

A APPENDIX

This appendix includes the following:

1. An extended discussion of related work in Section A.1.
2. An extended description of robust residual block in Section A.2.
3. Additional results of topology in Section A.3
4. An extended discussion of Squeeze-n-Excitation in Section A.4
5. Additional results of convolution kernel size in Section A.5
6. Impact of normalization in Section A.6
7. An extended description of RobustResNet in Section A.7.

A.1 EXTENDED RELATED WORK

This section provides a general review of the related concepts and existing work.

Adversarial white-box attacks. Since the first demonstration that high performant DNNs are vulnerable to small perturbation in inputs (a.k.a. adversarial examples) (Szegedy et al., 2014), there is a plethora of efforts devoted to crafting more delicate adversarial examples (AEs) – fast gradient sign method (FGSM) (Goodfellow et al., 2015) is one of the earliest methods that applies a single gradient step to generate AEs; projected gradient descent (PGD) (Madry et al., 2018) is a widely studied method that is considered to perform well in most cases while being computationally efficient; Carlini & Wagner (CW) (Carlini & Wagner, 2017) introduces an alternative loss that is shown to exhibit stronger attack performance; AutoAttack (AA) (Croce & Hein, 2020) is an aggregated attack that forms an ensemble of four complementary attacks.

Adversarial Training as a Defense. Among many defense strategies proposed in the literature, adversarial training (AT) has emerged as one of the most effective ways to guard against adversarial attacks. The basic idea of AT is to leverage AEs during the training process of a DNN model. Since the primary work that feeds inputs perturbed by PGD back into the training data (Madry et al., 2018), the procedure of applying AT has been extended in various directions – sophisticated loss functions to balance the trade-off between natural and robust accuracy (Zhang et al., 2019) or make use of misclassified natural examples (Wang et al., 2020); advanced training procedures such as early stopping to prevent *robust overfitting* (Rice et al., 2020) and weight ensembling (Chen et al., 2021; Wang & Wang, 2022); more diverse data for training by generative modeling (Gowal et al., 2021; Sehwag et al., 2022) or data augmentation (Rebuffi et al., 2021).

Robust Architecture. Adversarial robustness has also been studied from an architecture perspective. On the one hand, several recent works attempt to find more robust DNN architectures via neural architecture search (NAS) – Guo et al. (2020) applies a one-shot NAS algorithm to design the topology of a cell structure (i.e., operations and connections among them) while leaving the network skeleton (i.e., width and depth) to human designs; Mok et al. (2021) incorporates the smoothness of a DNN model’s input loss landscape as an additional regularizer for NAS, among others (Ning et al., 2020; Chen et al., 2020; Liu & Jin, 2021). On the other hand, there are also recent works that aim to gain understandings of adversarial robustness from an architectural perspective – Huang et al. (2021a) investigates the impact of network width and depth on the adversarial robustness of adversarially trained DNN models.

A.2 EXTENDED DESCRIPTION OF ADVERSARIALLY ROBUST RESIDUAL BLOCK

We provide the visual comparison between post- and pre-activation in Figure 11, where the post-activation He et al. (2016a) places activation function posterior to the weights while the pre-activation proposed by He et al. (2016b) places activation function a-priori to the weights.

Besides, PyTorch-like implementation of our proposed RobustResBlock is summarized in Algorithm 1.

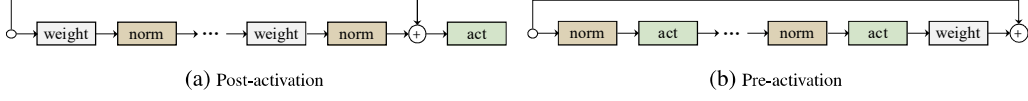


Figure 11: A pictorial illustration of (a) the standard post-activation and (b) pre-activation arrangements.

Algorithm 1 RobustResBlock: PyTorch-like Pseudocode

```
# C_in, C_out: number of input / output channels
# s, c, w: scales, cardinality, and hidden size
# NormAct: Batch normalization followed by a ReLU activation
class RobustResBlock(nn.Module):
    expansion = 4
    def __init__(self, C_in, C_out, S, G, W, stride, **kwargs):
        width = math.floor(C_out * (W / 160)) * G # True width
        conv1 = Conv1x1(C_in, width * S)
        conv2 = [Conv3x3(width, width, groups=G) for i in range(S - 1)]
        conv3 = Conv1x1(width * S, C_out * expansion)
        se = SE(C_out * expansion, reduction=64) # Squeeze-and-Excitation
        if stride > 1 or (C_in != C_out * expansion):
            shortcut = Conv1x1(C_in, C_out * expansion)
            pool = AvgPool3x3(stride)

    def forward(self, x):
        out = NormAct(x) # Pre-Act: Input firstly goes through a NormAct
        shortcut = shortcut(out) if shortcut else x
        inputs = torch.chunk(out, S, dim=1) # Split input into number of scales chunks
        outs = [pool(inputs[-1])] if stride > 1 else inputs[-1]
        for i in range(S - 1):
            x = inputs[i] if (i == 0 or stride > 1) else inputs[i] + outs[-1]
            outs.append(NormAct(conv2[i](x)))
        out = conv3(torch.cat(outs, 1))
        return out + shortcut + se(out)
```

A.3 ADDITIONAL RESULTS OF TOPOLOGY

In this subsection, we present extra results of topology over CIFAR10 in Figure 12. Detailed comparison for the non-residual block with post- and pre-activation is shown in (a). Both activation arrangements have similar performance and can not alleviate the lack of residual connection. Besides, we also studied two more model capacities for aggregated and hierarchical convolutions. Furthermore, (g, h) comparing aggregated (cardinality = 4) and hierarchical (scales = 8) bottlenecks to other blocks under PGD²⁰.

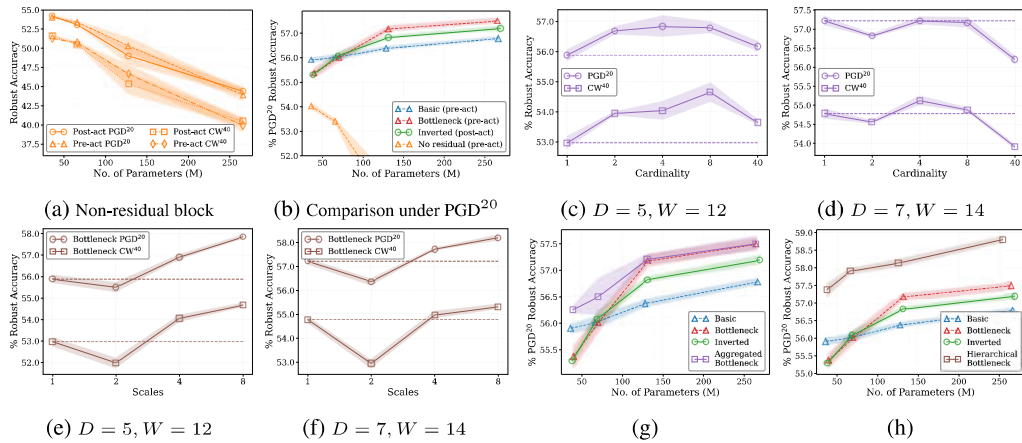


Figure 12: Extra results for comparing (a) non-residual block with post- and pre-activation, (b) four blocks with best settings. (c, d) and (e, f) show the robustness of aggregated and hierarchical bottleneck in two kinds of models, respectively. (g, h) Comparing aggregated (cardinality = 4) and hierarchical (scales = 8) bottleneck to other blocks under PGD²⁰.

A.4 EXTENDED DESCRIPTION OF SQUEEZE-N-EXCITATION

Squeeze-n-Excitation is a well-recognized operator on standard tasks. We have tried five variants of SE for adversarial robustness in this work and provide more details of those variants in Figure 13. Specifically, (a - e) show their structures, and (f) explains each variant. Besides, additional results are found in (g, h). Our proposed residual SE is simple as standard SE Hu et al. (2018) and improves the robustness while others fail. Furthermore, we studied different reduction ratios for residual SE and identified that a higher reduction ratio (i.e., $r = 64$) provides a good trade-off between parameters and robustness.

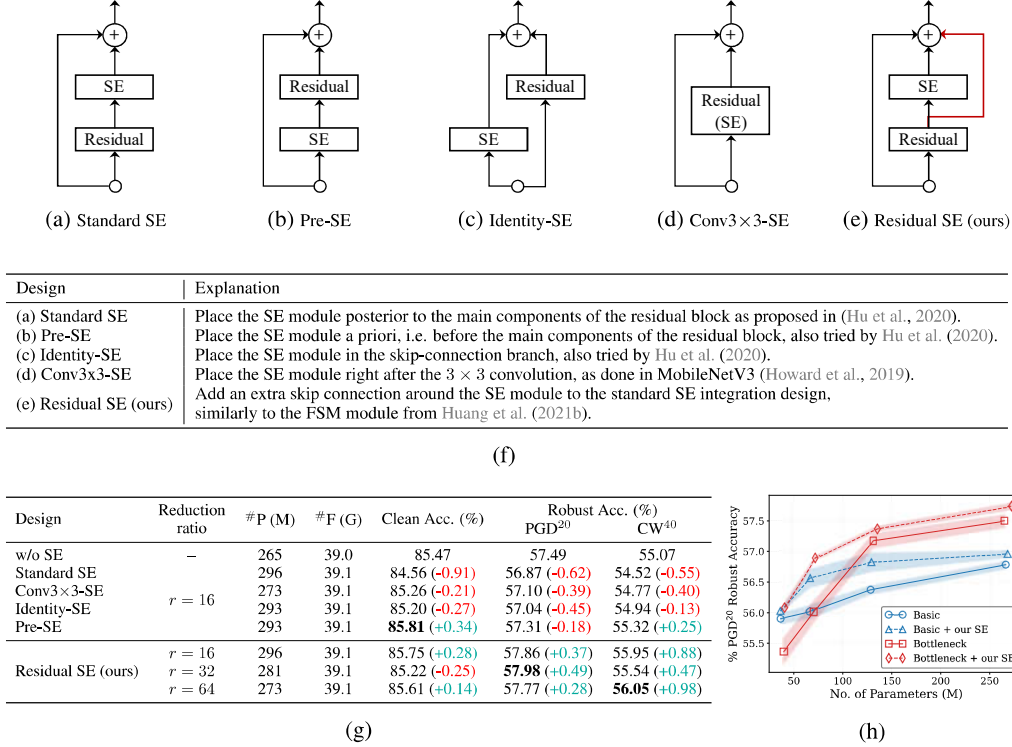


Figure 13: (a) - (e) An overview of SE integration designs studied in this work. (f) Description and (g) ablation results of the SE integration designs shown in (a) - (e).

A.5 ADDITIONAL RESULTS OF CONVOLUTION KERNEL SIZE

We provide more experimental results of convolution kernel size in Figure 14. The observations from CW⁴⁰ in the main paper are consistent under different attackers, i.e., FGSM and AutoAttack.

A.6 IMPACT OF NORMALIZATION

In this section, we investigate the relationship between normalization methods and adversarial robustness. In addition to baseline of Batch Normalization (BN), we consider three other normalization methods, i.e., Group Normalization (GN) (Wu & He, 2018), Layer Normalization (LN) (Ba et al., 2016), and Instance Normalization (IN) (Ulyanov et al., 2016). We also confine all blocks in a DNN model to use a single choice of normalization method and repeat the experiment for each method three times. The experimental results are summarized in Table 3. Evidently, the baseline normalization method (i.e., BN) outperforms all other alternative normalization methods, particularly on Tiny-ImageNet. It is worth noting that LN fails to converge on all three datasets.

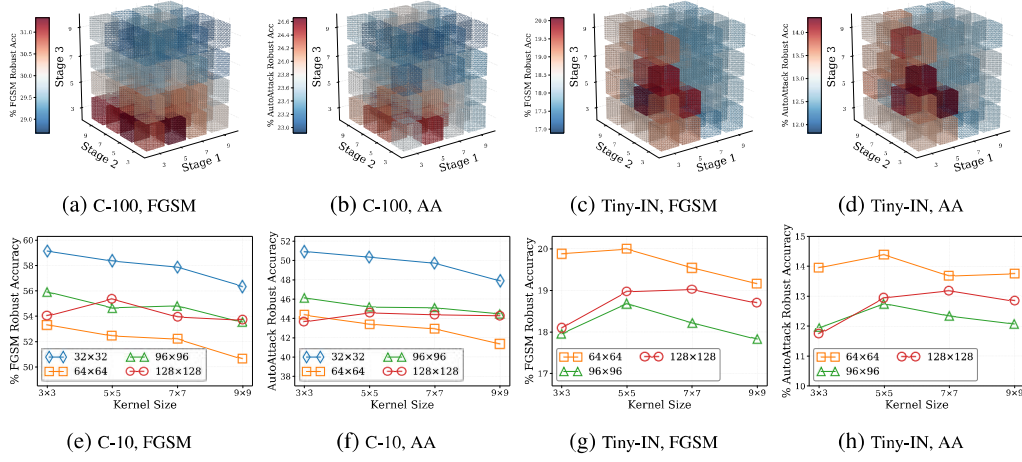


Figure 14: Additional results of convolution kernel size.

Table 3: The adversarial robustness of the considered normalization methods. We highlight the best results of each section in bold. “-” indicates that the training fails to converge.

Options	CIFAR-10				CIFAR-100				Tiny-ImageNet			Average Rank
	Nat.	PGD ²⁰	CW ⁴⁰	AA	Nat.	PGD ²⁰	CW ⁴⁰	AA	Nat.	PGD ²⁰	AA	
BN	85.11	55.36	53.02	51.43	55.77	29.91	26.23	25.35	42.09	20.68	16.25	1.5
GN	85.28	55.82	52.76	51.23	56.60	29.86	26.26	25.09	30.99	16.87	13.01	1.7
IN	85.34	54.49	50.82	49.34	56.56	28.41	24.17	22.68	17.25	10.69	8.18	2.7
LN	-	-	-	-	-	-	-	-	-	-	-	4.0

A.7 EXTENDED DISCUSSION OF ROBUSTRESNETS

Table 4 provides the detailed settings of RobustResBlockA1-A4. Besides, we add extra comparisons between blocks and scalings in Figure 15. The derived compound scaling rule reduces the parameters significantly and further improves the robustness over RobustResBlock.

Table 4: The stage wise setting is presented using $[k \times k, \text{\#Ch}]$, where k denotes the convolution filter size, \#Ch denotes the number of output channels, and $[\cdot]$ indicates residual connection.

	RobustResNet-A1	RobustResNet-A2	RobustResNet-A3	RobustResNet-A4
Stem	$3 \times 3, 16, \text{stride } 1$			
Stage 1	$\begin{bmatrix} 1 \times 1, 160 \\ 3 \times 3, 160 \\ 1 \times 1, 320 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 224 \\ 3 \times 3, 224 \\ 1 \times 1, 448 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 320 \\ 3 \times 3, 320 \\ 1 \times 1, 640 \end{bmatrix} \times 3$
Stage 2	$\begin{bmatrix} 1 \times 1, 448 \\ 3 \times 3, 448 \\ 1 \times 1, 896 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 576 \\ 3 \times 3, 576 \\ 1 \times 1, 1152 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 704 \\ 3 \times 3, 704 \\ 1 \times 1, 1408 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 896 \\ 3 \times 3, 896 \\ 1 \times 1, 1792 \end{bmatrix} \times 3$
Stage 3	$\begin{bmatrix} 1 \times 1, 384 \\ 3 \times 3, 384 \\ 1 \times 1, 768 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 1024 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 640 \\ 3 \times 3, 640 \\ 1 \times 1, 1280 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 768 \\ 3 \times 3, 768 \\ 1 \times 1, 1536 \end{bmatrix} \times 3$
Tail	Global average pool			

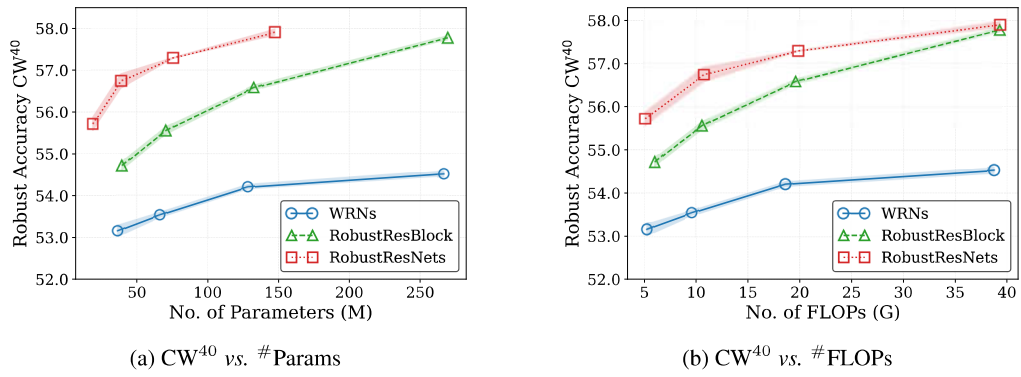


Figure 15: Comparisons between blocks and scalings. Both WRN and RobustResBlock adopt standard scaling, while RobustResNets utilize the proposed compound scaling and RobustResBlock.