A. Appendix

A.1. Computational Cost of Compression

We justify our claim that our network compression functions $f(\boldsymbol{\omega}, \gamma)$ can be computed in real-time. Consider a network layer with weights $\boldsymbol{\omega}_l$. The computational cost of a forward pass is $\mathcal{O}(|\boldsymbol{\omega}_l|L)$, where L is the spatial dimension (in the case of a 2D convolution, $L = HW$, where H, W are the height and width of the output buffer). In the LCS+L algorithm, the cost of computing $\boldsymbol{\omega}^*$ is $\mathcal{O}(|\boldsymbol{\omega}_l|)$, which is far less than the cost of a forward pass. In the case of LCS+P, no computation is needed. Similarly, the cost of computing $\gamma(\alpha)$ is $O(1)$.

Now, consider the cost of computing $f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha))$. Our unstructured sparsity method takes $\mathcal{O}(|\boldsymbol{\omega}_l|)$ for each layer (to calculate the threshold, then discard parameters below the threshold). Our structured sparsity method takes $\mathcal{O}(1)$ for each layer, since we only need to mark each layer with the number of filters to ignore. Since most inference libraries support tensor slicing operations, we can simply pass a subset of the filters to the underlying matrix multiplication or convolution. Our quantization method [12] takes $\mathcal{O}(|\boldsymbol{\omega}_l|)$ for each layer (to calculate the affine transform parameters and apply them). In all cases, the complexity of computing $f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha))$ is at least L times lower than the cost of the convolutional forward passes, meaning our compression methods can be considered real-time.

A.2. Overhead Calculation

Unstructured Sparsity: In this setting, we prune individual network weights. As such, the maximum number of compressed network configurations is determined by the layer with the largest number of parameters. Let L denote the number of parameters in a given model's largest layer and $[0, s]$ a compression sparsity range. Then the maximum number of configurations is given by $\lfloor Ls \rfloor$. In this setting, the number of pre-calibrated BatchNorm parameters that we need to store do not vary for each configuration. Consequently, the total number of additional parameters that must be stored is $B(L-1)$ where B is the total number of BatchNorm parameters in the uncompressed model (note that we subtract 1 from L since storing an uncompressed model requires storing one set of BatchNorm parameters anyway). Hence, enabling compression at every possible configuration would incur a total overhead of

$$\left(\frac{100B(L-1)}{T}\right)\%, \tag{1}$$

where T is the total number of model parameters.

Structured Sparsity: Let M denote the width of the widest layer of a given model, $[w, 1]$ a compression width factor range, and B the total number of BatchNorm parameters. Then $L = \lfloor M(1-w) \rfloor$ gives the maximum number of compressed network configurations obtainable after channel pruning. For each $\ell \in \{1, \ldots, L\}$, pruning ℓ channels from the widest layer corresponds to compressing the network with a width factor of $1 - \ell/M$. Hence, for each compressed network configuration, storing pre-calibrated BatchNorm statistics would require storing an additional $B(1-\ell/M)$ parameters. To enable compression at every possible configuration would therefore require storing a total of $\sum_{\ell=1}^{L} B(1-\ell/M)$ additional parameters, incurring a total overhead of

$$\left(\frac{100B}{T}\sum_{\ell=1}^{L}(1-\ell/M)\right)\%, \tag{2}$$

where T is the total number of model parameters.

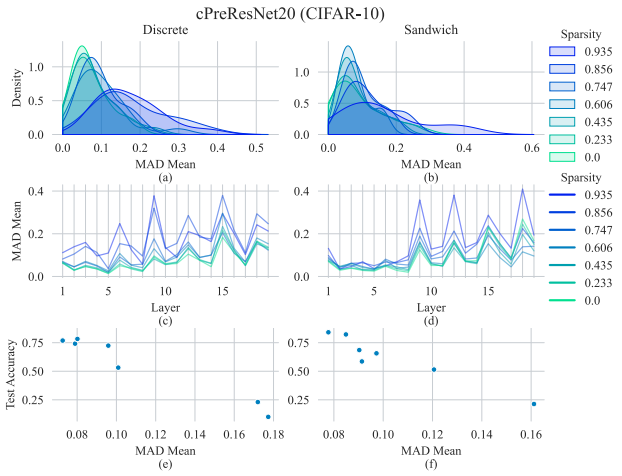

Figure 8: Analysis of the mean absolute difference between observed batch-wise means $\hat{\boldsymbol{\mu}}$ and stored BatchNorm means $\boldsymbol{\mu}$ during testing for cPreResNet models trained with $Discrete$ and $Sandwich$. (a)-(b): The distribution of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ across all layers. (c)-(d): The average value of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ for each individual BatchNorm layer. (e)-(f): The correlation between the average of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ and test set error.

A.3. Further BatchNorm Analysis

In Figure 3, we analyzed the inaccuracies of BatchNorm statistics for models trained with unstructured sparsity. We show a similar analysis for the case of $Sandwich$ and $Discrete$ in Figure 8. We show the case of quantization in Figure 9.

A.4. Linear Subspace Analysis

In Figure 10, we provide additional experimental evidence that our linear subspace method (LCS+L) trains a subspace specialized for high-accuracy at one end and high-efficiency at the other end. We plot the validation accuracy along our subspace, as well as the validation accuracy along

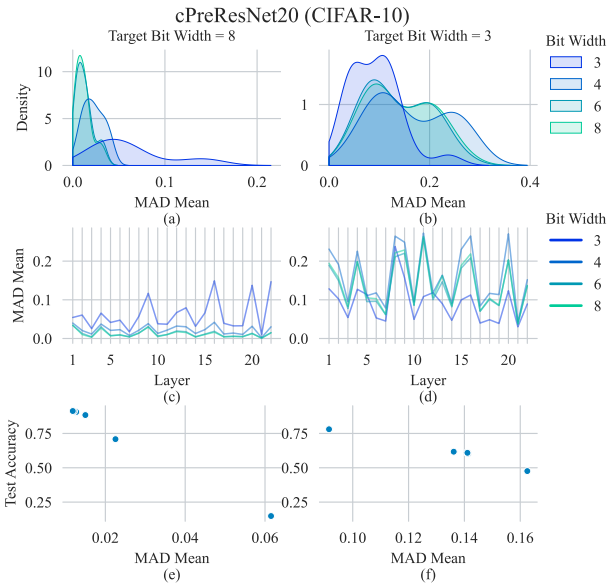

Figure 9: Analysis of the mean absolute difference between observed batch-wise means $\hat{\boldsymbol{\mu}}$ and stored BatchNorm means $\boldsymbol{\mu}$ during testing for cPreResNet models trained with different quantization bit widths. (a)-(b): The distribution of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ across all layers. (c)-(d): The average value of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ for each individual BatchNorm layer. (e)-(f): The correlation between the average of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ and test set error.

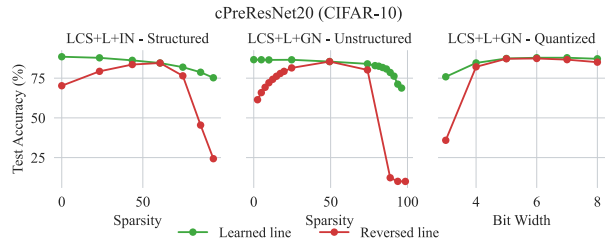

Figure 10: Standard evaluation of a linear subspace with network $f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha))$ (Learned line), and evaluation when evaluating with reversed compression levels, $f(\boldsymbol{\omega}^*(\alpha), \gamma(1-\alpha))$ (Reversed line).

our subspace when compressing with $\tilde{f}(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha)) \equiv f(\boldsymbol{\omega}^*(\alpha), \gamma(1-\alpha))$. In other words, the weights that were trained for low compression levels are evaluated with high compression levels, and vice versa. We see that this leads to a large drop in accuracy, confirming that our method has conditioned one side of the line to achieve high accuracy at high sparsities, and the other side of the line to achieve high accuracy at low sparsities.

Table 3: Our baseline models' (with BatchNorm) accuracies. cPreResNet20 is trained on CIFAR-10 and all other models on ImageNet.

Model	BatchNorm Baseline Accuracy (%)
cPreResNet20	91.69
ResNet18	70.72
VGG19	62.21
MnasNet-B1	72.58
MobileNetV2	70.03
MobileNetV3-Small	66.5
MobileNetV3-Large	73.09
DeiT-Ti	72.7
DeiT-Ti + Distillation	73.1
DeiT-S	80.3
CaiT-XXS	76.02

A.5. Global Model Details

Our CNNs warm up to an initial learning rate of 0.1 (0.045 for MobileNetV2) over 5 epochs, which then decays to 0 over 85 epochs (or 195 for cPreResNet20) using a cosine schedule. We use a batch size of 128 on a single GPU for cPreResNet20 and ResNet18. We use the version of VGG19 provided by [16]. This implementation modifies VGG19 slightly by adding BatchNorm layers and removing the last two fully connected layers. For VGG19, we use a batch size of 256 with 4 GPUs. We train MnasNet, MobileNetV2, MobileNetV3-Small, and MobileNetV3-Large with a batch size of 128 using two GPUs. We train transformer models using a batch size of 1024 with 8 GPUs. For cPreResNet20, VGG19, and ResNet18, we use a weight decay of 5×10^{-4}. For MobileNetV2, we use a weight decay of 4×10^{-5}, and 10^{-5} for MnasNet, MobileNetV3-Small, and MobileNetV3-Large.

A.6. Unstructured Sparsity Details

It is typical to include a warmup phase when training models with TopK sparsity [32]. In our baselines in Section 4.1, we increase the sparsity level from 0% to its final value over the first 80% of training epochs. For our method, sparsity values fall within a range, so there is no single target sparsity value to warm up to.

For our point method (LCS+P), we simply train for the first 80% of training with the lowest sparsity value in our sparsity range. We finish training by sampling uniformly between the lowest and highest sparsity levels.

For our line method (LCS+L), our choice of sparsity level is tied to our choice of weight-space parameters through α. We implement our warmup by simply adjusting $\gamma(\alpha)$ to apply less sparsity early in training, warming up to our final sparsity rates over the first 80% of training.

For transformer models in particular, our LCS+P train-

ing resembles our LCS+L method whereby $\gamma(\alpha)$ applies less sparsity early in training and gradually warms up to our final sparsity rates over a fraction of the training epochs. Moreover, we do not apply sparsity to the patch embedding layer.

In detail, let $\alpha_{\min}$ and $\alpha_{\max}$ correspond to our minimum and maximum alpha values (for example, for ResNet18 in Section 4.1, $\alpha_{\min} = 0.005$, and $\alpha_{\max} = 0.05$). As motivated in Section 3.3, we bias sampling of $\boldsymbol{\alpha}$ towards the endpoints of our line. We set $\boldsymbol{\alpha} = [\alpha_{\min}]$ with 25% probability, $\boldsymbol{\alpha} = [\alpha_{\max}]$ with 25% probability, and $\boldsymbol{\alpha} = [U(\alpha_{\min}, \alpha_{\max})]$ with 50% probability, where $U(a, b)$ samples uniformly in the range $[a, b]$. To warm up our sparsity rates, we choose

$$d = \max(1 - c/t, 0) \tag{3}$$

$$\gamma(\alpha) = (1 - \alpha)(1 - d), \tag{4}$$

where c is the current iteration number, and t is the total number of iterations in the first 80% of training. At the beginning of training, $d = 1$, and $\gamma(\alpha) = 0$, corresponding to a sparsity level of 0 for all values of α. Once 80% of training is finished, $d = 0$, and $\gamma(\alpha) = 1 - \alpha$ for the remainder of training. This corresponds to our final sparsity range.

Our methods use GroupNorm in the unstructured setting. For cPreResNet20, ResNet18, and VGG19, we set the number of groups to $g = 32$ (we set $g = c$ for the first few layers of cPreResNet20, because it has fewer than 32 channels). For MnasNet, MobileNetV2, MobileNetV3-Small, and MobileNetV3-Large, we use $g = 8$. Transformer models use LayerNorm (equivalent to $g = 1$).

A.7. Structured Sparsity Details

When training our method with a line in the structured sparsity setting, we do not use two sets of weights (e.g., $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$, Section 3.1) for convolutional filters. Instead, we only use two sets of weights for affine transforms in GroupNorm [28] layers. For the convolutional filters, we instead use a single set of weights, similar to our point formulation (and similar to US [30] and NS [31]). By contrast, we use two sets of weights for convolutional filters as well as for affine transforms when experimenting with unstructured sparsity (Section 4.1) and quantization (Section 4.3).

The reason for only using one set of convolutional filters in the structured sparsity setting is that the filters themselves are able to specialize, even without an extra copy of network weights. Some filters are only used in larger networks, so they can learn to identify different signals than the filters used in all subnetworks. Note that this filter specialization argument does not apply to our unstructured or quantized settings.

In preliminary experiments, we found that using a single set of weights for convolutions in our structured sparsity experiments gave a slight improvement over using two sets of weights (roughly 2% for cPreResNet20 [7] on CIFAR-10 [14]). We hypothesize that this slight difference may be attributed to the ease of learning fewer network parameters.

A.8. Quantization Details

Our method applied to quantization trains without quantizing the activations for the first 80% of training, and then adds activation quantization for the remainder of training. Weights are quantized throughout training.

The number of groups in our GroupNorm layers is the same as described in Appendix A.6.

A.9. Additional Results

Unstructured Sparsity: We present additional results in the unstructured wide sparsity regime in Figure 11 and Figure 12. For MnasNet, our α range is $[0.0325, 1]$. For the remaining models, our α range is $[0.025, 1]$. All other training details are unchanged. We find that our method is able to produce a higher accuracy over a wider range of sparsities than our baselines.

We also present results for vision transformer models in the high sparsity regime in Figure 13. For these models, our α range is $[0.05, 0.2]$. Additionally, for these models we use a 65% warm-up phase instead of 80% as was discussed in Appendix A.6.

We also provide a table of unstructured sparsity results in the high sparsity regime (Table 4) and the wider sparsity regime (Table 5), showing memory usage and FLOPS.

Additional results on lightweight networks in the structured sparsity setting are presented in Figure 15 and Figure 14. Our width factor ranges for MnasNet, MobileNetV2, MobileNetV3-Small, and MobileNetV3-Large are $[0.5, 1]$, $[0.55, 1]$, $[0.57, 1]$, and $[0.4427, 1]$, respectively. Our method yields a better accuracy-efficiency trade-off for a wider range of sparsity levels.

Quantization: We present quantization results for ResNet18 in Figure 16. We find that our models approach the accuracy of models trained at a single bit width, and our models generalize better to other bit widths.

We also provide a table of quantization results in Table 6, showing memory usage of the models.

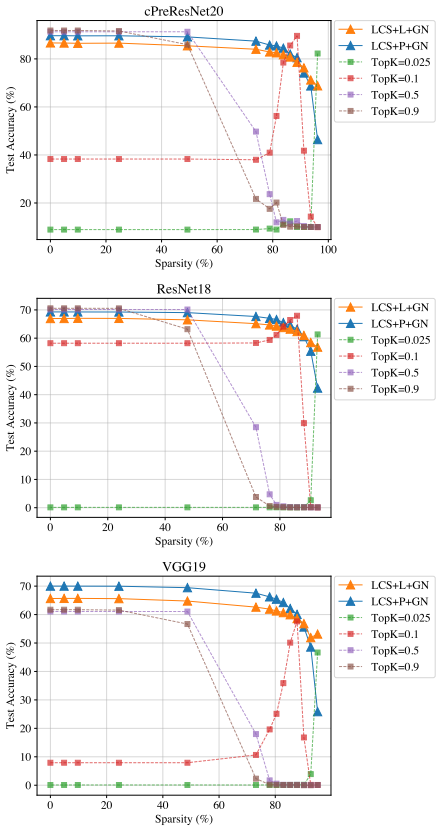

Figure 11: Our method for unstructured sparsity using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) compared to networks trained for a particular TopK target.

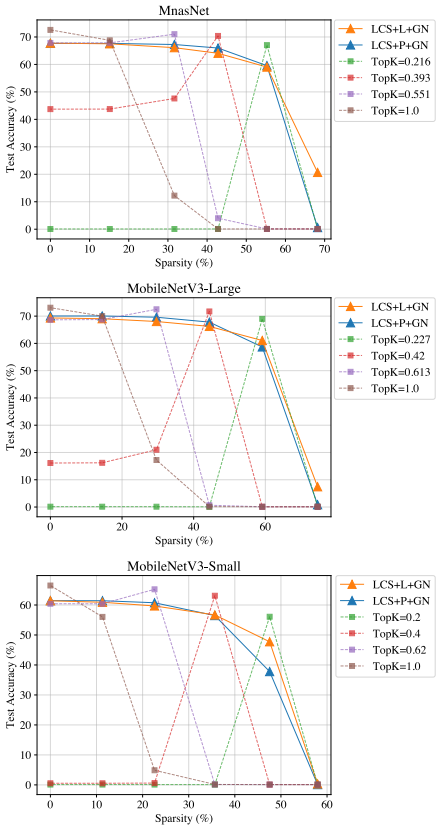

Figure 12: Our method for unstructured sparsity using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) compared to networks trained for a particular TopK target.

Table 4: Results for unstructured sparsity in the high sparsity regime. Note that models of a particular architecture and sparsity level all have the same runtime characteristics (memory and FLOPS), so we only report one value. Runtime was not measured because it requires specialized hardware. So, we follow the standard practice of only reporting memory and flops. Memory consumption refers to the size of *nonzero* model weights in the currently executing model.

cPreResNet20 (CIFAR-10)	Sparsity (%)	95.66	96.15	96.64	97.14	97.63	98.12
	FLOPS ($\times 10^6$)	1.46	1.29	1.13	0.96	0.79	0.63
	Memory (MB)	0.04	0.03	0.03	0.02	0.02	0.02
	Acc (LCS+P+GN)	70.18	69.22	67.74	**63.78**	54.91	24.92
	Acc (LCS+L+GN)	**75.53**	**72.30**	67.02	52.86	39.63	34.12
	Acc (TopK=0.04)	14.83	12.83	10.8	9.66	10.01	9.79
	Acc (TopK=0.02)	10.25	10.53	**78.7**	10.56	10.05	10.09
	Acc (TopK=0.01)	10.02	9.97	9.96	10.64	**59.2**	10.79
	Acc (TopK=0.005)	10.0	10.08	9.81	10.03	10.0	**41.44**
ResNet18 (ImageNet)	Sparsity (%)	92.67	93.15	93.62	94.1	94.58	95.06
	FLOPS ($\times 10^6$)	169.42	160.94	152.46	143.98	135.51	127.03
	Memory (MB)	3.42	3.2	2.98	2.76	2.53	2.31
	Acc (LCS+P+GN)	51.5	51.1	50.37	48.62	44.8	30.69
	Acc (LCS+L+GN)	**58.63**	**56.96**	54.70	**51.02**	44.87	39.41
	Acc (TopK=0.04)	5.96	0.9	0.18	0.11	0.1	0.1
	Acc (TopK=0.02)	24.66	45.37	**59.92**	3.84	0.1	0.11
	Acc (TopK=0.01)	0.12	0.1	0.1	0.11	**53.95**	0.11
	Acc (TopK=0.005)	0.1	0.12	0.11	0.1	0.11	**46.35**

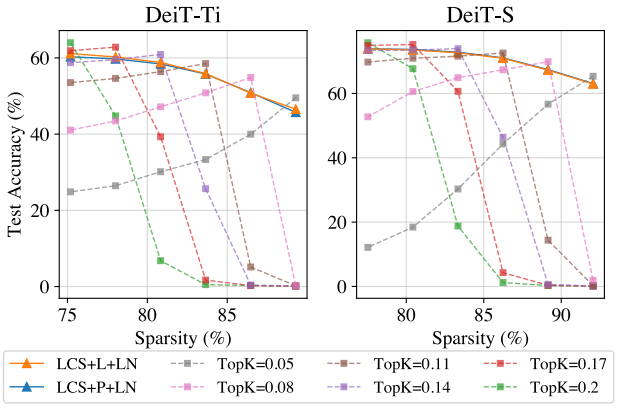

Figure 13: Our method for unstructured sparsity using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) compared to networks trained for a particular TopK target.

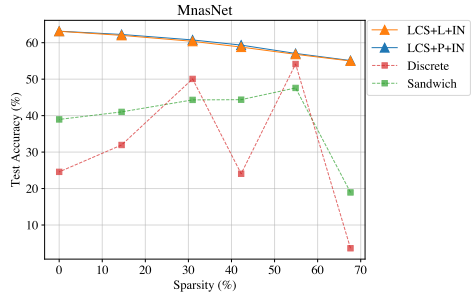

Figure 14: Our method for structured sparsity using a linear subspace (LCS+L+IN) and a point subspace (LCS+P+IN), compared to *Sandwich* and *Discrete*.

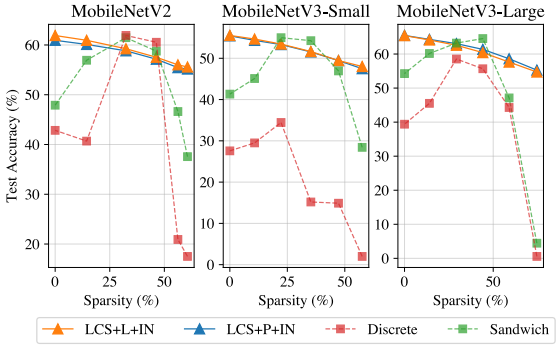

Figure 15: Our method for structured sparsity using a linear subspace (LCS+L+IN) and a point subspace (LCS+P+IN), compared to *Sandwich* and *Discrete*.

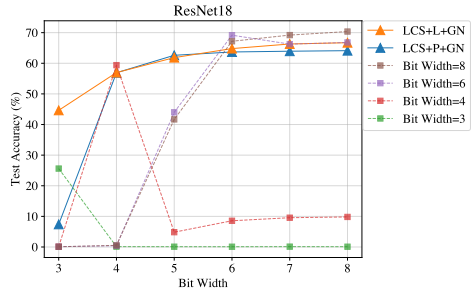

Figure 16: Our method for quantization using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) compared to networks trained for a particular bit width target.

Table 5: Results for unstructured sparsity in the wide sparsity regime. Note that models of a particular architecture and sparsity level all have the same runtime characteristics (memory and FLOPS), so we only report one value. Runtime was not measured, because it requires specialized hardware (so most unstructured pruning works report memory and flops). Memory consumption refers to the size of *nonzero* model weights in the currently executing model.

cPreResNet20 (CIFAR-10)	Sparsity (%)	0.0	49.31	86.29	91.22	93.68	96.15
	FLOPS ($\times 10^6$)	33.75	17.11	4.62	2.96	2.13	1.29
	Memory (MB)	0.87	0.44	0.12	0.08	0.05	0.03
	Acc (LCS+P+GN)	89.64	89.34	**82.34**	75.24	67.55	47.1
	Acc (LCS+L+GN)	86.65	85.45	80.78	**76.27**	**71.22**	68.78
	Acc (TopK=0.9)	**91.66**	83.17	10.54	10.0	9.76	10.0
	Acc (TopK=0.5)	91.16	**91.17**	10.64	10.14	10.0	10.0
	Acc (TopK=0.1)	40.74	40.74	78.56	39.54	11.67	10.26
	Acc (TopK=0.025)	9.64	9.64	10.65	10.0	9.98	**82.15**
ResNet18 (ImageNet)	Sparsity (%)	0.0	47.77	83.59	88.37	90.76	93.15
	FLOPS ($\times 10^6$)	1814.1	966.32	330.49	245.72	203.33	160.94
	Memory (MB)	46.72	24.4	7.66	5.43	4.32	3.2
	Acc (LCS+P+GN)	69.25	69.12	64.53	60.36	55.58	41.48
	Acc (LCS+L+GN)	66.94	66.49	63.20	**60.96**	**58.44**	56.83
	Acc (TopK=0.9)	**70.57**	63.17	0.1	0.1	0.1	0.1
	Acc (TopK=0.5)	70.15	**70.15**	0.24	0.17	0.1	0.12
	Acc (TopK=0.1)	58.21	58.21	**66.44**	29.93	0.24	0.1
	Acc (TopK=0.025)	0.12	0.12	0.13	0.17	2.64	**61.33**
VGG19 (ImageNet)	Sparsity (%)	0.0	48.75	85.31	90.19	92.62	95.06
	FLOPS ($\times 10^6$)	19533.52	9822.65	2539.51	1568.42	1082.88	597.34
	Memory (MB)	82.12	42.09	12.06	8.06	6.06	4.06
	Acc (LCS+P+GN)	**70.0**	**69.45**	**62.11**	55.62	48.58	25.87
	Acc (LCS+L+GN)	65.64	64.76	59.93	**56.80**	**51.89**	**53.15**
	Acc (TopK=0.9)	61.72	56.67	0.1	0.0	0.0	0.1
	Acc (TopK=0.5)	61.07	61.07	0.15	0.09	0.06	0.1
	Acc (TopK=0.1)	7.91	7.91	50.13	16.8	0.13	0.09
	Acc (TopK=0.025)	0.09	0.09	0.1	0.16	3.91	46.66

Table 6: Results for quantization. Note that models of a particular architecture and quantization bit width all use the same memory, so we only report one value. Runtime was not measured, because it requires specialized hardware. Memory consumption refers to the size of model weights in the currently executing model.

cPreResNet20 (CIFAR-10)	Bit Widths	8	7	6	5	4	3
	Memory (MB)	0.22	0.19	0.17	0.14	0.11	0.08
	Acc (LCS+P+GN)	89.97	90.0	89.88	**89.26**	86.25	65.26
	Acc (LCS+L+GN)	87.20	87.86	87.86	87.40	84.59	**75.86**
	Acc (Bit Width=8)	**91.36**	**91.02**	90.47	87.98	65.91	16.65
	Acc (Bit Width=6)	91.07	90.89	**91.26**	87.12	63.1	18.78
	Acc (Bit Width=4)	84.93	84.65	84.77	82.37	**88.22**	25.19
	Acc (Bit Width=3)	55.71	55.56	57.01	55.66	44.83	73.89
ResNet18 (ImageNet)	Memory (MB)	11.69	10.23	8.77	7.31	5.84	4.38
	Acc (LCS+P+GN)	63.59	63.51	63.15	**61.82**	55.89	5.48
	Acc (LCS+L+GN)	66.72	66.30	64.80	61.84	56.96	**44.63**
	Acc (Bit Width=8)	**70.36**	**69.2**	67.18	41.67	0.54	0.08
	Acc (Bit Width=6)	66.8	66.26	**69.17**	44.0	0.43	0.1
	Acc (Bit Width=4)	9.82	9.57	8.57	4.84	**59.39**	0.1
	Acc (Bit Width=3)	0.1	0.12	0.09	0.1	0.12	25.61
VGG19 (ImageNet)	Memory (MB)	20.542	17.974	15.406	12.839	10.271	7.703
	Acc (LCS+P+GN)	57.83	57.74	57.24	55.64	47.11	0.25
	Acc (LCS+L+GN)	**64.38**	**63.89**	**61.95**	**57.72**	50.28	**31.85**
	Acc (Bit Width=8)	60.85	59.32	56.13	39.73	0.3	0.09
	Acc (Bit Width=6)	56.52	55.35	59.89	23.24	0.16	0.1
	Acc (Bit Width=4)	0.13	0.13	0.14	0.11	**51.33**	0.1
	Acc (Bit Width=3)	0.12	0.1	0.1	0.12	0.09	20.72