Learning Compact and Efficient Networks through Explicit Connection Sensitivity Pruning and Quantization

Arvind Subramaniam
arvindsu@andrew.cmu.edu

Diksha Agarwal
dikshaa@andrew.cmu.edu

Nitish Garikoti
ngarikot@andrew.cmu.edu

Vinay Patil
vspatil@andrew.cmu.edu

Abstract

In this work, we propose an approach to reduce the memory footprint and inference time of classical Deep Neural Networks (DNNs) while simultaneously maintaining test performance. First, we perform pruning by considering the magnitude of both the weights and the corresponding gradients of the reference network after training the model for one epoch. We call this approach Explicit Connection Sensitivity Pruning (ECS). Next, the memory of the pruned model is further reduced by quantizing the network weights via kmeans clustering and 8-bit quantization. Our work seeks to perform both pruning and quantization to develop compact and efficient DNNs, as opposed to existing literature which has primarily focused on either pruning or quantization. The code for our implementation can be found at https://github.com/vspatil173/Deep-Pruning-Quantization-approach.

1 Introduction

Over the years, Deep Neural Networks (DNNs) have been pivotal in improving performance benchmarks on challenging tasks such as image classification, object detection, speech processing, and medical applications. As a result, they have been increasingly deployed on computers and web applications over the years. Due to the increase in the number of smartphones, it has become important to deploy these networks on such devices as well. However, this presents an interesting challenge since classical DNNs occupy too much memory and require considerable computation resources for training and inference tasks. Existing literature has made use of pruning and quantization strategies to reduce the memory footprint and accelerate inference of classical DNNs.

Initial efforts in pruning leverage multi-steps to eliminate less relevant weights according to the corresponding gradients[18,19].

Building on it we propose a novel approach to jointly optimize the time and space complexities of Deep Neural Networks. We are exploring an approach that prunes the weights with just 1 epoch of training by checking corresponding gradients. Following pruning, we propose to speed up the inference by applying kmeans-based quantization followed by 8-bit quantization to the pruned network.

A high-level representation of our approach is provided in Fig. 1. We start with the reference model, which is the unpruned, non-quantized model. After training the model for 1 epoch, we sequentially perform pruning and quantization on the reference model to increase its efficiency, by first reducing its memory footprint through pruning and then speeding up inference via quantization. An important point to note is that we perform single-shot pruning, as opposed to iterative pruning. In other words, we do not prune the model between successive training cycles. Rather, we obtain the sparse pruned

11785 Course Project (NeurIPS 2021), Pittsburgh, USA.

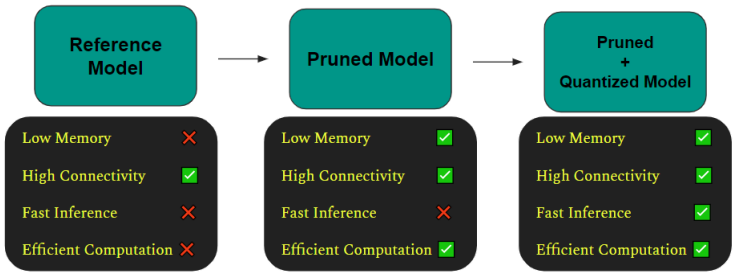


Figure 1: Stages of the proposed framework.

model after the first epoch and permanently fix redundant weights to zero during the entire training process.

2 Related Work

2.1 Pruning

Over the years, pruning has been extensively explored to reduce the memory footprint of classical Deep Neural Networks. We categorize previous pruning strategies into three categories:

- **Pruning after Training:** This pruning is performed after the training stage is completed by performing checks on trained weights and reducing redundancy in weights to obtain a sparse network which helps optimize performance. Previous papers have followed two approaches to prune networks after training: pruning based on the Hessian Matrix of the network [5, 6] and pruning based on the magnitude of the weights of the pre-trained network [3, 4]. The second approach assumes that weights with a smaller magnitude have lesser importance and can be deemed redundant. Both the aforementioned methods work well at moderate compression ratios of 2x - 5x, but are ineffective at higher compression ratios of > 10x.
- **Pruning while training:** Eliminating redundant weights in every cycle of training can help in reducing the weights and thus ensure that only important weights are kept. However, these methods are expensive to deploy since they usually involve multiple prune-train cycles. One of the most popular approach involves uniformly increasing the sparsity of the baseline network over the course of training by pruning either the network weights or the gradients [1, 8, 9]. Our approach differs from these methods in the aspect that we jointly consider both the weights as well as the gradients to prune the baseline network.
- **Pruning before training:** Spurred by the Lottery Ticket Hypothesis [2], there has been an increased interest in developing sparse learning representations which are pruned in a single shot [7]. This mode of pruning is relatively much faster and simpler, it has been implemented using techniques such as low-rank approximation, matrix factorization, and other structured simplification methods [10, 11, 21].

2.2 Quantization

Network Quantization has been widely used as an effective technique to accelerate inference in classical DNNs, thereby enabling their deployment on resource-constrained devices. Initial explorations in Network Quantization primarily involved binarization and ternary quantization, in which the network weights/activations were -1,1 and -1,0,1 respectively [12, 13, 14]. Works like Group-Net [15] and ABC-Net [16] used multiple binary layers to achieve state of the art performance, at the cost of higher computational expense. Recent techniques have explored the combination of pruning and quantization, thereby reducing both the time and space complexities of the network, with a marginal drop in the overall performance. Song et al. [3] employed pruning and trained quantization to achieve faster inference and a high compression rate while maintaining a marginal drop in performance. Lou et al. [17] automatically quantized every filter and activation layer of a pre-trained DNN using hierarchical Deep Reinforcement Learning. Our work explores a combination of network pruning

and quantization, and aims to produce compact DNNs which have relatively faster inference, while maintaining the performance of the baseline model.

3 Datasets

3.1 CIFAR-10

CIFAR-10 is a publicly available widely used dataset for image classification consisting of 10 classes. There are a total of 50000 training examples and 10000 validation examples. Every image is a 32 X 32 dimensional image which has 3 color vectors denoting RGB [20].

3.2 CIFAR-100

CIFAR-100 is a publicly available widely used dataset for image classification consisting of 100 classes. There are a total of 50000 training examples and 10000 validation examples. Every image is a 32 X 32 dimension which has 3 color vectors denoting RGB [20].

4 Methods

4.1 Method 0 - Baseline (SNIP)

Any pruning aims to learn the small subset of weights that can mimic the performance of the pristine model. The contemporary methods use prune-retrain methods repetitively every epoch and these logical designs can't be extended to other/new models. In SNIP, the idea is to identify the important connections before training by acquiring the information from the data. So, with the known amount of compression we remove the redundant weights by pruning once before training and learn the non-pruned weights by training.

This method provides various advantages such as simplistic model as we prune once in the beginning and train in the normal procedure. It also provides diversity on models on which we apply this method. This methods chooses the structurally important connections leading to robustness thus applicable to diverse model designs like Convolution Networks and Recurrent Neural Networks.

If D is the dataset, $m \in \{0, 1\}$ i.e. is connectivity of θ i.e. weights. Given the sparsity level T, the loss function can be defined as:

$$\min_{\mathbf{m},\boldsymbol{\theta}} J(\mathbf{m} \odot \boldsymbol{\theta}; D) = \min_{\mathbf{m},\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} jl(\mathbf{m} \odot \boldsymbol{\theta}; (x_i, y_i))$$

$$s.t\ \boldsymbol{\theta} \in R^k$$

$$\|\mathbf{m}\|_0 \leq \mathrm{T}$$

Keeping the model intact but just removing connection j affects the loss like the following:

$$\Delta J_j(\boldsymbol{\theta}; D) = J(1 \odot \boldsymbol{\theta}; D) - J((1 - q_j) \odot \boldsymbol{\theta}, D)$$

$$\Delta J_j(\boldsymbol{\theta}; D) \approx \mathrm{f}_j(\boldsymbol{\theta}; D) = J(1 \odot \boldsymbol{\theta}; D) - J((1 - q_j) \odot \boldsymbol{\theta}, D) = \frac{\partial J_j(\boldsymbol{\theta}; D)}{\partial m_j} | c = 1 = \lim \delta -> 0 \frac{J(\mathbf{m} \odot \boldsymbol{\theta}; D) - J((\mathbf{m} - q_j) \odot \boldsymbol{\theta}, D)}{\delta} |_{c=1}$$

As f_j is derivative, Normalisation of these derivatives gives us connection sensitivity.

$$css_j = \frac{|f_j(\boldsymbol{\theta}; D)|}{\sum_{i=1}^{k} |f_i(\boldsymbol{\theta}; D)|}$$

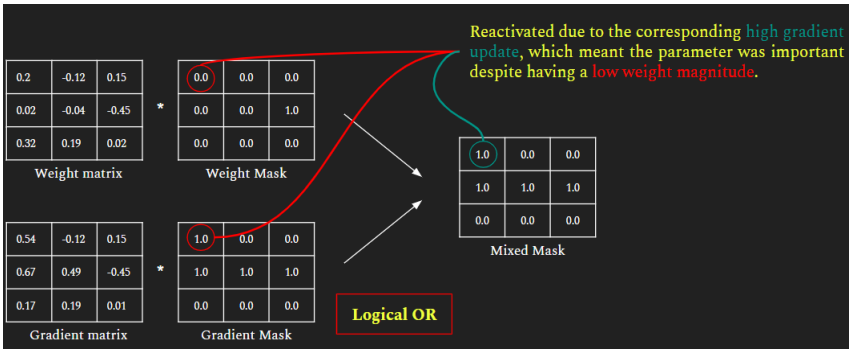


Figure 2: Obtaining the final pruning mask from the weight mask and gradient mask.

4.2 Method 1 - Explicit Connection Sensitivity pruning (ECS)

Our model can check the magnitude of connectivity with peer neuron, as we remove weights and corresponding gradients which are lesser than a pre-specified threshold. Hence, our sparsity mask is a logical *OR* between the top x% of the weights and the top x% of the gradients. Therefore, the sparsity mask would have a final sparsity spanning anywhere between $x\% - 2x\%$.

We call our approach Explicit Connection Sensitivity Pruning (ECS) since we explicitly examine the magnitudes of both weights and gradients and retain x% of each, as opposed to SNIP which implicitly looks at the impact of weights on the overall loss of the model. Algorithm 1 shown below is a formal description of our method.

Algorithm 1 Explicit Connection Sensitivity pruning (ECS)

Result: Obtaining the binary pruning mask

1. Initialize all parameters and train the model for 1 epoch.
2. **for** *every layer in model*
   weight.append(*layer.flatten*())
   gradient.append(*layer.grad.flatten*())
3. Obtain a weight mask by retaining top x% of weights in every layer.
4. Obtain a gradient mask by retaining top x% of gradients in every layer.
5. $Final\_Mask = Weight\_Mask | Gradient\_Mask$
6. Apply Pruning Mask: W = $Final\_Mask \odot W$

We leverage information from both - the weights and the gradients of a network to identify redundant parameters. Our hypothesis is that by examining the magnitude of both - the weight and its corresponding gradient, it is possible to ascertain the importance of the parameter. Weights with smaller magnitudes are relatively less important than weights with larger magnitudes, provided their corresponding gradient updates are smaller as well. In other words, a parameter with a small weight magnitude and a small gradient magnitude is deemed unimportant by the model. As shown in Fig. 2, despite being erroneously removed due to a relatively low weight magnitude, the parameter has been reactivated since its gradient update is relatively large.

Memory Leakage: We believe that the following approach may inadvertently allow redundant parameters to be a part of the final mask. Since, parameters with the high weight and gradient magnitude are passed, it is always possible for a parameter's gradient update to cancel out it's weight magnitude. Consequently, this would lead to a weight with a small magnitude to be a part of the final mask. We hope to resolve this "memory leakage" by comparing the values of the weights and corresponding gradients.

4.3 Quantization

We have performed quantization on convolution and linear layers of the trained model's weights which reduces the memory footprint and also speeds up the inference of the model by 1.5 - 2x. We

have combined k-means clustering algorithm to make the model compact and followed by 8-bit quantization to speed up the integer valued weights.
By performing quantization we have managed to reduce 98% unique variants from weight matrix. This combined with the dynamic programming can further improve performance of the model as efforts to calculate the same operations can be saved.

Algorithm 2 K-means and 8 bit Quantization algorithm

Result: Quantization Algorithm

1.Iterate through feature and classifier layers of the model to perform quantization on conv2d linear layer.

2. Obtain all the pruned weights from the layer and flatten it.

3. Perform k-means clustering algorithm on flattened weights to obtain k=5 different clusters and their centroids.

Until convergence,

$For\ every\ i\ =\ 0\ to\ k,\ set$

$c^{(i)} := arg\ min||x^{(i)} - \mu_j||^2$

4. Replace weights with the cluster centroids for every cluster c_i.

5. Perform 8 bit quantization on the updated weights.

$Q(x, scaling\_factor, zero\_point) = int((x/scaling\_factor) + zero\_point)$

6.Reshape and assign the quanitzed weights to the layer.

5 Experimental Results

We plan to evaluate our approach against SNIP, a competent benchmark proposed at ICLR 2019 [7]. SNIP is a connection-sensitivity based approach proposed in 2019, and has been extensively used as a benchmark in recent years. To allow for fair comparison, we have used the same training procedure and hyper-parameters for both SNIP and our approach. We train each network on CIFAR-10 and CIFAR-100 for 300 epochs with a batch size of 128. We use stochastic gradient descent with a lr of 0.05, momentum of 0.9 and a weight decay of 0.5 after every 30 epochs. Additionally, to prevent the model from overfitting, we use a weight decay of 5×10^{-4}.

We have demonstrated the results of deploying SNIP and our approach on networks listed below:

- ResNet34
- ResNet50
- VGG16

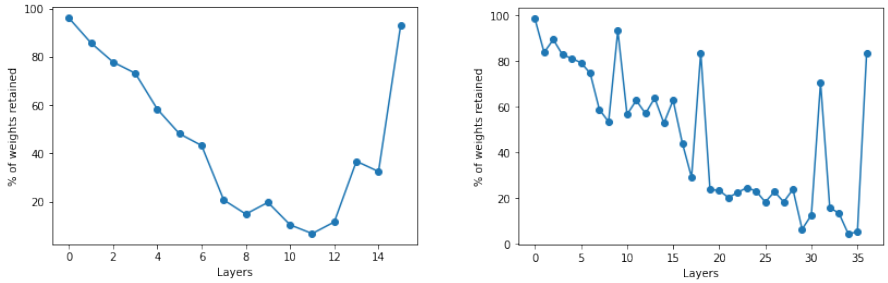
Table 1 shows the results of using the proposed approach as compared to SNIP on CIFAR-10. We run SNIP on each of the three networks at compression rates of **5x** (20%), **10x** (10%) and **20x** (5%). To ensure consistency, we have run each experiment thrice and reported the resulting mean accuracy and standard deviation.

5.1 Interpretation

5.1.1 Explicit Connection Sensitivity Pruning

As one can observe in Table 1, SNIP is an extremely competitive baseline with a marginal reduction in accuracy (< 2%) of the pruned models as compared to the reference models, even at 20x compression (5%). As expected, it is easier to obtain a higher compression rate without much reduction in accuracy on VGG16, since it has over 138M parameters as compared to ResNet34 and ResNet50, which have approximately 23M parameters each. Table 2 reports the results of SNIP and ECS pruning on CIFAR100. Similar to CIFAR10, we are able to match and even surpass SNIP on ResNet34 and ResNet50 at compression rates of 10x and 20x. There is a larger reduction in accuracy on CIFAR100 owing to the larger number of classes in the dataset (100 classes in CIFAR100 vs 10 classes in CIFAR10).

We denote our complete approach consisting of pruning and quantization steps as ECS(P + Q). ECS(P) denotes the results of the pruning portion of our approach. As shown in Table 1, our pruning approach



(a) VGG16 (b) ResNet34

Figure 3: Percentage of weights retained in each layer for VGG16 (left) and ResNet50 (right) at a compression of 10x on CIFAR10.

is almost at par with the baseline method for all the three models, across all the three compression rates. At higher compression rates of 20x, there is a difference of not more than 2% between our model and SNIP. In case of VGG16, the difference in accuracy is less than 1% between our pruning approach and SNIP.

5.1.2 Explicit Connection Sensitivity Pruning + Quantization

Further, we report the results of applying quantization on the pruned model. The quantized pruned model is denoted by ECS(P+Q). We achieved a reduction of over 90x in the number of unique parameters of the pruned model, while ensuring a marginal drop in the accuracy of the ECS(P+Q) as compared to ECS(P). For instance, the maximum drop in accuracy between ECS(P) and ECS(P+Q) is just over 2.5% for ResNet50, 1% for ResNet34, and less than 1% for VGG16 .

5.2 Analysis

5.2.1 Trends across different Architectures

In the case of both Pruning and Quantization, we observed that the difference in accuracy between the reference model and the pruned, quantized models is more pronounced in ResNet34 and ResNet50, as compared to VGG16. This is in line with our expectation that the number of redundant parameters in a larger network (more parameters) is proportionally more than the number of redundant parameters in a smaller network. Hence, since ResNet34 and ResNet50 have less than one-fifth the number of parameters in VGG16, we found that there was a higher reduction in accuracy at the same level of compression and quantization as compared to VGG16. This trend has been consistent for both pruning and quantization.

Model	Method	CIFAR-10			
		100%	20%	10%	5%
ResNet34	SNIP	94.01 ± 0.11	93.54 ± 0.28	92.81 ± 0.87	90.91 ± 0.87
	ECS (P)	–	92.13 ± 0.28	91.45 ± 0.45	89.56 ± 0.78
	ECS (P + Q)	–	91.86 ± 0.35	91.45 ± 0.45	88.45 ± 0.67
ResNet50	SNIP	95.15 ± 0.17	94.63 ± 0.21	93.27 ± 0.39	92.29 ± 0.33
	ECS (P)	–	93.59 ± 0.11	92.32 ± 0.62	90.66 ± 0.89
	ECS (P + Q)	–	92.95 ± 0.45	91.47 ± 0.37	88.21 ± 0.63
VGG16	SNIP	94.56 ± 0.21	94.01 ± 0.48	93.85 ± 0.93	93.21 ± 0.67
	ECS (P)	–	93.85 ± 0.05	93.12 ± 0.38	92.53 ± 0.71
	ECS (P + Q)	–	93.56 ± 0.55	92.78 ± 0.67	91.78 ± 0.34

Table 1: Performance of our approach (ECS) w.r.t. the baseline method (SNIP) on CIFAR-10.

Model	Method	CIFAR-100			
		100%	20%	10%	5%
ResNet34	SNIP	73.13 ± 0.13	71.72 ± 0.25	68.28 ± 0.42	65.11 ± 1.01
	ECS (P)	–	70.31 ± 0.14	69.21 ± 0.23	68.36 ± 0.57
	ECS (P + Q)	–	70.02 ± 0.13	67.86 ± 0.31	66.81 ± 0.49
ResNet50	SNIP	75.17 ± 0.32	72.12 ± 0.52	69.79 ± 0.26	66.76 ± 0.78
	ECS (P)	–	71.21 ± 0.09	70.42 ± 0.27	69.33 ± 0.54
	ECS (P + Q)	–	70.19 ± 0.18	69.88 ± 0.29	68.76 ± 0.52
VGG16	SNIP	74.52 ± 0.13	73.82 ± 0.13	72.35 ± 0.19	70.17 ± 0.56
	ECS (P)	–	73.45 ± 0.12	71.98 ± 0.34	69.12 ± 0.56
	ECS (P + Q)	–	73.12 ± 0.23	71.10 ± 0.32	68.56 ± 0.45

Table 2: Performance of our approach (ECS) w.r.t. the baseline method (SNIP) on CIFAR-100.

5.2.2 Layerwise Compression Ratios

Since, we apply a global threshold to get the weights and gradients, the individual layer-wise compression ratio will be different from the overall compression ratio. Fig. 3 shows the percentage of weights retained in each layer for VGG16 and ResNet 34 after applying the global threshold. We observed that there is a significantly larger percentage of weights retained in the initial layers as compared to the intermediate and final layers. Also, the last layer of the model is also pruned less aggressively since it determines the final outputs of the model. This is in line with our earlier prediction that since the initial layers capture more fundamental aspects of the dataset such as color, orientation and, shape they are more essential to the learning capabilities of the model.

5.2.3 Quantization Analysis

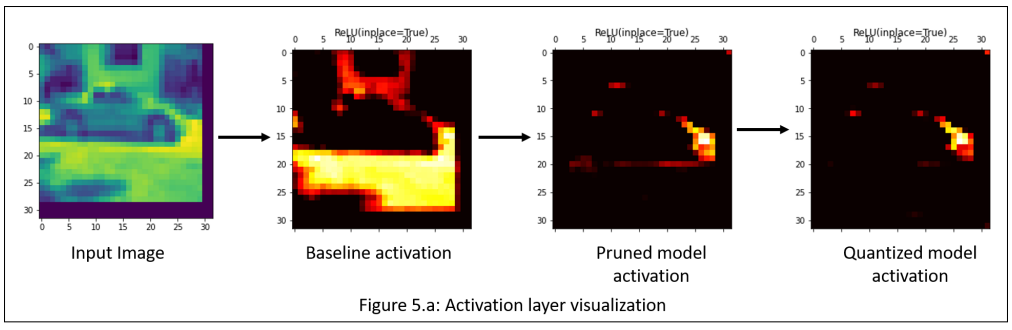


Figure 5.a: Activation layer visualization

Figure 5.a shows Activation layer visualization for baseline model, pruned model and pruned + quantized model. In the baseline model, the initial activation layer is doing most of the pattern detection which implies that the weights are not efficiently distributed. Whereas in pruned model most of the weights have became zero which makes the model detect only important details. At the end after quantizing the pruned weights the model has learned similar results as pruned but with much lesser weight variants and memory.

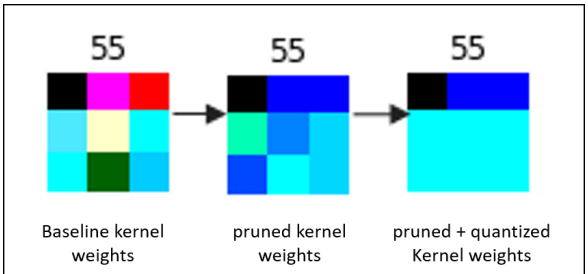


Figure 5.b) Convolution layer kernel visualization

To further explain the effect of pruning and quantization on the convolution layer, Figure 5.b shows the visualization of kernel from convolution layer. The colors from 3X3 matrix represents the unique weight values present in the kernel. As we prune the VGG16 model we set few less important weights with zeros. Post quantization, he total number of unique weights in the model are reduced by more than 50x.

#	Layer name	old # of unique weights	new # unique weights	% reduction in weights
1	Conv2d	1728	32	98.14814815
2	BatchNorm2d	64	32	50
3	Conv2d	36858	32	99.91318031
4	BatchNorm2d	64	32	50
5	Conv2d	73696	32	99.95657838
6	BatchNorm2d	128	32	75
7	Conv2d	147322	32	99.97827887
8	BatchNorm2d	128	32	75
9	Conv2d	294354	32	99.98912874
10	BatchNorm2d	256	32	87.5
11	Conv2d	587498	32	99.99455317
12	BatchNorm2d	256	32	87.5
13	Conv2d	587610	32	99.99455421
14	BatchNorm2d	256	32	87.5
15	Conv2d	1171255	32	99.99726789
16	BatchNorm2d	512	32	93.75
17	Conv2d	2324975	32	99.99862364
18	BatchNorm2d	512	32	93.75
19	Conv2d	2325805	32	99.99862413
20	BatchNorm2d	512	32	93.75
21	Conv2d	2327781	32	99.9986253
22	BatchNorm2d	512	32	93.75
23	Conv2d	2330797	32	99.99862708
24	BatchNorm2d	512	32	93.75
25	Conv2d	2328190	32	99.99862554
26	BatchNorm2d	512	32	93.75
27	Linear	261385	8298	96.82537253
28	BatchNorm1d	507	235	53.64891519
29	Linear	261957	874	99.66635746
30	BatchNorm1d	512	236	53.90625
31	Linear	5120	1318	74.2578125

#	Layer name	old # of unique weights	new # unique weights	% reduction in weights
Total		**15071574**	**11793**	**99.92175336**

Above table represents the quantization analysis on VGG16 model. Here, we have shown the reduction/compression rates of weights of the model.

6 Team Member Contributions

Arvind Subramaniam	Formulated and Coded ECS pruning layerwise pruning plots
Nitish Garikoti	Coded SNIP (baseline method) from scratch layerwise iterative variants of SNIP for all models on CIFAR10
Vinay Patil	Coded k-means quantization activation layer and scatter plots visualization
Diksha Agarwal	Trained all models on CIFAR100 Coded 8 bit quantization, kernel heatmap visualization

Table 4: Contribution of each Team Member

7 Conclusion

We performed both pruning and quantization to reduce the memory and the inference time of classical DNNs such as ResNet34, ResNet50 and VGG16 while ensuring a marginal drop in accuracy. We obtained a compression of 20x on all models across all datasets, and were also able to reduce the number of unique parameters in the pruned model by over 50x. We introduced Explicit Connection Sensitivity pruning, a robust one-shot pruning paradigm which looks at both the weights as well as the corresponding gradients to identify and remove redundant parameters in the model. Next, we applied K-means and 8-bit quantization to further reduce the memory footprint of the model, and simultaneously increase its inference speed. As an extension of our work, we hope to examine the robustness of our approach on larger datasets such as ImageNet, and also explore its applicability on Language models such as GRUs, LSTMs and Transformers.

8 References

[1] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. arXiv preprint arXiv:1907.04840, 2019.

[2] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635, 2018.

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

[4] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In Advances in neural information processing systems, pages 1135–1143, 2015.

[5] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In IEEE international conference on neural networks, pages 293–299. IEEE, 1993.

[6] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In Advances in neural information processing systems, pages 598–605, 1990.

[7] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In International Conference on Learning Representations, 2019. URL https://openreview.net/ forum?id=B1VZqjAcYX.

[8] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In Proceeding

[9] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. arXiv preprint arXiv:1902.05967, 2019

[10] Ameya Prabhu, Girish Varma, and Anoop Namboodiri. Deep expander networks: Efficient deep networks from graph theory. In Proceedings of the European Conference on Computer Vision (ECCV), pages 20–35, 2018.

[11] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866, 2014.

[12] Zhiyong Cheng, Daniel Soudry, Zexi Mao, and Zhenzhong Lan. Training binary multilayer neural networks for image classification using expectation backpropagation. arXiv preprint arXiv:1503.03562, 2015.

[13] Huizi Mao Chenzhuo Zhu, Song Han and William J. Dally. Trained ternary quantization. In ICLR, 2017

[14] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. arXiv preprint arXiv:1605.04711, 2016.

[15] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Structured binary neural networks for accurate image classification and semantic segmentation. In CVPR, 2019.

[16] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In NeurIPS, 2017.

[17] Qian Lou, Feng Guo, Minje Kim, Lantao Liu, and Lei Jiang. AUTOQ: Automated kernel-wise neural network quantization In ICLR, 2020.

[18] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In IEEE international conference on neural networks, pages 293–299. IEEE, 1993.

[19] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In Advances in neural information processing systems, pages 598–605, 1990.

[20] Alex Krizhevsky, Learning Multiple Layers of Features from Tiny Images, 2009.

[21] Subramaniam, Arvind and A. Sharma. "N2NSkip: Learning Highly Sparse Networks using Neuron-to-Neuron Skip Connections." BMVC (2020).