Supplementary of FLOAT: Fast Learnable Once-for-All Adversarial Training for Tunable Trade-off between Accuracy and Robustness

Souvik Kundu[1,2]*, Sairam Sundaresan[1], Massoud Pedram[2], Peter A. Beerel[2]
[1]Intel Labs, USA [2]Universiy of Southern California, Los Angeles, USA
{souvikk.kundu, sairam.sundaresan}@intel.com {pedram, pabeerel}@usc.edu

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 (96×96 as image height/width compared to 32×32 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_∞ norm with ϵ as $8/255$. To reproduce the results with OAT and OATS we used the official repository of [6] and used their recommended λ distribution, encoding dimension, sampling scheme as well as σ 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).

*Part of the work was done when the first author was with USC.

Table 1. Model parameters and FLOPs details.

Model type	Parameters (M)		FLOPs (G)	
	OAT	FLOAT	OAT	FLOAT
ResNet34	31.4	21.28	1.18	1.165
WRN40-2	3.22	2.25	3	2.96
WRN16-8	15.43	10.97	2.1	2.01

1.4. Trained Scaling Factors

The trained noise scaling-factor α^l for each layer of a FLOAT model is shown in Fig. 1. The α^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.

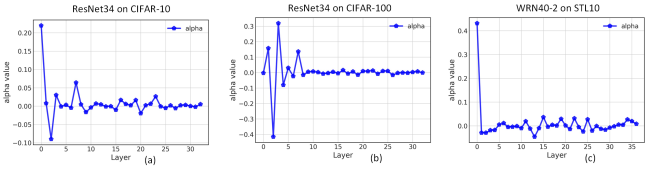

Figure 1. Trained noise scaling factor value (layer-wise) for (a) ResNet34 on CIFAR-10, (b) ResNet34 on CIFAR-100, and (c) WRN40-2 on STL10.

1.5. Alternate Results Plots in Terms of λ 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_λ used for training.

The comparison of training times of OAT and FLOAT as a function of different numbers of λs is shown in Fig. 3. Because FLOAT always has two possible training λs, the time is constant. However, for OAT, the training time increases when supporting more training λs. Note, in the

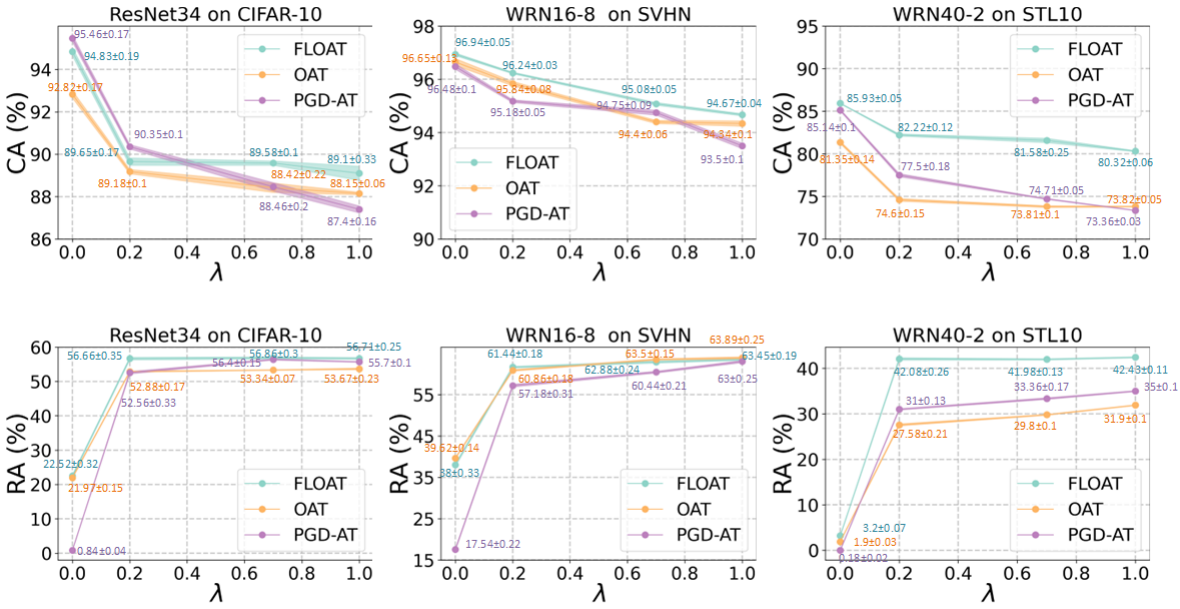

Figure 2. Comparison of trade-off between accuracy and robustness of FLOAT, OAT, and PGD-AT. (a)-(c) and (d)-(f) show CA and RA plots vs different λ values, respectively. The values corresponds to the mean $\pm$ std error for each point.

main manuscript, we report the training time comparisons for OAT training with four training λs (0.0, 0.2, 0.7, 1.0).

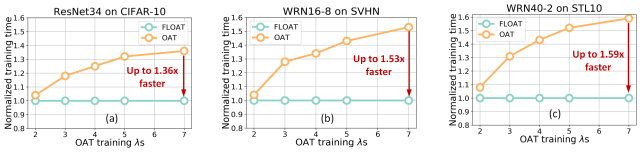

Figure 3. Comparison of per epoch normalized training time between FLOAT and OAT for different number of OAT training λs, on (a) CIFAR-10, (b) SVHN, and (c) STL10. Note, lower normalized value implies faster training time. meaning better training efficiency.

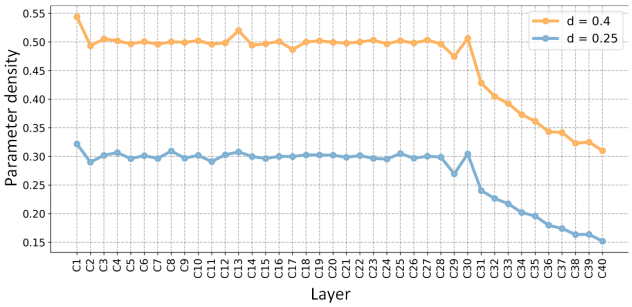

Figure 4. Per layer parameter density plot for two different target parameter density (d) with irregular pruning. We used WRN40-2 on STL10 for training.

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 tread 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 $\sim$0.1 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

Alogirthm 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.

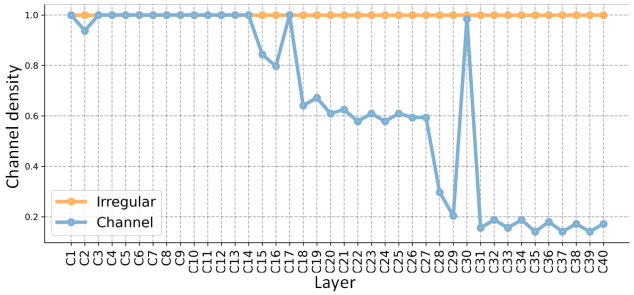

Figure 5. Per layer channel density for irregular vs channel pruning for the same target parameter density (d) of 0.3. We used WRN40-2 on STL10 for training.

Algorithm 1: FLOAT Algorithm

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

Output: trained model parameters $\boldsymbol{\Theta}$, $\boldsymbol{\alpha}$.
for $i \leftarrow 0$ **to to** ep **do**
  for $j \leftarrow 0$ **to** $n_{\mathcal{B}}$ **do**
    Sample clean image-batch of size $\mathcal{B}/2$ ($\boldsymbol{X}_{0:\mathcal{B}/2}$, $\boldsymbol{Y}_{0:\mathcal{B}/2}$) $\sim D$
    $\mathcal{L}_C \leftarrow \texttt{computeLoss}(\boldsymbol{X}_{0:\mathcal{B}/2}, \boldsymbol{\Theta}, \lambda = 0, \boldsymbol{\alpha}; \boldsymbol{Y}_{0:\mathcal{B}/2})$ // condition to use weights w/o noise
    $\hat{\boldsymbol{X}}_{\mathcal{B}/2:\mathcal{B}} \leftarrow \texttt{createAdv}(\boldsymbol{X}_{\mathcal{B}/2:\mathcal{B}}, \boldsymbol{Y}_{\mathcal{B}/2:\mathcal{B}})$ // adversarial image creation
    $\mathcal{L}_A \leftarrow \texttt{computeLoss}(\hat{\boldsymbol{X}}_{\mathcal{B}/2:\mathcal{B}}, \boldsymbol{\Theta}, \lambda = 1, \boldsymbol{\alpha}; \boldsymbol{Y}_{\mathcal{B}/2:\mathcal{B}})$ // condition to use transformed weights
    $\mathcal{L} \leftarrow 0.5 * \mathcal{L}_C + 0.5 * \mathcal{L}_A$
    $\texttt{updateParam}(\boldsymbol{\Theta}, \boldsymbol{\alpha}, \nabla_{\mathcal{L}})$
  end
end

Algorithm 2: FLOATS slim Algorithm

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

, **Output:** trained model parameters with density d.
$\boldsymbol{\Theta} \leftarrow \texttt{applyMask}\,(\boldsymbol{\Theta}, \boldsymbol{\Pi})$
for $i \leftarrow 0$ **to to** ep **do**
  for $j \leftarrow 0$ **to** $n_{\mathcal{B}}$ **do**
    $\mathcal{B}/2\ (\boldsymbol{X}_{0:\mathcal{B}/2}, \boldsymbol{Y}_{0:\mathcal{B}/2}) \sim D$
    for *slimming-factor* s_f *in sorted* S_f **do**
      $\texttt{sampleSBN}(s_f)$ //sample the BN_C and BN_A // corresponding to w
      $\mathcal{L}_C \leftarrow \texttt{computeLoss}(\boldsymbol{X}_{0:\mathcal{B}/2}, \boldsymbol{\Theta}, \lambda = 0, \boldsymbol{\alpha}; \boldsymbol{Y}_{0:\mathcal{B}/2})$
      $\hat{\boldsymbol{X}}_{\mathcal{B}/2:\mathcal{B}} \leftarrow \texttt{createAdv}(\boldsymbol{X}_{\mathcal{B}/2:\mathcal{B}}, \boldsymbol{Y}_{\mathcal{B}/2:\mathcal{B}})$
      $\mathcal{L}_A \leftarrow \texttt{computeLoss}(\hat{\boldsymbol{X}}_{\mathcal{B}/2:\mathcal{B}}, \boldsymbol{\Theta}, \lambda = 1, \boldsymbol{\alpha}; \boldsymbol{Y}_{\mathcal{B}/2:\mathcal{B}})$
      $\mathcal{L} \leftarrow 0.5 * \mathcal{L}_C + 0.5 * \mathcal{L}_A$
      $\texttt{accumulateGrad}(\mathcal{L})$
    end
    $\texttt{updateParam}(\boldsymbol{\Theta}, \boldsymbol{\alpha}, \nabla_{\mathcal{L}}, \boldsymbol{\Pi})$
  end
  $\texttt{updateLayerMomentum}\,(\boldsymbol{\mu})$
  $\texttt{pruneRegrow}\,(\boldsymbol{\Theta}, \boldsymbol{\Pi}, \boldsymbol{\mu}, d)$// Prune fixed % of active and regrow fraction of inactive weights
  $\boldsymbol{\Pi} \leftarrow \texttt{updateMask}\,(\boldsymbol{\Pi}, t_p, \boldsymbol{\mu})$
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

[1] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020.

[2] Xiaohan Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, Ji Liu, et al. Global sparse momentum sgd for pruning very deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[3] Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 588–597, 2019.

[4] Souvik Kundu, Mahdi Nazemi, Peter A Beerel, and Massoud Pedram. Dnr: A tunable robust pruning framework through dynamic network rewiring of dnns. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pages 344–350, 2021.

[5] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[6] Haotao Wang, Tianlong Chen, Shupeng Gui, Ting-Kuei Hu, Ji Liu, and Zhangyang Wang. Once-for-all adversarial training: In-situ tradeoff between robustness and accuracy for free. *arXiv preprint arXiv:2010.11828*, 2020.