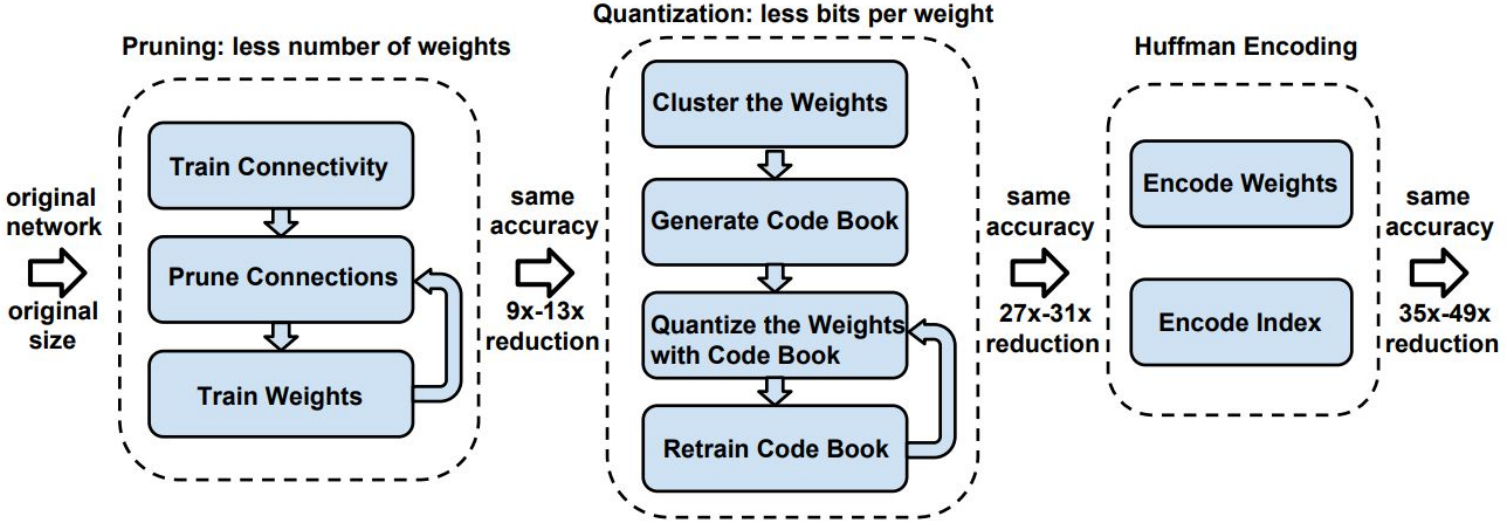
[1] An example of ConvNet Structure

[1] Adapted from https://github.com/gwding/draw_convnet

Background: Model Compression

Model weight compression:

- pruning, quantization, coding of **weights**



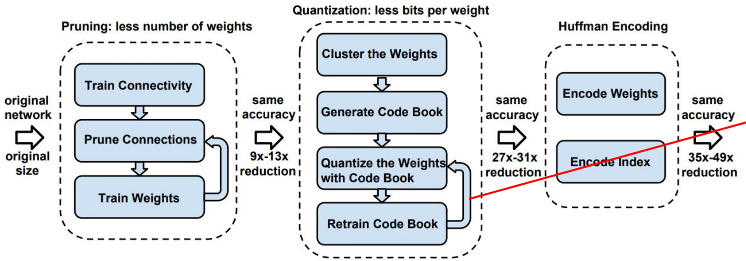
[1] Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding.

[1] Han, Song et al. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding." *arXiv: Computer Vision and Pattern Recognition* (2015): n. pag.

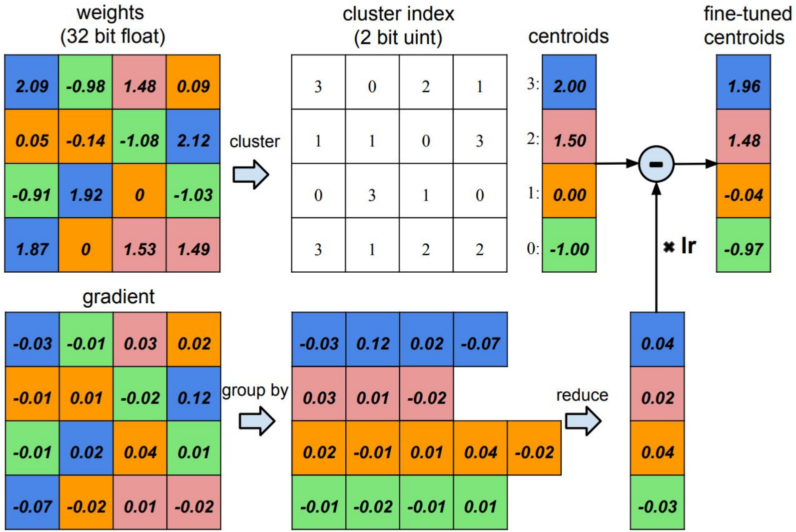
Background: Model Compression

Model weight compression:

- pruning, quantization, coding of **weights**



[1] Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding.



[1] Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom)

[1] Han, Song et al. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding." *arXiv: Computer Vision and Pattern Recognition* (2015): n. pag.

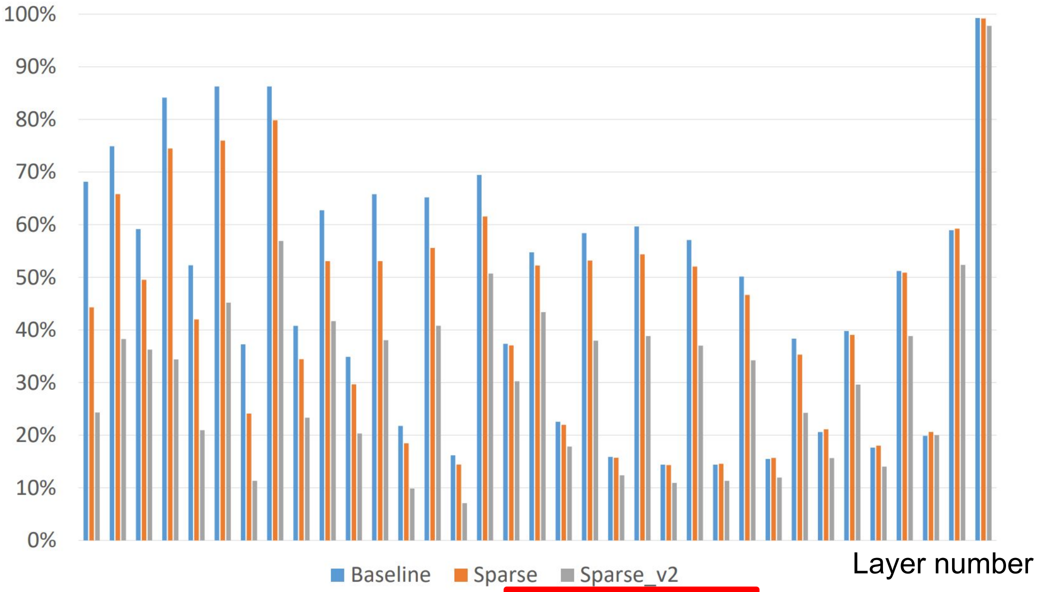
Background: Model Compression

- However, hidden layer's activation map is much larger than the weight
  - Inception-V3's second layer [1]:
    - Input: $149 \times 149 \times 32$
    - Output: $147 \times 147 \times 32$
    - Total 1,401,920 values
    - Weight between: $32 \times 32 \times 3 \times 3 = 9216$

[1] Georgiadis, Georgios. "Accelerating Convolutional Neural Networks via Activation Map Compression." *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018): 7078-7088.

Background: Model Compression

- hidden layer's activation map is sparse



[1] Figure 1. Percentage of non-zero activations (above)

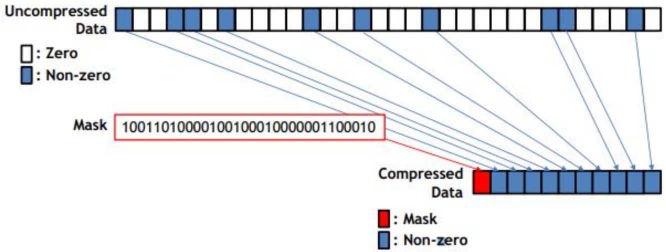


Figure 8: Zero-value compression.

[2] background: ZVC (zero value compression)

[1] Georgiadis, Georgios. "Accelerating Convolutional Neural Networks via Activation Map Compression." *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018): 7078-7088.
[2]M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, Austria, 2018, pp. 78-91, doi: 10.1109/HPCA.2018.00017.
keywords: {Graphics processing units;Training;Feature extraction;Bandwidth;Neural networks;Resource management;Backpropagation;GPU;Compression},

Propose

- Learning $\boxed{\text{sparser}}$ activation maps
- $\boxed{\text{Quantization}}$ of activation maps
- Entropy $\boxed{\text{coding}}$ of activation maps

Methodology: Sparsification

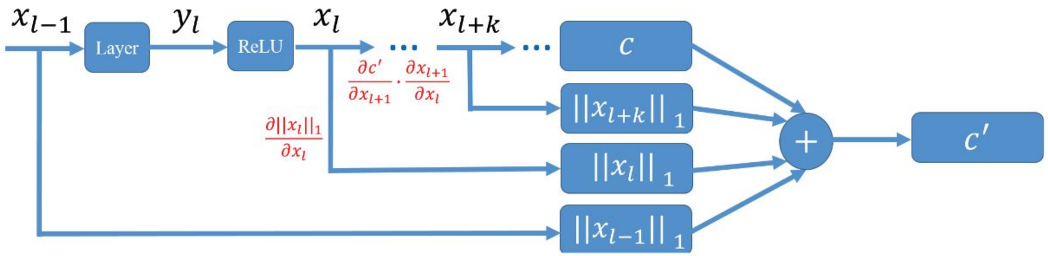
- Cost Function of vanilla CNN: $$E_0(w) = \frac{1}{N}\sum_{n=1}^{N} c_n(w) + \lambda_w r(w),$$
    - *n*: index of training samples
    - *w*: model weight
    - *c*: Cost function, e.g. cross-entropy loss $H(p,q) = -\sum_{x\in\mathcal{X}} p(x)\,\log q(x).$
    - *λ*: Regularization strength
    - *r*: Regularization term, e.g. L2
        - Purpose of L2 regularization:
            - Regularize on weight value
            - Preventing overfitting by preventing over-rely on certain feature

Methodology: Sparsification

- Sparsifying the activation map:
  - Applying L1 Loss on the activations.
    - The new cost function:

$$\begin{aligned} E(w) = & \quad E_0(w) + \frac{1}{N}\sum_{n=1}^{N}\sum_{l=0}^{L}\alpha_l \|x_{l,n}\|_1 \\ = & \quad \frac{1}{N}\sum_{n=1}^{N} c'_n(w) + \lambda_w r(w), \end{aligned}$$

    - *l*: layer index
    - $x_{l,n}$: the activation of sample *n* at layer *l*
    - *α*: L1 Loss strength, hyperparameter to tune.
    - y: Logits, the value before activation layer.



Methodology: Why L1 Loss Prompts Sparsity?

- L1 Loss: $L_1 = \sum_i |x_i|$
- Gradient of L1 Loss with sub-gradient: $$\frac{\partial |x_i|}{\partial x_i} = \begin{cases} +1 & \text{if } x_i > 0 \\ -1 & \text{if } x_i < 0 \\ [-1, 1] & \text{if } x_i = 0 \end{cases}$$
  - Case 1: Uniform Shrinkage:
    - The subgradient of L1 norm is constant (+1 or -1) for non-zero weights, shrinking all weights linearly.
  - Case 2: Zero Lock-in Effect:
    - At zero, the subgradient allows any value in [-1, 1], meaning the optimizer has no strict reason to move away from zero, promoting zero-valued weights.
- Compare to L2 Loss:
  - For L2 regularization, the penalty is w_j^2, and its derivative is $2w_j$
    - Proportional to the weight's size, so large weights shrink faster than small weights.
  - With L2 regularization, small weights shrink slowly and rarely reach zero. Instead, all weights get smaller without any of them becoming exactly zero, leading to a dense solution.

Methodology: Sparsification

- Cost Function of vanilla CNN: $E_0(w) = \frac{1}{N}\sum_{n=1}^{N} c_n(w) + \lambda_w r(w),$
- Specification of the activation map is achieved by applying L1 Loss on the activations.
  - The new cost function: $E(w) = \quad E_0(w) + \frac{1}{N}\sum_{n=1}^{N}\sum_{l=0}^{L} \alpha_l \|x_{l,n}\|_1$
- Computing the gradient w.r.t *x*:

$$\frac{\partial c_n'}{\partial x_{l,n}^j} = \alpha_l \frac{\partial \|x_{l,n}\|_1}{\partial x_{l,n}^j} = \begin{cases} +\alpha_l, & \text{if } x_{l,n}^j > 0 \\ -\alpha_l, & \text{if } x_{l,n}^j < 0 \\ 0, & \text{if } x_{l,n}^j = 0 \end{cases}$$

Methodology: Quantization

- The method then quantize floating point activation maps, x_l, to q bits using linear (uniform) quantization:

$$x_l^{\text{quant}} = \frac{x_l - x_l^{\text{min}}}{x_l^{\text{max}} - x_l^{\text{min}}} \times (2^q - 1),$$

  - Quantization is applied per-layer base.
    - x_l^{min} and x_l^{max} are selected from each layer.

Methodology: Why Quantization?

- Quantization reduces the bit-width of these activation values:
  - A 32-bit floating-point activation map of size 64×64×128 requires:
    - $64 \times 64 \times 128 \times 4$ bytes ≈ 2 MB
  - If quantized to 8-bit integers, the same activation map would take only:
    - $64 \times 64 \times 128 \times 1$ byte ≈ 512 KB
  - This 4x reduction in memory allows efficient usage of memory resources.
- Reducing the bit-width results in reducing entropy.
- Reducing entropy leads to shorter average codelenght for lossless compression.
- Side effect: Quantization introduces noise:
  - Help the model generalize better by preventing it from overfitting.
  - To learn more about Quantization-aware training: https://arxiv.org/pdf/1712.05877
  - To learn more about how noise helps model training: https://arxiv.org/abs/1909.03172

Methodology: Entropy Coding

- Purpose: Store sparse matrices while preserving fast arithmetic operations
- Problem: Common algorithms usually assume entire matrix available prior to storage
- Need: Data is often streamed and computation done on-the-fly so we need algorithm to encode one element at a time

Methodology: Golomb Coding

Given a nonnegative integer n and a positive integer divisor $m > 0$, the Golomb code of n with respect to m, denoted $G_m(n)$, constructed as follows:

Step 1. Form the unary code of quotient $\lfloor n/m \rfloor$

(The unary code of integer q is defined as q 1s followed by a 0)

Step2. Let k=$\lceil \log_2 m \rceil, c = 2^k - m, r = n \bmod m,$ and compute truncated remainder r' such that

$$r' = \begin{cases} r \text{ truncated to } k\text{-1 bits} & 0 \le r < c \\ r + c \text{ truncated to } k \text{ bits} & \text{otherwise} \end{cases}$$

Step 3. Concatenate the results of steps 1 and 2.

Computer Science Engineering Concepts. (2020, May 6). *Golomb Coding*. YouTube.

https://www.youtube.com/watch?v=eJQf55fwAE0

Methodology: Exponential-Golomb

- Separate successively sub-vectors of 2^k, 2^{k+1}, … binary zeros
- Encode rest of run-length as a binary number

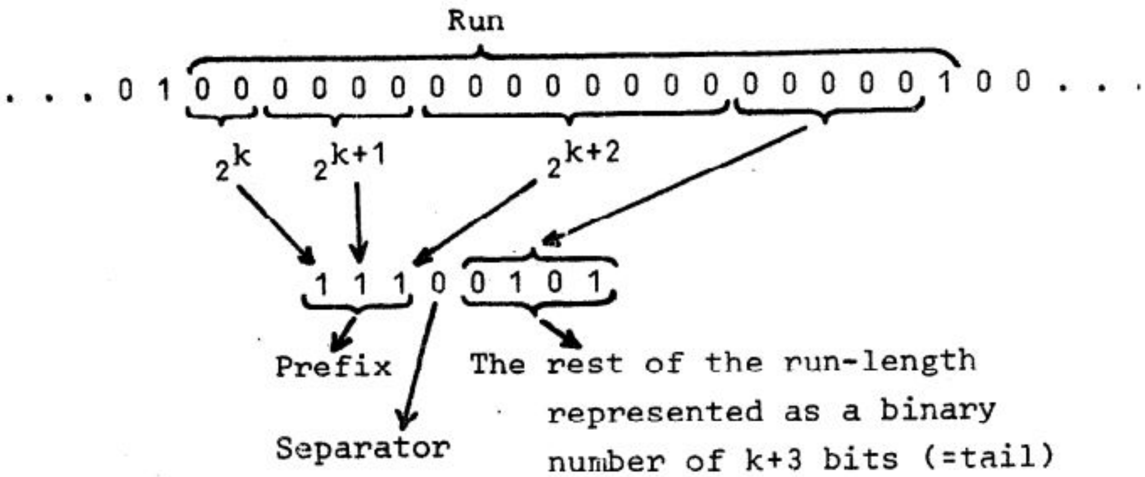


Fig. 2. An example of the exp-Golomb code (here $k = 1$).

Fig. from Jukka Teuhola. A compression method for clustered bit-vectors. *Information processing letters*, 1978.

Methodology: Exponential-Golomb

- Let s=run-length
- Step 1: Determine n such that $\sum_{i=k}^{n} 2^i \leq s \leq \sum_{i=k}^{n+1} 2^i$
- Step 2: Form the prefix of n-k+1 1-bits
- Step 3: Insert the separator (0-bit)
- Step 4: Form the tail: express the value of $s - \sum_{i=k}^{n} 2^i$ as a binary number with n+1 bits

Methodology: Exponential-Golomb

- Exponential-Golomb encoding is optimal when
  1. The activation maps are mostly sparse
  2. The first-order probability distribution of the activation maps have a long tail (e.g. geometric distribution)
- In this case, we base things off an exponential number of zeros instead of hardcoding due to the on-the-fly/streamed data that needs to be processed in real time
- The authors of the paper chose to use exponential-golomb simply from reading the histograms and suspecting the distributions were near geometric

Methodology: Sparse-exponential-Golomb

- Algorithm based on older exponential-Golomb algorithm
- Exponential-Golomb with $k=0$ parameter assigns a code word of length 1 for $x=0$
- Unfortunately, if we use $k>0$, then $x=0$ is no longer 1 bit code word
- Solution: Dedicate '1' for $x=0$ and pre-append everything else with '0'

Algorithm 1 Sparse-exponential-Golomb

```
Input: Non-negative integer x, Order k
Output: Bitstream y
function encode_sparse_exp_Golomb (x, k)
{
   If k == 0:
      y = encode_exp_Golomb(x, k)
   Else:
      If x == 0:
            Let y = '1'
      Else:
            Let y = '0' + encode_exp_Golomb(x − 1, k)
   Return y
}
Input: Bitstream x, Order k
Output: Non-negative integer y
function decode_sparse_exp_Golomb (x, k)
{
   If k == 0:
      y = decode_exp_Golomb(x, k)
   Else:
      If x[0] == '1':
            Let y = 0
      Else:
            Let y = 1 + decode_exp_Golomb(x[1: ], k)
   Return y
}
```

[1] Georgiadis, Georgios. "Accelerating Convolutional Neural Networks via Activation Map Compression." *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018): 7078-7088.

Experiment: Acceleration

Dataset	Model	Variant	Top-1 Acc.	Top-5 Acc.	Acts. (%)	Speed-up
MNIST	LeNet-5	Baseline	98.45%	-	53.73%	1.0×
		Sparse	**98.48% (+0.03%)**	-	**23.16%**	**2.32×**
CIFAR-10	MobileNet-V1	Baseline	89.17%	-	47.44%	1.0×
		Sparse	**89.71% (+0.54%)**	-	**29.54%**	**1.61×**
ImageNet	Inception-V3	Baseline	75.76%	92.74%	53.78%	1.0×
		Sparse	**76.14% (+0.38%)**	**92.83% (+0.09%)**	33.66%	1.60×
		Sparse_v2	68.94% (-6.82%)	88.52% (-4.22%)	**25.34%**	**2.12×**
	ResNet-18	Baseline	69.64%	88.99%	60.64%	1.0×
		Sparse	**69.85% (+0.21%)**	**89.27% (+0.28%)**	49.51%	1.22×
		Sparse_v2	68.62% (-1.02%)	88.41% (-0.58%)	**34.29%**	**1.77×**
	ResNet-34	Baseline	73.26%	91.43%	57.44%	1.0×
		Sparse	**73.95%(+0.69%)**	**91.61% (+0.18%)**	46.85%	1.23×
		Sparse_v2	67.73% (-5.53%)	87.93% (-3.50%)	**29.62%**	**1.94×**

Table 2. Accelerating neural networks via sparsification. Numbers in brackets indicate change in accuracy. Acts. (%) shows the percentage of non-zero activations.

- Sparse: targeting at accuracy with sparsity
- Sparse_v2: targeting at high sparsity

$$E(w) = E_0(w) + \frac{1}{N}\sum_{n=1}^{N}\sum_{l=0}^{L}\alpha_l \|x_{l,n}\|_1$$

Experiment: Acceleration

Dataset	Model	Variant	Top-1 Acc.	Top-5 Acc.	Acts. (%)	Speed-up
MNIST	LeNet-5	Baseline	98.45%	-	53.73%	1.0×
		Sparse	**98.48% (+0.03%)**	-	**23.16%**	**2.32**×
CIFAR-10	MobileNet-V1	Baseline	89.17%	-	47.44%	1.0×
		Sparse	**89.71% (+0.54%)**	-	**29.54%**	**1.61**×
ImageNet	Inception-V3	Baseline	75.76%	92.74%	53.78%	1.0×
		Sparse	**76.14% (+0.38%)**	**92.83% (+0.09%)**	33.66%	1.60×
		Sparse_v2	68.94% (-6.82%)	88.52% (-4.22%)	**25.34%**	**2.12**×
	ResNet-18	Baseline	69.64%	88.99%	60.64%	1.0×
		Sparse	**69.85% (+0.21%)**	**89.27% (+0.28%)**	49.51%	1.22×
		Sparse_v2	68.62% (-1.02%)	88.41% (-0.58%)	**34.29%**	**1.77**×
	ResNet-34	Baseline	73.26%	91.43%	57.44%	1.0×
		Sparse	**73.95%(+0.69%)**	**91.61% (+0.18%)**	46.85%	1.23×
		Sparse_v2	67.73% (-5.53%)	87.93% (-3.50%)	**29.62%**	**1.94**×

Table 2. Accelerating neural networks via sparsification. Numbers in brackets indicate change in accuracy. Acts. (%) shows the percentage of non-zero activations.

- Increasing sparsity can also increase accuracy.
- Hyperparameter need to be carefully selected.

Experiment: Acceleration

Network	Algorithm	Top-1 Acc. Change	Top-5 Acc. Change	Speed-up
ResNet-18	Ours (Sparse)	+0.21%	+0.28%	18.4%
	Ours (Sparse_v2)	**-1.02%**	**-0.58%**	**43.5%**
	LCCL [10]	-3.65%	-2.30%	34.6%
	BWN [50]	-8.50%	-6.20%	50.0%
	XNOR [50]	-18.10%	-16.00%	98.3%
ResNet-34	Ours (Sparse)	+0.69%	+0.18%	18.4%
	Ours (Sparse_v2)	**-5.53%**	**-3.50%**	**48.4%**
	LCCL [10]	-0.43%	-0.17%	24.8%
	PFEC [36]	-1.06%	-	24.2%
LeNet-5	**Ours (Sparse)**	**+0.03%**	-	**56.9%**
	[18] ($p = 70\%$)	-0.12%	-	7.3%
	[18] ($p = 80\%$)	-0.57%	-	14.7%

- Compare with previous SoTA model acceleration methods
- Sparse_v2 achieve better speed-up with a balance with of accuracy.

Experiment: Quantization

Model	Variant	Measurement	float32	uint16	uint12	uint8
LeNet-5 (MNIST)	Baseline	Top-1 Acc. Compression	98.45% -	98.44% (-0.01%) 3.40× (1.70×)	98.44% (-0.01%) 4.40× (1.64×)	98.39% (-0.06%) 6.32× (1.58×)
	Sparse	Top-1 Acc. Compression	98.48% (+0.03%) -	98.48% (+0.03%) 6.76× (3.38×)	98.49% (+0.04%) 8.43× (3.16×)	98.46% (+0.01%) **11.16× (2.79×)**
MobiletNet-V1 (CIFAR-10)	Baseline	Top-1 Acc. Compression	89.17% -	89.18% (+0.01%) 5.52× (2.76×)	89.15% (-0.02%) 7.09× (2.66×)	89.16% (-0.01%) 9.76× (2.44×)
	Sparse	Top-1 Acc. Compression	89.71% (+0.54%) -	89.72% (+0.55%) 5.84× (2.92×)	87.72 (+0.55%) 7.33× (2.79×)	89.62% (+0.45%) **10.24× (2.56×)**

Table 5. Effect of quantization on compression on SEG. LeNet-5 is compressed by 11× and MobileNet-V1 by 10×. In brackets, we report change in accuracy and compression gain over the float32 baseline.

- Quantizing the model can achieve model compression will not affect the model performance.
- Quantization can also increase the model performance in some cases.

Experiment: Compression

Dataset	Model	Variant	Bits	Top-1 Acc.	Top-5 Acc.	SEG	EG [61]	HC [18]	ZVC [52]	ZLIB [1]
MNIST	LeNet-5	Baseline	float32	98.45%	-	-	-	-	-	-
		Baseline	uint16	98.44% (-0.01%)	-	**3.40× (1.70×)**	2.30× (1.15×)	2.10× (1.05×)	3.34× (1.67×)	2.42× (1.21×)
		Sparse		**98.48% (+0.03%)**	-	**6.76× (3.38×)**	4.54× (2.27×)	3.76× (1.88×)	6.74× (3.37×)	3.54× (1.77×)
CIFAR-10	MobileNet-V1	Baseline	float32	89.17%	-	-	-	-	-	-
		Baseline	uint16	89.18% (+0.01%)	-	**5.52× (2.76×)**	3.70× (1.85×)	2.90× (1.45×)	5.32× (2.66×)	3.76× (1.88×)
		Sparse		**89.72% (+0.55%)**	-	**5.84× (2.92×)**	3.90× (1.95×)	3.00× (1.50×)	5.58× (2.79×)	3.90× (1.95×)
ImageNet	Inception-V3	Baseline	float32	75.76%	92.74%	-	-	-	-	-
		Baseline	uint16	75.75% (-0.01%)	92.74% (+0.00%)	**3.56× (1.78×)**	2.42×(1.21×)	2.66× (1.33×)	3.34× (1.67×)	2.66× (1.33×)
		Sparse		**76.12% (+0.36%)**	**92.83% (+0.09%)**	**5.80× (2.90×)**	4.10× (2.05×)	4.22× (2.11×)	5.02× (2.51×)	3.98× (1.99×)
		Sparse_v2		68.96% (-6.80%)	88.54% (-4.20%)	**6.86× (3.43×)**	5.12× (2.56×)	5.12× (2.56×)	6.36× (3.18×)	4.90× (2.45×)
	ResNet-18	Baseline	float32	69.64%	88.99%	-	-	-	-	-
		Baseline	uint16	69.64% (+0.00%)	88.99% (+0.00%)	**3.22× (1.61×)**	2.32× (1.16×)	2.54× (1.27×)	3.00× (1.50×)	2.32× (1.16×)
		Sparse		**69.85% (+0.21%)**	**89.27% (+0.28%)**	**4.00× (2.00×)**	2.70× (1.35×)	3.04× (1.52×)	3.60× (1.80×)	2.68× (1.34×)
		Sparse_v2		68.62% (-1.02%)	88.41% (-0.58%)	**5.54× (2.77×)**	3.80× (1.90×)	4.02× (2.01×)	4.94× (2.47×)	3.54× (1.77×)
	ResNet-34	Baseline	float32	73.26%	91.43%	-	-	-	-	-
		Baseline	uint16	73.27% (+0.01%)	91.43% (+0.00%)	**3.38× (1.69×)**	2.38× (1.19×)	2.56× (1.28×)	3.14× (1.57×)	2.46× (1.23×)
		Sparse		**73.96% (+0.70%)**	**91.61% (+0.18%)**	**4.18× (2.09×)**	2.84× (1.42×)	3.04× (1.52×)	3.78× (1.89×)	2.84× (1.42×)
		Sparse_v2		67.74% (-5.52%)	87.90% (-3.53%)	**6.26× (3.13×)**	4.38× (2.19×)	4.32× (2.16×)	5.58× (2.79×)	4.02× (2.01×)

Table 6. Compressing activation maps. We report the Top-1/Top-5 accuracy, with the numbers in brackets indicating the change in accuracy. The total compression gain is reported for various state-of-the-art algorithms (in brackets we also report the compression gain without including gains from quantization). SEG outperforms other state-of-the-art algorithms in all models and datasets.

- Evaluation on compression gain and accuracy.
- SEG achieves the best compression rate with good accuracy.

Zlib: https://www.euccas.me/zlib/

Conclusion

- This paper proposed a three-stage compression and acceleration pipeline that sparsifies, quantizes and encodes activation maps of CNN's:
  - Sparsify:
    - Increases the number of zero values leading to model acceleration on specialized hardware
  - Quantization and Encoding:
    - Contribute to compression by effectively utilizing the lower entropy of the sparser activation maps
- Experiments demonstrate that the proposed pipeline effectively reduces the computational and memory requirements while reaching good performance.

Limitations

- Sparse activation is effective, however, it only happens to CNN.
- In the age of Transformer, sparse attention map is more popular and the compression methods are mainly quantization.
- However, similar techniques are still useful in infra-level:
  - Input compression
  - Checkpoint/gradient compression for speeding up training
  - Weight compression for speeding up inference