Associative Convolutional Layers: Supplementary Materials

Hamed Omidvar Vahideh Akhlaghi Hao Su Massimo Franceschetti Rajesh K. Gupta

1 Estimating the Cardinality of the Code Vector Space

In this appendix we make the statements in Section 3.2 more precise. We first take the 4-D DCT-II of each slice defined in Section 3.2. The 4-D DCT-II that we use, after removing the scaling factors, is stated as follows.

$$K[u,v,w,t] := \sum_{i=0}^{\hat{s}_1-1}\sum_{j=0}^{\hat{s}_2-1}\sum_{k=0}^{\hat{s}_3-1}\sum_{l=0}^{\hat{s}_4-1} k[i,j,k,l]\times \cos\left(\frac{\pi}{\hat{s}_1}\left(i+\frac{1}{2}\right)u\right)\times\cos\left(\frac{\pi}{\hat{s}_2}\left(j+\frac{1}{2}\right)v\right)\times \cos\left(\frac{\pi}{\hat{s}_3}\left(k+\frac{1}{2}\right)w\right)\times\cos\left(\frac{\pi}{\hat{s}_4}\left(l+\frac{1}{2}\right)t\right)$$

After taking the 4-D DCT-II, we then remove the elements of the slice in the transformed domain that were smaller than a threshold. We then took the inverse transform. The inverse 4-D DCT transform, after neglecting its scaling factors, can be stated as follows.

$$K[i,j,k,l] := \sum_{u=0}^{\hat{s}_1-1}\sum_{v=0}^{\hat{s}_2-1}\sum_{w=0}^{\hat{s}_3-1}\sum_{t=0}^{\hat{s}_4-1} k[u,v,w,t]\times \cos\left(\frac{\pi}{\hat{s}_1}\left(u+\frac{1}{2}\right)u\right)\times\cos\left(\frac{\pi}{\hat{s}_2}\left(v+\frac{1}{2}\right)v\right)\times \cos\left(\frac{\pi}{\hat{s}_3}\left(w+\frac{1}{2}\right)w\right)\times\cos\left(\frac{\pi}{\hat{s}_4}\left(t+\frac{1}{2}\right)t\right)$$

In order to measure the similarity between the inverse transformed version of the slice and the original slice, inspired by image compression similarity measures, we use a variation of a known measure called PSNR (Welstead, 1999) which we define as follows. Let $\hat{k^*}$ denote the inverse DCT of the pruned DCT of the slice $\hat{k}$. We re-scale the elements of the slices and their corresponding approximate version to $[0,1]$ with a bit of abuse of notation we represent the re-scaled versions with the same notations.

$$PSNR^* = 10\log\frac{1^2}{MSE}, \tag{1}$$

where

$$MSE = \frac{1}{\hat{s}_1\hat{s}_2\hat{s}_3\hat{s}_4}||\hat{k}-\hat{k}^*||_2^2. \tag{2}$$

We chose the threshold for keeping the elements in the DCT domain such that the average PSNR* is above 20dB which from image compression literature is expected to result in images that are still recognizable (see, for instance (Veldhuizen, 2010)). We then calculated the mean of the number of remaining elements in the DCT domain after the pruning step. This suggests that a code size of 20 times fewer elements than its corresponding slice would be sufficient. Based on these estimates, in most of our experiments we choose code vectors whose number of elements is 18 times smaller than that of their corresponding slices.

2 Training Convergence

In this section of the appendix we provide the proof of Theorem 2.

Proof of Theorem 2. First of all we note that since the number of weights in the convolutional layer is a polynomial function of $|\hat{\mathcal{C}}|$, it has replaced the m in Theorem 2. Now, let

$$C = \left[c_1, ..., c_{|\hat{\mathcal{C}}|}\right], \tag{3}$$

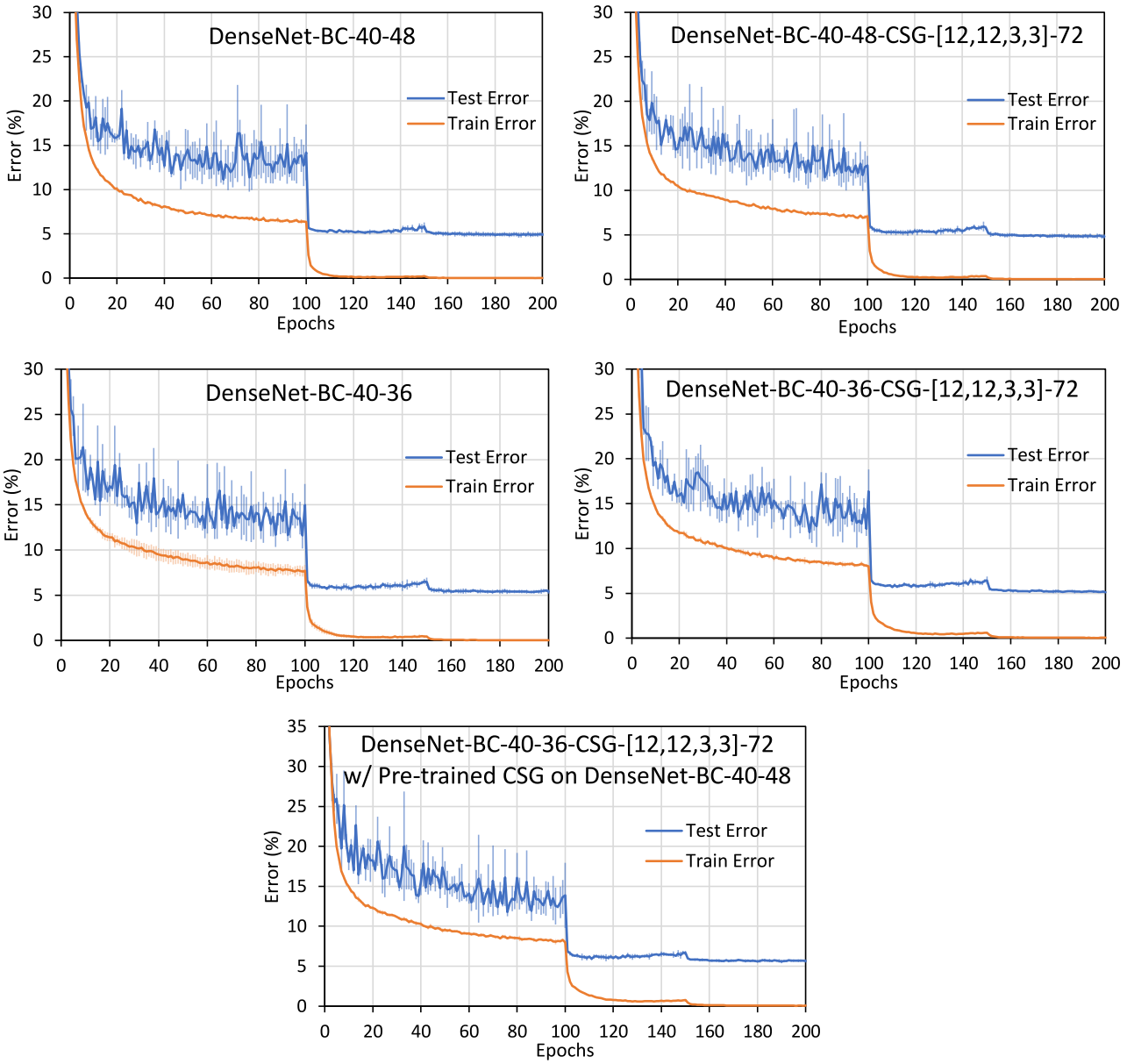

Figure 1: Training and test error for DenseNet-BC-40-48, DenseNet-BC-40-36 and their CSG-augmented versions on CIFAR-10 dataset over the course of 200 epochs with batch size of 128. For the first 100 epochs the learning rate was set to 0.05 and for the two final 50 epochs to 5×10^{-3} and 5×10^{-4}, respectively.

denote a matrix whose columns are the code vectors corresponding to the slices of the convolutional layer.

Now, instead of assuming that the convolutional filter is first generated and then it is used for the convolution operation, equivalently, using associativity, we can assume that each column of the matrix A_{CSG} denotes a

vectorized version of a slice of a convolutional filter. It means that, for each column of A_{CSG}, we need to calculate the convolution of a slice for its $|\hat{\mathcal{C}}|$ possible locations in the filter. But each of these would be an ordinary convolution with appropriate zero-paddings. Now, the matrix C can be viewed as an additional fully connected layer before the final classification layers. Hence we are dealing with a CNN with an additional fully connected layer at the final stage for which the results in (Allen-Zhu et al., 2019) and specially Theorem 1 hold. □

3 Training on CIFAR-10 Dataset

The training and test error of different models for CIFAR-10 dataset and their CSG-augmented versions reported in Table 1 can be found in figures 1 through 4.

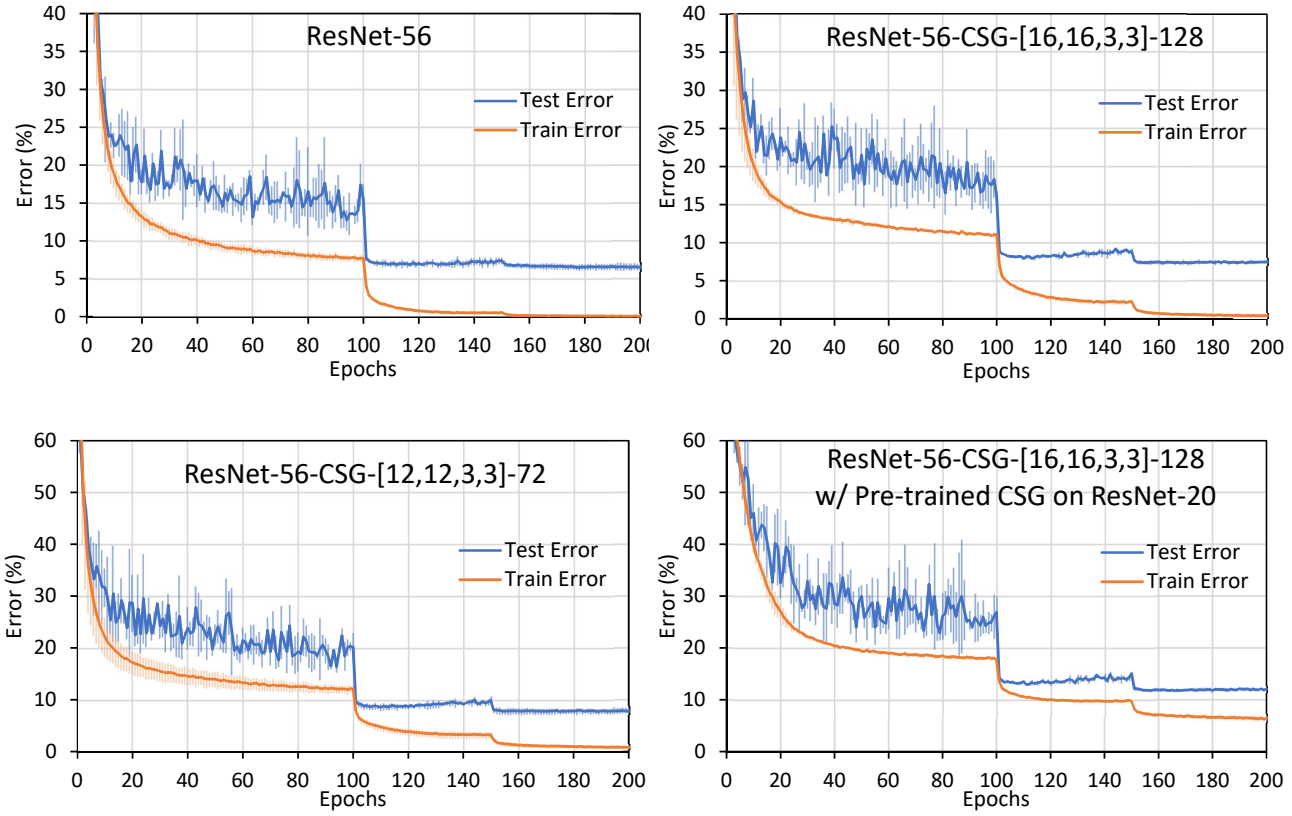

Figure 2: Training and test error for the ResNet-56 and its CSG-augmented versions reported in Table 1 over the course of 200 epochs. The batch size was set to 192 in all experiments. For the first 100 epochs the learning rate was set to 0.05 and for the two final 50 epochs it was set to 5×10^{-3} and 5×10^{-4} respectively.

3.1 Using Pre-Trained CSG

When using pre-trained CSG parameters during the training of the CSG-augmented CNNs, the number of parameters to be trained reduces to $|\hat{\mathcal{C}}| + |\hat{\mathcal{O}}|$. This can result in significant reduction in the number of the parameters of the network depending on its architecture. For ResNet-56, in the case of using fixed pre-trained parameters for the CSG that was trained alongside ResNet-20 architecture (in ResNet-20, due to the small size of the network, use of the CSG-augmented network does not result in parameter reduction, i.e., $\hat{\mathcal{G}}$ is larger than the number of parameters - training and test details of ResNet-20 are available in supplementary materials), the number of parameters reduces from about $850K$ to merely $50K$, a reduction of more than $16\times$ but at the cost of higher accuracy loss. For DenseNet-BC when $L = 40$, $K = 48$, using pre-trained CSG trained alongside DenseNet-BC with $L = 40$, $K = 36$ reduces the number of parameters from ≈ 2.7 million to ≈ 1.3 million (i.e., $2.06\times$) while also improving the accuracy. For DenseNet-BC when $L = 40$, $K = 36$, this approach (using a

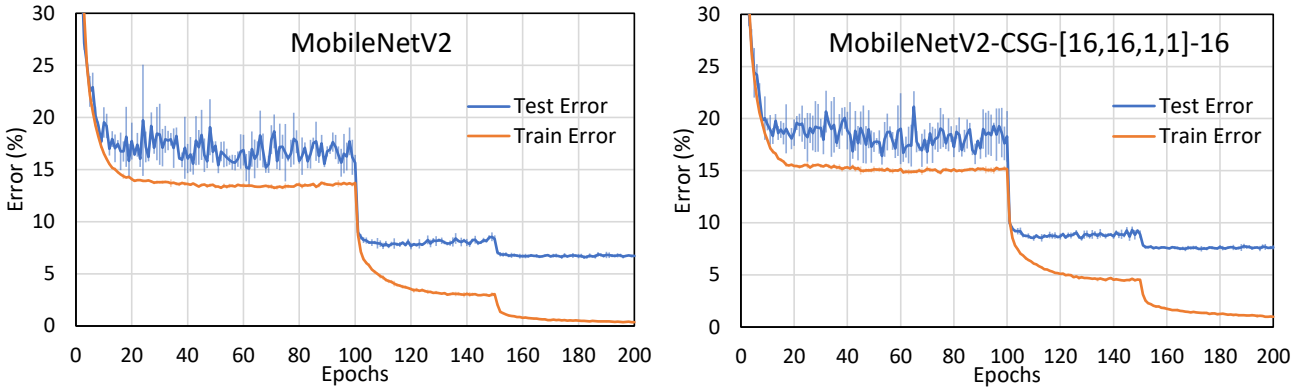

Figure 3: Training and test error for the MobileNet V2 (CIFAR Version) and its CSG-augmented versions reported in Table 1 over the course of 200 epochs. The batch size was set to 128 in all experiments. For the first 100 epochs the learning rate was set to 0.05 and for the two final 50 epochs it was set to 5×10^{-3} and 5×10^{-4} respectively.

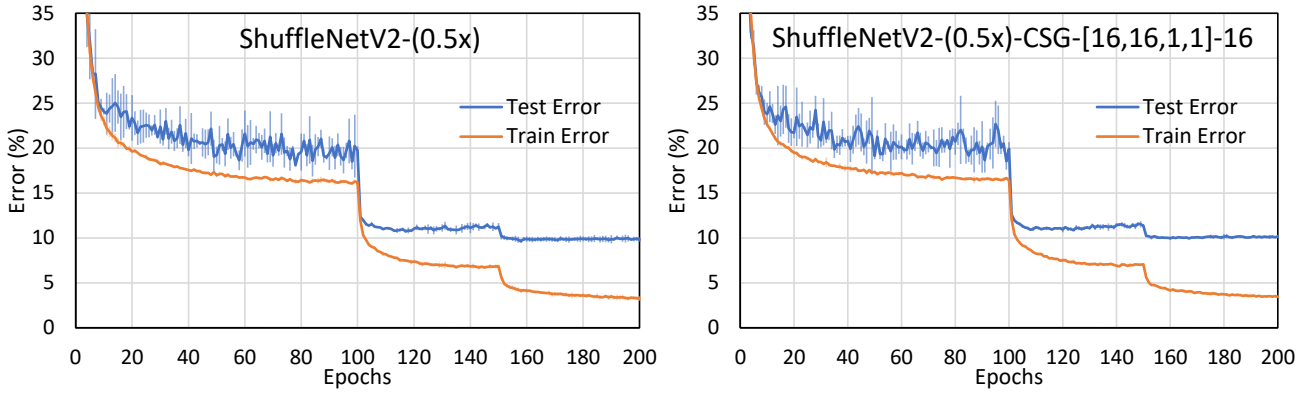

Figure 4: Training and test error for the ShuffleNet V2 (0.5x) (CIFAR Version) and its CSG-augmented versions reported in Table 1 over the course of 200 epochs. The batch size was set to 128 in all experiments. For the first 100 epochs the learning rate was set to 0.05 and for the two final 50 epochs it was set to 5×10^{-3} and 5×10^{-4} respectively.

pre-trained CSG obtained from DenseNet-BC with $L = 40$, $K = 48$) reduces the number of parameters from ≈ 1.5 million to ≈ 0.75 million with only a slight degradation in the accuracy.

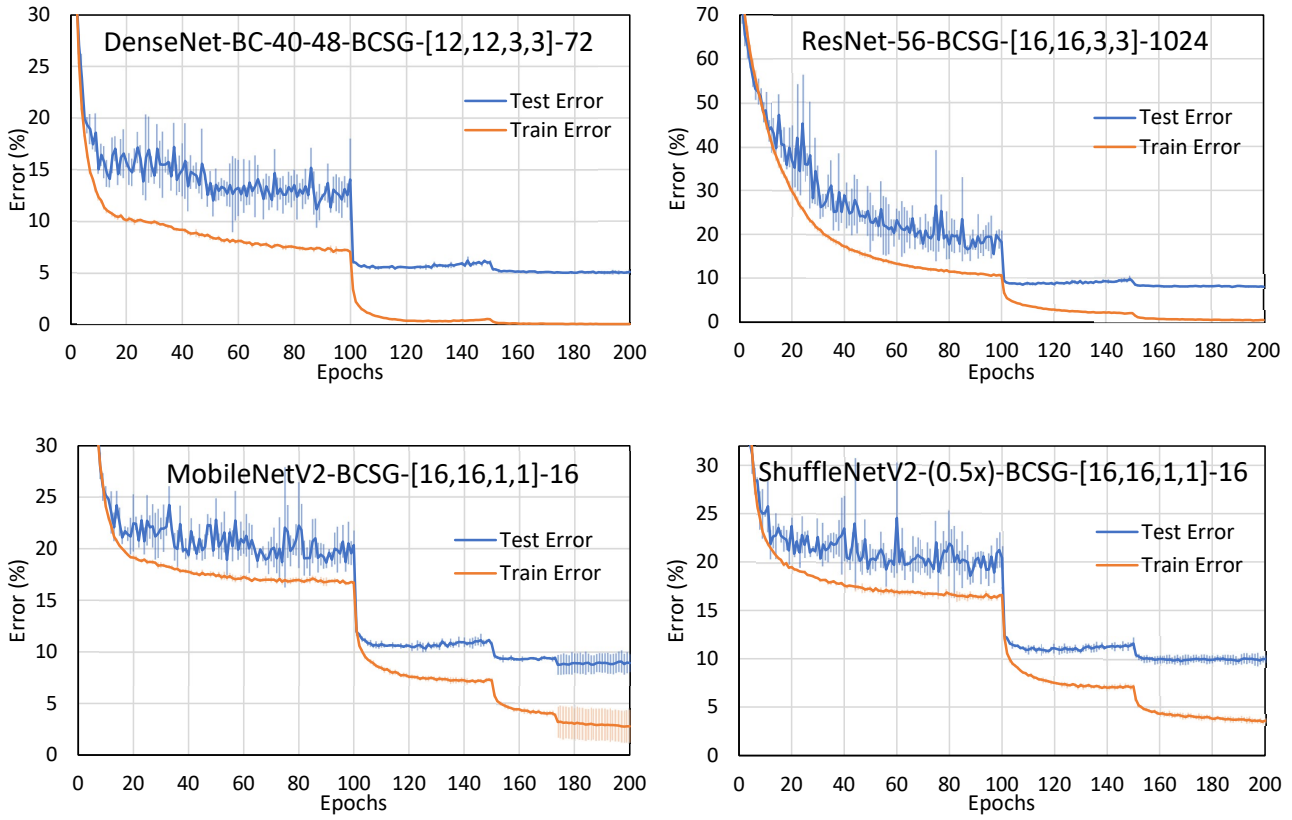

Figure 5: Training and test error for the BCSG-augmented versions of various models for CIFAR-10 dataset over the course of 200 epochs. The batch size was set to 128 in all experiments. For the first 100 epochs the learning rate was set to 0.05 and for the two final 50 epochs it was set to 5×10^{-3} and 5×10^{-4} respectively.

3.2 Trying Different Slice Shapes

In this section we provide a series of results for various shapes of the slice used in our CSG-augmented networks. As we can see in Table 1 the choices we have by estimating the cardinality of the code vector space explained above provide a very good trade-off between compression and accuracy for the shape of the slices.

Table 1: Training results on CIFAR-10 dataset for different slice shapes. In the "Test Err.' column the best test errors among all epochs and on the test error at epoch 200 with the format "Average ± Std." are reported.

Network Architecture	# Param.	Test Err. (%)	Ratio
DenseNet-BC-40-48-CSG-[12,12,3,3]-72	1,416,394	4.83 ± 0.24	1.92×
DenseNet-BC-40-48-CSG-[6,6,3,3]-72	1,595,242	4.77 ± 0.08	1.71×
DenseNet-BC-40-48-CSG-[24,24,3,3]-72	1,634,122	4.84 ± 0.37	1.67×
ResNet-56-CSG-[16,16,3,3]-128	347,162	7.24 ± 0.11	2.45×
ResNet-56-CSG-[8,8,3,3]-128	267,290	7.56 ± 0.31	3.19×
ResNet-56-CSG-[32,32,3,3]-128	1,179,648	6.68 ± 0.09	0.71×
ResNet-56-CSG-[12,12,3,3]-72	160,450	8.01 ± 0.27	5.31×
ResNet-56-CSG-[6,6,3,3]-72	238,354	7.74 ± 0.13	3.58×
ResNet-56-CSG-[24,24,3,3]-72	396,178	7.27 ± 0.24	2.15×

Table 2: Training time, inference time and model sizes (Mega Bytes (MB)) and host to device model data transfer. Training and inference time for each epoch are reported in the following format: "mean" ± "standard deviation" over all 200 epochs. Model size reports the size of the PyTorch model in MB. "H to D" column reports the amount of model related data transmitted from the host (main memory) to the device (GPU memory) collected using the 'nvprof' tool in MB. DN and RN stand for DenseNet and ResNet.

Network Architecture	Train Time	Test Time	Size	H to D
DN-BC-40-48 (Original)	68.30s ± 0.55s	4.40s ± 0.06s	10.50	11.97
DN-BC-40-48-CSG-[12,12,3,3]-72	67.53s ± 0.51s	4.33s ± 0.04s	5.52	6.69
DN-BC-40-36 (Original)	51.87s ± 0.43s	3.33s ± 0.03s	6.34	7.18
DN-BC-40-36-CSG-[12,12,3,3]-72	51.58s ± 0.40s	3.31s ± 0.04s	3.31	4.38
RN-56 (Original)	14.51s ± 0.32s	1.11s ± 0.02s	3.33	4.30
RN-56-CSG-[16,16,3,3]-128	14.74s ± 0.27s	1.30s ± 0.03s	1.40	2.28
RN-56-CSG-[12,12,3,3]-72	14.97s ± 0.29s	1.40s ± 0.06s	0.67	1.53

3.3 Using Binary CSG

When using random and fixed binary CSG parameters during the training of the CSG-augmented CNNs, the number of parameters to be trained again reduces to $|\hat{\mathcal{C}}| + |\hat{\mathcal{O}}|$. This can result in significant reduction in the number of the parameters of the network depending on its architecture for small networks. The summary of our results are found in Table 1 in the paper. In Fig. 5 in this appendix the training course of the Binary CSG-augmented versions of DenseNet, ResNet, MobileNet, and ShuffleNet are plotted.

3.4 End-to-End Timings

We evaluated the training and inference time of CSG-augmented CNNs on a single GPU for different CNN models, and compared it with the baseline ones. We measured the execution time of training and inference stages on the entire train and test datasets. The average epoch time for each network is summarized in Table 2. The results show that for DenseNet, both inference time and training time are slightly improved in the CSG-augmented models compared to the baseline. For ResNet-56, execution times reported for the CSG-augmented model are slightly higher than the baseline model. The results indicate that although CSG is added to each convolutional layer in CSG-augmented CNNs, the execution time on a single GPU remains almost the same. The reason is that in CSG-augmented networks, due to the reduced number of parameters, costly memory accesses like DRAM accesses are decreased; thus, the communication and memory accesses cost are reduced. Therefore, the cost of more computation performed in CSG-augmented CNN models (i.e., additional operations related to matrix multiplications in CSG) does not increase the execution time. This results in a slight improvement of timing in DenseNet models, however, in case of ResNet, due to its smaller size, the lower memory access cost does not fully compensate the additional cost of CSG. We expect the execution time to be improved on specialized hardware architectures such as Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs), mostly used in edge devices. Since, in these hardware architectures, the required off-chip memory bandwidth for the model parameters are relatively high compared to other data such as input/output feature maps (Chen et al., 2014), and the energy and time of memory accesses are higher than operations (Chen et al., 2016), both inference time and energy will be improved by lowering the number of model parameters. In contrast, on a GPU, because of data parallelization, the intermediate results such as feature maps for different input images consume a more significant part of the on-chip and off-chip memory bandwidth. Therefore, in these devices reducing the size of model does not improve the inference time significantly. In addition to evaluating the timing, we report the size of the model related parameters that are transferred from the main memory of the CPU (Host) to DRAM of the GPU (Device) (i.e., Host to Device ("H to D" column in Table 2)) for the inference. The numbers, extracted from an Nvidia profiling tool (nvprof) show that with CSG-augmented CNNs, the required communication bandwidth between host and device memories to transfer the model is reduced by, on average, 2.03×, compared to the baseline models.

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. *International Conference on Machine Learning*, 2019.

Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 269–284, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2305-5. doi: 10.1145/2541940.2541967. URL `http://doi.acm.org/10.1145/2541940.2541967`.

Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 367–379, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-4673-8947-1. doi: 10.1109/ISCA.2016.40. URL `https://doi.org/10.1109/ISCA.2016.40`.

Todd Veldhuizen. Measures of image quality. *CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision*, 2010.

Stephen T Welstead. *Fractal and wavelet image compression techniques*. SPIE Optical Engineering Press Bellingham, Washington, 1999.