training to [41]. Furthermore, our approach outperforms the state-of-the-art robust binary networks [17], achieving performance on par with or even better than the state-of-the-art robust full-precision ones [44] while producing much more compact networks. Finally, we conduct preliminary investigations on the structure of the robust subnetworks obtained and find some interesting patterns. Altogether, our work sheds some light on understanding the structure of robust networks and obtaining compressed models robust to adversarial attacks.

Notation. We use light letters, lowercase bold letters, and uppercase bold letters to represent scalars, vectors, and higher dimensional tensors, respectively. $\odot$ is the elementwise multiplication operation. We use the term *adversarial budget* to represent the range of allowable perturbations. Specifically, the adversarial budget $\mathcal{S}_\epsilon$ is based on the l_∞ norm and defined as $\{\Delta | \|\Delta\|_\infty \leq \epsilon\}$, with ϵ the strength of the adversarial budget. We refer to the proportion of pruned parameters over the total number of parameters in a layer or a model as the *pruning rate* r.

2 Related Work

Adversarial Robustness. Deep neural networks have been shown to be vulnerable to adversarial attacks [45, 37]. To generate adversarial examples, the *Fast Gradient Sign Method* (FGSM) [16] perturbs the input in the direction of the input gradient. The *Iterative Fast Gradient Sign Method* (IFGSM) [30] improves FGSM by running it iteratively. *Projected Gradient Descent* (PGD) [36] uses random initialization and multiple restarts on top of IFGSM to further strengthen the attack. Recently, AutoAttack (AA) [12] has become the state-of-the-art attack method by ensembling different types of attacks; it is used to reliably benchmark the robustness of models [10] and we are thus using it in our experiments.

Many works have proposed defense mechanisms against these adversarial attacks. Early ones [4, 38, 48] used obfuscated gradients [3, 12] and thus were ineffective against adaptive attacks. As a consequence, adversarial training [36] and its variants [1, 5, 24, 42, 29, 47, 52, 53] have in practice become the mainstream approach to obtain robust models. Specifically, given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, a model f parameterized by $\mathbf{w}$ and a loss function $\mathcal{L}$, adversarial training solves the min-max optimization problem

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} \max_{\Delta_i \in \mathcal{S}_\epsilon} \mathcal{L}(f(\mathbf{w}, \mathbf{x}_i + \Delta_i), y_i) \ . \tag{1}$$

In practice, this is achieved by first generating adversarial examples $\mathbf{x}_i + \Delta_i$, usually by PGD, and then using these examples to train the model parameters.

While effective, adversarial training was shown to require a larger model capacity [36, 49]. Specifically, as the model capacity decreases, adversarial training first fails to converge while the training on clean inputs still yields non-trivial performance. Conversely, as the model capacity increases, the performance of training on clean inputs saturates before that of adversarial training. This highlights the challenge of finding robust yet compact models. Here, we introduce a solution to this problem.

Model Compression. There are many ways to compress deep neural networks to achieve lower memory consumption and faster inference, including pruning, quantization, and parameter encoding. Pioneer works in the 1990s use information-theoretic methods [31] or second-order derivatives [21] to compress models by removing unimportant weights. The seminal work [20] proposed to prune the model parameters with the smallest absolute values for deep networks. This motivated many follow-up works, performing either irregular pruning [18, 50, 54], which removes individual parameters, or regular pruning [23, 34], which aims to discard entire convolutional kernels. In contrast to pruning, quantization [56, 39, 26] seeks to reduce the memory consumption and inference time by using low-precision parameters. An extreme case of quantization is binarization, which can take the form of only binarizing the model parameters [7, 8] or binarizing both model parameters and intermediate activations [25]. The models can be further compressed by combining pruning with quantization and Huffman coding [19].

Recently, some efforts have been made to incorporate adversarial training into model compression. [15, 32, 40] suggest quantization as a defense against adversarial attacks. [51] use the alternating direction method of multipliers (ADMM) to alternatively conduct adversarial training and network pruning. [17]

extends this framework and proposes *Adversarially Trained Model Compression* (ATMC) to include other model compression techniques, such as quantization. Furthermore, [44] introduces the HYDRA framework, which improves the performance of compressed robust models by a three-phase method: Pretraining, score-based pruning, and fine-tuning. Here, we follow a different strategy: instead of performing adversarial training, we search for a robust binary subnetwork in a randomly-initialized one. We show that our approach outperforms those based on adversarial training.

Lottery Ticket Hypothesis. [14] first introduced the *Lottery Ticket Hypothesis*: overparameterized neural networks contain sparse subnetworks that can be trained in isolation to achieve competitive performance. These subnetworks are called the *winning tickets*. Based on this interesting observation, [57, 41] further proposed the *Strong Lottery Ticket Hypothesis*. They showed that there exist winning tickets with competitive performance even without training. Recently, [6] proposed an iterative randomization scheme to reduce the size of the network in which one searches for the winning tickets. Motivated by the *Strong Lottery Ticket Hypothesis*, we aim to find the winning tickets that achieve not only competitive performance but also robustness against adversarial attacks.

3 Methodology

3.1 Edge Popup under Adversarial Attacks

Let us consider a neural network f parameterized by $\mathbf{w} \in \mathbb{R}^n$. The neural network outputs $f(\mathbf{w}, \mathbf{x})$ for the input data $(\mathbf{x}, y)$. $\mathcal{L}(f(\mathbf{w}, \mathbf{x}), y)$ then represents the training loss objective, where $\mathcal{L}$ is the softmax cross-entropy loss. Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, an adversarial budget $\mathcal{S}_\epsilon$, and a predefined pruning rate r, we search for a binary pruning mask $\mathbf{m}$ that solves the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{m}} \frac{1}{N} \sum_{i=1}^{N} \max_{\Delta_i \in \mathcal{S}_\epsilon} \mathcal{L}\left(f(\mathbf{w} \odot \mathbf{m}, \mathbf{x}_i + \Delta_i), y_i\right) \\ &s.t.\ \mathbf{m} \in \{0,1\}^n, sum(\mathbf{m}) = (1-r)n. \end{aligned} \tag{2}$$

Here *sum* indicates the function that calculates the summation of all the elements in a vector. Note that, in contrast to adversarial training, we do not optimize the model parameters $\mathbf{w}$ in (2); instead $\mathbf{w}$ contains randomly-initialized parameters that are kept fixed during optimization. As such, our algorithm aims to find a pruned network structure, encoded via the n-dimensional binary vector $\mathbf{m}$, corresponding to a robust subnetwork.

Since the mask $\mathbf{m}$ is a discrete vector, it cannot be directly optimized by gradient-based methods. To overcome this, we replace it with a continuous "score" variable, $\mathbf{s} \in \mathbb{R}^n$, from which we calculate the mask as

$$\mathbf{m} = M(\mathbf{s}, r)\ , \tag{3}$$

where M is a binarization function, which constructs a binary mask from the continuous-valued scores based on the pruning strategy and the required pruning rate r. Specifically, the pruning strategy first determines the number of parameters retained in each layer. Then, for a layer assigned m post-pruned parameters, we retain the parameters with the top m highest scores and prune the rest.

To update the scores $\mathbf{s}$, we use the same *edge-popup* strategy as in [41], except that we first generate adversarial perturbations of the input using PGD. The strategy works by exploiting the M function to construct the binary mask in the forward pass, while treating M as the identity function in the backward pass. This allows the gradient to pass through and update all elements in the scores $\mathbf{s}$, although we only use a subnetwork in the forward pass. Note that we need the gradient of the score $\frac{\partial \mathcal{L}}{\partial \mathbf{s}}$ only in the outer minimization of (2). Therefore, the approximation of the function M in the back-propagation does not affect the inner maximization. Our experiments show that we can effectively generate adversarial examples by PGD. We provide the pseudo-code of our algorithm in Appendix C.

3.2 Adaptive Pruning

In addition to the given pruning rate r, the binarization function M in Equation (3) also depends on the pruning strategy, which determines the number of parameters retained in each layer. The pruning strategy can greatly affect the performance and stability of the edge-popup method, particularly in the case of adversarial training, as adversarial training solves a more challenging optimization problem than training on clean samples [33]. In this section, we introduce an *adaptive pruning* strategy that relies on an adaptive rate for layers of different sizes, and discuss its advantages over other strategies.

Let us first review the pruning strategy used in [41]. Specifically, [41] employ a *fixed pruning rate* that retains the same proportion of parameters in each layer. To define this formally, let us consider an L-layer network with n_1, n_2, ..., n_L parameters in its successive layers, from which we retain m_1, m_2, ..., m_L parameters, respectively, after pruning. Then, a *fixed pruning rate* translates to obtaining a mask $\mathbf{m}$ such that $1 - r = \frac{m_1}{n_1} = \frac{m_2}{n_2} = ... = \frac{m_L}{n_L}$. [1] In Appendix A.1, we show that this strategy approximately maximizes the size of the search space of the subnetwork.

While this strategy may seem intuitive, it does not take the differences in layer size into account. In practice, the number of parameters in different layers can vary widely. For example, residual networks [22] have much fewer parameters in the first and last layers than in the middle ones. Using *fixed pruning rate* thus yields very few parameters after pruning within such small layers. For example, when $r = 0.99$, only 17 parameters are left after pruning for a convolutional layer with 3 input channels, 64 output channels and a kernel size of 3. Such a small number of parameters has two serious drawbacks: 1) It greatly limits the expression power of the network; 2) it makes the edge-popup algorithm less stable, because adding or removing a single parameter then has a large impact on the network's output. This instability becomes even more pronounced in the presence of adversarial samples, because the gradients of the model parameters are more scattered than when training on clean inputs [33].

To overcome these drawbacks, we study an alternative strategy aiming to maximize the total number of paths from the input to the output in the pruned network. For a feedforward network, the total number of such paths is upper bounded by $\Pi_{i=1}^{L} m_i$. The following theorem demonstrates that the pruning strategy that maximizes this upper bound consists of retaining the same number of parameters in every layer, except for the layers that initially have too few parameters, for which all parameters should then be retained.

Theorem 1. *Consider an L-layer feedforward neural network with $n_1, n_2, ..., n_L$ parameters in its successive layers, from which we retain $m_1, m_2, ..., m_L$ parameters, respectively, after pruning. Given a predefined sparsity ratio $r = 1 - \frac{\sum_{i=1}^{L} m_i}{\sum_{i=1}^{L} n_i}$, the numbers of post-pruning parameters $\{m_i\}_{i=1}^{L}$ that maximize the upper bound of the total number of the input-output paths $\Pi_{i=1}^{L} m_i$ have the following property: $\forall 1 \leq j \leq L$, m_j satisfies either of the following two conditions: 1) $m_j = n_j$; 2) $\forall 1 \leq k \leq L, m_j \geq m_k - 1$.*

The two conditions in Theorem 1 mean we retain the same number of parameters for each layer except for ones totally unpruned. We defer the proof to Appendix B.1, where we use proof by contraction. We refer to the corresponding pruning strategy as *fixed number of parameters*.

While this *fixed number of parameters* strategy addresses the problem of obtaining too small layers arising in the *fixed pruning rate* one, it suffers from overly emphasizing the influence of the small layers. That is, the smaller layers end up containing too many parameters. In the extreme case, some layers are totally unpruned when the pruning rate r is small. This is problematic in our settings, since the model parameters are random and not updated. The unpruned layers based on random parameters provide a large amount of noise in the forward process. Furthermore, this strategy significantly sacrifices the expression power of the big layers.

In other words, the two strategies discussed above are two extremes: the *fixed pruning rate* one suffers when r is big, whereas the *fixed number of parameters* one suffers when r is small. To address this, we propose a strategy in-between these two extremes. Specifically, we determine the number of parameters

[1] In practice, this equality is only approximately satisfied, because $\{m_i\}_{i=1}^{L}$ needs to be all integers.

retained in each layer by solving the following system of equations:

$$1 - r = \frac{\sum_{i=1}^{L} m_i}{\sum_{i=1}^{L} n_i}, \frac{m_1}{n_1^p} = \frac{m_2}{n_2^p} = ... = \frac{m_L}{n_L^p}, \tag{4}$$

where $p \in [0, 1]$ is a hyper-parameter controlling the trade-off between the two extreme cases. When $p = 0$, the strategy (4) is close to the *fixed number of parameters* one. When $p = 1$, the strategy becomes the *fixed pruning rate* one. By setting $0 < p < 1$, we can retain a higher proportion of parameters in the smaller layers without sacrificing the big layers too much. We call this strategy *adaptive pruning*.

As discussed above, the strategy obtained with $p = 1$ tends to fail with a big r, while the strategy resulting from setting $p = 0$ tends to fail with a small r. This indicates that we need to assign small values of p given a big r and big values of p otherwise. We validate this and study the influence of p on the results of our approach in our experiments.

3.3 Binary Initialization and Last Normalization Layer

In this section, we introduce a binary initialization scheme for the model parameters. Such binary parameter values allow for further model compression and acceleration. Furthermore, we introduce a normalization layer to facilitate training and boost performance.

The empirical studies of [41] demonstrate the importance of the initialization scheme on the performance of the pruned network. To this end, they suggest the *Signed Kaiming Constant* initialization: the parameters in layer i are uniformly sampled from the set $\left\{-\sqrt{\frac{2}{l_{i-1}(1-r)}}, \sqrt{\frac{2}{l_{i-1}(1-r)}}\right\}$, where l_{i-1} represents the fan-out of the previous layer. Correspondingly, the scores $\mathbf{s}$ are initialized based on a uniform distribution $U[-\sqrt{\frac{1}{l_{i-1}}}, \sqrt{\frac{1}{l_{i-1}}}]$.

The magnitude of the *Signed Kaiming Constant* initialization is carefully calculated to keep the variance of the intermediate activations stable from the input to the output. In modern deep neural networks, the convolutional layers, potentially together with activation functions, are typically followed by a batch normalization layer. In [41] and our settings, these batch normalization layers only estimate the running statistics of their inputs, they do not have trainable parameters representing affine transformations. Because of these batch normalization layers, the magnitudes of the convolutional layers do not affect the outputs of the "convolution-batch norm" blocks. Furthermore, the fully-connected layers on top of the convolutional ones are homogeneous[2] because their bias terms are always initialized to zero. The activation functions we use, such as ReLU or leaky ReLU [35], are also homogeneous. Therefore, the magnitudes of parameters in these fully-connected layers do not change the predicted labels of the model either.

Based on the analysis above, we can conclude that the magnitudes of the model parameters at initialization do not change the predicted labels. Therefore, we propose to scale the model parameters $\mathbf{w}$ in all linear layers, i.e., convolutional and fully-connected ones, so that they are all sampled from $\{-1, +1\}$. Correspondingly, the scores $\mathbf{s}$ are initialized based on a uniform distribution $[-a, a]$ where a is a factor controlling the variance. *Binary initialization* is beneficial to model compression and acceleration, since there are no longer multiplication operations in linear layers. We discuss the efficiency improvement of the binary networks in detail in Appendix A.2. Theoretically, for the ResNet34 models we use in this paper, binary initialization can save approximately 45% and 32% FLOP operations compared with its full precision counterpart. Since we use irregular pruning in our method, fully take advantage of this improvement needs lower-level and hardware customization.

Although scaling the model parameters does not affect the expression power of the network, it does change the optimization landscape of the problem (2), because the softmax cross-entropy function used to calculate the loss objective is not homogeneous. With the *Signed Kaiming Constant* method, after multiplying the parameters initialized in the last layer by $\sqrt{\frac{l_{L-1}(1-r)}{2}}$, the output logits fed to the softmax cross-entropy function are also multiplied by the same factor. In practice, $\sqrt{\frac{l_{L-1}(1-r)}{2}} \gg 1$

[2]We call a function f homogeneous if it satisfies $\forall x \; \forall a \in \mathbb{R}^+$, $f(ax) = af(x)$.

greatly increases the output logits. Large logits will cause numerical instability and thus greatly worsen the optimization performance. In particular, our detailed analysis in Appendix A.3 shows that such scaling causes gradient vanishing for correctly classified inputs and, even worse, gradient exploding for misclassified ones.

To address this issue, we propose to add another 1-dimensional batch normalization layer at the end of the model, just before the softmax layer. The analysis in Appendix A.3 shows this normalization layer cancels out the multiplication factor applied to the weights in the last layer and thus facilitates the optimization. In our experiments below, we show that this normalization layer greatly improves the performance of both the *Signed Kaiming Constant* method and our *Binary Initialization* one. Furthermore, the last normalization layer also makes the performance more robust to different score $\mathbf{s}$ initializations.

4 Experiments

In this section, we present extensive experimental results to validate our approach. First, we describe an ablation study and sensitivity analysis to evaluate the performance of our proposed methods. Then, we compare our performance with existing works, which achieve robustness and compression in either full-precision or binary cases. We also include adversarial training [36] as a baseline. Finally, we analyze the structure of the pruned networks that we obtain. We show some interesting patterns of these post-pruning networks, suggesting the potential of our approach for effective compression.

Unless explicitly stated otherwise, we use a 34-layer Residual Network (ResNet34) [22], the same as the one in [41, 44].[3] We train the models for 400 epochs and use a cosine annealing learning rate scheduler with an initial value of 0.1. We use the CIFAR10 dataset [28] in the ablation study; we also use the CIFAR100 dataset [28] in the comparison with the baselines. We employ PGD attacks [36] to generate adversarial examples during training, but we use AutoAttack (AA) [12] for our robustness evaluation. While PGD is faster than AutoAttack and thus suitable for training, AutoAttack is the current state-of-the-art attack method, and we thus consider it a more reliable metric of robustness. We use an l_∞ norm-based adversarial budget, and the perturbation strength ϵ is 8/255 for CIFAR10 and 4/255 for CIFAR100. More details about the experimental settings and hyper-parameters are listed in Appendix D.1.

4.1 Ablation Study and Sensitivity Analysis

4.1.1 Pruning Strategy and Pruning Rates

Prune Strategy	$r = 0.5$	$r = 0.8$	$r = 0.9$	$r = 0.95$	$r = 0.99$	$r = 0.995$	$r = 0.998$
$p = 0.0$	2.16	6.86	23.01	41.61	44.60	40.70	**34.97**
$p = 0.1$	4.35	15.03	28.12	42.65	**44.88**	**40.97**	33.09
$p = 0.2$	8.01	19.21	27.99	43.72	42.92	40.52	32.99
$p = 0.5$	9.21	32.70	42.84	43.62	42.45	40.55	30.08
$p = 0.8$	28.90	41.51	**43.64**	**43.88**	39.12	33.61	28.07
$p = 0.9$	39.09	41.71	43.07	42.28	38.68	33.89	17.43
$p = 1.0$	**42.85**	**43.23**	42.13	41.12	34.57	26.67	20.56

Table 1: Robust accuracy (in %) on the CIFAR10 test set under different pruning rates r and values of p in *adaptive pruning*. The best result for each pruning rate is marked in bold.

In this section, we focus on binary initialization and on the models with the last batch normalization layer (LBN). We compare the performance of our method under different pruning rates r and *adaptive pruning strategies* with different values of p. The scores $\mathbf{s}$ are initialized from a uniform distribution $U[-0.01, 0.01]$.

[3] Note that the ResNet34 used in these papers and ours differs from the WideResNet34-10 used in [36, 47], which is larger and has almost twice the number of trainable parameters.

Our results are summarized in Table 1, in which we include 7 different values of pruning rate r and 7 different values of p in the *adaptive pruning strategy*. First, we notice that the best performance is achieved when $r = 0.99$ and $p = 0.1$. For the *fixed pruning rate* strategy ($p = 1$), the best performance is achieved when $r = 0.8$. Compared with the vanilla (e.g., non-adversarial) case in [41], which uses the *fixed pruning rate* strategy and shows that $r = 0.5$ achieves the best clean accuracy, the best performance for robust accuracy is achieved at a much higher pruning rate. This interesting observation is also consistent with the existing work [9], which shows that adversarial training implicitly encourages sparse convolutional kernels.

Table 1 further demonstrates the benefits of our *adaptive pruning strategy*. For larger pruning rates r, a smaller value of p prevails; for smaller pruning rates, a bigger value of p prevails. This is consistent with our analysis in Section 3.2. In particular, compared with the best results for a fixed pruning rate strategy ($p = 1.0$, $r = 0.8$), which is the pruning strategy in [41], our best adaptive pruning ($p = 0.1$, $r = 0.99$) achieves not only better performance but also a higher pruning rate. That is to say, using our *adaptive pruning strategy* improves both robustness and compression rates.

In Figure 4 of Appendix D.2.4, we provide the learning curves of the experiments in Table 1 when $r = 0.99$ and when $r = 0.5$. Regardless of the pruning rate r, these curves indicate the importance of the pruning strategy: a well chosen p value not only improves the performance but also makes training more stable.

4.1.2 Initialization Schemes and Last Normalization Layer

We then study the initialization scheme and how the last batch normalization layer (LBN) introduced in Section 3.3 affects the performance.

We focus on the *binary initialization* first and report the performance of models with and without the last normalization layer under different values of a, the hyper-parameter controlling the variance of the initial score $\mathbf{s}$. Based on the results of Table 1, we use the *adaptive pruning strategy* with $p = 0.1$ and the prune rate $r = 0.99$.

Model	Value of a in score initialization			
	0.001	0.01	0.1	1
no LBN	33.08	39.96	**41.01**	31.04
LBN	**45.06**	44.88	44.63	44.41

Table 2: Robust accuracy (in %) on the CIFAR10 test set for models with and without the last batch normalization layer (LBN) under different values of a for score $\mathbf{s}$ initialization. The best results are marked in bold.

The results are listed in Table 2 and clearly show that the last batch normalization layer (LBN) greatly improves the performance. Furthermore, LBN makes performance much less sensitive to the initialization of the scores, which in practice facilities the hyper-parameter selection.

We then compare the performance of the *binary initialization* with the *Signed Kaiming Constant*. We fix the pruning rate to $r = 0.99$ and employ an adaptive pruning strategy with different values of p. Our results are summarized in Table 3. For binary initialization, we use the optimal initialization scheme of the score $\mathbf{s}$ from Table 2; for *Signed Kaiming Constant* initialization, we use the optimal setting from [41] to initialize $\mathbf{s}$.

Based on the results in Table 3, we can conclude that the *binary initialization* achieves a comparable performance with the *Signed Kaiming Constant*. In addition, the last batch normalization layer can also benefit performance when using the *Signed Kaiming Constant*. We show in Table 7 of Appendix D.2.1 that these conclusions are also valid in a non-adversarial setting.

Prune Strategy	Signed KC no LBN	Signed KC LBN	Binary no LBN	Binary LBN
$p = 0.0$	39.38	42.83	40.94	44.65
$p = 0.1$	39.62	45.01	**41.01**	**45.06**
$p = 0.2$	36.66	**45.04**	37.85	41.58
$p = 0.5$	**39.98**	42.64	40.61	39.95
$p = 0.8$	37.96	41.71	35.15	38.95
$p = 0.9$	34.75	40.14	35.64	35.81
$p = 1.0$	36.88	39.32	30.02	30.62

Table 3: Robust accuracy (in %) on the CIFAR10 test set with the *Signed Kaiming Constant* (Signed KC) and the binary initialization. We include models both with and without the last batch normalization layer (LBN). The best results are marked in bold.

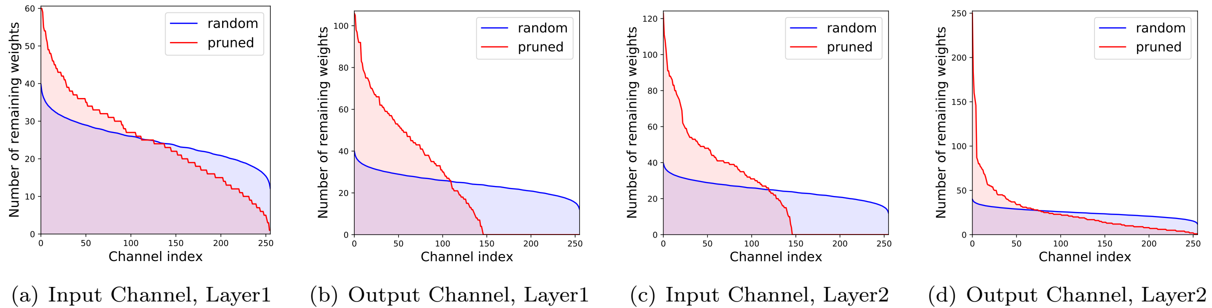
Method	Architecture	Pruning Strategy	Pruning Rate	CIFAR10 FP	CIFAR10 Binary	CIFAR100 FP	CIFAR100 Binary
AT	ResNet34	-	-	<u>43.26</u>	40.34	<u>36.63</u>	26.49
AT	ResNet34-LBN	-	-	42.39	39.58	35.15	32.98
HYDRA	ResNet34	$p = 0.1$	0.99	42.73	29.28	33.00	23.60
HYDRA	ResNet34	$p = 1.0$	0.99	40.51	26.40	31.09	18.24
HYDRA	ResNet34-LBN	$p = 0.1$	0.99	40.55	33.99	13.63	24.69
HYDRA	ResNet34-LBN	$p = 1.0$	0.99	32.93	26.23	29.96	17.75
ATMC	ResNet34	Global	0.99	34.14	25.62	25.10	11.09
ATMC	ResNet34	$p = 0.1$	0.99	34.58	24.65	25.37	11.04
ATMC	ResNet34	$p = 1.0$	0.99	30.50	20.21	22.28	2.53
ATMC	ResNet34-LBN	Global	0.99	33.55	19.01	23.16	15.73
ATMC	ResNet34-LBN	$p = 0.1$	0.99	31.61	22.88	25.16	17.33
ATMC	ResNet34-LBN	$p = 1.0$	0.99	27.88	13.22	22.12	9.55
AT	Small ResNet34-p0.1	-	-	42.01	32.54	28.46	16.18
AT	Small ResNet34-p1.0	-	-	38.81	26.03	27.68	15.85
Ours	ResNet34-LBN	$p = 0.1$	0.99	-	**45.06**	-	**34.83**
Ours	ResNet34-LBN	$p = 1.0$	0.99	-	34.57	-	26.32
Ours (Fast)	ResNet34-LBN	$p = 0.1$	0.99	-	40.77	-	34.45
Ours (Fast)	ResNet34-LBN	$p = 1.0$	0.99	-	29.68	-	24.97

Table 4: Robust accuracy (in %) on the CIFAR10 and CIFAR100 test sets for AT, HYDRA, ATMC and our proposed method. "ResNet34-LBN" represents ResNet34 with the last batch normalization layer. "Ours (Fast)" represents the our method with acceleration based on FGSM and ATTA. The best results for full precision (FP) models are underlined; the best results for binary models are marked in bold.

4.2 Comparison with Existing Methods

In this section, we compare our approach with the state-of-the-art methods targeting model compression and robustness. Specifically, we include HYDRA [44] and ATMC [17], as well as adversarial training (AT) [36] with early stopping [43]. Given our previous results, we fix the pruning rate to $r = 0.99$. For HYDRA and ATMC, we evaluate the performance on ResNet34 models both with and without the last normalization layer. [4] For ATMC, we also include *global pruning*, where we prune all parameters of different layers together, as it is the pruning strategy in the original paper [17]. For adversarial training, we use the full ResNet34 model and some smaller networks of approximately the same parameters as our pruned models. These smaller networks have the same architecture as the ResNet34 except that they have fewer channels so that they have approximately the same number of parameters per layer as the pruned network. For example, "Small ResNet34-p0.1" represents a small network with approximately the same number of parameters per layer as the pruned ResNet34 with $p = 0.1$. The details of these small networks are shown in Table 6 of Appendix D.1. Unlike our method, we follow the official implementations of HYDRA and ATMC, the normalization layers used in all baselines have the affine operation and thus

[4] In the original implementation of HYDRA and ATMC, the models were trained without the last batch normalization layer using the *fixed pruning rate* strategy.



(a) Input Channel, Layer1 (b) Output Channel, Layer1 (c) Input Channel, Layer2 (d) Output Channel, Layer2

Figure 1: Number of retained parameters in each input and output channel of layer1 and layer2 in the same residual block. We sort the numbers and plot the curves from the largest on the left to the smallest on the right. The red curves represent the mask obtained by our method; the blue curves depict what happens when randomly pruning the corresponding layer.

trainable parameters.

ATMC supports quantization but its parameterization introduces learnable quantized values. That is, the models obtained by ATMC's 1-bit quantization have only two parameter values in each layer; these values are different from layer to layer and are not necessarily -1 and $+1$. This means that, compared with the binary networks obtained with our method, those from ATMC have more flexibility. Nevertheless, we still include ATMC for comparison in the case of binary networks. HYDRA and AT are not designed for quantization and do not inherently support binary networks. To address this, we use *BinaryConnect* [7] to replace the model's linear layers so that their parameters are binary. *BinaryConnect* generates binarized model parameters by continuous alternatives, which can be updated by gradients in backpropagation.

Our method uses *binary initialization* and the last batch normalization layer, so the models we obtained are inherently binary. In addition to using PGD-based adversarial examples, we accelerate our method by using adversarial examples based on FGSM [16] with ATTA [55]. FGSM with ATTA generates adversarial examples by one-step attacks with accumulated perturbations across epochs. This is much cheaper than the ten-step PGD attacks.

The results of all the methods are shown in Table 4, where we report the robust accuracy under AutoAttack (AA). Our method using the *adaptive pruning* strategy ($p = 0.1$) achieves better performance than all baselines in case of binary models. We also achieves better performance than methods that aim to compress full-precision robust models. Furthermore, our method achieve comparable results with AT on the original unpruned models with $100\times$ trainable parameters. Our method based on accelerated adversarial training also achieves better performance than all baselines in the case of binary networks.

The proposed *adaptive pruning* strategy ($p = 0.1$) consistently achieves better performance than the *fix pruning rate* strategy and than *global pruning* in all methods. This indicates the importance of the pruning strategy for both our method and the baselines. Furthermore, the normalization layer on top of the model does not necessarily improve the performance of the baselines studied here. On the contrary, it even hurts the performance in the full precision case. This is because HYDRA, ATMC and AT all include the optimization for the model parameters $\mathbf{w}$. In the full precision case, the magnitude of $\mathbf{w}$, and thus of the output logits, is automatically adjusted during training. The issue resulting from large output logits that we pointed out in Section 3 does thus not happen in this case, so the last batch normalization layer is not necessary. In practice, we notice this layer slows down the training convergence.

Finally, we report the vanilla accuracy of all methods in Table 8. We show that our proposed method also has competitive performance on clean inputs. Specifically, we achieve the best performance among all methods for binary networks. Combining the results in Table 4 and Table 8, we can conclude that our proposed method yields a better trade-off between vanilla accuracy and robust accuracy.

4.3 Analysis of the Subnetwork Pattern

In this work, we use irregular pruning. Compared with regular pruning, irregular pruning is more flexible but less structured, which means that it requires lower-level customization to fully take advantage of parameter sparsity for acceleration. However, visualizing the masks $\mathbf{m}$ of the convolutional layers in our pruned binary network with a pruning rate $r = 0.99$ allowed us to find that the mask is structured to some degree. For example, we visualize the mask of a convolutional layer with 256 input channels and 256 output channels in Figure 5 of Appendix D.2.3. We notice that the retained parameters are quite concentrated: A large portion of the retained parameters concentrate on few input or output channels, while many other channels (40% of the total) are completely pruned.

Furthermore, we visualize two consecutive convolutional layers in the same residual block of the ResNet34 model. We call them *layer1* and *layer2* following the forward pass. In Figure 1, we plot the distribution of the retained parameters in each input channel and in each output channel, respectively. We find that many output channels of layer1 and the input channels of layer2, 40% of all channels in this case, are totally pruned. As a reference, we also plot the distribution of random pruning, based on the average of 500 simulations. Our analysis in Appendix A.4 demonstrates that, in a randomly pruned network, it is almost impossible to have even one entirely pruned channel. The comparison indicates that our mask $\mathbf{m}$ is structured.

Furthermore, in Figure 3, we visualize the pruned channels in both layers and find that they are aligned. That is, some neurons representing the output channels of layer1 and the input channels of layer2 are entirely removed. We defer additional discussions, figures and results to Appendix D.2.3. The pattern of the structures learned by our method indicates the potential of regular pruning for a randomly-initialized neural network under adversarial attacks. We leave this as our future work.

5 Conclusion

We have proposed a method to obtain robust binary models by pruning randomly-initialized networks, thus extending the *Strong Lottery Ticket Hypothesis* to the adversarial case. In contrast to the state-of-the-art methods, we learn the structure of robust subnetworks without updating the parameters. Furthermore, we have proposed an *adaptive pruning* strategy and last batch normalization layer to stabilize the training and improve performance. Finally, we have relied on binary initialization to obtain more compact models.

Our extensive results on various benchmarks have demonstrated that our approach outperforms existing methods for training compressed robust models. Furthermore, we have observed interesting structured patterns occurring in the parameters retained in the subnetworks. This opens the door to further investigations on the structure of the robust subnetworks and on the design of regular pruning strategies in the adversarial scenario.

References

[1] Jean-Baptiste Alayrac, Jonathan Uesato, Po-Sen Huang, Alhussein Fawzi, Robert Stanforth, and Pushmeet Kohli. Are labels required for improving adversarial robustness? In *Advances in Neural Information Processing Systems*, pages 12192–12202, 2019.

[2] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *European Conference on Computer Vision*, pages 484–501. Springer, 2020.

[3] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[4] 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.

[5] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and Percy S Liang. Unlabeled data improves adversarial robustness. In *Advances in Neural Information Processing Systems*, pages 11190–11201, 2019.

[6] Daiki Chijiwa, Shin'ya Yamaguchi, Yasutoshi Ida, Kenji Umakoshi, and Tomohiro INOUE. Pruning randomly initialized neural networks with iterative randomization. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[8] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[9] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. *AISTATS 2019*, 2019.

[10] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020.

[11] Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *International Conference on Machine Learning*, pages 2196–2205. PMLR, 2020.

[12] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International Conference on Machine Learning*, 2020.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[14] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[15] Angus Galloway, Graham W. Taylor, and Medhat Moussa. Attacking binarized neural networks. In *International Conference on Learning Representations*, 2018.

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

[17] Shupeng Gui, Haotao N Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework. In *Advances in Neural Information Processing Systems*, pages 1283–1294, 2019.

[18] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in Neural Information Processing Systems*, 29:1379–1387, 2016.

[19] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[20] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *Advances in neural information processing systems*, 28:1135–1143, 2015.

[21] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[22] 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*, pages 770–778, 2016.

[23] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[24] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *International Conference on Machine Learning*, pages 2712–2721, 2019.

[25] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[26] Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2146–2156, 2020.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[29] Nupur Kumari, Mayank Singh, Abhishek Sinha, Harshitha Machiraju, Balaji Krishnamurthy, and Vineeth N Balasubramanian. Harnessing the vulnerability of latent layers in adversarially trained models. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2779–2785. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[30] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[31] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[32] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. In *International Conference on Learning Representations*, 2019.

[33] Chen Liu, Mathieu Salzmann, Tao Lin, Ryota Tomioka, and Sabine Süsstrunk. On the loss landscape of adversarial training: Identifying challenges and how to overcome them. *Advances in Neural Information Processing Systems*, 33, 2020.

[34] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[35] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. Citeseer.

[36] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[37] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[38] Tianyu Pang, Kun Xu, Yinpeng Dong, Chao Du, Ning Chen, and Jun Zhu. Rethinking softmax cross-entropy loss for adversarial robustness. In *International Conference on Learning Representations*, 2020.

[39] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *International Conference on Learning Representations*, 2018.

[40] Adnan Siraj Rakin, Jinfeng Yi, Boqing Gong, and Deliang Fan. Defend deep neural networks against adversarial examples via fixed and dynamic quantized activation functions. *arXiv preprint arXiv:1807.06714*, 2018.

[41] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902, 2020.

[42] Sylvestre-Alvise Rebuffi, Sven Gowal, Dan Andrei Calian, Florian Stimberg, Olivia Wiles, and Timothy Mann. Data augmentation can improve robustness. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[43] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pages 8093–8104. PMLR, 2020.

[44] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. Hydra: Pruning adversarially robust neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19655–19666. Curran Associates, Inc., 2020.

[45] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[47] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. *Advances in Neural Information Processing Systems*, 33, 2020.

[48] Chang Xiao, Peilin Zhong, and Changxi Zheng. Enhancing adversarial defense by k-winners-take-all. In *International Conference on Learning Representations*, 2020.

[49] Cihang Xie and Alan Yuille. Intriguing properties of adversarial training at scale. In *International Conference on Learning Representations*, 2020.

[50] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[51] Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. Adversarial robustness vs. model compression, or both. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 2, 2019.

[52] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In *Advances in Neural Information Processing Systems*, pages 227–238, 2019.

[53] 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*, pages 7472–7482, 2019.

[54] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199, 2018.

[55] Haizhong Zheng, Ziqi Zhang, Juncheng Gu, Honglak Lee, and Atul Prakash. Efficient adversarial training with transferable adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1181–1190, 2020.

[56] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *International Conference on Learning Representations*, 2017.

[57] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in Neural Information Processing Systems*, 32:3597–3607, 2019.

Ï

A Analysis

A.1 Analysis of the *Fixed Pruning Rate* Strategy

We consider a L-layer neural network and each layer has n_1, n_2, ..., n_L parameters, we retrain m_1, m_2, ..., m_L parameters after pruning. As such, the total number of combinations $\Pi_{i=1}^{L} \binom{n_i}{m_i}$ is the size of search space of the subnetworks. The following theorem shows, the *fixed pruning rate* strategy is the strategy which approximates the maximization of the total number of combinations.

Theorem 2. *Consider an L-layer neural network with n_1, n_2, ..., n_L parameters in each layer, we retain m_1, m_2, ..., m_L parameters after pruning. Given a predefined pruning rate $r = 1 - \frac{\sum_{i=0}^{L} m_i}{\sum_{i=0}^{L} n_i}$, the optimal numbers of post-pruning parameters $\{m_i\}_{i=1}^{L}$ that maximizing the total number of combinations $\Pi_{i=1}^{L} \binom{n_i}{m_i}$ satisfy the following inequality:*

$$\forall 1 \leq j, k \leq L, \ \left| \frac{m_j}{n_j} - \frac{m_k}{n_k} \right| < \frac{1}{n_j} + \frac{1}{n_k} \tag{5}$$

We defer the proof to Appendix B.2. Specifically, we let n_k in (5) be the largest layer in the network without the loss of generality, we then have $\forall i \leq j \leq L, j \neq k$, $\left| m_j - \frac{m_k}{n_k} n_j \right| < \frac{n_j}{n_k} + 1 \leq 2$. m_j is the number of retained parameters and thus an integer, so Theorem 2 indicates the pruning rate of each layer is close to each other when we aim to maximize the total number of combinations. Therefore, we can consider the *fixed pruning rate* strategy, i.e., $1 - r = \frac{m_1}{n_1} = \frac{m_2}{n_2} = ... = \frac{m_L}{n_L}$, as an approximation to maximize the total number of combinations.

A.2 Analysis of Acceleration by Binary Initialization

In this section, we analyze the acceleration benefit of binary initialization. Since most of the forward and backward computational complexity for the models studied in this paper is consumed by the "Convolutional-BatchNorm-ReLU" block, the acceleration rate on such blocks is a good approximation of that on the whole network. Therefore, we concentrate on the "Convolution-BatchNorm-ReLU" block here.

For simplicity, we assume the feature maps and convolutional kernels are all squares. Without the loss of generality, we consider r_{in}-channel input feature maps of size s, the size of the convolutional kernel is c and the convolutional layer outputs r_{out} channels. Here, the binary layers represent the layer whose parameters are either -1 or $+1$.

Forward Pass For full-precision dense networks, the number of FLOP operations of the convolutional layer is $2c^2s^2r_{in}r_{out}$. By contrast, the complexity can be reduced to $c^2s^2r_{in}r_{out}$ for binary dense layers, convolution operation with a binary kernel does not include any multiplication operations. Correspondingly, for sparse layers whose pruning ratio is r, the complexity of full-precision sparse networks and of the binary sparse networks can be reduced to $2(1-r)c^2s^2r_{in}r_{out}$ and $(1-r)c^2s^2r_{in}r_{out}$, respectively.

The batch normalization layer will consume $3s^2r_{out}$ FLOP operations during inference and $10s^2r_{out}$ during training. The additional operations during training are due to the update of running statistics. Note that, the batch normalization layer in our random initialized network does not contain any trainable parameters, so there is no scaling parameters after normalization. The ReLU layer will always consume s^2r_{out} FLOP operations.

To sum up, we calculate the complexity ratio of the binary "Convolution-BatchNorm-ReLU" block over its full-precision counterpart in the forward pass. For dense layers, the ratio is $\frac{c^2r_{in}+11}{2c^2r_{in}+11}$ for the training

time and $\frac{c^2 r_{in}+4}{2c^2 r_{in}+4}$ for the inference. For sparse layers, the ratio is $\frac{(1-r)c^2 r_{in}+11}{2(1-r)c^2 r_{in}+11}$ for the training time and $\frac{(1-r)c^2 r_{in}+4}{2(1-r)c^2 r_{in}+4}$ for the inference.

Backward Pass Compared with the forward pass, the backward pass has some computational overhead, because we need to calculate the gradient with respect to the score variable $\mathbf{s}$ associated with the convolutional kernels. For both dense and sparse networks, the overhead is $2c^2 s^2 r_{in} r_{out} + c^2 r_{in} r_{out}$ for full precision layers and $2c^2 s^2 r_{in} r_{out}$ for binary layers. Note that, the overhead is independent of the pruning rate r because the pruning function is treated as the identity function in the backward pass. In addition, the difference here between the full precision layer and binary layer arises from the multiplication when we backprop the gradient through the weights.

To sum up, we calculate the complexity ratio of the binary "Convolution-BatchNorm-ReLU" block over its full-precision counterpart in the backward pass. We only back propagate the gradient in the training time, so the batch normalization layer is always the training mode. For dense layers, the ratio is $\frac{3c^2 s^2 r_{in}+4s^2}{4c^2 s^2 r_{in}+4s^2+c^2 r_{in}}$. For sparse layers, the ratio is $\frac{3(1-r)c^2 s^2 r_{in}+4s^2}{4(1-r)c^2 s^2 r_{in}+4s^2+c^2 r_{in}}$.

Network	Forward Pass		Backward Pass
	Training	Evaluation	Training
FP	$2(1-r)c^2 s^2 r_{in} r_{out} + 11 s^2 r_{out}$	$2(1-r)c^2 s^2 r_{in} r_{out} + 4s^2 r_{out}$	$4(1-r)c^2 s^2 r_{in} r_{out} + 4s^2 r_{out} + c^2 r_{in} r_{out}$
Binary	$(1-r)c^2 s^2 r_{in} r_{out} + 11 s^2 r_{out}$	$(1-r)c^2 s^2 r_{in} r_{out} + 4s^2 r_{out}$	$3(1-r)c^2 s^2 r_{in} r_{out} + 4s^2 r_{out}$

Table 5: The complexity in FLOP operations of the sparse'Convolution-BathNorm-ReLU" block in both full precision (FP) and binary case. The pruning rate is r.

Discussion We summarize the complexity in FLOP operations of the sparse "Convolution-BatchNorm-ReLU" block in different scenarios. We can now conclude that compared with the full precision block, the binary block decrease the overall complexity in two places: 1) we save $(1-r)c^2 s^2 r_{in} r_{out}$ FLOPs for the convolution and transpose convolution operations in the forward and backward pass, respectively; 2) for the backpropagation, we save $c^2 r_{in} r_{out}$ FLOPs, because there is no multiplication when we backprop the gradient through the weights for binary blocks.

We consider the practical settings: $r = 0.99$, $c = 3$, $r_{in} = r_{out} = 128$, $s = 16$. The complexity ratio of the binary block over the full precision block in the forward pass is $\frac{(1-r)c^2 r_{in}+11}{2(1-r)c^2 r_{in}+11} = 0.6616$ for the training mode and $\frac{(1-r)c^2 r_{in}+4}{2(1-r)c^2 r_{in}+4} = 0.5740$ for the evaluating mode, respectively. The complexity ratio in the backward pass is $\frac{3(1-r)c^2 s^2 r_{in}+4s^2}{4(1-r)c^2 s^2 r_{in}+4s^2+c^2 r_{in}} = 0.7065$. That is to say, compared with the full precision block, the binary block under this setting can save around 34% and 29% time in the forward and backward passes during training; for inference, it can save 43% time.

A.3 Analysis of the Normalization Layer before Softmax

We consider a L-layer neural network and each layer has l_1, l_2, ..., l_L neurons. Let $\mathbf{u} \in \mathbb{R}^{l_{L-1}}$, $\mathbf{W} \in \mathbb{R}^{l_L \times l_{L-1}}$, $\mathbf{o} \in \mathbb{R}^{l_L}$ be the output of the penultimate's output, the weight matrix of the last fully-connected layer and the last layer's output, respectively. In addition, we use $c \in \{1, 2, ..., l_L\}$ to denote the label of the data and omit the bias term of the last layer since it is initialized as 0 and is not updated. For the 1-dimensional batch normalization layer, we use $\mathbf{b} \in \mathbb{R}^{l_L}$ and $\mathbf{v} \in \mathbb{R}^{l_L}$ to represent the running mean and running standard deviation, respectively.

Therefore, the loss objective $\mathcal{L}_{wo}$ and its gradient of the model without the 1-dimensional batch normalization layer is:

$$\begin{aligned} \mathcal{L}_{wo} &= -log \frac{e^{\mathbf{o}_c}}{\sum_{i=1}^{l_L} e^{\mathbf{o}_i}} \\ \frac{\partial \mathcal{L}_{wo}}{\partial \mathbf{o}_j} &= \frac{e^{\mathbf{o}_j}}{\sum_{i=1}^{l_L} e^{\mathbf{o}_i}} - \mathbf{1}(j = c) \end{aligned} \tag{6}$$

Correspondingly, the loss objective $\mathcal{L}_{wi}$ and its gradient of the model with the 1-dimensional batch

normalization layer is:

$$
\begin{aligned}
\mathcal{L}_{wi} &= -log \frac{e^{(\mathbf{o}_c - \mathbf{b}_c)/\mathbf{v}_c}}{\sum_{i=1}^{l_L} e^{(\mathbf{o}_i - \mathbf{b}_i)/\mathbf{v}_i}} \\
\frac{\partial \mathcal{L}_{wi}}{\partial \mathbf{o}_j} &= \frac{1}{\mathbf{v}_j} \left(\frac{e^{(\mathbf{o}_c - \mathbf{b}_c)/\mathbf{v}_c}}{\sum_{i=1}^{l_L} e^{(\mathbf{o}_i - \mathbf{b}_i)/\mathbf{v}_i}} - \mathbf{1}(j = c) \right)
\end{aligned}
\tag{7}
$$

Now we consider the case when the model parameter $\mathbf{W}$ is multiplied by a factor $\alpha > 1$: $\mathbf{W}' = \alpha \mathbf{W}$ and assume the output of the penultimate layer is unchanged. In practice, α is far more than 1. For example, if the penultimate layer has 512 neurons, α will be 16 when we change kaiming constant initialization to binary initialization. Based on this, the new output of the last layer is $\mathbf{o}' = \alpha \mathbf{o}$. For the model with the normalization layer, the new statistics are $\mathbf{b}' = \alpha \mathbf{b}$ and $\mathbf{v}' = \alpha \mathbf{v}$. In this regard, we can then recalculate the gradient of the loss objective as follows:

$$
\begin{aligned}
\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j} &= \frac{e^{\mathbf{o}'_j}}{\sum_{i=1}^{l_L} e^{\mathbf{o}'_i}} - \mathbf{1}(j = c) = \frac{e^{\alpha \mathbf{o}_j}}{\sum_{i=1}^{l_L} e^{\alpha \mathbf{o}_i}} - \mathbf{1}(j = c) \\
\frac{\partial \mathcal{L}'_{wi}}{\partial \mathbf{o}'_j} &= \frac{1}{\mathbf{v}'_j} \left(\frac{e^{(\mathbf{o}'_c - \mathbf{b}'_c)/\mathbf{v}'_c}}{\sum_{i=1}^{l_L} e^{(\mathbf{o}'_i - \mathbf{b}'_i)/\mathbf{v}'_i}} - \mathbf{1}(j = c) \right) = \frac{1}{\alpha \mathbf{v}_j} \left(\frac{e^{(\mathbf{o}_c - \mathbf{b}_c)/\mathbf{v}_c}}{\sum_{i=1}^{l_L} e^{(\mathbf{o}_i - \mathbf{b}_i)/\mathbf{v}_i}} - \mathbf{1}(j = c) \right)
\end{aligned}
\tag{8}
$$

We first study the case without the normalization layer. The first term $\frac{e^{\alpha \mathbf{o}_j}}{\sum_{i=1}^{l_L} e^{\alpha \mathbf{o}_i}}$ of the gradient $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ converge to $\mathbf{1}(j = \text{argmax}_i \mathbf{o}_i)$ exponentially. For correctly classified inputs, $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ converge to 0 exponentially with α. In addition, the gradient $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{u}} = \mathbf{W}'^T \frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j} = \alpha \mathbf{W}^T \frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ also vanish with α. $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{u}}$ is backward to previous layers, leading to gradient vanishing. For incorrectly classified inputs, $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ converge to $\mathbf{1}(j = \text{argmax}_i \mathbf{o}_i) - \mathbf{1}(j = c)$, which is a vector with c-th element being -1, the element corresponding to the output label being $+1$ and the rest elements being 0. In this case, the gradient backward $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{u}} = \alpha \mathbf{W}^T \frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ will be approximately multiplied by α, causing gradient exploding.

By contrast, in the case of the model with the normalization layer, $\frac{\partial \mathcal{L}'_{wi}}{\partial \mathbf{o}'_j} = \frac{1}{\alpha} \frac{\partial \mathcal{L}_{wi}}{\partial \mathbf{o}_j}$. The factor $\frac{1}{\alpha}$ is cancelled out when we calculate $\frac{\partial \mathcal{L}'_{wi}}{\partial \mathbf{u}} = \mathbf{W}'^T \frac{\partial \mathcal{L}'_{wi}}{\partial \mathbf{o}} = \mathbf{W}^T \frac{\partial \mathcal{L}_{wi}}{\partial \mathbf{o}}$. This means the gradient backward remains unchanged if we use the 1-dimensional batch normalization layer, which maintains the stability of training if we scale the model parameters.

To conclude, the 1-dimensional batch normalization layer is crucial to maintain the stability of training if we use binary initialization. Without this layer, the training will suffer from gradient vanishing for correctly classified inputs and gradient exploding for incorrectly classified inputs.

A.4 Analysis of the structure of a randomly pruned network

In this section, we provide preliminary analysis of the structure of a randomly pruned network.

As a starting point, we first estimate the probability of k retained parameters in a 3×3 kernel. Given the pruning rate r_i for the layer i with n_i weights, the number of the retained parameters is $m_i := (1 - r_i) n_i$. We assume m_i lies in a proper range: $9 \ll m_i < \frac{1}{9} n_i$. This is true when n_i is large and $r_i > \frac{8}{9}$.

For each kernel j, we use X_j to represent its number of retained parameters. It is difficult to calculate $P(X_j = k)$ directly because $\{X_j\}_j$ are constrained by: 1) $\sum_j X_j = m_i$; 2) $\forall j, 0 \le X_j \le 9$. However, in the case of random pruning, we have $E[X_j] = \frac{9 m_i}{n_i} = 9(1 - r_i) < 1$. In this regard, we can make the approximation by removing the constraint $X_j \le 9$.

Therefore, we can reformulate the problem of calculating $P(X_j = k)$ as: *Given m_i steps, randomly select one box out of the total $\frac{n_i}{9}$ boxes and put one apple in it. $P(X_j = k)$ is then the probability for the box j to have k apples.*

In this approximation, it is straightforward to have $P(X_j = k) = \binom{m_i}{k} P_i^k \cdot (1 - P_i)^{m_i - k}, 0 \le k \le 9$ where $P_i = \frac{9}{n_i}$. Based on the assumption that n_i is large, $P_i \approx 0$. Therefore, $P(X_j = 0) = (1 - \frac{9}{n_i})^{m_i} \approx e^{9(1-r_i)}$. For $k > 1$, we apply Stirling approximation $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$ to the binomial coefficient, then

$$
\begin{aligned}
P(X_j = k) &\approx \frac{m_i^{m_i+0.5}}{\sqrt{2\pi} k^{k+0.5} (m_i - k)^{m_i - k + 0.5}} \cdot (\frac{9}{n_i})^k \cdot (1 - \frac{9}{n_i})^{m_i - k} \\
&= \sqrt{\frac{m_i}{2\pi k(m_i - k)}} \cdot (\frac{9(1 - r_i)}{k})^k \cdot (1 + \frac{k}{m_i - k})^{m_i - k} \cdot (1 - \frac{9}{n_i})^{m_i - k}
\end{aligned} \tag{9}
$$

The second equality is based on the fact $m_i = (1 - r_i)n_i$. Since $m_i \gg 9 > k$ by the assumption and $n_i \gg m_i$, we can approximate $(1 - \frac{9}{n_i})^{m_i - k}$ to $1 - \frac{9(m_i - k)}{n_i} \approx 1$, then

$$
P(X_j = k) \approx \sqrt{\frac{m_i}{2\pi k(m_i - k)}} \cdot (\frac{9e(1 - r_i)}{k})^k = \sqrt{\frac{m_i}{2\pi k(m_i - k)}} \cdot (\frac{c}{k})^k \tag{10}
$$

where $c = 9e(1 - r_i)$ is a constant.

As shown in the equation above, $P(X_j = k)$ decreases drastically when k increases. Therefore, in a randomly pruned layer i with $n_i = 3 \times 3 \times 256 \times 256 = 589824$ and $r_i = 0.99$, it is almost impossible to see kernels who have at least 4 retained parameters, because according to the above formula, the estimated number of kernels in that layer having 3 retained parameters is $\frac{n_i}{9} \times P(X_j = 3) \approx 8.19$, and the number of kernels having 4 retained parameters is ≈ 0.18.

Now we consider the number of retained parameters in a channel. For the layer of r_{in} input channels and r_{out} output channels, it has $r_{in} \times r_{out} \times 3 \times 3$ parameters. We use Y_j to represent the number of the retained parameters for the input channel j. Similarly, for the random pruning, we have

$$
P(Y_j = k) \approx \sqrt{\frac{m_i}{2\pi k(m_i - k)}} \cdot (\frac{c'}{k})^k \cdot (1 - \frac{1}{r_{in}})^{m_i - k} \tag{11}
$$

where $c' = 9er_{out}(1 - r_i)$ is a constant.

By plotting the distribution $P(Y_j)$, it is easy to find that the distribution of Y_j concentrates around the neighborhood of $k = \frac{m_i}{a}$, and decreases significantly as Y_j deviates from it.

B Proofs of Theoretical Results

B.1 Proof of Theorem 1

Proof. We proof the theorem by contradictory. We assume the optimal $\{m_i\}_{i=1}^L$ does not satisfy the proporty mentioned in Theorem 1. This means $\exists 1 \le j \le L$ such that $m_j < n_j$ and $\exists 1 \le k \le L, m_j < m_k - 1$. Based on this, we then construct a new sequence $\{\widehat{m}_i\}_{i=1}^L$ as follows:

$$
\widehat{m}_j = m_j + 1; \widehat{m}_k = m_k - 1; \forall i \ne j, i \ne k, \widehat{m}_i = m_i. \tag{12}
$$

We then calculate the ratio of $\Pi_{i=1}^L \widehat{m}_i$ and $\Pi_{i=1}^L m_i$:

$$
\frac{\Pi_{i=1}^L \widehat{m}_i}{\Pi_{i=1}^L m_i} = \frac{(m_j + 1)(m_k - 1)}{m_j m_k} = 1 + \frac{m_k - m_j - 1}{m_j m_k} > 1 \tag{13}
$$

The last inequality is based on the assumption $m_j < m_k - 1$. (13) indicates $\Pi_{i=1}^L \widehat{m}_i > \Pi_{i=1}^L m_i$, which contradicts the optimality of $\{m_i\}_{i=1}^L$. □

B.2 Proof of Theorem 2

Proof. We pick arbitrary $0 < j, k \le L$ and generates two sequences $\{\widehat{m}_i\}_{i=1}^L$, $\{\widetilde{m}_i\}_{i=1}^L$ as follows:

$$\begin{aligned} \widehat{m}_j = m_j - 1, \widehat{m}_k = m_k + 1, \widehat{m}_i = m_i \forall i \neq j, i \neq k. \\ \widetilde{m}_j = m_j + 1, \widetilde{m}_k = m_k - 1, \widetilde{m}_i = m_i \forall i \neq j, i \neq k. \end{aligned} \tag{14}$$

Consider $\{m_i\}_{i=1}^L$ the optimality that maximizes the combination number $\Pi_{i=1}^L \binom{n_i}{m_i}$. We have the following inequality:

$$\begin{aligned} 1 > \frac{\Pi_{i=1}^L \binom{n_i}{\widehat{m}_i}}{\Pi_{i=1}^L \binom{n_i}{m_i}} = \frac{m_j}{n_j - m_j + 1} \frac{n_k - m_k}{m_k + 1} \\ 1 > \frac{\Pi_{i=1}^L \binom{n_i}{\widetilde{m}_i}}{\Pi_{i=1}^L \binom{n_i}{m_i}} = \frac{n_j - m_j}{m_j + 1} \frac{m_k}{n_k - m_k + 1} \end{aligned} \tag{15}$$

Reorganize the inequalities above, we obtain:

$$-\left(\frac{1}{n_k} + \frac{m_k - m_j + 1}{n_j n_k}\right) < \frac{m_k}{n_k} - \frac{m_j}{n_j} < \left(\frac{1}{n_j} + \frac{m_j - m_k + 1}{n_j n_k}\right) \tag{16}$$

Consider $1 \le m_j \le n_j$ and $1 \le m_k \le n_k$, we have $\frac{m_k - m_j + 1}{n_j n_k} \le \frac{1}{n_j}$ and $\frac{m_j - m_k + 1}{n_j n_k} \le \frac{1}{n_k}$. As a result, we have the following inequality:

$$\forall j, k, -\left(\frac{1}{n_j} + \frac{1}{n_k}\right) < \frac{m_k}{n_k} - \frac{m_j}{n_j} < \left(\frac{1}{n_j} + \frac{1}{n_k}\right) \tag{17}$$

This concludes the proof. □

C Algorithm

We provide the pseudo-code of the edge pop-up algorithm for adversarial robustness as Algorithm 1. We use PGD to generate adversarial attacks. $\Pi_{\mathcal{S}_\epsilon}$ mean projection into the set $\mathcal{S}_\epsilon$.

D Experiments

D.1 Experimental Settings

General The ResNet34 architecture we use in this paper is the same as the one in [41, 44], and it has 21265088 trainable parameters. The bias terms of all linear layers are initialized 0, and are thus disabled. We also disable the learnable affine parameters in batch normalization layers, following the setup of [41]. The adversarial budget in this paper is based on l_∞ norm and the perturbation strength ϵ is 8/255 for CIFAR10 and 4/255 for CIFAR100. The PGD attacks used in our experiments have 10 iterations and the step size is one-quarter of the ϵ, respectively. The AutoAttack (AA) consists of the following four attacks: 1) the untargeted 100-iteration AutoPGD based on cross-entropy loss; 2) the targeted 100-iteration AutoPGD based on difference of logits ratio (DLR) loss; 3) the targeted 100-iteration FAB attack [11]; 4)

Algorithm 1: Edge pop-up algorithm for adversarial robustness.

Input: training set $\mathcal{D}$, batch size B, PGD step size α and iteration number n, adversarial budget $\mathcal{S}_\epsilon$, pruning rate r, mask function M, the optimizer.
Random initialize the model parameters $\mathbf{w}$ and the scores $\mathbf{s}$.
for Sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^B \sim \mathcal{D}$ **do**
  for i = 1, 2, ..., B **do**
    Sample a random noise δ within the adversarial budget $\mathcal{S}_\epsilon$.
    $\mathbf{x}_i^{(0)} = \mathbf{x}_i + \delta$
    for j = 1, 2, ..., n **do**
      $\mathbf{x}_i^{(j)} = \mathbf{x}_i^{(j-1)} + \alpha \nabla_{\mathbf{x}_i^{(j-1)}} \mathcal{L}(f(\mathbf{w} \odot M(\mathbf{s}, r), \mathbf{x}_i^{(j-1)}), y_i)$
      $\mathbf{x}_i^{(j)} = \mathbf{x}_i + \Pi_{\mathcal{S}_\epsilon}\left(\mathbf{x}_i^{(j)} - \mathbf{x}_i\right)$
    end for
  end for
  Calculate the gradient $\mathbf{g} = \frac{1}{B}\sum_{i=1}^B \nabla_{\mathbf{s}} \mathcal{L}(f(\mathbf{w} \odot M(\mathbf{s}, r), \mathbf{x}_i^{(n)}), y_i)$
  Update the score $\mathbf{s}$ using the optimizer.
end for
Output: the pruning mask $M(\mathbf{s}, r)$.

the black-box 5000-query Square attack [2]. We use the same hyper parameters in all these component attacks as in the original AutoAttack implementation.[5]

We train the model using an SGD optimizer, with the momentum factor being 0.9 and the weight decay factor being 5×10^{-4}. The learning rate is initially 0.1 and decays following the cosine annealing scheduler. Finally, since adversarial training suffers from severe overfitting [43], we use a validation set consisting of 2% of the training data to select the best model during training.

Adversarial Training We apply the same settings as above to adversarial training, except the choice of optimizer and learning rate. For full precision networks, we use an SGD optimizer with an initial learning rate of 0.1 and decreases by a factor of 10 in the 200th and 300th epoch. For binary networks, we use Adam optimizer [27] suggested in [7] and have a cosine annealing learning rate schedule with an initial learning rate of 1×10^{-4}.

HYDRA & ATMC Our results on HYDRA and ATMC are based on their original implementation publicly available [6] except that we use the validation set to pick the best model during training. In HYDRA and ATMC, the batch normalization layers in the model have affine operations and are learnable. This introduces additional trainable parameters and is different from our method.

Smaller ResNet34 Variants Based on the adaptive pruning strategy, we designed several smaller ResNet34 variants with approximately the same number of parameters as the pruned networks. These variants have the same topology as ResNet34 but have fewer channels in each layer. In Table 6, we provide architecture details based on different values of p when the pruning rate r is 0.99.

D.2 Additional Experimental Results

D.2.1 Ablation Study in the Vanilla Case

In the vanilla case, we train the models using clean inputs and report the clean accuracy in Table 7. Other hyper-parameters here are the same as in Table 3. Our conclusions from Table 3 also hold true here: the *binary initialization* can achieves comparable performance as the *Signed Kaiming Constant*; the last batch normalization layer helps improve performance for both initialization schemes.

[5] AutoAttack: `https://github.com/fra31/auto-attack`.
[6] HYDRA: `https://github.com/inspire-group/hydra`; ATMC: `https://github.com/VITA-Group/ATMC`.

layer name	Small ResNet34-p0.1	Small ResNet34-p1.0
conv1	3×3, 23	3×3, 6
Block1	$\begin{bmatrix} 3 \times 3, 23 \\ 3 \times 3, 23 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 6 \\ 3 \times 3, 6 \end{bmatrix} \times 3$
Block2	$\begin{bmatrix} 3 \times 3, 25 \\ 3 \times 3, 25 \end{bmatrix} \times 4$	$\begin{bmatrix} 3 \times 3, 13 \\ 3 \times 3, 13 \end{bmatrix} \times 4$
Block3	$\begin{bmatrix} 3 \times 3, 27 \\ 3 \times 3, 27 \end{bmatrix} \times 6$	$\begin{bmatrix} 3 \times 3, 26 \\ 3 \times 3, 26 \end{bmatrix} \times 6$
Block4	$\begin{bmatrix} 3 \times 3, 29 \\ 3 \times 3, 29 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 51 \\ 3 \times 3, 51 \end{bmatrix} \times 3$
	average pool, 10d-fc, softmax	
#params	201078	216360

Table 6: ResNet34 variants that have similar layer sizes as the pruned ResNet34 obtained by different p values. $3 \times 3 \times 23$ means the kernel size is 3×3 and there are 23 output channels.

Prune Scheme	Kaiming Constant Initialization		Binary Initialization	
	without Last BN	with Last BN	without Last BN	with Last BN
$p = 0.0$	93.25	93.99	93.64	**94.05**
$p = 0.1$	92.12	93.98	**93.84**	93.99
$p = 0.2$	92.96	**94.35**	89.27	93.87
$p = 0.5$	**93.44**	94.29	90.85	94.00
$p = 0.8$	90.93	92.57	90.37	92.42
$p = 0.9$	91.31	92.26	90.51	90.12
$p = 1.0$	89.27	89.12	87.58	89.03

Table 7: The accuracy (in %) of vanilla trained models on the CIFAR10 test set under various settings. The best result for each setting is marked in bold and the second best is underlined.

D.2.2 Clean accuracy of Models in Table 4

Table 8 shows the vanilla accuracy of the models in Table 4. In the CIFAR10 dataset, our pruned network achieves the highest vanilla accuracy among all binary networks. Although the accuracy is lower than full-precision networks by ATMC, our model performs notably better ($> 10\%$) under AutoAttack. In the CIFAR100 dataset, our model has the second best vanilla accuracy among both full-precision networks and binary networks, just below the adversarially trained full-precision ResNet34 network. These results indicate that our models can achieve competitive robust accuracy without losing too much vanilla accuracy, hence more powerful in real applications where both robust and vanilla accuracy are important.

D.2.3 Mask of the Pruned Network

We have demonstrated that the masks of the pruned network obtained by our method are structured to some degree in Section 4.3. We have also analyzed the structure of a randomly pruned network in Appendix A.4.

Figure 5 shows the one of the convolutional layers in our pruned ResNet34 network. We resize the layer parameters in the shape of $(r_{out} \times 3, r_{in} \times 3)$ for visualization. The pruning rate for this layer is $r = 0.99$. We highlight the input channels that are totally pruned in orange. We also use a white bar at the top of the figure to indicate these empty input channels.

Section 4.3 demonstrates that the retained parameters concentrate in a few channels. This is also true for kernels: there are a few kernels, which is 3×3, totally unpruned. We show the distribution of the

Method	Architecture	Pruning Strategy	Pruning Rate	CIFAR10		CIFAR100	
				FP	Binary	FP	Binary
AT	ResNet34	-	-	80.99	74.37	61.48	47.87
AT	ResNet34-LBN	-	-	80.96	74.17	57.73	60.08
HYDRA	ResNet34	$p = 0.1$	0.99	75.31	62.09	55.92	45.96
HYDRA	ResNet34	$p = 1.0$	0.99	73.89	59.69	54.88	37.84
HYDRA	ResNet34-LBN	$p = 0.1$	0.99	74.37	68.43	53.65	46.09
HYDRA	ResNet34-LBN	$p = 1.0$	0.99	66.59	57.15	26.64	37.16
ATMC	ResNet34	Global	0.99	81.85	72.97	57.15	36.39
ATMC	ResNet34	$p = 0.1$	0.99	81.37	73.34	59.99	32.68
ATMC	ResNet34	$p = 1.0$	0.99	74.33	57.01	54.06	5.19
ATMC	ResNet34-LBN	Global	0.99	80.72	68.25	55.69	40.51
ATMC	ResNet34-LBN	$p = 0.1$	0.99	76.39	51.31	57.34	43.97
ATMC	ResNet34-LBN	$p = 1.0$	0.99	69.83	19.26	49.69	28.78
AT	Small ResNet34-p0.1	-	-	74.76	58.69	52.77	28.81
AT	Small ResNet34-p1.0	-	-	69.77	52.38	50.09	28.05
Ours	ResNet34-LBN	$p = 0.1$	0.99	-	76.59	-	60.16
Ours	ResNet34-LBN	$p = 1.0$	0.99	-	65.70	-	47.21
Ours (Fast)	ResNet34-LBN	$p = 0.1$	0.99	-	**81.63**	-	**63.73**
Ours (Fast)	ResNet34-LBN	$p = 1.0$	0.99	-	70.72	-	50.98

Table 8: The accuracy (in %) on the clean inputs of the methods studied in Section 4.2. "ResNet34-LBN" represents ResNet34 with the last batch normalization layer. "Ours (Fast)" represents our methods with acceleration of FGSM and ATTA. The best results in the full precision (FP) cases are underlined and the best results in the binary cases are marked in bold.

retained parameters in each kernel in Figure 2. Results in Figure 2 show there are kernels totally or almost totally unpruned. We also plot the distribution of a randomly pruned layer (labeled "random") by taking the average of 500 simulations. Based on both analysis and simulation, we find that in a random pruned layer there is no kernel with more than 3 retained parameters.

In Section 4.3, we also point out the aligned pruning pattern in the two consecutive layers, layer1 and layer2, of the same residual block in ResNet34. Figure 3 shows their pruning masks. The side bars show which channel is non-empty(colored in blue). For convenience, layer1 is resized in $(r_{out} \times 3, r_{in} \times 3)$, and layer2 is organized in $(r_{in} \times 3, r_{out} \times 3)$. It is interesting that the pruned input channels of layer2 are well aligned with the pruned output channels of layer1.

Note that our finding also holds in the vanilla settings, i.e. pruning with clean examples. We think this observation enables a possible way for regular pruning.

D.2.4 Learning Curves of Adaptive Pruning with Different p Values

We plot the learning curves when we use the *adaptive pruning* strategy with different values of p in Figure 4. Here, we use $r = 0.99$ and $r = 0.5$ as two examples. Based on the results of Table 1, our method achieves the best performance under $p = 0.1$ when $r = 0.99$ and under $p = 1.0$ when $r = 0.5$.

The learning curves in Figure 4 indicate that the training process is quite unstable when using the inappropriate pruning strategy, leading to suboptimal performance.

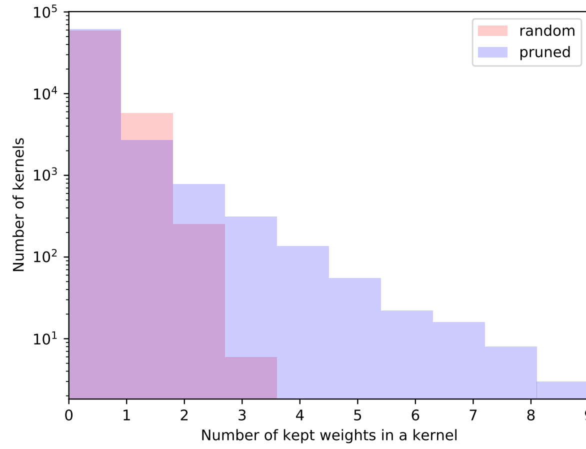


Figure 2: Distribution of kernels in terms of #retained parameters.

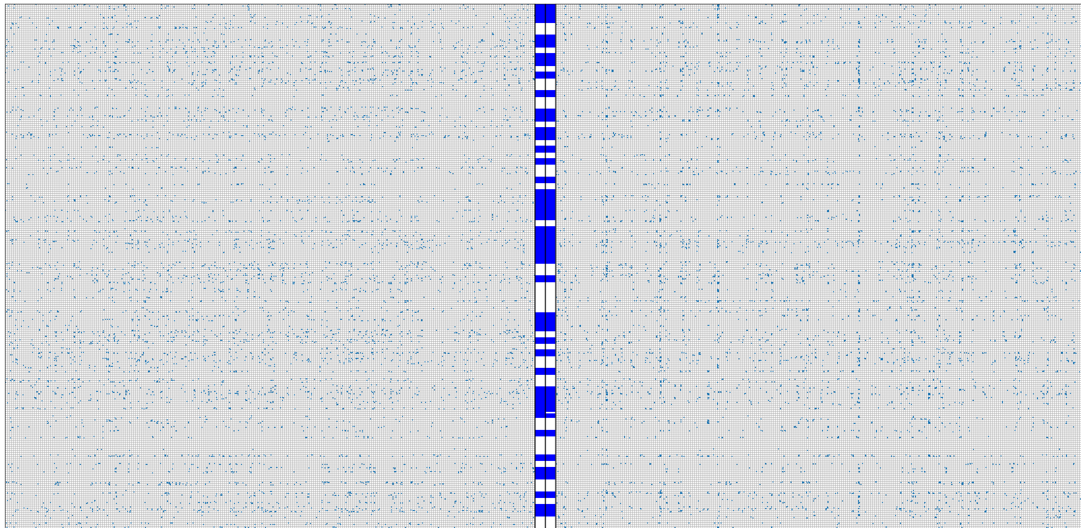
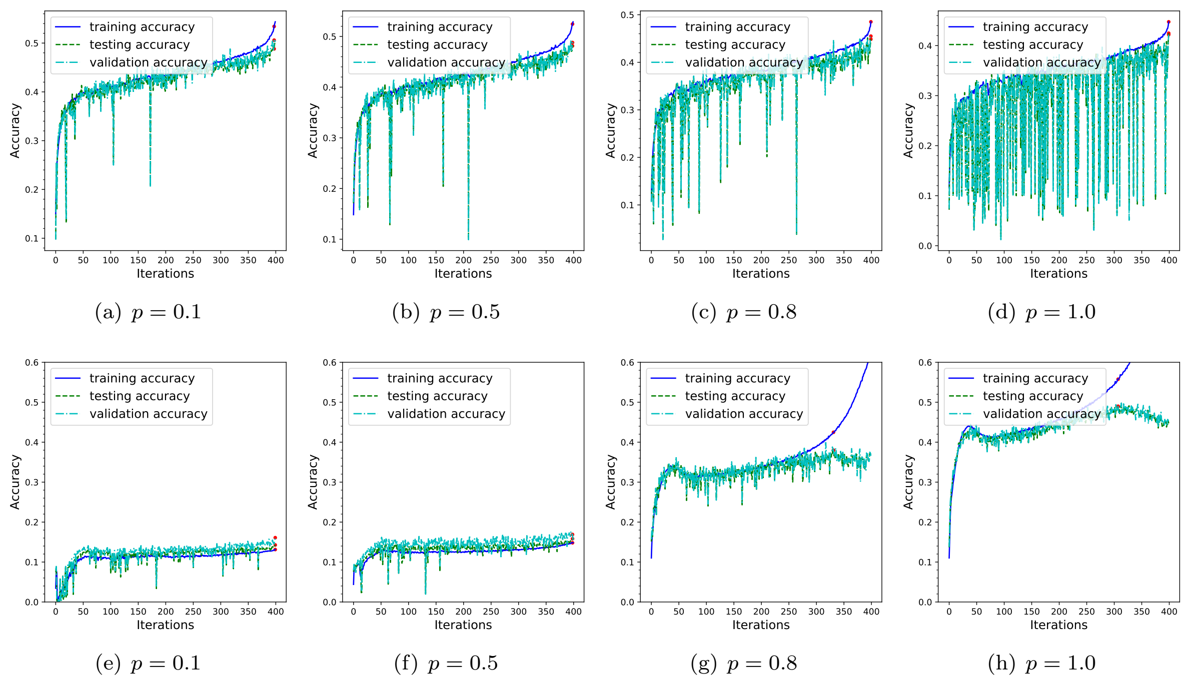


Figure 3: Distribution of weights in two consecutive layers. In layer1 (left), the masks are reshaped into ($r_{out} \times 3$, $r_{in} \times 3$) while masks in layer2 (right) are reshaped into ($r_{in} \times 3$, $r_{out} \times 3$). The output channels totally pruned in layer1 and the input channels totally pruned in layer2 are highlighted as the white bars in the middle.



(a) $p = 0.1$ (b) $p = 0.5$ (c) $p = 0.8$ (d) $p = 1.0$

(e) $p = 0.1$ (f) $p = 0.5$ (g) $p = 0.8$ (h) $p = 1.0$

Figure 4: Learning curves of our proposed method under adaptive pruning strategy with different values of p. The pruning ratio is 0.99 for figure (a) - (d) and is 0.5 for figure (e) - (h).

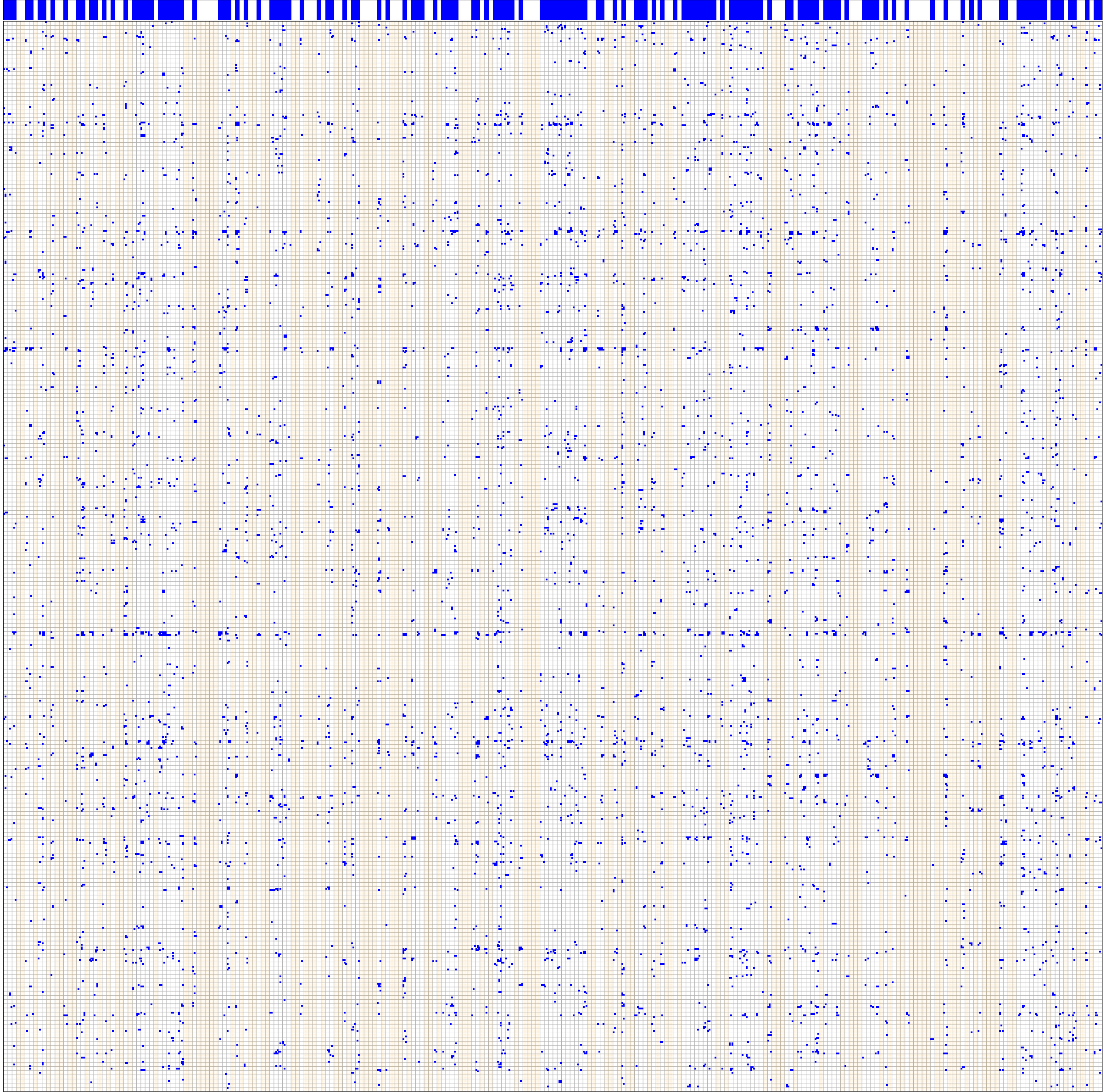


Figure 5: Mask visualization of the weight of a random convolutional layer in our model. The parameters retained is highlighted as blue dots. The dimension of the convolutional kernel is (r_{out}, r_{in}, 3, 3). We reshape this kernel in rectangle of shape ($r_{out} \times 3$, $r_{in} \times 3$). Channels with no remaining weight are colored orange. The top bar indicates whether the channel is empty (white) or not (blue).