A. Appendix

A.1. Characterization of g

For the training dynamics of s, we propose some desired properties for choosing $g : \mathbb{R} \to \mathbb{R}_{++}$:

- $0 < g(s)$, $\lim_{s\to-\infty} g(s) = 0$, and $\lim_{s\to\infty} g(s) = \infty$.
- $\exists\, G \in \mathbb{R}_{++} \ni 0 < g'(s) \leq G \;\forall\, s \in \mathbb{R}$.
- $g'(s_{\text{init}}) < 1$ providing us a handle on the dynamics of s.

For simplicity, the choice of g were the logistic sigmoid function, $g(s) = \frac{k}{1+e^{-s}}$, and the exponential function, $g(s) = ke^s$, for $k \in \mathbb{R}_k$, since in most of the experimental scenarios, we almost always have $s < 0$ throughout the training making it satisfy all the desired properties in $\mathbb{R}_-$. One can choose k as an appropriate scaling factor based on the final weight distribution of a given DNN. All the CNN experiments in this paper we use the logistic sigmoid function with $k = 1$, as the weights' final learnt values are typically $\ll 1$, and low-rank RNN use the exponential function with $k = 1$. It should be noted that better functional choices might exist for g and can affect the expressivity and dynamics of STR parameterization for inducing sparsity.

A.2. Gradient w.r.t. $\{s_l\}_{l\in[L]}$

The gradient of $s_l \;\forall\, l \in [L]$ takes an even interesting form

$$\begin{aligned}\nabla_{s_l}\mathcal{L}\big(\widetilde{\mathbf{W}}_l(s_l)\big) &= \nabla_{s_l}\mathcal{L}\left(\mathcal{S}_g(\mathbf{W}_l, s_l)\right)\\ &= -g'(s_l)\mathcal{P}\left(\mathbf{W}_l, g(s_l)\right)\end{aligned} \tag{5}$$

Where $\mathcal{P}\left(\mathbf{W}_l, g(s_l)\right) := \left\langle \nabla_{\widetilde{\mathbf{W}}_l(s_l)}\mathcal{L}\big(\widetilde{\mathbf{W}}(s_l)\big), \text{sign}\left(\mathbf{W}_l\right) \odot \mathbf{1}\left\{\widetilde{\mathbf{W}}_l(s_l) \neq 0\right\}\right\rangle$. Thus the final update equation for $s_l \;\forall\, l \in [L]$ becomes

$$s_l^{(t+1)} \leftarrow s_l^{(t)} + \eta_t g'(s_l^{(t)})\mathcal{P}\left(\mathbf{W}_l^{(t)}, g\left(s_l^{(t)}\right)\right) - \eta_t\lambda s_l^{(t)} \tag{6}$$

where λ is its ℓ_2 regularization hyperparameter.

A.3. ResNet50 Learnt Budgets and Backbone Sparsities

Table 6 lists the non-uniform sparsity budgets learnt through STR across the sparsity regimes of 80%, 90%, 95%, 96.5%, 98% and 99% for ResNet50 on ImageNet-1K. The table also lists the backbone sparsities of every budget. It is clear that STR results in a higher than expected sparsity in the backbones of CNNs resulting in efficient backbones for transfer learning.

Table 7 summarizes all the sparsity budgets for 90% sparse ResNet50 on ImageNet-1K obtained using various methods. This table also shows that the backbone sparsities learnt through STR are considerably higher than that of the baselines.

One can use these budgets directly for techniques like GMP and DNW for a variety of datasets and have significant accuracy gains as shown in the Table 4.

A.4. MobileNetV1 Sparsity and FLOPs Budget Distributions

Table 8 summarizes all the sparsity budgets for 90% sparse MobileNetV1 on ImageNet-1K obtained using various methods. Note that GMP here makes the first and depthwise (dw) convolution layers dense, hence it is not the standard uniform sparsity. This table also shows that the backbone sparsities learnt through STR are considerably higher than that of GMP.

Figure 8 shows the sparsity distribution across layers when compared to GMP and Figure 9 shows the FLOPs distribution across layers when compared to GMP for 90% sparse MobileNetV1 models on ImageNet-1K.

It is interesting to notice that STR automatically keeps depthwise separable (the valleys in Figure 8) convolution layers less sparse than the rest to maximize accuracy which is the reason GMP keeps them fully dense.

Algorithm 1 PyTorch code for $\mathrm{STRConv}$ with per-layer threshold.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from args import args as parser_args

def softThreshold(x, s, g=torch.sigmoid):
   # STR on a weight x (can be a tensor) with "s" (typically a scalar, but can be a tensor) with function "g".
   return torch.sign(x)*torch.relu(torch.abs(x)-g(s))

class STRConv(nn.Conv2d): # Overloaded Conv2d which can replace nn.Conv2d
   def __init__(self, *args, **kwargs):
      super().__init__(*args, **kwargs)
      # "g" can be chosen appropriately, but torch.sigmoid works fine.
      self.g = torch.sigmoid
      # parser_args gets arguments from command line. sInitValue is the initialization of "s" for all layers. It
           can take in different values per-layer as well.
      self.s = nn.Parameter(parser_args.sInitValue*torch.ones([1, 1]))
      # "s" can be per-layer (a scalar), global (a shared scalar across layers), per-channel/filter (a vector)
          or per individual weight (a tensor of the size self.weight). All the experiments use per-layer "s" (a
          scalar) in the paper.

   def forward(self, x):

      sparseWeight = softThreshold(self.weight, self.s, self.g)
      # Parameters except "x" and "sparseWeight" can be chosen appropriately. All the experiments use default
          PyTorch arguments.
      x = F.conv2d(x, sparseWeight, self.bias, self.stride, self.padding, self.dilation, self.groups)

      return x

# FC layer is implemented as a 1x1 Conv2d and STRConv is used for FC layer as well.
```

Table 6. The non-uniform sparsity budgets for various sparsity ranges learnt through STR for ResNet50 on ImageNet-1K. FLOPs distribution per layer can be computed as $\frac{100-s_i}{100} * \text{FLOPs}_i$, where s_i and FLOPs_i are the sparsity and FLOPs of the layer i.

Metric	Fully Dense Params	Fully Dense FLOPs	Sparsity (%)															
Overall	25502912	4089284608	79.55	81.27	87.70	90.23	90.55	94.80	95.03	95.15	96.11	96.53	97.78	98.05	98.22	98.79	98.98	99.10
Backbone	23454912	4087136256	82.07	83.79	90.08	92.47	92.77	96.51	96.71	96.84	97.64	97.92	98.82	98.99	99.11	99.46	99.58	99.64
Layer 1 - conv1	9408	118013952	51.46	51.40	63.02	59.80	59.83	64.87	67.36	66.96	72.11	69.46	73.29	73.47	72.05	75.12	76.12	77.75
Layer 2 - layer1.0.conv1	4096	12845056	69.36	73.24	87.57	83.28	85.18	89.60	91.41	91.11	92.38	91.75	94.46	94.51	94.60	95.95	96.53	96.51
Layer 3 - layer1.0.conv2	36864	115605504	77.85	76.26	90.87	89.48	87.31	94.79	94.27	95.04	95.69	96.07	97.36	97.77	98.35	98.51	98.59	98.84
Layer 4 - layer1.0.conv3	16384	51380224	74.81	74.65	86.52	85.80	85.25	91.85	92.78	93.67	94.13	94.69	96.61	97.03	97.37	98.04	98.21	98.47
Layer 5 - layer1.0.downsample.0	16384	51380224	70.95	72.96	83.53	83.34	82.56	89.13	90.62	90.17	91.83	92.69	95.48	94.89	95.68	96.98	97.56	97.72
Layer 6 - layer1.1.conv1	16384	51380224	80.27	79.58	89.82	89.89	88.51	94.56	96.64	95.78	95.81	96.81	98.79	98.90	98.98	99.13	99.62	99.47
Layer 7 - layer1.1.conv2	36864	115605504	81.36	80.95	91.75	90.60	89.61	94.70	95.78	96.18	96.42	97.26	98.65	99.07	99.40	99.11	99.31	99.56
Layer 8 - layer1.1.conv3	16384	51380224	84.45	80.11	91.22	91.70	90.21	95.17	97.05	95.81	96.34	97.23	98.68	98.76	98.90	99.16	99.57	99.46
Layer 9 - layer1.2.conv1	16384	51380224	78.23	79.79	90.12	88.07	89.36	94.62	95.94	94.74	96.23	96.75	97.96	98.41	98.72	99.38	99.35	99.46
Layer 10 - layer1.2.conv2	36864	115605504	76.01	81.53	91.06	87.03	88.27	93.90	95.63	94.26	96.24	96.11	97.54	98.27	98.44	99.32	99.19	99.39
Layer 11 - layer1.2.conv3	16384	51380224	84.47	83.28	94.95	90.99	92.64	95.76	96.95	96.01	96.87	97.31	98.38	98.60	98.72	99.38	99.27	99.51
Layer 12 - layer2.0.conv1	32768	102760448	73.74	73.96	86.78	85.95	85.90	92.32	94.79	93.86	94.62	95.64	97.19	98.22	98.52	98.48	98.84	98.92
Layer 13 - layer2.0.conv2	147456	115605504	82.56	85.70	91.31	93.91	94.03	97.54	97.43	97.65	98.38	98.62	99.24	99.23	99.40	99.61	99.67	99.63
Layer 14 - layer2.0.conv3	65536	51380224	84.70	83.55	93.04	93.13	92.13	96.61	97.37	97.21	97.59	98.14	98.80	98.95	99.18	99.29	99.47	99.43
Layer 15 - layer2.0.downsample.0	131072	102760448	85.10	87.66	92.78	94.96	95.13	98.07	97.97	98.15	98.70	98.88	99.37	99.35	99.40	99.69	99.68	99.71
Layer 16 - layer2.1.conv1	65536	51380224	85.42	85.79	94.04	95.31	94.94	97.92	98.53	98.21	98.84	99.06	99.46	99.53	99.72	99.78	99.81	99.80
Layer 17 - layer2.1.conv2	147456	115605504	76.95	82.75	87.63	91.50	91.76	95.59	97.22	96.07	97.32	97.80	98.24	98.24	98.60	99.24	99.66	99.33
Layer 18 - layer2.1.conv3	65536	51380224	84.76	84.71	93.10	93.66	93.23	97.00	98.18	97.35	98.06	98.41	98.96	99.21	99.32	99.55	99.58	99.59
Layer 19 - layer2.2.conv1	65536	51380224	84.30	85.34	92.70	94.61	94.76	97.72	97.91	98.21	98.54	98.98	99.24	99.35	99.50	99.62	99.63	99.77
Layer 20 - layer2.2.conv2	147456	115605504	84.28	85.43	92.99	94.86	94.90	97.52	97.21	98.11	98.19	99.04	99.28	99.37	99.46	99.63	99.59	99.72
Layer 21 - layer2.2.conv3	65536	51380224	82.19	84.21	91.12	93.38	93.53	96.89	97.14	97.59	97.77	98.66	98.96	99.15	99.25	99.49	99.51	99.57
Layer 22 - layer2.3.conv1	65536	51380224	83.37	84.41	90.46	93.26	93.50	96.71	97.89	96.99	98.14	98.36	99.10	99.23	99.33	99.53	99.75	99.60
Layer 23 - layer2.3.conv2	147456	115605504	82.83	84.03	91.44	93.21	93.25	96.83	98.02	96.96	98.45	98.30	98.97	99.06	99.26	99.31	99.81	99.68
Layer 24 - layer2.3.conv3	65536	51380224	82.93	85.65	91.02	94.14	93.56	97.20	97.97	97.04	98.16	98.36	98.88	98.97	99.20	99.32	99.67	99.62
Layer 25 - layer3.0.conv1	131072	102760448	76.63	77.98	85.99	88.85	88.60	94.26	95.07	94.97	96.21	96.59	97.75	98.04	98.30	98.72	99.11	99.06
Layer 26 - layer3.0.conv2	589824	115605504	87.35	88.68	94.39	96.14	96.19	98.51	98.77	98.72	99.11	99.23	99.53	99.59	99.64	99.73	99.80	99.81
Layer 27 - layer3.0.conv3	262144	51380224	81.22	83.22	90.58	93.19	93.05	96.82	97.38	97.32	97.98	98.28	98.88	99.03	99.16	99.39	99.55	99.53
Layer 28 - layer3.0.downsample.0	524288	102760448	89.75	90.99	96.05	97.20	97.16	98.96	99.21	99.20	99.50	99.58	99.78	99.82	99.86	99.91	99.94	99.93
Layer 29 - layer3.1.conv1	262144	51380224	85.88	87.35	93.43	95.36	96.12	98.64	98.77	98.87	99.22	99.33	99.64	99.67	99.72	99.82	99.88	99.84
Layer 30 - layer3.1.conv2	589824	115605504	85.06	86.24	92.74	95.06	95.30	98.09	98.28	98.36	98.75	99.08	99.46	99.48	99.54	99.69	99.76	99.76
Layer 31 - layer3.1.conv3	262144	51380224	84.34	86.79	92.15	94.84	94.90	97.75	98.15	98.11	98.56	98.94	99.30	99.36	99.45	99.65	99.79	99.70
Layer 32 - layer3.2.conv1	262144	51380224	87.51	89.15	94.15	96.77	96.46	98.81	98.83	98.96	99.19	99.44	99.67	99.71	99.74	99.82	99.85	99.89
Layer 33 - layer3.2.conv2	589824	115605504	87.15	88.67	94.09	95.59	96.14	98.86	98.69	98.91	99.21	99.20	99.64	99.72	99.76	99.85	99.84	99.90
Layer 34 - layer3.2.conv3	262144	51380224	84.86	86.90	92.40	94.99	94.99	98.19	98.19	98.42	98.76	98.97	99.42	99.56	99.62	99.76	99.75	99.88
Layer 35 - layer3.3.conv1	262144	51380224	86.62	89.46	94.06	96.08	95.88	98.70	98.71	98.77	99.01	99.27	99.58	99.66	99.69	99.83	99.87	99.87
Layer 36 - layer3.3.conv2	589824	115605504	86.52	87.97	93.56	96.10	96.11	98.70	98.82	98.89	99.19	99.31	99.68	99.73	99.77	99.88	99.87	99.93
Layer 37 - layer3.3.conv3	262144	51380224	84.19	86.81	92.32	94.94	94.91	98.20	98.37	98.43	98.82	99.00	99.51	99.57	99.64	99.81	99.81	99.87
Layer 38 - layer3.4.conv1	262144	51380224	85.85	88.40	93.55	95.49	95.86	98.35	98.44	98.55	98.79	98.96	99.54	99.59	99.60	99.82	99.86	99.87
Layer 39 - layer3.4.conv2	589824	115605504	85.96	87.38	93.27	95.66	95.63	98.41	98.58	98.56	99.19	99.26	99.64	99.69	99.67	99.87	99.90	99.92
Layer 40 - layer3.4.conv3	262144	51380224	83.45	85.76	91.75	94.49	94.35	97.67	98.09	97.99	98.65	98.94	99.49	99.52	99.48	99.77	99.86	99.85
Layer 41 - layer3.5.conv1	262144	51380224	83.33	85.77	91.79	95.09	94.24	97.46	97.89	97.92	98.71	98.90	99.35	99.52	99.58	99.76	99.79	99.83
Layer 42 - layer3.5.conv2	589824	115605504	84.98	86.67	92.48	94.92	95.13	97.88	98.14	98.32	98.91	99.00	99.44	99.58	99.69	99.80	99.83	99.87
Layer 43 - layer3.5.conv3	262144	51380224	79.78	82.23	89.39	93.14	92.76	96.59	97.04	97.30	98.10	98.41	99.03	99.25	99.44	99.61	99.71	99.75
Layer 44 - layer4.0.conv1	524288	102760448	77.83	79.61	87.11	90.32	90.64	95.39	95.84	95.92	97.17	97.35	98.36	98.60	98.83	99.20	99.37	99.42
Layer 45 - layer4.0.conv2	2359296	115605504	86.18	88.00	93.53	95.66	95.78	98.31	98.47	98.55	99.08	99.16	99.54	99.63	99.69	99.81	99.85	99.86
Layer 46 - layer4.0.conv3	1048576	51380224	78.43	80.48	87.85	91.14	91.27	96.00	96.40	96.47	97.53	97.92	98.81	99.00	99.15	99.45	99.57	99.61
Layer 47 - layer4.0.downsample.0	2097152	102760448	88.49	89.98	95.03	96.79	96.90	98.91	99.06	99.11	99.45	99.51	99.77	99.82	99.85	99.92	99.94	99.94
Layer 48 - layer4.1.conv1	1048576	51380224	82.07	84.02	90.34	93.69	93.72	97.15	97.56	97.76	98.45	98.75	99.27	99.36	99.54	99.67	99.76	99.80
Layer 49 - layer4.1.conv2	2359296	115605504	83.42	85.23	91.16	93.98	93.93	97.26	97.58	97.71	98.36	98.67	99.25	99.34	99.50	99.68	99.76	99.80
Layer 50 - layer4.1.conv3	1048576	51380224	78.08	79.96	86.66	90.48	90.22	95.22	95.76	95.89	96.88	97.65	98.70	98.85	99.13	99.45	99.58	99.66
Layer 51 - layer4.2.conv1	1048576	51380224	76.34	77.93	84.98	87.57	88.47	93.90	93.87	94.16	95.55	95.91	97.66	97.97	98.15	98.88	99.08	99.22
Layer 52 - layer4.2.conv2	2359296	115605504	73.57	74.97	82.32	84.37	86.01	91.92	91.66	92.22	94.02	94.16	96.65	97.13	97.29	98.44	98.74	99.00
Layer 53 - layer4.2.conv3	1048576	51380224	68.78	70.38	78.11	80.29	81.73	89.64	89.43	89.65	91.40	92.65	96.02	96.72	96.93	98.47	98.83	99.15
Layer 54 - fc	2048000	2048000	50.65	52.46	60.48	64.50	65.12	75.20	75.73	75.80	78.57	80.69	85.96	87.26	88.03	91.11	92.15	92.87
AP - adaptive average pool before fc	0	100352	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 7. The non-uniform sparsity budgets learnt multiple methods for 90% sparse ResNet50 on ImageNet-1K. FLOPs distribution per layer can be computed as $\frac{100-s_i}{100} * \text{FLOPs}_i$, where s_i and FLOPs_i are the sparsity and FLOPs of the layer i.

Metric	Fully Dense Params	Fully Dense FLOPs	Sparsity (%)					
			STR	Uniform	ERK	SNFS	VD	GS
Overall	25502912	4089284608	90.23	90.00	90.07	90.06	90.27	89.54
Backbone	23454912	4087136256	92.47	90.00	89.82	89.44	91.41	90.95
Layer 1 - conv1	9408	118013952	59.80	90.00	58.00	2.50	31.39	35.11
Layer 2 - layer1.0.conv1	4096	12845056	83.28	90.00	0.00	2.50	39.50	56.05
Layer 3 - layer1.0.conv2	36864	115605504	89.48	90.00	82.00	2.50	67.87	75.04
Layer 4 - layer1.0.conv3	16384	51380224	85.80	90.00	4.00	2.50	64.87	70.31
Layer 5 - layer1.0.downsample.0	16384	51380224	83.34	90.00	4.00	2.50	60.38	66.88
Layer 6 - layer1.1.conv1	16384	51380224	89.89	90.00	4.00	2.50	61.35	75.09
Layer 7 - layer1.1.conv2	36864	115605504	90.60	90.00	82.00	2.50	64.38	80.42
Layer 8 - layer1.1.conv3	16384	51380224	91.70	90.00	4.00	2.50	65.83	80.00
Layer 9 - layer1.2.conv1	16384	51380224	88.07	90.00	4.00	2.50	68.75	75.21
Layer 10 - layer1.2.conv2	36864	115605504	87.03	90.00	82.00	2.50	70.86	74.95
Layer 11 - layer1.2.conv3	16384	51380224	90.99	90.00	4.00	2.50	54.05	79.28
Layer 12 - layer2.0.conv1	32768	102760448	85.95	90.00	43.00	2.50	57.10	70.89
Layer 13 - layer2.0.conv2	147456	115605504	93.91	90.00	91.00	62.90	78.65	85.39
Layer 14 - layer2.0.conv3	65536	51380224	93.13	90.00	52.00	11.00	85.49	83.54
Layer 15 - layer2.0.downsample.0	131072	102760448	94.96	90.00	71.00	66.10	79.96	88.36
Layer 16 - layer2.1.conv1	65536	51380224	95.31	90.00	52.00	32.60	72.07	88.25
Layer 17 - layer2.1.conv2	147456	115605504	91.50	90.00	91.00	61.60	84.41	85.37
Layer 18 - layer2.1.conv3	65536	51380224	93.66	90.00	52.00	20.80	79.19	86.53
Layer 19 - layer2.2.conv1	65536	51380224	94.61	90.00	52.00	29.10	73.94	86.40
Layer 20 - layer2.2.conv2	147456	115605504	94.86	90.00	91.00	63.90	78.48	88.29
Layer 21 - layer2.2.conv3	65536	51380224	93.38	90.00	52.00	22.90	78.09	85.87
Layer 22 - layer2.3.conv1	65536	51380224	93.26	90.00	52.00	27.60	78.66	84.87
Layer 23 - layer2.3.conv2	147456	115605504	93.21	90.00	91.00	65.30	84.38	87.14
Layer 24 - layer2.3.conv3	65536	51380224	94.14	90.00	52.00	25.70	82.07	86.84
Layer 25 - layer3.0.conv1	131072	102760448	88.85	90.00	71.00	48.70	66.56	78.40
Layer 26 - layer3.0.conv2	589824	115605504	96.14	90.00	96.00	90.20	87.92	92.93
Layer 27 - layer3.0.conv3	262144	51380224	93.19	90.00	76.00	73.30	92.19	86.19
Layer 28 - layer3.0.downsample.0	524288	102760448	97.20	90.00	86.00	93.70	88.76	94.66
Layer 29 - layer3.1.conv1	262144	51380224	95.36	90.00	76.00	81.10	91.79	93.60
Layer 30 - layer3.1.conv2	589824	115605504	95.06	90.00	96.00	90.40	92.47	93.07
Layer 31 - layer3.1.conv3	262144	51380224	94.84	90.00	76.00	78.10	88.88	90.54
Layer 32 - layer3.2.conv1	262144	51380224	96.77	90.00	76.00	80.40	84.86	93.44
Layer 33 - layer3.2.conv2	589824	115605504	95.59	90.00	96.00	90.80	91.50	93.73
Layer 34 - layer3.2.conv3	262144	51380224	94.99	90.00	76.00	79.30	81.59	91.13
Layer 35 - layer3.3.conv1	262144	51380224	96.08	90.00	76.00	80.70	76.64	93.18
Layer 36 - layer3.3.conv2	589824	115605504	96.10	90.00	96.00	90.70	91.26	93.63
Layer 37 - layer3.3.conv3	262144	51380224	94.94	90.00	76.00	79.00	85.46	91.63
Layer 38 - layer3.4.conv1	262144	51380224	95.49	90.00	76.00	79.40	85.33	91.98
Layer 39 - layer3.4.conv2	589824	115605504	95.66	90.00	96.00	91.00	91.57	94.21
Layer 40 - layer3.4.conv3	262144	51380224	94.49	90.00	76.00	79.00	86.19	91.63
Layer 41 - layer3.5.conv1	262144	51380224	95.09	90.00	76.00	78.30	84.64	90.72
Layer 42 - layer3.5.conv2	589824	115605504	94.92	90.00	96.00	91.00	91.14	93.43
Layer 43 - layer3.5.conv3	262144	51380224	93.14	90.00	76.00	78.20	84.09	89.56
Layer 44 - layer4.0.conv1	524288	102760448	90.32	90.00	86.00	85.80	77.90	85.35
Layer 45 - layer4.0.conv2	2359296	115605504	95.66	90.00	98.00	97.60	96.53	95.07
Layer 46 - layer4.0.conv3	1048576	51380224	91.14	90.00	88.00	93.20	93.52	89.21
Layer 47 - layer4.0.downsample.0	2097152	102760448	96.79	90.00	93.00	98.80	93.80	96.72
Layer 48 - layer4.1.conv1	1048576	51380224	93.69	90.00	88.00	94.10	94.96	92.69
Layer 49 - layer4.1.conv2	2359296	115605504	93.98	90.00	98.00	97.70	97.76	93.85
Layer 50 - layer4.1.conv3	1048576	51380224	90.48	90.00	88.00	94.20	94.53	89.84
Layer 51 - layer4.2.conv1	1048576	51380224	87.57	90.00	88.00	93.60	94.19	85.91
Layer 52 - layer4.2.conv2	2359296	115605504	84.37	90.00	98.00	97.90	94.92	87.14
Layer 53 - layer4.2.conv3	1048576	51380224	80.29	90.00	88.00	94.50	89.64	80.65
Layer 54 - fc	2048000	2048000	64.50	90.00	93.00	97.10	77.17	73.43
AP - adaptive average pool before fc	0	100352	0.00	0.00	0.00	0.00	0.00	0.00

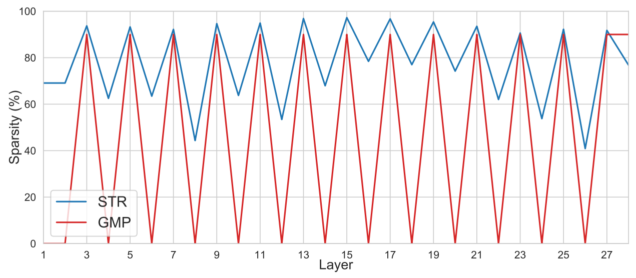

Figure 8. Layer-wise sparsity budget for the 90% sparse MobileNetV1 models on ImageNet-1K using various sparsification techniques.

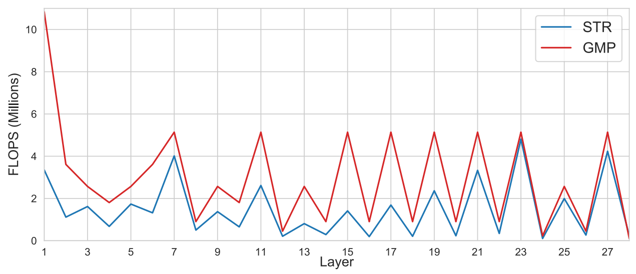

Figure 9. Layer-wise FLOPs distribution for the 90% sparse MobileNetV1 models on ImageNet-1K using various sparsification techniques.

Table 8. The non-uniform sparsity budgets learnt multiple methods for 90% sparse MobileNetV1 on ImageNet-1K. FLOPs distribution per layer can be computed as $\frac{100-s_i}{100} * \text{FLOPs}_i$, where s_i and FLOPs_i are the sparsity and FLOPs of the layer i.

Metric	Fully Dense Params	Fully Dense FLOPs	Sparsity (%) STR	Sparsity (%) GMP
Overall	4209088	568740352	89.01	89.03
Backbone	3185088	567716352	92.93	88.71
Layer 1	864	10838016	69.10	0.00
Layer 2 (dw)	288	3612672	69.10	0.00
Layer 3	2048	25690112	93.70	90.00
Layer 4 (dw)	576	1806336	62.50	0.00
Layer 5	8192	25690112	93.25	90.00
Layer 6 (dw)	1152	3612672	63.45	0.00
Layer 7	16384	51380224	92.19	90.00
Layer 8 (dw)	1152	903168	44.36	0.00
Layer 9	32768	25690112	94.65	90.00
Layer 10 (dw)	2304	1806336	63.76	0.00
Layer 11	65536	51380224	94.91	90.00
Layer 12 (dw)	2304	451584	53.43	0.00
Layer 13	131072	25690112	96.86	90.00
Layer 14 (dw)	4608	903168	67.93	0.00
Layer 15	262144	51380224	97.25	90.00
Layer 16 (dw)	4608	903168	78.43	0.00
Layer 17	262144	51380224	96.71	90.00
Layer 18 (dw)	4608	903168	77.00	0.00
Layer 19	262144	51380224	95.40	90.00
Layer 20 (dw)	4608	903168	74.22	0.00
Layer 21	262144	51380224	93.52	90.00
Layer 22 (dw)	4608	903168	62.02	0.00
Layer 23	262144	51380224	90.64	90.00
Layer 24 (dw)	4608	225792	53.78	0.00
Layer 25	524288	25690112	92.23	90.00
Layer 26 (dw)	9216	451584	40.89	0.00
Layer 27	1048576	51380224	91.76	90.00
Layer 28 (fc)	1024000	1024000	76.81	90.00
AP (average pool before fc)	0	50176	0.00	0.00

A.5. STR Adaptations

Algorithm 1 has comments suggesting the simple modifications required for global and per-weight sparsity.

A.5.1. STR for Global Sparsity

STR can be trivially modified to learn the global threshold to induce global sparsity like in (Han et al., 2015; Frankle & Carbin, 2019). Instead of having an s_l per layer l, share all the s_l to create one single learnable global threshold s_g. This can be implemented by a simple modification in Algorithm 1. STR's capability to induce global sparsity was evaluated on ResNet50 for ImageNet-1K for 90-98% sparsity regimes.

Table 9 shows the performance of STR-GS that learns the global threshold to induce global sparsity. While the accuracies are comparable to the state-of-the-art if not better, they do come at cost of $\sim 2\times$ inference cost compared to layer-wise sparsity due to poor non-uniform sparsity distribution which is a result of difference converged values of weights in each of the layers. STR-GS has numbers similar to IMP (Frankle & Carbin, 2019) while being able to learn the threshold stablely.

Table 9. STR can stablely learn the global threshold to induce global sparsity resulting in models with comparable accuracies as layer-wise sparsity but with $\sim 2\times$ the inference cost.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
STR-GS	74.13	2.42M	89.54	596M
STR-GS	71.61	1.58M	93.84	363M
STR-GS	67.95	1.01M	96.06	232M
STR-GS	62.17	0.54M	97.91	142M

A.5.2. STR for Filter/Channel Pruning

Let us assume there are n_{out} filters of size $k \times k \times n_{in}$ in a given layer. Typically in channel/filter pruning techniques, each of these n_{out} filters have an importance factor that represents the utility of the filter and is used to scale the corresponding filter. For a filter f_i there exists a importance scalar m_i learnt or obtained in some fashion and is used to get the effective filter in use $\hat{f}_i = m_i \cdot f_i$ where m_i is broadcasted to scale f_i. In practice, m_i is heuristically made to go to 0 to induce structured sparsity through channel/filter pruning. Let us stack all the importance scalars of the filters in the layer, $\{m_i\}_{i\in[n_{out}]}$, as vector $\boldsymbol{m}_l$ where l is the layer index. Now, this reduces to the same problem of inducing sparsity in a vector as in the learning of low-rank in RNN presented in Section 4.2.1. STR will be applied to each of the $\{m_l\}_{l\in[L]}$ where L is the total number of layers in a deep neural network. The inference will use the importance scalars through STR ensuring channel/filter pruning due to the induced sparsity. This is very similar to the work-flow we used to induce low-rank in RNNs.

A.5.3. STR for Per-weight Pruning or Mask Learning

The adaptation of STR for per-weight pruning or mask learning is simple and is similar to layer-wise or global sparsity. Changing $s_l \to \mathbf{S}_l$ ie., changing the layer-wise thresholds from a scalar to a tensor of the size of $\mathbf{W}_l$ will hep STR adapt to do per-weight pruning or mask learning as discussed in the recent works (Zhou et al., 2019; Savarese et al., 2019; Ramanujan et al., 2020). We have explored this using a couple of experiments on CIFAR-10 (Krizhevsky et al., 2009) and ImageNet-1K. We observed that high amounts of sparsity were induced and the routine is very aggressive compared to other sparsification methods. For example, we were able to get 90% accuracy on CIFAR-10 using ResNet18 at a staggering 99.63% sparsity (270× lesser parameters than the dense model) which results in 41K parameters pushing it into very under parameterized regime. We suggest caution when running per-weight sparsity experiments with any method due to the high variance in the final accuracy.

A.6. Hyperparameters for Reproducibility

All the ResNet50 experiments use a batchsize of 256, cosine learning rate with warm-up as in (Wortsman et al., 2019) and trained for 100 epochs. λ is the weight-decay hyperparameter. s_{init} is the initial value of all s_i where i is the layer number. The hyper parameter setting for each of the sparse model can be found in Table 10.

All the MobileNetV1 experiments use a batchsize of 256, cosine learning rate with warm-up as in (Wortsman et al., 2019) and trained for 100 epochs. λ is the weight-decay hyperparameter. s_{init} is the initial value of all s_i where i is the layer number. The hyper parameter setting for each of the sparse model can be found in Table 11.

All the CNN experiments use $g(s) = \frac{1}{1+e^{-s}}$ for the STR.

All the FastGRNN experiments use a batchsize of 100, learning rate and optimizers as suggested in (Kusupati et al., 2018) and trained for 300 epochs. Weight-decay parameter, λ is applied to both $\mathbf{m_W}, \mathbf{m_U}$ resulting in the rank setting obtained. Each hyperparamter setting can lead to multiple low-rank setting over the course of training. s_{init} set such that $g(s_{\text{init}}) \approx 0$ for the initialization of soft threshold

Table 10. The hyperparameters for various sparse ResNet50 models on ImageNet-1K using STR. λ is the weight-decay parameter and s_{init} is the initialization of all s_i for all the layers in ResNet50.

Sparse Model (%)	Weight-decay (λ)	s_{init}
79.55	0.00001700000000	-3200
81.27	0.00001751757813	-3200
87.70	0.00002051757813	-3200
90.23	0.00002251757813	-3200
90.55	0.00002051757813	-800
94.80	0.00003051757813	-3200
95.03	0.00003351757813	-12800
95.15	0.00003051757813	-1600
96.11	0.00003051757813	-100
96.53	0.00004051757813	-12800
97.78	0.00005217578125	-12800
98.05	0.00005651757813	-12800
98.22	0.00006051757813	-12800
98.79	0.00007551757813	-12800
98.98	0.00008551757813	-12800
99.10	0.00009051757813	-12800

Table 11. The hyperparameters for various sparse MobileNetV1 models on ImageNet-1K using STR. λ is the weight-decay parameter and s_{init} is the initialization of all s_i for all the layers in MobileNetV1.

Sparse Model (%)	Weight-decay (λ)	s_{init}
75.28	0.00001551757813	-100
79.07	0.00001551757813	-25
85.80	0.00003051757813	-3200
89.01	0.00003751757813	-12800
89.62	0.00003751757813	-3200

Table 12. Hyperparameters for the low-rank FastGRNN with STR. The same weight-decay parameter λ is applied on both $\mathbf{m_W}, \mathbf{m_U}$. Multiple rank setting can be acheived during the training course of the FastGRNN model. $g(s_{\text{init}}) \approx 0$ ie., $s_{\text{init}} \le -10$ for all the experiments.

Google-12		HAR-2	
(r_W, r_U)	Weight-decay (λ)	(r_W, r_U)	Weight-decay (λ)
(12, 40)	0.001	(9, 8)	0.001
(11, 35)	0.001	(9, 7)	0.001
(10, 31)	0.002	(8, 7)	0.001
(9, 24)	0.005		

pruning scalar for the low-rank vectors.

All the RNN experiments use $g(s) = e^s$ for the STR.

A.7. Dataset and Model Details

ImageNet-1K: ImageNet-1K has RGB images with 224×224 dimensions. The dataset has 1.3M training images, 50K validation images and 1000 classes. Images were transformed and augmented with the standard procedures as in (Wortsman et al., 2019).

Google-12: Google Speech Commands dataset (Warden, 2018) contains 1 second long utterances of 30 short words (30 classes) sampled at 16KHz. Standard log Mel-filter-bank featurization with 32 filters over a window size of 25ms and stride of 10ms gave 99 timesteps of 32 filter responses for a 1-second audio clip. For the 12 class version, 10 classes used in Kaggle's Tensorflow Speech Recognition challenge were used and the remaining two classes were noise and background sounds (taken randomly from the remaining 20 short word utterances). The datasets were zero mean - unit variance normalized during training and prediction. Google-12 has 22,246 training points, 3,081 testing points. Each datapoint has 99 timesteps with each input being 32 dimensional making the datapoint 3,168 dimensional.

HAR-2: Human Activity Recognition (HAR) dataset was collected from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone. The features available on the repository were directly used for experiments. The 6 activities were merged to get the binarized version. The classes {Sitting, Laying, Walking_Upstairs} and {Standing, Walking, Walking_Downstairs} were merged to obtain the two classes. The dataset was zero mean - unit variance normalized during training and prediction. HAR-2 has 7,352 training points and 2,947 test points. Each datapoint has 1,152 dimensions, which will be split into 128 timesteps leading to dimensional per timestep inputs.

ResNet50: ResNet50 is a very popular CNN architecture and is widely used to showcase the effectiveness of sparsification techniques. ResNet50 has 54 parameter layers (including fc) and a couple of pooling layers (which contribute minimally to FLOPs). All the batchnorm parameters are left dense and are learnt during the training. STR can be applied per-layer, per-channel and even per-weight to obtain unstructured sparsity and the aggressiveness of sparsification increases in the same order. This paper only uses per-layer STR which makes it have 54 additional learnable scalars. The layer-wise parameters and FLOPs can be seen in Tables 7 and 6. All the layers had no bias terms.

MobileNetV1: MobileNetV1 is a popular efficient CNN architecture. It is used to showcase the generalizability of sparsification techniques. MobileNetV1 has 28 parameter layers (including fc) and a couple of pooling layers (which contribute minimally to FLOPs). All the batchnorm parameters are left dense and are learnt during the training. STR can be applied per-layer, per-channel and even per-weight to obtain unstructured sparsity and the aggressiveness of sparsification increases in the same order. This paper only uses per-layer STR which makes it have 28 additional learnable scalars. The layer-wise parameters and FLOPs can be seen in Tables 8. All the layers had no bias terms.

FastGRNN: FastGRNN's update equations can be found in (Kusupati et al., 2018). FastGRNN, in general, benefits a lot from the low-rank reparameterization and this enables it to be deployed on tiny devices without losing any accuracy. FastGRNN's biases and final classifier are left untouched in all the experiments and only the input and hidden projection matrices are made low-rank. All the hyperparameters were set specific to the datasets as in Kusupati et al. (2018).

A.8. Hard Threshold vs Soft Threshold

Figure 10 shows the difference between hard thresholding and soft thresholding for the same threshold value of $\alpha = 2$. It is clear from Figure 10 that soft-threshold is a continuous function that is sub-differentiable. The abrupt change in hard-threshold leads to instability in training sometimes increasing dependence on fine tuning of the obtained sparse network. Soft-threshold is robust to such issues.

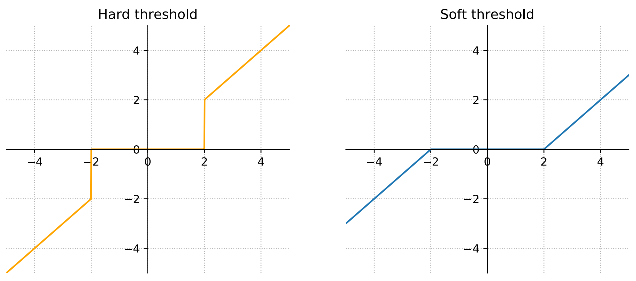

Figure 10. A visualization of hard-threshold (*left*) and soft-threshold (*right*) functions with the threshold $\alpha = 2$. x-axis is the input and y-axis is the output.