1. Training Details

1.1. Hyperparameters

We trained the models for 200 epochs with a starting learning rate (LR) of 0.1 and a cosine annealing LR scheduler [5]. We used the SGD optimizer with a weight decay and momentum value of $5e-4$ and 0.9, respectively. For CIFAR-10, CIFAR-100, and SVHN we used a batch-size of 128 and for STL-10 which has a comparatively larger resolution ($96 \times 96$ as image height/width compared to $32 \times 32$ of others), we used a batch-size of 64. We used half of each image-batch as clean and the remaining half as perturbed (using PGD-7 attack). For test-time random autoattack [1] generation we used $L_\infty$ norm with $\epsilon$ as $8/255$. To reproduce the results with OAT and OATS we used the official repository of [6] and used their recommended $\lambda$ distribution, encoding dimension, sampling scheme as well as $\sigma$ value for attack.

1.2. ImageNet Training

To train on ImageNet dataset on limited compute resources, we first sampled 100 classes from it, with 500 training and 50 test samples from each class, each with resolution of $224 \times 224 \times 3$. We then trained a ResNet18 on the sampled subset of ImageNet with FLOAT for 60 epochs. We used multi step LR scheduler that decayed by a factor of 0.1 respectively at epoch 30 and 45. We used a training batch size of 50 and partitioned each batch equally to create clean and adversarial samples. To create train and test adversarial samples we used similar adversarial strengths as other datasets.

1.3. Models

We used ResNet34 to evaluate CIFAR-10 and CIFAR-100, WRN16-8 to evaluate SVHN, and WRN40-2 to evaluate STL10. Table 1 shows the models’ parameter and FLOP count (contributed by the convolutions and FC layers as they attribute to majority of the FLOPs and parameters).

1.4. Trained Scaling Factors

The trained noise scaling-factor $\alpha^l$ for each layer of a FLOAT model is shown in Fig. 1. The $\alpha^l$ values are generally high at the initial layers of the models while and reduce to near zero at later layers. [3] has also observed a similar trend in trained alpha values while training targeting only robustness.

1.5. Alternate Results Plots in Terms of $\lambda$ vs. Accuracy

To bolster Fig. 7 in the original manuscript, Fig. 2 show an alternate view of the CA-RA comparisons between FLOAT, OAT, and PGD-AT.

1.6. Training Time as a Function of Size of $S_\lambda$ used for training.

The comparison of training times of OAT and FLOAT as a function of different numbers of $\lambda$s is shown in Fig. 3. Because FLOAT always has two possible training $\lambda$s, the time is constant. However, for OAT, the training time increases when supporting more training $\lambda$s. Note, in the
Figure 2. Comparison of trade-off between accuracy and robustness of FLOAT, OAT, and PGD-AT. (a)-(c) and (d)-(f) show CA and RA plots vs different $\lambda$ values, respectively. The values corresponds to the mean $\pm$ std error for each point.

Figure 3. Comparison of per epoch normalized training time between FLOAT and OAT for different number of OAT training $\lambda$s, on (a) CIFAR-10, (b) SVHN, and (c) STL10. Note, lower normalized value implies faster training time, meaning better training efficiency.

2. More Analysis on Pruned Models

We now show per layer parameter density for a specific target global density $d$. In particular, Fig. 4 shows the layer-wise density of non-zero weight. As we can see in the Fig, the earlier layers generally have higher parameter density (for both the densities). This trend is similar to that observed in [4]. Also, earlier literature have used the parameter density as a proxy of the layer sensitivity [4, 2]. Here sensitivity of a layer can be measured by the accuracy reduction caused by pruning a certain ratio of parameters from that layer. Thus, we can use FLOATS as an automated to evaluate layer-sensitivity for a target $d$ eliminating the need for heavy human works to tune the sparsity ratios as hyper-parameters.

Fig. 5 shows the density of non-zero channel present per layer for a model pruned via irregular and channel pruning. Interestingly, as shown in the figure, irregular pruning keeps the channel density close to 1.0 for all the layers, while channel pruning can reduce the density to $\sim0.1$ for some layers. We attribute this large reduction of channels for some layers to the non-significant accuracy drop of the channel pruned models compared to the baseline.

2.1. FLOAT and FLOATS Slim Training Algorithms

Algorithm 1 and 2 describes the training algorithms of FLOAT and FLOATS slim respectively. In particular, FLOATS slim has an additional loop running over the set of SFs that the network can dynamically slim down to during inference.
Figure 5. Per layer channel density for irregular vs channel pruning for the same target parameter density (d) of 0.3. We used WRN40-2 on STL10 for training.

Algorithm 1: FLOAT Algorithm

Data: Training set $X \sim D$, model parameters $\Theta$, trainable noise scaling factor $\alpha$, binary conditioning parameter $\lambda$, mini-batch size $B$.

Output: trained model parameters $\Theta$, $\alpha$.

1. for $i \leftarrow 0$ to $\epsilon_B$ do
   2. for $j \leftarrow 0$ to $n_B$ do
      3. Sample clean image-batch of size $B/2$ ($X_{0:B/2}$, $Y_{0:B/2}$) $\sim D$
      4. $L_C \leftarrow \text{computeLoss}(X_{0:B/2}$, $\Theta$, $\lambda = 0$, $\alpha$; $Y_{0:B/2}$) // condition to use weights w/o noise
      5. $\hat{X}_{B/2:B} \leftarrow \text{createAdv}(X_{B/2:B}$, $Y_{B/2:B})$ // adversarial image
      6. $L_A \leftarrow \text{computeLoss}(\hat{X}_{B/2:B}$, $\Theta$, $\lambda = 1$, $\alpha$; $Y_{B/2:B}$) // condition to use transformed weights
      7. $L \leftarrow 0.5 \ast L_C + 0.5 \ast L_A$
      8. updateParam($\Theta$, $\alpha$, $\nabla L$)
   9. end
10. end
11. end

Algorithm 2: FLOATS slim Algorithm

Data: Training set $X \sim D$, model parameters $\Theta$, trainable noise scaling factor $\alpha$, binary conditioning parameter $\lambda$, mini-batch size $B$, global parameter density $d$, initial mask $\Pi$, prune type (irregular/channel) $t_p$, slimming-factor set $S_f$.

Output: trained model parameters with density $d$.

1. $\Theta \leftarrow \text{applyMask} (\Theta, \Pi)$
2. for $i \leftarrow 0$ to $\epsilon_B$ do
   3. for $j \leftarrow 0$ to $n_B$ do
      4. $B/2 (X_{0:B/2}$, $Y_{0:B/2}) \sim D$
      5. $\hat{X}_{B/2:B} \leftarrow \text{createAdv}(X_{B/2:B}$, $Y_{B/2:B})$
      6. $L_C \leftarrow \text{computeLoss}(X_{0:B/2}$, $\Theta$, $\lambda = 0$, $\alpha$; $Y_{0:B/2}$)
      7. $L_A \leftarrow \text{computeLoss}(\hat{X}_{B/2:B}$, $\Theta$, $\lambda = 1$, $\alpha$; $Y_{B/2:B}$)
      8. $L \leftarrow 0.5 \ast L_C + 0.5 \ast L_A$
      9. accumulateGrad($L$)
   10. end
11. updateLayerMomentum($\mu$)
12. pruneRegrow($\Theta$, $\Pi$, $\mu$, $d$) // Prune fixed % of active and regrow fraction of inactive weights
13. $\Pi \leftarrow \text{updateMask} (\Pi, t_p, \mu)$
14. end

3. Reproducibility Statement

We have detailed our training algorithms for both FLOAT and FLOATS slim in Supplementary Algorithm 1 and 2, respectively. We have further provided detailed hyperparameter and model information in the main manuscript and further additional details in the Supplementary section 1.1. Finally, the code for this project will be made publicly available upon acceptance of the paper.

References


