APPENDIX

A ADDITIONAL BASELINE COMPARISON

We compare our approach with additional pruning and quantization approaches in Table 2 for ResNet-50 trained on ImageNet. We see that we continue to achieve high levels of model compression along with slice sparsity for inference speedups. Yuan et al. (2020) achieve high levels of sparsity but are unstructured requiring dedicated hardware to obtain speedups. A similar case holds for the quantization approaches of Zhao et al. (2019); Jain et al. (2020) which can obtain inference speedups but with hardware optimized for 4-bit and 8-bit integer arithmetic. Additionally, they typically require post-hoc training stages (Jain et al., 2020) to improve performance after quantization while our approach is a single stage trained end-to-end.

Table 2: Comparison of our approach with other pruning and quantization approaches for ResNet-50 trained on ImageNet. We continue to achieve the most compression along with high slice sparsity. * denotes that the sparsity is unstructured and do not directly translate to computational benefits.

Algorithm	Size (MB)	Error (Top-1 %)	Sparsity (%)
ResNet-50 (ImageNet)			
Uncompressed	102.00 (1×)	23.7	0.0
Savarese et al. (2020)	8.36 (12×)	24.5	91.8*
Yuan et al. (2020)	38.76 (3×)	24.8	38.8
Zhao et al. (2019) (4 bit)	12.75 (8×)	33.8	0.0
Zhao et al. (2019) (8 bit)	25.5 (4×)	25.3	0.0
Jain et al. (2020) (4 bit w/ retraining)	12.75 (8×)	25.6	0.0
Jain et al. (2020) (8 bit w/o retraining)	25.5 (4×)	25.7	0.0
Jain et al. (2020) (8 bit w/ retraining)	25.5 (4×)	24.6	0.0
LilNetX (Best)	3.96 (26×)	25.4	66.7
LilNetX (Extreme)	**2.96 (34×)**	25.8	**81.7**

B HISTOGRAM OF WEIGHTS FOR DENSE UNCOMPRESSED MODEL

We obtain the histogram of weights of the various types of layers of a dense uncompressed ResNet-50 model trained on ImageNet with only the cross entropy loss. We do not apply any weight decay in order to avoid enforcing any distribution on the weights. Results are shown in Fig. 7. We show histograms for 1×1, 3×3, 7×7 convolutions as well as for the dense layer. For 3×3 and 7×7 convolutions, we pick a random dimension from a 9-dimensional or a 49-dimensional slice respectively, to highlight the histograms, as a single probability model is fit to each dimension as shown in Eq. (3). We see that the distributions naturally follow unimodality and are more or less zero-centered even without any weight decay regularization. The 7×7 convolution weight distribution is less continuous due to relatively fewer weight values per dimension (192) but still weakly exhibits the property of unimodality and symmetry. This shows that networks trained with vanilla cross entropy loss prefer such distributions naturally. However, the probability models in Eq. (3) do not enforce any such distribution and can take on any random distribution. Thus, enforcing a Gaussian prior as proposed in Sec. 3.2 promotes unimodality and symmetry of the weight distributions which can be beneficial for network performance.

C HISTOGRAM OF WEIGHTS FOR QUANTIZED LATENTS

To provide insights into the effect of our quantization, we visualize the histogram of the quantized latents as well for different weight groups. Results are shown in Fig. 8. We see that we obtain high levels of 0s on almost all weight groups spanning different types of convolutional layers as well as the final dense layer. Fewer number of zeros are present in the initial 7×7 convolution similar to the uncompressed weights as shown in Fig. 7 highlighting its importance in the network. Additionally, high amount of elements are zeros in 3×3 convolutions highlighting their redundancy and potential for compression compared to other convolutional layers or the dense layer.

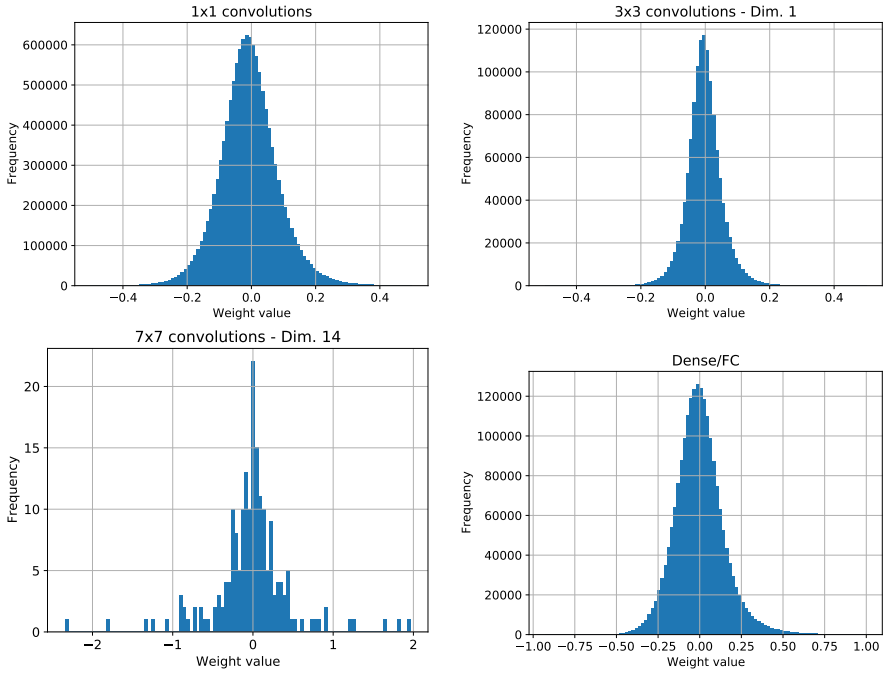

Figure 7: Histogram of the weights for various types of layers of a ResNet-50 model trained on ImageNet without weight decay. For 3×3 or 7×7 convolutions we pick a random dimension of a 9-dimensional or 49-dimensional slice to highlight the weight distribution for each dimension. Note the unimodal and zero-centered nature of each of these distributions, even without enforcing weight decay, highlighting the importance of the Gaussian prior as proposed in Sec. 3.2

D Comparison of model compression with entropy coding and sparse matrix formats

Instead of entropy coding, the sparse matrices can additionally be compressed using sparse matrix formats. We choose the two popular formats of Compressed Sparse Row (CSR) or Coordinate Format(COO). Results are summarized in Table 3 for our best run for ResNet-50 shown in Table 1 in the main paper. We see that entropy coding far outperforms the sparse formats of CSR and COO with COO obtaining better compression rates than CSR. This is expected as CSR/COO achieves high levels of compression only with extremely high levels of sparsity. With an unstructured sparsity level of $\sim$80%, storing only the non zero weights itself (and not their indices) provides a maximum compression of $5\times$.

Table 3: **Sparse formats:** Comparison of the effect of entropy coding *vs*. sparse matrix formats of CSR, COO on model compression of a ResNet-50 trained on ImageNet. We show the model size in MB of the latent weights along with the sparsity of the model weights.

Entropy Coding	CSR	COO	Slice Sparsity (%)	Unstructured Sparsity (%)
3.96 (26$\times$)	57 (2$\times$)	30 (3$\times$)	66.7	78.8

E Parameter groups for various networks

We share weight decoders and probability models for different parameter groups of a network which can be seen as being drawn from similar weight distributions. This limits the overhead in storing the weights of the corresponding decoders. We list the types of parameter groups for each network as follows:

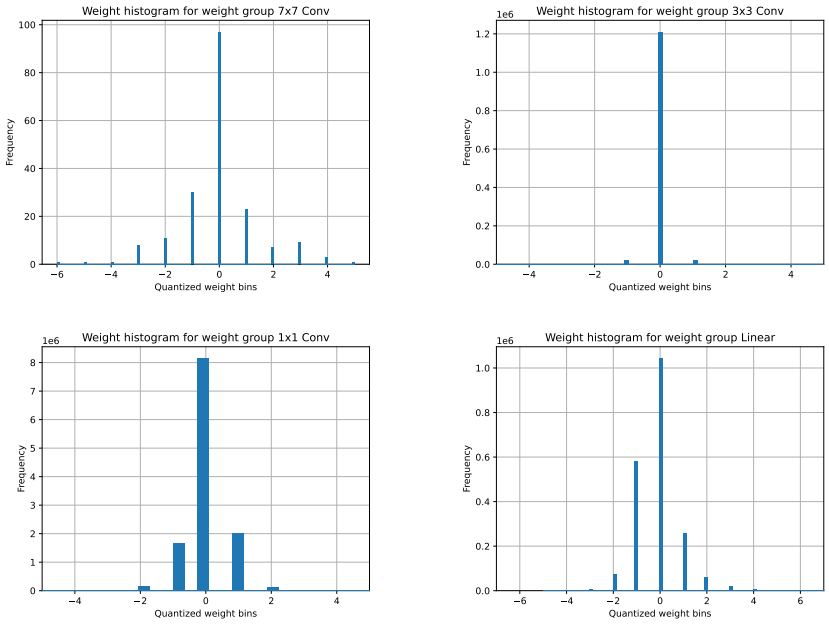

Figure 8: Histogram of the quantized latents for various types of layers of a ResNet-50 model trained on ImageNet. For 3×3 or 7×7 convolutions we pick a random dimension of a 9-dimensional or 49-dimensional slice to highlight the weight distribution for each dimension. Note the high level of 0s obtained due to our sparsity priors.

- VGG-16 consists of a parameter group for each dense layer and a parameter group for all 3×3 convolutions leading to four weight decoders/probability models for each parameter group.
- For ResNet-20-4 we use zero padding shortcut type A as defined in He et al. (2016), which leads to only 2 parameter groups, one for the final dense layer and the other for all 3×3 convolutions.
- For ResNet-18 trained on ImageNet, we use three parameter groups, for the initial 7x7 convolution, 3×3 convolutions, as well as the dense layer.
- ResNet-50 consists of an additional parameter group for 1×1 convolutions compared to ResNet-18.
- MobileNet-V2 consists of 3 parameter groups for the initial 3x3 convolution, final dense layer and the remaining 3x3 convolution.

F Standard Error for Multiple Runs

Sec. 5 in the main paper shows results when averaged across 3 seeds. In this section, we additionally provide the standard errors across the 3 random seeds. Results are summarized in Fig. 9 for the two datasets of CIFAR-10/100. CIFAR-10 shows little to no standard error both in the x-axis (model size) and y-axis (top-1 validation accuracy). This suggests that the training is stable for different random seeds. For CIFAR-100 however, we observe large error in the top-1 validation accuracy. We attribute this to the slow convergence for CIFAR-100 also highlighted in Fig. 10.

CIFAR-100 Convergence:We analyze the convergence of 3 different runs for ResNet-20-4 trained on the CIFAR-100 dataset with varying values of λ_S and λ_U. Results are shown in Fig. 10 when trained for 200 epochs. We see that validation accuracy (on the right y-axis) continues to increase towards the end of training between 190-200 epochs. At the same time, validation loss (on the left y-axis) also decreases. This suggests that the model hasn't fully converged by the end of 200 epochs. We hypothesize that this is an artifact of the dataset as well as the cosine decay schedule

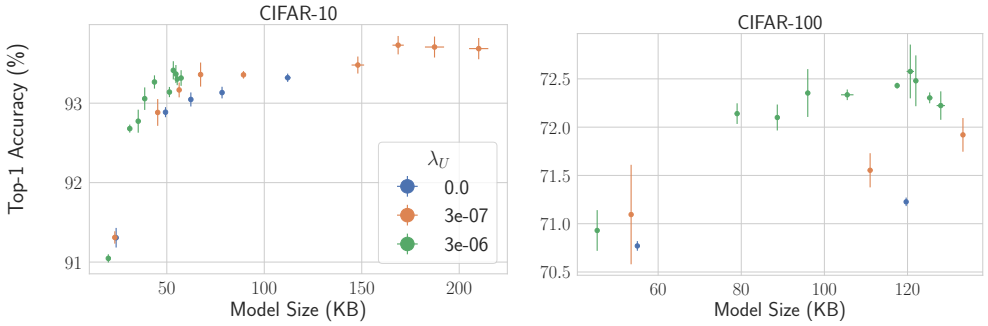

Figure 9: Scatter plots with horizontal and vertical error bars for ResNet-20-4 trained on CIFAR-10/100. For a different random seed, model size changes leading to the error bar in the x-axis while the vertical bar represents the top-1 validation accuracy error on the y-axis. There is very little variance in CIFAR-10 and slightly higher for CIFAR-100 due to slow convergence as shown in Fig. 10

where learning rate decreases drastically towards the end of training and is not maintained for longer for better convergence.

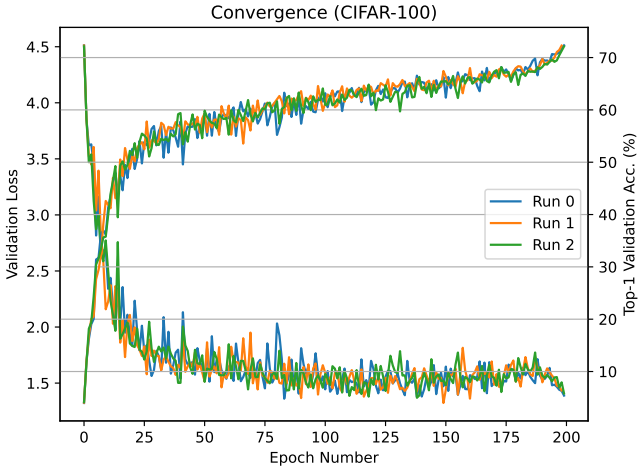

Figure 10: Convergence plots for 3 ResNet-20-4 runs on CIFAR-100. We see that loss (left axis) as well as top-1 validation accuracy (right axis) do not stabilize towards the end of training and respectively decrease/increase sharply suggesting that the training has not fully converged.

G l_2 *vs.* l_1 *vs.* l_∞ NORM

In this section, we analyze the effect of different types of norm for both individual weights and groups. For individual weights, we compare the l_2 norm with the l_1 norm while for the group norm, we compare the l_2 norm with the l_∞ norm (l_1 weight norm is same as l_1 group norm due to sum of absolutes). Results are summarized in Fig. 11 where top/bottom rows are for CIFAR-10/100 respectively. We see that l_2 group norm outperforms its l_∞ counterpart for both datasets. However, l_1 norm has little additional effect in terms of l_2 weight norm. Additionally, the l_2 group norm yields lesser slice sparsity for a given sparsity (c,g) highlighting the importance of l_∞ for high structured sparsity. While l_∞ leads to higher sparsity, it also shows higher model size for a given slice sparsity. Thus, there is an inherent tradeoff for l_∞ which leads to more sparsity but also larger model sizes (d,h).

H INITIALIZATION OF CONTINUOUS SURROGATES

The initialization of the continuous surrogate $\widehat{\boldsymbol{W}}$ of a latent space weight $\widetilde{\boldsymbol{W}}$ and the decoder matrix $\boldsymbol{\Psi}$ plays an important in the neural network training. Naïve He initialization (He et al., 2015) commonly used in training ResNet classifiers does not work in our case since small values of $\widehat{\boldsymbol{W}}$ get rounded to zero before decoding. Such an initialization results in zero gradients for updating the parameters and the loss becomes stagnant. To overcome this issue, we propose a modification to the initialization of the different parameters. In our framework, we recap that the decoded weights used in a forward pass are obtained using

$$\boldsymbol{W} = \text{reshape}(\widetilde{\boldsymbol{W}}\boldsymbol{\Psi}) \tag{9}$$

where $\widetilde{\boldsymbol{W}}$ is a matrix in $\mathbb{Z}^{C_{\text{in}}C_{\text{out}} \times l}$ and $\boldsymbol{\Psi}$ is a matrix in $\mathbb{Z}^{l \times l}$ (where $l = 1$ for dense weights (and biases) while $l = K^2$ for convolutional weights).

Our goal is to initialize $\widehat{\boldsymbol{W}}$ and $\boldsymbol{\Psi}$ such that the decoded weights $\boldsymbol{W}$ follow He initialization. First, since $\widehat{\boldsymbol{W}}$ is rounded to nearest integer (to obtain latent space weights $\widetilde{\boldsymbol{W}}$), we assume its elements to be drawn from a uniform distribution in $[-b, b]$ where $b > 0.5$ in order to enforce atleast some non-zero weights after rounding to nearest integer. Next, we take the elements of $\boldsymbol{\Psi}$ to be a normal distribution with mean 0 and variance v.

Assuming the parameters to be i.i.d., and $\text{Var}(\boldsymbol{X})$ denoting the variance of any individual element in matrix $\boldsymbol{X}$,

$$\text{Var}(\boldsymbol{W}) = l \times \text{Var}(\boldsymbol{\Psi}) \times \text{Var}(\widetilde{\boldsymbol{W}}) \tag{10}$$

Assuming a RELU activation, with f denoting the total number of channels (fan-in or fan-out) for a layer, LHS of Eq. (10), using the He initializer becomes $\frac{2}{f}$, RHS on the other hand can be obtained analytically

$$\begin{aligned} \frac{2}{f} &= l \times v \times \frac{(2b+1)^2 - 1}{12} \\ \implies b &= \frac{\sqrt{\frac{24}{lvf} + 1} - 1}{2}, v = \frac{24}{lf\left((2b+1)^2 - 1\right)} \end{aligned} \tag{11}$$

Eq. (11) gives us a relationship between b (defining the uniform distribution of $\widehat{\boldsymbol{W}}$) and v (defining the normal distribution of $\boldsymbol{\Psi}$). Note that l and f values are constant and known for each layer.

For a weight decoder corresponding to a parameter group, the maximum value of f in that group enforces the smallest value of b which should be above a minimum limit $b_{\min}$. Denoting $f_{\max}$ as the maximum fan-in or fan-out value for a parameter group, we get

$$\begin{aligned} v &= \frac{24}{lf_{\max}\left((2b_{\min}+1)^2 - 1\right)} \\ \implies b &= \frac{\sqrt{\frac{f_{\max}}{f}\left((2b_{\min}+1)^2 - 1\right) + 1} - 1}{2} \end{aligned} \tag{12}$$

The hyperparameter $b_{\min}$ then refers to the minimum boundary any latent space parameter can take in the network. By calculating the values of v based on $f_{\max}, b_{\min}$ and b for various parameters based on the corresponding value of f, we then initialize the elements of $\widehat{\boldsymbol{W}}$ to be drawn from a uniform distribution in the interval $[-b, b]$ and elements of $\boldsymbol{\Psi}$ to be drawn from $\mathcal{N}(0, v)$.

Note that $f = f_{\max} \implies b = b_{\min}$ which shows that the minimum boundary corresponds to the layer with maximum channels (fan-in or fan-out) f.

By choosing an appropriate value of $b_{\min}$ we obtain good initial values of the gradient which allows the network to converge well as training progresses. $b_{\min}$ offers an intuitive way of initializing the discrete weights. Too small a value leads to most of the weights being set to zero while too large a value can lead to exploding gradients. In practice, we find that this initialization approach works well for Cifar experiments. For ImageNet experiments, we assume a normal distribution instead of uniform distribution for $\widehat{\boldsymbol{W}}$ with a sufficiently high variance for the network to train.

I LICENSE

Table 4: **Licenses of datasets**.

Dataset	License
CIFAR-10 Krizhevsky et al. (2009)	MIT
CIFAR-100 Krizhevsky et al. (2009)	MIT
ImageNet Deng et al. (2009)	BSD 3-Clause

Table 4 lists all datasets we used and their licenses.

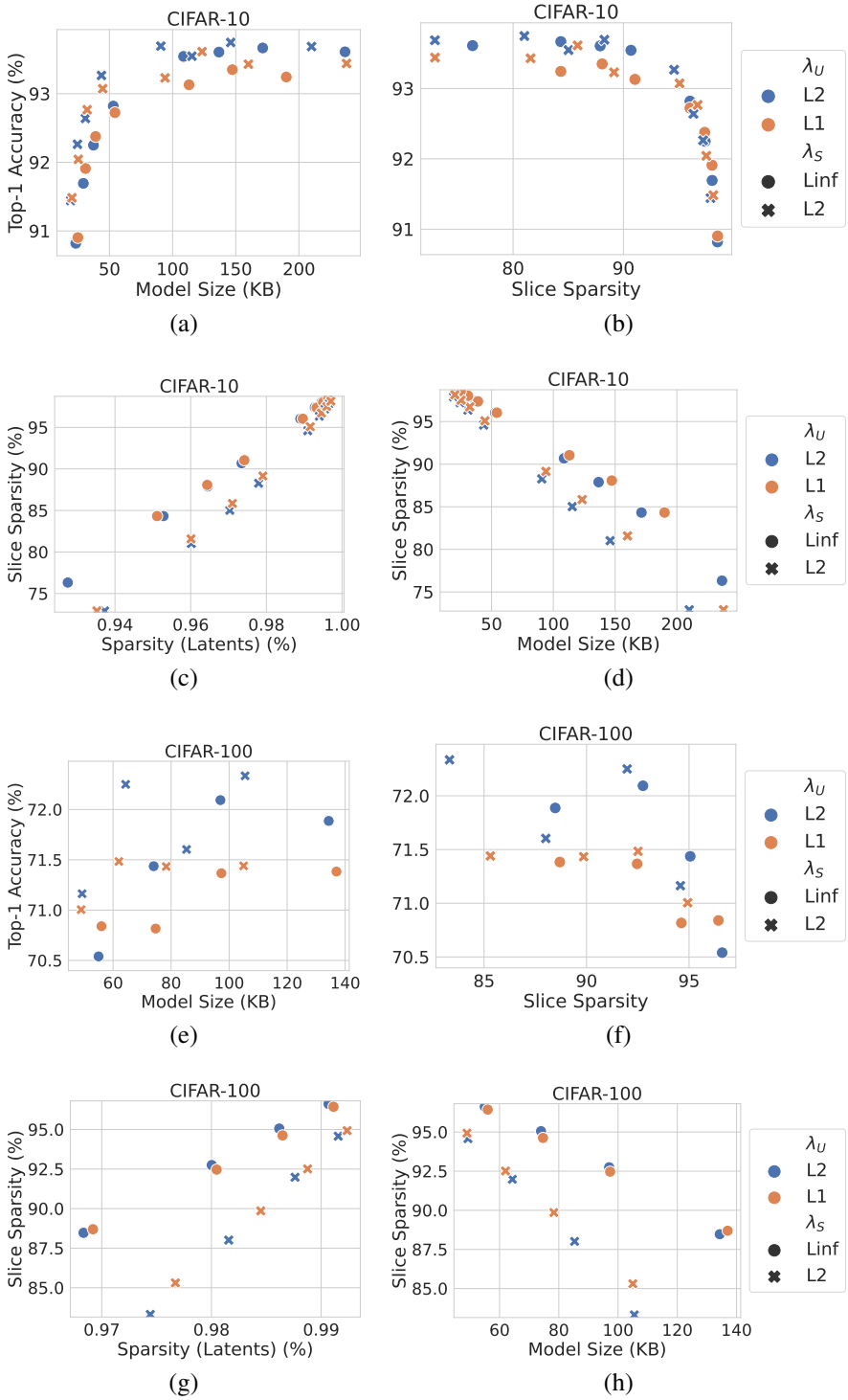

Figure 11: Comparison of l_2 *vs.* l_1*vs.* l_∞ norm for various metrics of sparsity and size for both CIFAR-10 (top row) and CIFAR-100 (bottom row). We see that l_2 group norm does better than l_∞ group norm in terms of accuracy vs model-size or slice sparsity (a,b,e,f). l_1 weight norm has little additional effect compared to l_2 weight norm. l_∞ favors higher slice sparsity for the same level of sparsity (c,g). l_∞ tends to result in higher model size for a given slice sparsity but also higher slice sparsity given a model size, which shows the tradeoff between compression and sparsification.