FLOAT: FAST LEARNABLE ONCE-FOR-ALL ADVERSARIAL TRAINING FOR TUNABLE TRADE-OFF BETWEEN ACCURACY AND ROBUSTNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training a model that can be robust against adversarially-perturbed images without compromising accuracy on clean-images has proven to be challenging. Recent research has tried to resolve this issue by incorporating an additional layer after each batch-normalization layer in a network that implements feature-wise linear modulation (FiLM). These extra layers enable in-situ calibration of a trained model, allowing the user to configure the desired priority between robustness and clean-image performance after deployment. However, these extra layers significantly increase training time, parameter count, and add latency which can prove costly for time or memory constrained applications. In this paper, we present *Fast Learnable Once-for-all Adversarial Training* (FLOAT) which transforms the weight tensors without using extra layers, thereby incurring no significant increase in parameter count, training time, or network latency compared to a standard adversarial training. In particular, we add configurable scaled noise to the weight tensors that enables a trade-off between clean and adversarial performance. Additionally, we extend FLOAT to slimmable neural networks to enable a three-way in-situ trade-off between robustness, accuracy, and complexity. Extensive experiments show that FLOAT can yield state-of-the-art performance improving both clean and perturbed image classification by up to $\sim 6.5\%$ and $\sim 14.5\%$, respectively, while requiring up to $1.47\times$ fewer parameters with similar hyperparameter settings compared to FiLM-based alternatives. Code for this project will be made available shortly.

1 INTRODUCTION

As artifical intelligence (AI)-enabled applications proliferate, the robustness concerns of deep neural networks (DNNs) has grown in importance Zhang et al. (2019). These concerns are largely associated with the vulnerability of machine learning (ML) classifiers to *adversarial examples*, i.e., images that have well-crafted barely-visible perturbations from corresponding clean images and have the ability to fool classifier models into making wrong predictions Carlini & Wagner (2017); Goodfellow et al. (2014). With the growing usage of DNNs in safety-critical and sensitive applications including autonomous-driving Bojarski et al. (2016) and medical image analysis Han et al. (2021), it therefore becomes crucial that they have high accuracy on both adversarially-perturbed and clean images. To improve the model performance of DNNs against such adversarial attacks, various defense mechanisms have been proposed including hiding gradients Tramèr et al. (2017), adding noise to parameters He et al. (2019), and detection of adversaries Meng & Chen (2017). In particular, adversarial training Madry et al. (2017) has proven to be a consistently effective approach.

These defenses, however, come at various costs. Firstly, most of these methods suffer from increased training time due to the additional back-propagation overhead to generate perturbed images. Secondly, adversarial defenses sometimes cause a significant drop in clean-image accuracy Tsipras et al. (2018), highlighting an accuracy-robustness trade-off that has been explored both theoretically and experimentally Sun et al. (2019); Tsipras et al. (2018); Schmidt et al. (2018). Moreover, the defenses rely on several hyperparameters whose settings force the model to work at a specific point along this trade-off. This is disadvantageous in applications in which the desired trade-off depends on context Wang et al. (2020).

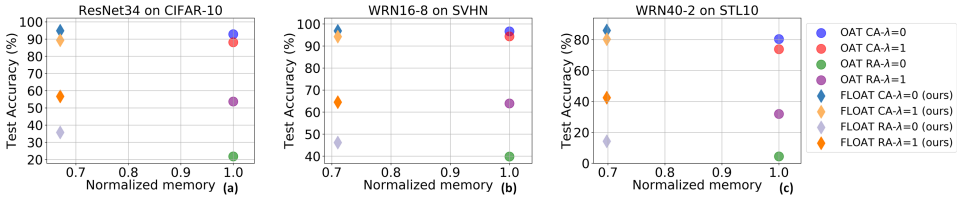

Figure 1: Normalized memory vs. test accuracy of FLOAT with existing state-of-the-art OAT for (a) ResNet34, (b) WRN16-8, and (c) WRN40-2, respectively. CA and RA represent clean-image classification accuracy and robust accuracy (accuracy on adversarial images), respectively. For each dataset we normalized the memory requirement with the maximum model memory needed for that.

A naive solution to this problem is to use multiple networks trained with different priorities between clean and adversarial images. However, this increases both training time and inference memory. Alternatively, recent work has proposed training a once-for-all adversarial network (OAT) that supports *conditional learning* Wang et al. (2020), enabling the network to adjust to different input distributions. In particular, after each batch-normalization (BN) layer, they add a feature-wise linear modulation (FiLM) module Perez et al. (2018) whose weights are controlled by a parameter λ. For inference, the user sets λ to enable an in-situ trade-off between accuracy and robustness. The disadvantage with this approach is that the added FiLM modules increase the parameter count, training time, and network latency, limiting applicability in resource-constrained, real-time applications.

Our contributions. In view of the concerns above, we present *fast learnable once-for-all adversarial training* (FLOAT). In FLOAT, we train a model using a novel mechanism wherein each weight tensor of the model is transformed by conditionally adding a noise tensor based on a binary parameter λ. This yields state-of-the-art (SOTA) test accuracy for clean and adversarial images by in-situ setting $\lambda = 0.0$ and 1.0, respectively. For inference, we further show that model robustness can be correlated to the strength of the noise-tensor scaling factor. This motivates a simple yet effective noise re-scaling approach controlled by an added floating-point parameter that can help the user to have a practical accuracy-robustness trade-off.

Because FLOAT does not require additional layers to perform conditioning, it incurs no increase in latency and causes only a negligible increase in parameter count compared to the baseline models. Moreover, compared to OAT, FLOAT training is up to $1.43\times$ faster, as it does not explicitly require learning intermediate fine-grained conditioning using different λs.

Moreover, we show how FLOAT can be extended to slimmable networks Yu et al. (2018) to enable an in-situ inference trade-off across three dimensions, namely, accuracy, robustness, and complexity.

To evaluate the merits of FLOAT, we conduct extensive experiments on CIFAR-10, CIFAR-100, Tiny-ImageNet, SVHN, and STL10 datasets with ResNet34 (on both CIFAR and Tiny-ImageNet datasets), WRN16-8, WRN40-2, respectively. As shown in Fig. 1, compared to existing methods, FLOAT models can provide improved accuracy of up to $\sim 6.5\%$, and $\sim 14.5\%$, on clean and perturbed images, respectively.

The remainder of this paper is organized as follows. Section 2 presents the necessary preliminaries and prior work. We present our approach in Section 3 and analyze experimental results in Section 4. Finally, the paper concludes in Section 5 with discussions of broader impact in Section 6.

2 Preliminaries

2.1 Notation

Consider a model Φ with L layers parameterized by $\boldsymbol{\Theta}$ that learns a function $f_\Phi(.)$. For a classification task on dataset $\boldsymbol{X}$ with distribution D, the model parameters $\boldsymbol{\Theta}$ are learned by minimizing the empirical risk (ERM) as follows

$$\underset{\boldsymbol{\Theta}}{\text{minimize}}\, \mathcal{L}(f_\Phi(\boldsymbol{x}, \boldsymbol{\Theta}; t)), \tag{1}$$

where t is the ground-truth class label, $\boldsymbol{x}$ is the vectorized input drawn from $\boldsymbol{X}$, and $\mathcal{L}$ is the cross-entropy loss function.

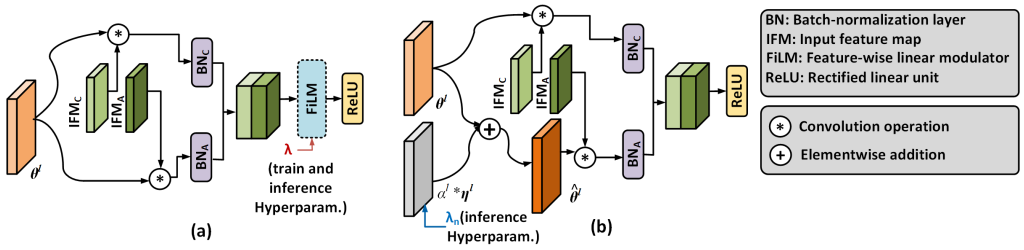

Figure 2: Comparison of a conditional layer between (a) existing FiLM based approach in OAT and (b) proposed approach in FLOAT.

2.2 Robust Model Training

Several forms of adversarial training (AT) have been proposed to improve robustness Madry et al. (2017); Samangouei et al. (2018); Buckman et al. (2018). They all use both clean and adversarially-perturbed images to train a model. Projected gradient descent (PGD) attack, recognized as one of the strongest L_∞ adversarial example generation algorithm Madry et al. (2017), is typically used to create adversarial images during training. The perturbed image for a PGD-k attack with k as the number of steps is given by,

$$\hat{\boldsymbol{x}}^k = \text{Proj}_{P_\epsilon(\boldsymbol{x})}(\hat{\boldsymbol{x}}^{k-1} + \sigma \times sign(\nabla_x \mathcal{L}(f_\Phi(\hat{\boldsymbol{x}}^{k-1}, \boldsymbol{\Theta}; t))) \tag{2}$$

Here, the scalar ϵ corresponds to the perturbation constraint that determines the severity of the perturbation. Proj projects the updated adversarial sample onto the projection space $P_\epsilon(\boldsymbol{x})$ which is the ϵ-L_∞ neighbourhood of the benign sample $\boldsymbol{x}$[1]. σ is the attack step-size. For PGD-AT, the model parameters are then learned by the following ERM

$$\underset{\boldsymbol{\Theta}}{\text{minimize}}[\underbrace{(1-\lambda)\mathcal{L}(f_\Phi(\boldsymbol{x}, \boldsymbol{\Theta}; t))}_{\mathcal{L}_C} + \underbrace{\lambda\mathcal{L}(f_\Phi(\hat{\boldsymbol{x}}, \boldsymbol{\Theta}; t))}_{\mathcal{L}_A}], \tag{3}$$

where $\mathcal{L}_C$ and $\mathcal{L}_A$ corresponds to the clean and adversarial image classification loss components, respectively, weighted by the scalar λ. Hence, for a fixed λ and adversarial strength, the model learns a fixed tradeoff between accuracy and robustness. For example, an AT with λ value of 1 will allow the model to completely focus on perturbed images, resulting in a significant drop in clean-image classification accuracy. Another strategy to improve model robustness is through the addition of noise to the model weight tensors. For example, He et al. (2019) introduced the idea of noisy weight tensors with a learnable noise scaling factor to improve model robustness against gradient-based attacks. However, this strategy also incurs a significant drop in clean accuracy.

2.3 Conditional Learning

Conditional learning involves training a model with multiple computational paths that can be selectively enabled during inference Wang et al. (2018). For example, Teerapittayanon et al. (2016); Huang et al. (2017); Kaya et al. (2019) enhanced a DNN model with multiple early exit branches at different architectural depths to allow early predictions of various inputs. Yu et al. (2018) introduced switchable BNs that enable the network to adjust the channel widths dynamically, providing a in-situ efficient trade-off between complexity and accuracy. Recently, Bulat & Tzimiropoulos (2021) used switchable BNs to support runtime bit-width selection of a mixed-precision network. Another conditional learning approach used feature transformation to modulate intermediate DNN features Huang & Belongie (2017); Yang et al. (2019); De Vries et al. (2017); Wang et al. (2020). In particular, Wang et al. (2020) used FiLM Perez et al. (2018) to adaptively perform a channel-wise affine transformation after each BN stage that is controlled by the hyperparameter λ of Equation 3. Each FiLM module is composed of two fully-connected (FC) layers with leaky ReLU activation functions and dimensions that match the output feature-map channel size. Despite requiring a relatively small number of additional FLOPs, however, the FiLM module can significantly increase the number of model parameters and associated memory access cost Horowitz (2014). Moreover, the increased number of layers can significantly increase inference latency Singh et al. (2019), potentially prohibiting its use in real-time applications.

[1]Note that the generated $\hat{\boldsymbol{x}}$ are clipped to a valid range which, for our experiments, is $[0, 1]$.

3 PROPOSED APPROACH

3.1 FLOAT

In this section, we disclose our fast learnable once-for-all adversarial training (FLOAT). We first focus on defining the conditions for a model being trained on either clean or adversarial images, as the two *boundary conditions* for training. We use a binary conditioning parameter λ, that during training, forces the model to focus on either of these two conditions. To formalize our approach, consider a L-layer DNN parameterized by $\boldsymbol{\Theta}$ and let $\boldsymbol{\theta}^l \in \mathbb{R}^{k^l \times k^l \times c_i^l \times c_o^l}$ represent the layer l weight tensor, where c_o^l and c_i^l represent the number of filters and channels per filter, respectively, and k^l represents the kernel height/width. We transform each parameter of $\boldsymbol{\theta}^l$, by adding a noise tensor $\boldsymbol{\eta}^l \in \mathbb{R}^{k^l \times k^l \times c_i^l \times c_o^l}$ scaled by a parameter α^l and conditioned by λ, as follows,

$$\hat{\theta}^l = \theta^l + \lambda \cdot \alpha^l \cdot \eta^l; \quad \eta^l \sim \mathcal{N}(0, (\sigma^l)^2). \tag{4}$$

Note that the standard deviation (σ^l) of the noise matches that of its weight tensor. For $\lambda = 0$ and 1, we obtain the original weight tensor or a noisy variant, respectively.

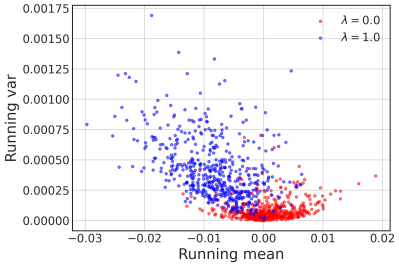

Figure 3: Comparison of the normalization statistics for a ResNet34 trained on CIFAR-10 with two different λs. We plot the distribution by taking the running mean and running variance statistics from the last BN layer placed before the FC layers.

As illustrated in Algorithm 1, we train our models by partitioning an image batch $\mathcal{B}$ into two equal sub-batches $\mathcal{B}_1$ and $\mathcal{B}_2$, one with clean (IFM_C) images and the other with perturbed variants (IFM_A) (lines 4 and 6 in Algorithm 1). We use the PGD-7 attack to generate perturbations on the image batch $\mathcal{B}_2$. As illustrated in Fig. 2(b), the original and noisy weight tensors are convolved only with clean and perturbed variants, respectively. Similar to He et al. (2019), we also train the noise scaling factor α^l (line 9). As exemplified in Fig. 3, the post-convolution feature maps for clean and adversarial inputs can differ significantly in their respective mean/variances Xie et al. (2020); Xie & Yuille (2019). Therefore, the use of a single BN to learn both distributions may limit model performance Wang et al. (2020). To solve this problem, we extend the λ-conditioning to choose between two BNs, BN_C and BN_A, dedicated for IFM_C and IFM_A, respectively.

This approach differs from previous efforts in several ways. Earlier research performed noise-injection via regularization Bietti et al. (2018); Lecuyer et al. (2019) and perturbed weight tensors He et al. (2019) to boost model robustness at the cost of a significant accuracy drop on clean images. In contrast, we use noise tensors to transform a shared weight tensor yielding a model that can be configured in-situ to provide SOTA accuracy on either clean or perturbed images. Our approach is similar to λ-conditioning used by Wang et al. (2020). However, instead of transforming activations using added FiLM-based layers trained with multiple values of λ Wang et al. (2020), we transform weight tensors using added noise conditioned by binary λ. Compared to Wang et al. (2020), we thus require models with significantly fewer parameters and training scenarios, yielding faster training (up to $1.43\times$).

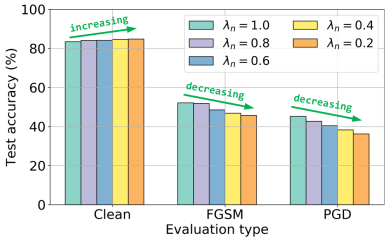

Figure 4: Post-training model performance on both clean and gradient-based attack-generated adversarial images, with different noise re-scaling factor λ_n.

3.2 FLOAT GENERALIZATION WITH NOISE RE-SCALING

One limitation of the FLOAT approach as proposed above is that it allows the user to choose between two boundary conditions only. This limits applicability when the user is not confident about which condition to use during inference. To motivate more continuous in-situ conditioning, we analyze a ResNet20 model with noisy weight tensors trained with PGD-AT on CIFAR-10 He et al. (2019). Post-training, we re-scaled α^l for each layer l, using a new floating-point parameter λ_n to yield $\lambda_n \cdot \alpha^l$. Interestingly, as shown in Fig. 4, as the re-scaling factor decreases, the model robustness decreases and the clean-image accuracy increases.

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 $\mathcal{B}/2$ $(\boldsymbol{X}_{0:\mathcal{B}/2}, \boldsymbol{Y}_{0:\mathcal{B}/2}) \sim D$
    $\mathcal{L}_C \leftarrow \text{computeLoss}(\boldsymbol{X}_{0:\mathcal{B}/2}, \boldsymbol{\Theta}, \lambda = 0, \boldsymbol{\alpha}; \boldsymbol{Y}_{0:\mathcal{B}/2})$ // condition to use weights w/o noise
    $\hat{\boldsymbol{X}}_{\mathcal{B}/2:\mathcal{B}} \leftarrow \text{createAdv}(\boldsymbol{X}_{\mathcal{B}/2:\mathcal{B}}, \boldsymbol{Y}_{\mathcal{B}/2:\mathcal{B}})$ // adversarial image creation
    $\mathcal{L}_A \leftarrow \text{computeLoss}(\hat{\boldsymbol{X}}_{\mathcal{B}/2:\mathcal{B}}, \boldsymbol{\Theta}, \lambda = 1, \boldsymbol{\alpha}; \boldsymbol{Y}_{\mathcal{B}/2:\mathcal{B}})$ // condition to use transformed weights
    $\mathcal{L} \leftarrow 0.5 * \mathcal{L}_C + 0.5 * \mathcal{L}_A$
    $\text{updateparam}(\boldsymbol{\Theta}, \boldsymbol{\alpha}, \nabla_{\mathcal{L}})$
  end
end

Based on this observation, we enable practical post-training in-situ calibration by adding the re-scaling parameter λ_n to the inference model[2]. This allows us to enable a practical accuracy-robustness trade-off in FLOAT during inference. We define a threshold λ_{th} such that for $\lambda_n > \lambda_{th}$ we select BN_A to perform inference and select BN_C otherwise. Wang et al. (2020) selected BN_C and BN_A when $\lambda = 0$ and $\lambda > 0$, respectively. We follow a similar approach by setting $\lambda_{th} = 0$.

3.3 FLOAT Extension to dynamically switchable models

As mentioned earlier, deployed DNN models, particularly on edge devices, can face dynamically changing resource budgets Lou et al. (2021). To better adjust to instantaneous budget changes, earlier research has proposed the idea of slimmable neural networks Yu et al. (2018) in which each layer's channel width can be dynamically shrunk. Let S_f be the predefined set of supported slimming-factors. For each of the floating-point slimming-factors $w (w \in (0, 1])$ in S_f, the model shrinks each convolution layer by w using a different set of BNs.

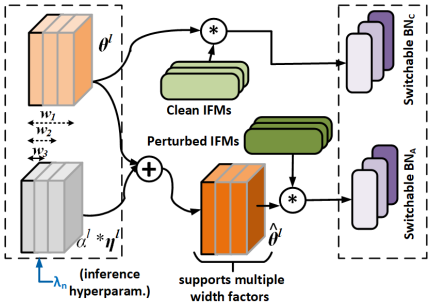

Figure 5: Modified conditional layer of FLOAT to support dynamic complexity switch in FLOATS.

To extend FLOAT to account for dynamically changing resource budgets, we herein present FLOAT Slim (FLOATS) to yield models that, in-situ, can trade-off between accuracy, robustness, and efficiency. Yu et al. (2018) introduced the idea of switchable BNs by having a dedicated BN for each slimming factor. Similarly, we create a dedicated set of BN_A and BN_C for each slimming factor in S_f. In tandem, we scale the noise tensor along with the weight tensors by the same factor to ensure the transformed noisy weights follow the same scaling. Fig. 5 shows an example of a slimmable layer of FLOATS with $S_f = 3$ and slimming factors w_1, w_2, and w_3, where $w_1 = 1.0$ and $1 > w_2 > w_3 > 0$ that operate on the clean and perturbed feature maps of scaled widths generated by the previous layer. Algorithm for FLOATS training is detailed in the Appendix.

4 Experimental Results and Analysis

4.1 Experimental Setup

Models and datasets. To evaluate the efficacy of FLOAT, we performed detailed experiments on five popular datasets, CIFAR-10, CIFAR-100 Krizhevsky et al. (2009), Tiny-ImageNet Hansen (2015) with ResNet34 He et al. (2016), SVHN Netzer et al. (2011) with WRN16-8 Zagoruyko

[2]During inference, we allow the user to control the trade-off using λ_n, since λ is binary in FLOAT. Also, $\lambda_n = 0$ and $\lambda_n = 1$, matches the training boundary conditions. OAT, on the other hand, can reuse the variable λ during both training and inference as it can assume any floating point value in $[0, 1]$.

& Komodakis (2016), and STL10 Coates et al. (2011) with WRN40-2 Zagoruyko & Komodakis (2016).

Hyperparameters and training settings. In order to facilitate a fair comparison, for CIFAR-10, SVHN, and STL10 we used similar hyperparameter settings as Wang et al. (2020)[3]. For CIFAR-100, we followed same hyperparameter settings as that with CIFAR-10. For Tiny-ImageNet we trained the model for 120 epochs with an initial learning rate of 0.1 with cosine decay. For adversarial image generation, we used the PGD-k attack with ϵ and k set to 8/255 and 7, respectively, for all experiments. We initialized the noise scaling-factor α^l for layer l to 0.25 as described in He et al. (2019). Further hyperparameter and model details are provided in the Appendix A.1 and A.2, respectively. We used the PyTorch API Paszke et al. (2017) to implement our models and trained them on a Nvidia GTX Titan XP GPU.

Evaluation metrics. Clean (standard) accuracy (CA): classification accuracy on the original clean test images. Robust Accuracy (RA): classification accuracy on adversarially perturbed images generated from the original test set. We use RA as the measure of robustness of a model. To directly evaluate the robustness vs accuracy trade-off, we evaluate the clean and robust accuracy values of models generated through FLOAT at various λ values compared with those yielded through OAT and PGD-AT. We used the average of the best CA and RA values over three different runs with varying random seeds, for each λ values to report in our results.

4.2 Performance of FLOAT

Sampling λ_n. Unless stated otherwise, to evaluate the performance of FLOAT during validation we chose a set of λ_ns as $S_{\lambda_n} = \{0.0, 0.2, 0.7, 1.0\}$. Note that setting λ_n to 0.0 or 1.0 corresponds to the values of λ used during training. Also, we measure the accuracy of FLOAT using two different settings of λ_{th}, 0.0 (similar to OAT) and 0.5. For $\lambda_{th} = 0.5$, we update the noise scaling factor by using the following simple equation

$$\alpha^l_{new} = \begin{cases} \alpha^l \cdot 2 \cdot \lambda_n; \text{ if } \lambda_n \leq 0.5 \\ \alpha^l \cdot 2 \cdot (\lambda_n - 0.5); \text{ if } 0.5 < \lambda_n \leq 1.0 \end{cases} \tag{5}$$

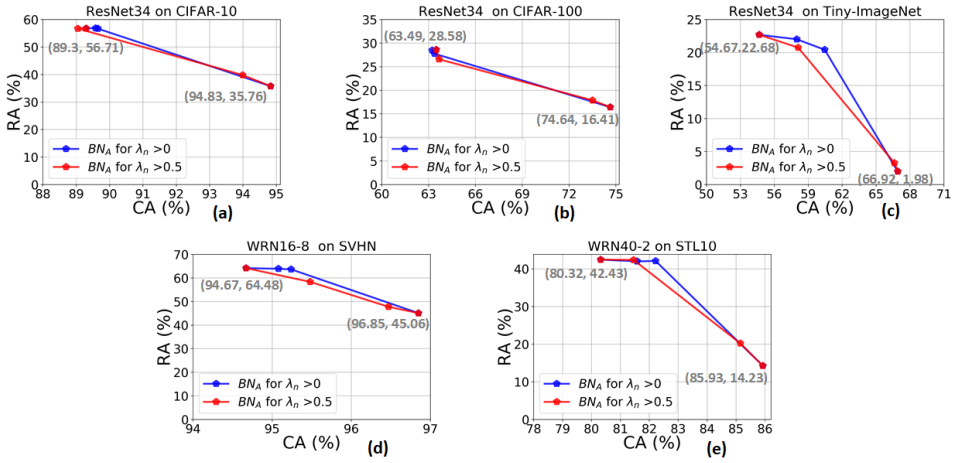

Figure 6: Performance of FLOAT on (a) CIFAR-10, (b) CIFAR-100, (c) Tiny-ImageNet, (d) SVHN, and (e) STL10 with various λ_n values sampled from S_{λ_n} for two different λ_{th} for BN_C to BN_A switching. The numbers in the bracket corresponds to (CA, RA) for the boundary conditions of $\lambda = 0$ and $\lambda = 1$. λ_n varies from largest to smallest value from top-left to bottom-right point.

As depicted in Fig. 6 (a)-(e), the FLOAT models generalize well to yield a semi-continuous accuracy-robustness trade-off. Also, across all the datasets, $\lambda_{th} = 0.5$ yields a more gradual transition between the two boundary conditions. Consider the setting where $\lambda_n = 0.2$. With $\lambda_{th} = 0.5$, we observe a 4.95% improvement in CA and a reduction in RA of 12.37% on average over all five

[3]We followed the official repository https://github.com/VITA-Group/Once-for-All-Adversarial-Training

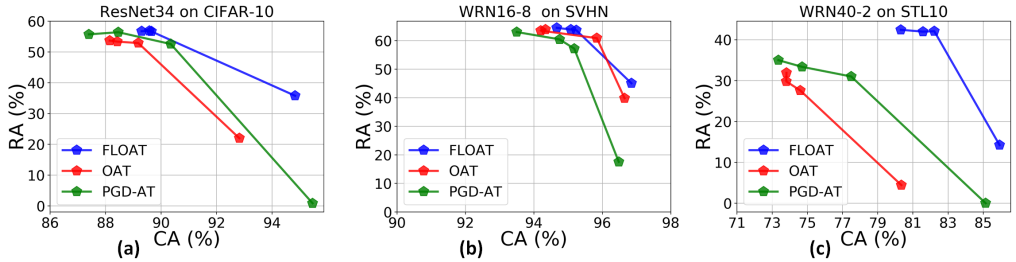

Figure 7: Performance comparison of FLOAT with OAT and PGD-AT generated models on (a) CIFAR10, (b) SVHN, and (c) STL10. λ varies from largest to smallest value in S_λ for the points from top-left to bottom-right.

datasets when compared with $\lambda_{th} = 0.0$. The improvement in clean accuracy here can be attributed to the use of BN_C. However, this configuration shows a drop in CA and an improvement in RA when compared to the configuration where $\lambda_n = 0.0$. This can be attributed to the use of noisy weights (refer to Eq. 5) during inference. Thus, it can be concluded that a user who cares more about clean image performance than adversarial robustness should set $\lambda_{th} > 0.0$ to see a less abrupt drop in CA. Note that, because the generation of adversarial images is noisy, it is not always true that increasing λ will always significantly improve robustness. Consequently, in some cases, we obtain improved clean image performance without a significant drop in robustness.

4.3 Comparison with OAT and PGD-AT

We trained the models by following OAT and PGD-AT with λs sampled from a set $S_\lambda = S_{\lambda_n}$ on three datasets, CIFAR-10, SVHN, and STL10.

Discussion on CA-RA trade-off. Fig. 7(a)-(c) show the comparison of FLOAT with OAT and PGD-AT in terms of CA-RA trade-offs. The FLOAT models show similar or superior performance at the boundary conditions as well as at intermediate sampled values of λ. In particular, compared to OAT and PGD-AT models, FLOAT models can provide an improved RA of up to 14.5% (STL10, $\lambda = 0.2$) and 34.92% (CIFAR-10, $\lambda = 0.0$), respectively. FLOAT also provides improved CA of up to 6.5% (STL10, $\lambda = 1.0$) and 6.96% (STL10, $\lambda = 1.0$), compared to OAT and PGD-AT generated models, respectively. Interestingly, for both FLOAT and OAT, in all the plots we generally see a sharp drop in robustness while moving from top-left to bottom-right. This can be attributed to to the switch from BN_A to BN_C based on the λ_{th}, in the forward pass of the inference model.

Discussion on training time and inference latency. Due to the presence of the additional FiLM

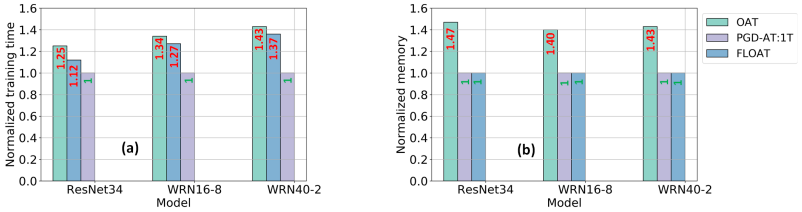

Figure 8: Comparison of FLOAT with OAT and PGD-AT in terms of (a) normalized training time per epoch and (b) model parameter storage (neglecting the storage cost for the BN and $\boldsymbol{\alpha}$). Note here, PGD-AT:1T yields 1 model for a specific λ choice.

modules, OAT requires more time than standard PGD-AT to train. However, a single PGD-AT training can only provide a fixed accuracy-robustness trade-off. For example, to have trade-off with 4 different λs PGD-AT training time increases proportionally by a factor of 4. FLOAT, on the contrary, due to absence of additional layers, trains faster than OAT. In particular, Fig. 8(a) shows the normalized per-epoch training time (averaged over 200 epochs) of OAT and PGD-AT are, respectively, up to $1.43\times$ and $1.37\times$ slower than FLOAT. More results on FLOAT-OAT training time is provided in Appendix A.6.

Network latency increases with the increase in the number of layers for both standard and mobile GPUs Li et al. (2021); Singh et al. (2019), primarily because layers are operated on sequentially

Singh et al. (2019). The additional FiLM modules in OAT significantly increase the layer count. For example, for each bottleneck layer in ResNet34, OAT requires two FiLM modules, yielding a total of four additional FCs per bottleneck. On the other hand, FLOAT, similar to a single PGD-AT trained model, requires no additional layers or associated latency, making it more attractive for real-time applications.

Discussion on model parameter storage cost. Unlike OAT, where the FiLM layer FCs significantly increase the parameter count, the additional BN layers and scaling factors of FLOAT represent a negligible increase in parameter count. In particular, assuming parameters are represented with 8-bits, a FLOAT ResNet34 has only 21.28 MB memory cost compared to 31.4MB for OAT. Fig. 8(b) shows that FLOAT models, similar to PGD-AT:1T, can yield up to $1.47\times$ lower memory.

Discussion on FLOPs. Compared to the standard PGD-AT, FLOAT incurs additional compute cost of addition of noise with the weight tensor during forward pass. For example, for ResNet34 with $\sim$21.28 M parameters, FLOAT needs similar number of additions for noisy weight transformation. However, compared to the total operations of $\sim$1.165 GFLOPs, the transformation adds on 1.182% additional computation. Moreover, as a single addition can be up to $32\times$ cheaper than a single FLOP Horowitz (2014), we can gracefully ignore such transformation cost in terms of FLOPs. OAT, on the other hand, also incurs negligibly less FLOPs overhead of up to only $\sim 1.7\%$ Wang et al. (2020).

4.4 Performance of FLOATS

To generate results with FLOATS we sample two representative slimming-factors 0.5 and 1.0 for training and validations with ResNet34 on CIFAR-10. We further trained a model with OATS to support above mentioned slimming-factors. As depicted in Fig. 9, FLOATS provides significantly improved performance through-out the sampling λ trade-off for both the slimming-factors. In particular, a FLOATS model can provide an improved CA and RA of up to 3.71% and 14.5%, respectively, compared to a model trained with OATS. Moreover, FLOATS enjoys $1.90\times$ faster training time compared to OATS along with the reduced latency and param. requirement as mentioned earlier.

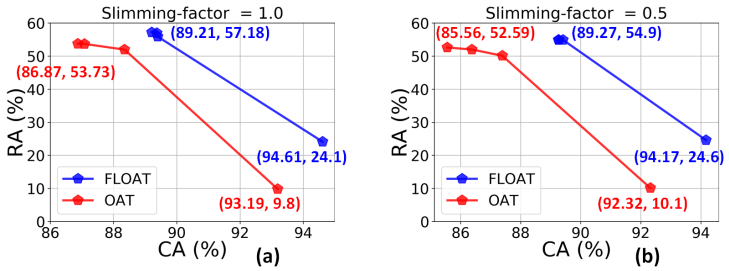

Figure 9: Performance comparison of FLOATS with OATS on slimming-factor of (a) 1.0 and (b) 0.5. We used ResNet34 on CIFAR-10 to evaluate the performance. λ varies from largest to smallest value in S_λ for the points from top-left to bottom-right.

4.5 Generalization on Various Perturbation Techniques

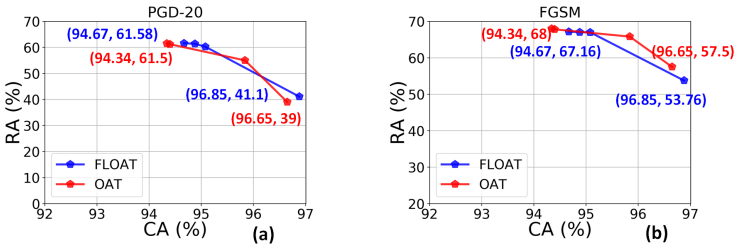

Figure 10: Performance comparison of FLOAT with OAT on (a) PGD20 and (b) FGSM attack generated images. We used WRN16-8 on SVHN to evaluate the performance. λ varies from largest to smallest value in S_λ for the points from top-left to bottom-right.

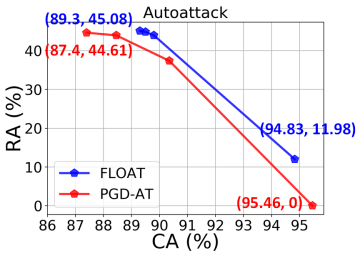

Figure 11: Performance comparison on autoattack with ResNet34 on CIFAR-10.

To demonstrate the generalization ability of FLOAT models on different attacks, we show their performance on images adversarially-perturbed through PGD-20 and FGSM attacks. We follow Wang et al. (2020) to generate the PGD-20 perturbations and use the number of steps as 20, keeping other hyperparameters the same as PGD-7. For FGSM, we make $\epsilon = 8/255$ following Wang et al. (2020). As shown in Fig. 10, under both the attacks, FLOAT can achieve in-situ accuracy-robustness trade-offs similar to that with OAT. To further demonstrate FLOAT's generalization ability on even stronger attacks, we have also analyzed its robustness with an ensemble of parameter-free attacks, namely autoattack Croce & Hein (2020). In particular, we have used the 'random' variant of the autoattack[4]. Details of the autoattack hyperparameter is provided in the Appendix. As depicted in 11, compared to the PGD-AT yielded models, FLOAT model consistently provide better RA with similar or improved CA.

5 Conclusions

This paper aims at addressing a largely unexplored problem of achieving in-situ inference trade-offs between accuracy, robustness, and complexity. We propose a fast learnable once-for-all adversarial training (FLOAT) which uses a novel form of model conditioning to capture the different feature-map distributions corresponding to clean and adversarial images. FLOAT transforms its weights using conditionally added scaled noise and dual batch normalization structures to distinguish between clean and adversarial images. The approach avoids increasing the layer count, unlike other state of the art alternatives, and thus does not suffer from increased network latency. We further show how FLOAT can be coupled with recently proposed slimmable networks to extend the in-situ trade-off to include model complexity. Extensive experiments show FLOAT's superiority in terms of improved CA-RA performance, reduced parameter count, and faster training time.

6 Broader Impact

DNNs are well-known to be susceptible to adversarial images Szegedy et al. (2013). As their use grows in various safety-critical applications, including autonomous driving Bojarski et al. (2016), medical imaging Han et al. (2021) and household robotics Tritschler (2021) achieving model robustness without sacrificing clean image accuracy is increasingly important. This is particularly important for scenarios that can change frequently such as in the case of a household robot. Moreover, the increasingly portable nature of these AI-enabled devices introduces stringent storage and energy-budget limitations. To address these challenging problems, this paper presents FLOAT models which can perform in-situ configuration of the DNNs and allow an application to dynamically adjust the model's accuracy, robustness, and complexity based on the context and scenario. Our approach does not require iterative (re-)training as with standard PGD-AT and trains significantly faster than SOTA alternatives. Moreover, for inference-only devices, our approach circumvents the need for re-loading alternate model parameters from the cloud to support different scenarios, avoiding the potentially significant data-transfer costs. With the increase of high-stake real-world applications that require robustness, we believe this research will form the foundation for practical AI-driven applications that can efficiently adapt to their environment. Finally, we hope this paper will motivate further work aimed at enhancing the continuity of the accuracy-robustness trade-off and developing a theoretical basis that can explain the benefits and limitations of FLOAT and conditional learning in general.

7 Reproducibility

We have detailed our training algorithms for both FLOAT and FLOATS in Algorithm 1 and 2, respectively. We have further provided detailed hyperparameter and model information in Section

[4]We have followed the official repo https://github.com/fra31/auto-attack, to generate the attack.

4.1 as well as in the Appendix A.1. Finally, the code for this project will be made publicly available shortly.

REFERENCES

Alberto Bietti, Grégoire Mialon, and Julien Mairal. On regularization and robustness of deep neural networks. 2018.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

Adrian Bulat and Georgios Tzimiropoulos. Bit-mixer: Mixed-precision networks with runtime bit-width selection. *arXiv preprint arXiv:2103.17267*, 2021.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pp. 39–57. IEEE, 2017.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

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

Harm De Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron Courville. Modulating early visual processing by language. *arXiv preprint arXiv:1707.00683*, 2017.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Tianyu Han, Sven Nebelung, Federico Pedersoli, Markus Zimmermann, Maximilian Schulze-Hagen, Michael Ho, Christoph Haarburger, Fabian Kiessling, Christiane Kuhl, Volkmar Schulz, et al. Advancing diagnostic performance and clinical usability of neural networks via adversarial training and dual batch normalization. *Nature Communications*, 12(1):1–11, 2021.

Lucas Hansen. Tiny ImageNet challenge submission. *CS 231N*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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*, pp. 588–597, 2019.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14. IEEE, 2014.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1501–1510, 2017.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning*, pp. 3301–3310. PMLR, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 656–672. IEEE, 2019.

Zhengang Li, Geng Yuan, Wei Niu, Pu Zhao, Yanyu Li, Yuxuan Cai, Xuan Shen, Zheng Zhan, Zhenglun Kong, Qing Jin, et al. Npas: A compiler-aware framework of unified network pruning and architecture search for beyond real-time mobile acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14255–14266, 2021.

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

Wei Lou, Lei Xun, Amin Sabet, Jia Bi, Jonathon Hare, and Geoff V Merrett. Dynamic-ofa: Runtime dnn architecture switching for performance scaling on heterogeneous embedded platforms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3110–3118, 2021.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 135–147, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.

Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *arXiv preprint arXiv:1804.11285*, 2018.

Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P Namboodiri. Hetconv: Heterogeneous kernel-based convolutions for deep cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4835–4844, 2019.

Ke Sun, Zhanxing Zhu, and Zhouchen Lin. Towards understanding adversarial examples systematically: Exploring data size, task and model factors. *arXiv preprint arXiv:1902.11019*, 2019.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

Charlie Tritschler, Sep 2021. URL `https://www.aboutamazon.com/news/devices/meet-astro-a-home-robot-unlike-any-other`.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

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.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 409–424, 2018.

Cihang Xie and Alan Yuille. Intriguing properties of adversarial training at scale. *arXiv preprint arXiv:1906.03787*, 2019.

Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 819–828, 2020.

Shuai Yang, Zhangyang Wang, Zhaowen Wang, Ning Xu, Jiaying Liu, and Zongming Guo. Controllable artistic text style transfer via shape-matching gan. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4442–4451, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*, pp. 7472–7482, 2019.

A Appendix

A.1 Hyperparameters

We trained the models for 200 epochs with a starting learning rate (LR) of 0.1 and a cosine annealing LR scheduler Loshchilov & Hutter (2016). 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 Croce & Hein (2020) generation we used L_∞ norm with ϵ as $8/255$. To reproduce the results with OAT and OATS we used the official repository of Wang et al. (2020) and used their recommended λ distribution, encoding dimension, sampling scheme as well as σ value for attack.

A.2 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).

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

A.3 Trained Scaling Factors

The trained noise scaling-factor α^l for each layer of a FLOAT model is shown in Fig. 12. The α^l values are generally high at the initial layers of the models while and reduce to near zero at later layers. He et al. (2019) has also observed a similar trend in trained alpha values while training targeting only robustness.

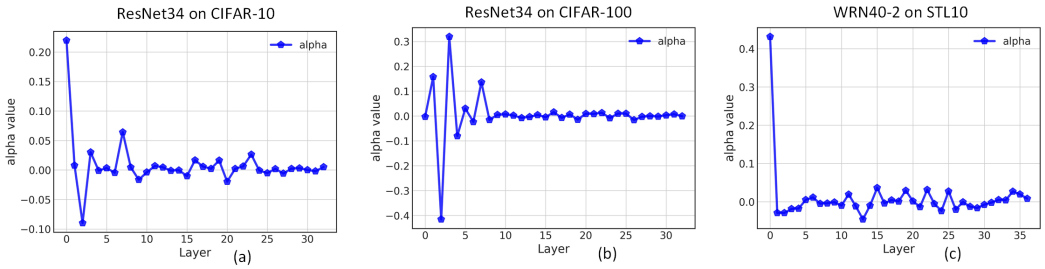

Figure 12: Trained noise scaling factor value (layer-wise) for (a) ResNet34 on CIFAR-10, (b) ResNet34 on CIFAR-100, and (c) WRN40-2 on STL10.

A.4 Alternate Results Plots in Terms of λ vs. Accuracy

To bolster Fig. 7 in Section 4, Fig. 13 show an alternate view of the CA-RA comparisons between FLOAT, OAT, and PGD-AT.

A.5 FLOAT performance on CW attack

Here we present FLOAT's performance on Carlini and Wagner (CW) attack Carlini & Wagner (2017). In particular, we used CW_∞ attack (optimized by PGD) with an attack step size of 20. As depicted in 14, the FLOAT yielded models consistently outperforms than that generated through PGD-AT. For example for $\lambda_n = 0.0$, FLOAT provides an improved robustness of up to 28.74% with similar clean image performance.

A.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. 15. 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 Section 4, we report the training time comparisons for OAT training with four training λs (0.0, 0.2, 0.7, 1.0).

A.7 FLOAT Slim Training Algorithm

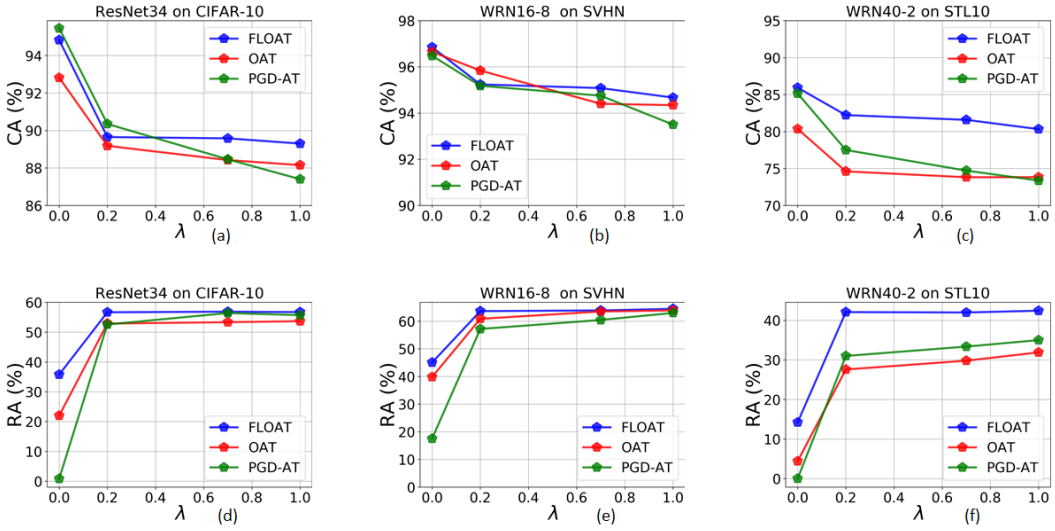

Figure 13: Comparison of trade-off between accuracy and robustness of FLOAT, OAT, and PGD-AT. (a)-(c) and (d)-(f) show CA and RA plots vs different λ values, respectively.

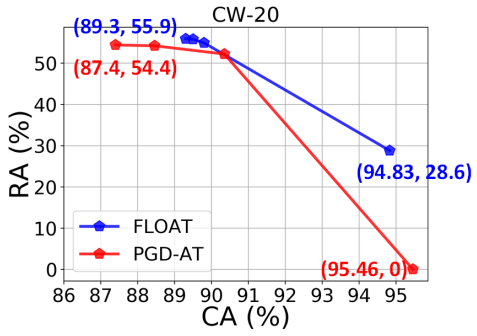

Figure 14: Performance comparison on CW attack with ResNet34 on CIFAR-10.

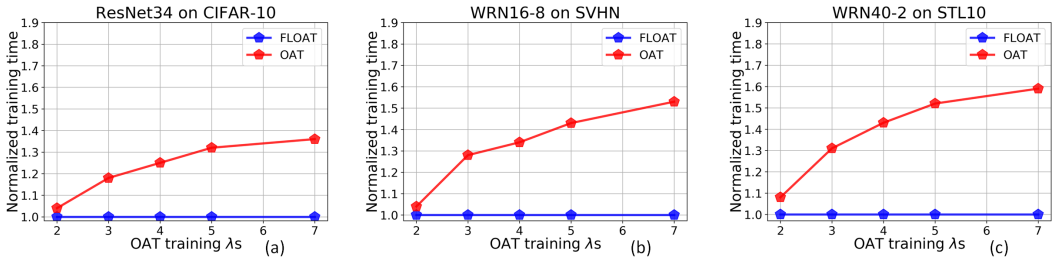

Figure 15: Comparison of per epoch normalized training time between FLOAT and OAT for different number of OAT training λs, on (a) CIFAR-10, (b) SVHN, and (c) STL10.

Algorithm 2: FLOATS 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 $\mathcal{B}$, slimming-factor set S_f.

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 $\mathcal{B}/2$ ($\boldsymbol{X}_{0:\mathcal{B}/2}$, $\boldsymbol{Y}_{0:\mathcal{B}/2}$) $\sim D$
    for *slimming-factor* w *in sorted* S_f **do**
      sampleSBN(w) //sample the BN_C and BN_A corresponding to w
      $\mathcal{L}_C \leftarrow$ computeLoss($\boldsymbol{X}_{0:\mathcal{B}/2}, \boldsymbol{\Theta}_{\boldsymbol{w}}, \lambda = 0, \boldsymbol{\alpha}_{\boldsymbol{w}}; \boldsymbol{Y}_{0:\mathcal{B}/2}$)
      $\hat{\boldsymbol{X}}_{\mathcal{B}/2:\mathcal{B}} \leftarrow$ createAdv($\boldsymbol{X}_{\mathcal{B}/2:\mathcal{B}}, \boldsymbol{Y}_{\mathcal{B}/2:\mathcal{B}}$) // adversarial image creation
      $\mathcal{L}_A \leftarrow$ computeLoss($\hat{\boldsymbol{X}}_{\mathcal{B}/2:\mathcal{B}}, \boldsymbol{\Theta}_{\boldsymbol{w}}, \lambda = 1, \boldsymbol{\alpha}_{\boldsymbol{w}}; \boldsymbol{Y}_{\mathcal{B}/2:\mathcal{B}}$)
      $\mathcal{L} \leftarrow 0.5 * \mathcal{L}_C + 0.5 * \mathcal{L}_A$
      accumulateGradient($\mathcal{L}$)
    end
    updateparam($\boldsymbol{\Theta}, \boldsymbol{\alpha}, \nabla_{\mathcal{L}}$)
  end
end