Are All Layers Created Equal?

Chiyuan Zhang
chiyuan@google.com

Samy Bengio
bengio@google.com

Yoram Singer
singer@google.com

Abstract

Understanding learning with deep architectures has been a major research objective in the recent years with notable theoretical progress. A main focal point of those studies stems from the success of excessively large networks. We study empirically the layer-wise functional structure of overparameterized deep models. We provide evidence for the heterogeneous characteristic of layers. To do so, we introduce the notion of (post training) re-initialization and re-randomization robustness. We show that layers can be categorized into either "robust" or "critical". In contrast to critical layers, resetting the robust layers to their initial value has no negative consequence, and in many cases they barely change throughout training. Our study provides evidence flatness or robustness analysis of the model parameters needs to respect the network architectures.

1 Introduction

Deep neural networks have been remarkably successful in many real world machine learning applications. Distilled understanding of the systems is at least as important as their state-of-the-art performance when applying them in many critical domains. Recent work on understanding why deep networks perform so well in practice focused on questions such as networks' performance under drifting or even adversarially perturbed data distribution. Another interesting and relevant to this work is research on how we can interpret or explain the decision function of trained networks. While related, this work takes a different angle as we focus on the role of the layers in trained networks and then relate the empirical results to robustness properties.

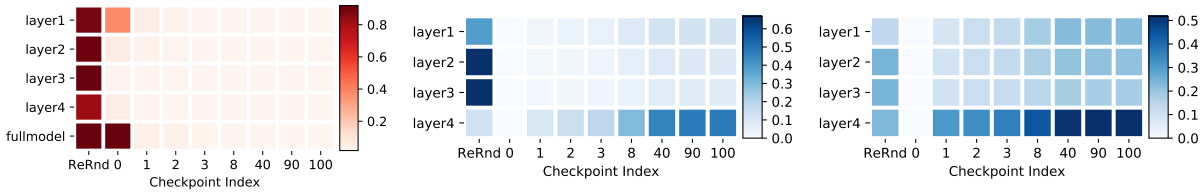
Theoretical research on the representation power of neural networks is well studied. It is known that a neural network with a single sufficiently wide hidden layer is universal approximator for continuous functions over a compact domain [11, 18, 2]. More recent research further examines whether *deep* networks can have superior representation power than *shallow* ones with the same number of units or edges [27, 5, 24, 35, 31, 9, 23, 29]. The capacity to represent arbitrary functions with finite samples is also extensively discussed in recent work [13, 40, 26, 39]. However, the constructions used in the aforementioned work for building networks approximating particular functions are typically "artificial" and are unlikely to be obtained by gradient-based learning algorithms. We focus instead on empirically studying the role different layers in deep architecture take *post* gradient-based training.

In particular, we show empirically that the layers in a deep network are not homogeneous in the role they play at representing a prediction function. Some layers are critical to forming good predictions while others are fairly robust to the assignment of their parameters along training. Moreover, depending on the capacity of the network and the complexity of the target function, gradient-based trained networks conserve the complexity by not using excess capacity.

2 Setting

Let $\mathcal{F}^D = \{f_\theta : \theta = (\theta_1, \dots, \theta_D)\}$ be the space of a particular neural network architecture with D (parametric) layers. We use the term *capacity* to refer to properties of the entire space $\mathcal{F}^D$ before training takes place (e.g. *Rademacher complexity*, *VC Dimension*). The term *complexity* refers to properties of a *single* neural network f_θ, often employing with notion of *norm* of the parameters θ and possibly normalized by empirical quantities such as the *margin*.

We are interested in analyzing post-training behavior of layers of popular deep networks. Such networks are typically trained using SGD with randomly sampled initial weights from pre-defined distributions $\theta_d^0 \sim \mathcal{P}_d$, typically depending



(a) Test error (b) $\|\theta_d^\tau - \theta_d^0\|$ (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 1: **Robustness results for FCN** 3×256 **on MNIST.** (a) Test error rate: each row corresponds to one layer in the network. The last row shows the full model performance at the corresponding epoch as reference. The first column designates robustness of each layer w.r.t re-randomization and the rest of the columns designate re-initialization robustness at different checkpoints. The last column shows the final performance (at the last checkpoint during training) as reference. (b-c) Weights distances: each cell in the heatmaps depict the normalized 2-norm (b) or ∞-norm (c) distance of trained parameters to their initial weights.

on the type, fan-in, and fan-out of of each layer. In a deep network, the representations at a particular layer recursively depend on all the layers beneath it. This complex dependency makes it challenging to isolate and inspect each layer independently in theoretical studies. In this paper, we introduce and use the following two empirical probes to inspect the individual layers in a trained neural network.

Re-initialization After training, for a given layer $d = 1, \dots, D$, we can *re-initialize* the parameters through assignment $\theta_d^T \longleftarrow \theta_d^0$, while keeping the parameters for the other layers unchanged. The model with the parameters $(\theta_1^T, \dots, \theta_{d-1}^T, \theta_d^0, \theta_{d+1}^T, \dots, \theta_D^T)$ is then evaluated. Unless noted otherwise, we use the term performance to designate *classification* error on *test* data. The performance of a network in which layer d was re-initialized is referred to as the *re-initialization robustness* of layer d. Note that here θ_d^0 denotes the random values realized at the beginning of the training. More generally, for k time steps $0 = \tau_1 < \tau_2 < \cdots < \tau_{k-1} < \tau_k = T$, we can *re-initialize* the d-th layer by setting $\theta_d^T \longleftarrow \theta_d^\tau$, and obtain the *re-initialization robustness* of layer d after τ updates.

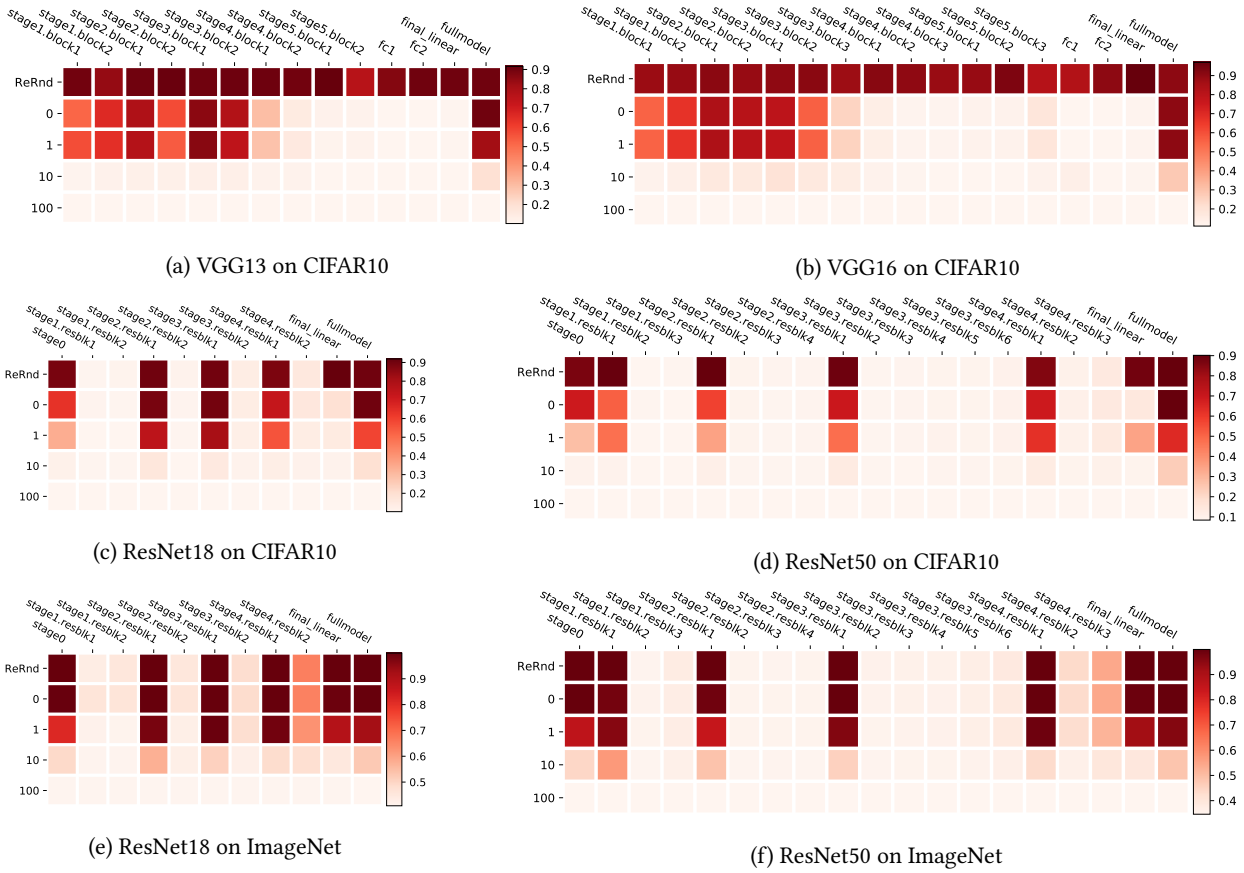
Re-randomization To go one step further, we also examine *re-randomization* of a layer d by re-sampling random values $\tilde{\theta}_d \sim \mathcal{P}_d$ and evaluate the model's performance for $(\theta_1^T, \dots, \theta_{d-1}^T, \tilde{\theta}_d, \theta_{d+1}^T, \dots, \theta_D^T)$. Analogously, we refer to the evaluated performance as the *re-randomization robustness* of layer d.

Note that there is *no* re-training or finetuning after re-initialization or re-randomization, and the network is evaluated directly with mixed weights. When a network exhibits no or negligible decrease in performance after re-initializing or re-randomizing of a layer, we say that the layer is robust, and otherwise the layer is called critical.

3 Robustness of individual layers

We start by examining robustness of fully-connected networks (FCN). A FCN $D \times H$ consists of D fully connected layers each of which of output dimension H and ReLU activation function. The extra final layer is a linear multiclass predictor with one output per class. As a starter, we trained an FCN 3×256 on the MNIST digit classification task, and applied the re-initialization and re-randomization analysis on the trained model. The results are shown in Figure 1(a). As expected, due to the intricate dependency of the classification function on each of the layers, re-randomizing any of the layers completely disintegrate the representations and classification accuracy drops to the level of random guessing. However, for re-initialization, we find that while the first layer is very sensitive, the rest of the layers are robust to re-initializing back to their pre-training random weights.

A plausible reason could be attributed to the fact that gradient norms increase during back-propagation to the point that the bottom layers are being updated more aggressively than the top ones. If this was the case, we would expect a smoother transition instead of a sharp contrast at the first layer. We thus measured how distant the weights of each layer from their initialization using the *normalized* 2-norm and the ∞-norm, in Figure 1(b) and (c), respectively. It turns out the robustness to re-initialization does not obviously correlate to either of the distances.



(a) VGG13 on CIFAR10

(b) VGG16 on CIFAR10

(c) ResNet18 on CIFAR10

(d) ResNet50 on CIFAR10

(e) ResNet18 on ImageNet

(f) ResNet50 on ImageNet

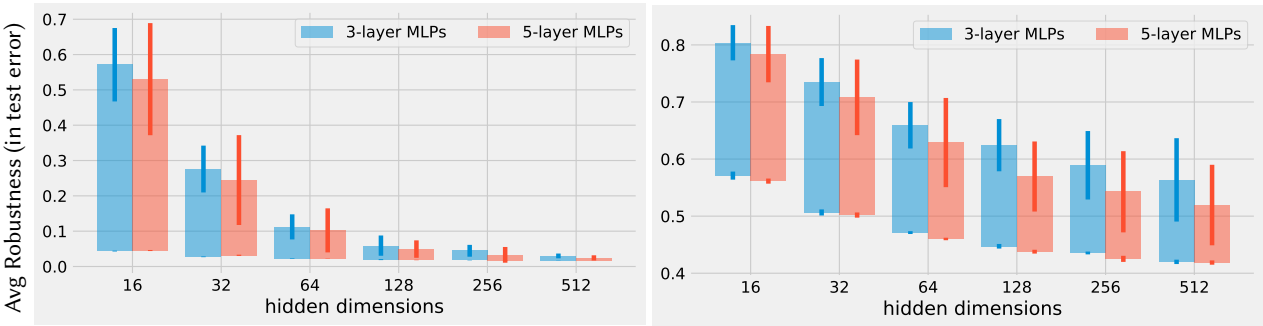
Figure 2: **Layer-wise robustness analysis with ConvNets.** The heatmaps use the same layout as in Figure 1, but they are transposed, to visualize the deeper architectures more effectively.

On large ConvNets that are used in practice, similar obversations could be found in Figure 2. Moreover, the layerwise robustness patterns for ResNets are quite unique. We found that each stage (in which the "image" size of each layer is the same) in a ResNet acts as a sub-network, and the robustness patterns *within* each stage resembles the VGGs and FCNs. The skip connections also allow the residual blocks to be robust not only to re-initialization, but also to re-randomization. Please see Appendix B for the full results and analysis.

To assess the effects of the network capacity and the task complexity on the layer robustness, we apply the same analysis procedure to a large number of different configurations. In Figure 3(a), we compare the average re-initialization robustness for all layers but the first with respect FCNs of varying hidden dimensions on MNIST. The upper layers become more robust as the hidden dimension increases. We believe that it reflects the fact that the wider FCNs have higher model capacity. When the capacity is small, all layers are vigil participants in representing the prediction function. As capacity increases, it suffices to use the bottom layer while the rest act as random projections with non-linearities. Similar observations can be found on CIFAR10, in Figure 3(b). These observations suggest that deep networks *automatically* adjust their de-facto complexity. When a big network is trained on an easy task, only a few layers seem to be playing critical roles.

4 Joint robustness and connection to other notions of robustness

In all the experiments so far, the analysis focuses on the robustness of each individual layer separately. A natural question is that for all the layers that are individually robustness, are they also jointly robust? In other words, if we re-initialize multiple layers jointly, would the trained network still retain the original performance? The answer is no. It turns out that jointly re-initialize many consequtive layers (that are individually robust) completely



(a) MNIST

(b) CIFAR10

Figure 3: **Re-initialization robustness of all layers but the first using checkpoint-0 for FCNs.** Each bar designates the difference in classification error between a fully trained model and a model with one layer re-initialized. The error bars designate one standard deviation obtained by running five experiments with different random initialization.

destroy the model performance. However, via some clever grouping schemes of the layers, multiple layers *can* be jointly re-initialized while maintaining resonable good model performances. Furthermore, if we explicitly constraing the learning algorithm to not update a subset of layers, much better joint robustness can be achieved. Please see Appendix D for full details.

The properties studied in this paper is closely related to a number of other notions of robustness. For example, the notion of "flatness" refers to the property that neural network weights, after training, can be locally perturbed via isotropic noises without changing the model performance. Our study is in a more restricted setting where we perturb by moving along the training trajectory. More importantly, our empirical results show that while global flatness is quite limited, (a subset of) the individual layers can have negligible affects when perturbed. Another interesting connection is to the notion of adversarial robustness to the perturbations applied on the model inputs. Please see Appendix E for a full study on those connections.

5 Conclusions

We investigated the functional structure on a layer-by-layer basis of over-parameterized deep models. We introduced the notions of re-initialization and re-randomization robustness. Using these notions we provided evidence for the heterogeneous characteristic of layers, which can be morally categorized into either "robust" or "critical". Resetting the robust layers to their initial value has no negative consequence on the model's performance. Our empirical results give evidence that optimization landscape based analysis (e.g. flatness or sharpness at the minimizer) is better performed respecting the network architectures due to the heterogeneous behaviors of different layers. For future work, we are interested in devising a new algorithm which learns the interleaving trained and partially random subnetworks within one large network.

Acknowledgments The authors would like to thank David Grangier, Lechao Xiao, Kunal Talwar and Hanie Sedghi for helpful discussions and comments.

References

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via Over-Parameterization. *CoRR*, arXiv:1811.03962, 2018.

[2] Martin Anthony and Peter L Bartlett. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, 2009.

[3] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *CoRR*, arXiv:1802.05296, 2018.

[4] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. In *ICLR*, 2017.

[5] Olivier Delalleau and Yoshua Bengio. Shallow vs. Deep Sum-Product Networks. In *NIPS*, pages 666–674, 2011.

[6] Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *CoRR*, arXiv:1811.03804, 2018.

[7] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *CoRR*, arXiv:1810.02054, 2018.

[8] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *UAI*, 2016.

[9] Ronen Eldan and Ohad Shamir. The Power of Depth for Feedforward Neural Networks. *CoRR*, arXiv:1512.03965, 2015.

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

[11] G Gybenko. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

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

[13] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. In *ICLR*, 2017.

[14] 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.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *CoRR*, arXiv:1503.02531, 2015.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[18] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[19] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8580–8589, 2018.

[20] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*, 2017.

[21] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *arXiv preprint arXiv:1902.06720*, 2019.

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

[23] Hrushikesh Mhaskar and Tomaso A. Poggio. Deep vs. shallow networks : An approximation theory perspective. *CoRR*, arXiv:1608.03287, 2016.

[24] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems (NIPS)*, pages 2924–2932, 2014.

[25] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *NIPS*, pages 2422–2430, 2015.

[26] Quynh Nguyen and Matthias Hein. Optimization Landscape and Expressivity of Deep CNNs. In *International Conference on Machine Learning*, pages 3727–3736, 2018.

[27] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[28] Tomaso Poggio, Qianli Liao, Brando Miranda, Andrzej Banburski, Xavier Boix, and Jack Hidary. Theory iiib: Generalization in deep networks. Technical report, MIT, 2018.

[29] David Rolnick and Max Tegmark. The power of deeper networks for expressing natural functions. *CoRR*, arXiv:1705.05502, 2017.

[30] Amir Rosenfeld and John K Tsotsos. Intriguing Properties of Randomly Weighted Networks: Generalizing While Learning Next to Nothing. *CoRR*, arXiv:1802.00844, 2018.

[31] Uri Shaham, Alexander Cloninger, and Ronald R Coifman. Provable approximation properties for deep neural networks. *CoRR*, arXiv:1509.07385, 2015.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] Samuel L Smith and Quoc V Le. A bayesian perspective on generalization and stochastic gradient descent. In *ICLR*, 2018.

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

[35] Matus Telgarsky. benefits of depth in neural networks. In Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1517–1539, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR.

[36] Vladimir N Vapnik. *Statistical Learning Theory*. Adaptive and learning systems for signal processing, communications, and control. Wiley, January 1998.

[37] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

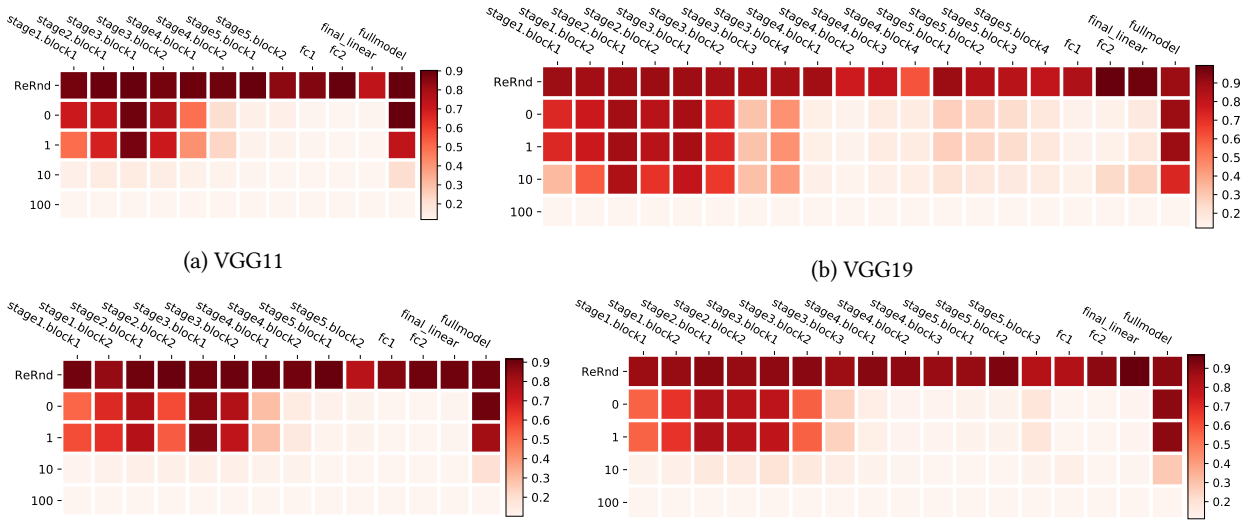
[38] Huan Wang, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. Identifying Generalization Properties in Neural Networks. *CoRR*, arXiv:1809.07402, 2018.

[39] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Finite sample expressive power of small-width relu networks. *CoRR*, arXiv:1810.07770, 2018.

[40] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

[41] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the ImageNet scale: a PAC-Bayesian compression approach. In *ICLR*, 2019.

[42] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Stochastic gradient descent optimizes over-parameterized deep ReLU networks. *CoRR*, arXiv:1811.08888, 2018.



(a) VGG11

(b) VGG19

(c) VGG13

(d) VGG16

Figure 4: **Layer-wise robustness analysis with VGG networks on CIFAR10.** The heatmaps use the same layout as in Figure 1, but they are transposed, to visualize the deeper architectures more effectively.

A Related work

Modern neural networks are typically over-parameterized and thus have plenty of redundancy in their representation capabilities. Previous work exploited over-parameterization to compress [12] or distill [16] a trained network. Rosenfeld and Tsotsos [30] found that one can achieve comparable performance by training only a small fraction of network parameters such as a subset of channels in each convolutional layer. Towards interpreting residual networks as ensemble of shallow networks, Veit et al. [37] found that residual blocks in a trained network can be deleted or permuted to some extent without hurting the performance too much. In another line of research, it is shown that under extreme overparameterization, such as when the network width is polynomial in the training set size and input dimension [1, 6, 7, 42], or even in the asymptotic regime of infinite width [19, 21], the network weights move slowly during training. The observations in this paper show that in more practical regime, different layers could behave very differently.

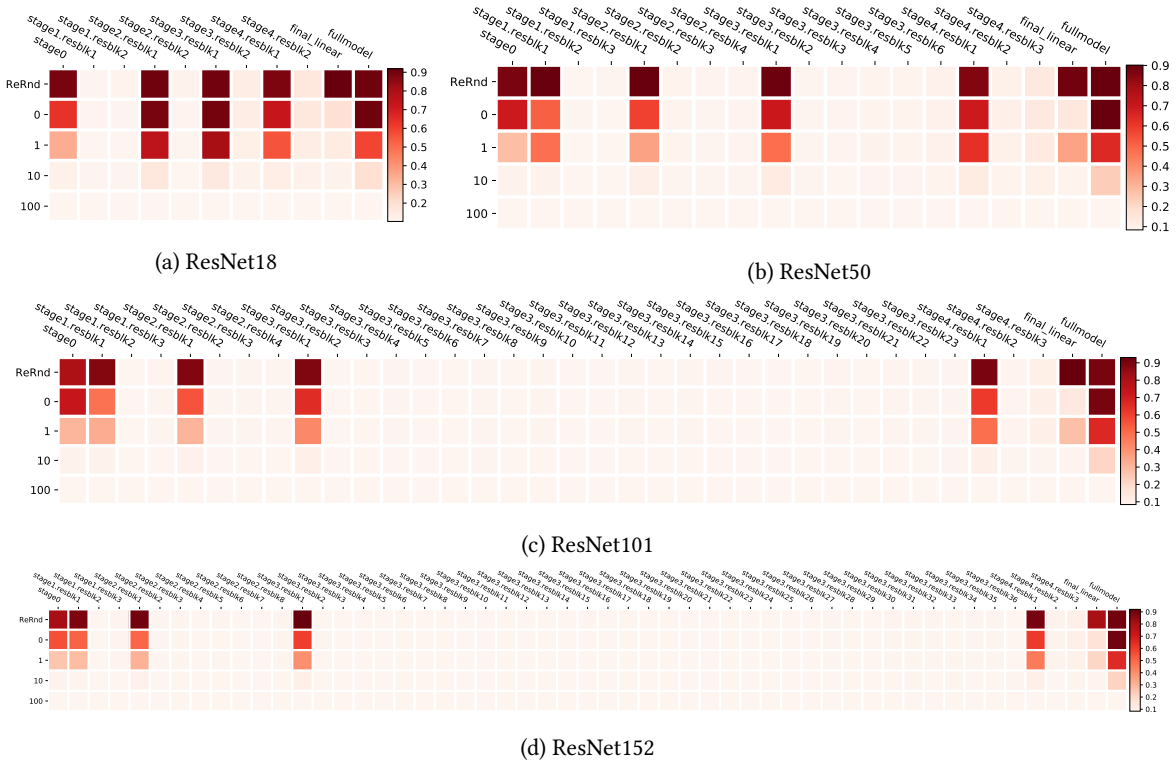
B Full results on layerwise robustness analysis of convolutional networks

On typical computer vision tasks beyond MNIST, densely connected FCNs are outperformed significantly by convolutional neural networks. VGGs and ResNets are among the most widely used convolutional network architectures. Figure 4 and Figure 5 show the robustness analysis on the two types of networks, respectively.

Since those networks are much deeper than the FCNs, we transpose the heatmaps to show the layers as columns. For VGGs, a large number of layers are sensitive to re-initialization, but the patterns are similar to the observations from the simple FCNs on MNIST: the bottom layers are more critical but the upper layers are robust to re-initialization.

The results for ResNets in Figure 5 is to be considered together with results on ImageNet in Figure 6. We found the robustness patterns for resnets more interesting mainly for two reasons:

ResNets re-distribute sensitive layers. Unlike the FCNs and VGGs which put the sensitive layers at the bottom of the network, ResNets distribute them across the network. To better understand the patterns, let us do a brief recap of the ResNets architectures. It is common in theoretical analysis to broadly define ResNets as any neural network architectures with *residual blocks*. In practice, a few "standard" architectures (and variants) that divide the network into a few "stages" are commonly used. At the bottom, there is a pre-processing stage (`stage0`) with



(a) ResNet18

(b) ResNet50

(c) ResNet101

(d) ResNet152

Figure 5: **Layer-wise robustness analysis on residual blocks of ResNets trained on CIFAR10.**

vanilla convolutional layers. It is followed by a few (typically 4) residual stages (`stage1` to `stage4`) consisting of residual blocks, and then global average pooling and the densely connected linear classifier (`final_linear`). The image size shrinks and the number of convolutional feature channels doubles from each residual stage to the next one[1]. As a result, while most of the residual blocks have real *identity* skip connections, the first block of each stage (`stage*.resblk1`) that connects to the previous stage has a *non-identity* skip connection due to different input / output shapes. Figure 7 illustrates the two types of residual blocks.

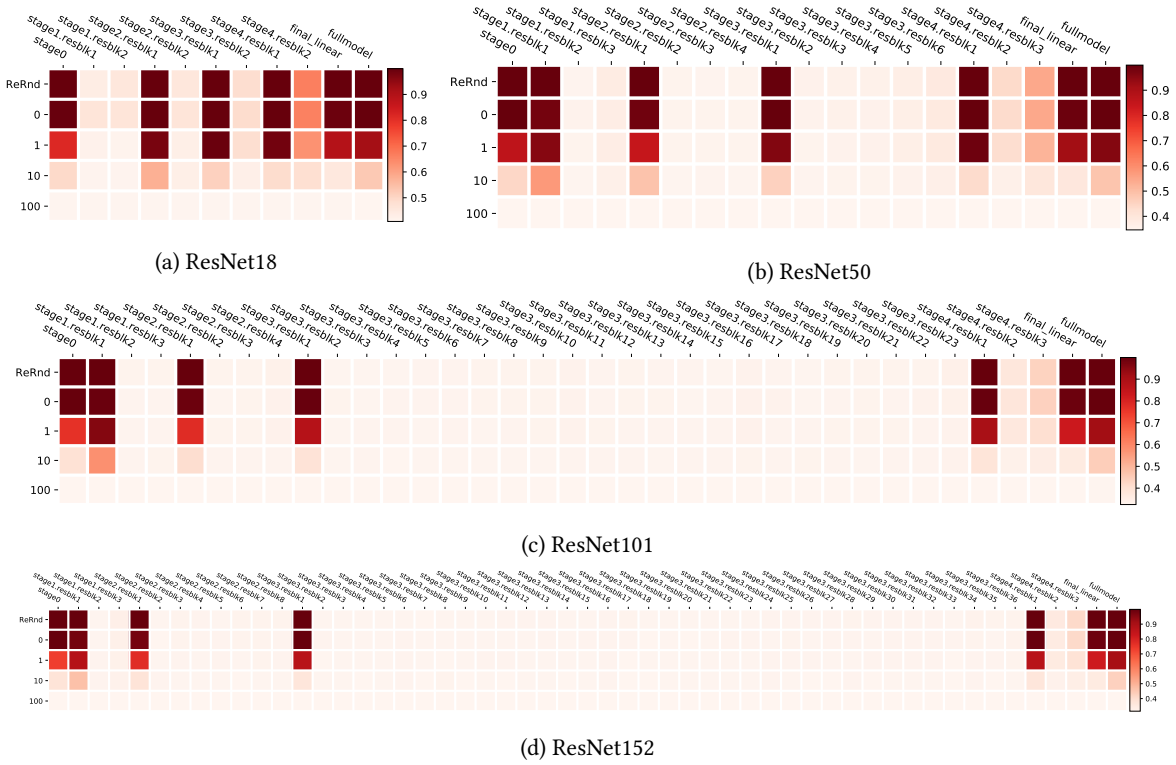
With a big picture of the ResNet architectures, we can see that each stage in a ResNet acts as a sub-network, and the layer-wise robustness patterns *within* each stage resembles the VGGs and FCNs.

Residual blocks can be robust to re-randomization. Among the layers that are robust to re-initialization, if the layer is a residual block, it is also robust to re-randomization: e.g. compare the `final_linear` layer and any of the robust residual blocks. A possible reason is that the identity skip connection dominates the residual branch in those blocks. It is known from previous lesion studies [37] that residual blocks in a ResNet can be removed without seriously hurting the performance. But our experiments put it in the context with other architectures and study the adaptive robustness with respect to the interplay between the model capacity and the task difficulties. In particular, comparing the results on CIFAR10 and ImageNet, we see that especially on ResNet18 from Figure 6(a), many residual blocks with real identity skip connection also become sensitive comparing to bigger models due to smaller capacity.

C Theoretical Implications on Generalization

As mentioned earlier, if some parameters can be re-assigned to the randomly initialized values without affecting the model performance, then the effective number of parameters is reduced as the random initialization is independent of the training data. The benefits on improving generalization is most easily demonstrated with a naive parameter

[1]There are more subtle details especially at `stage1` depending on factors like the input image size, whether residual blocks contain a bottleneck, and the version of ResNets, etc.



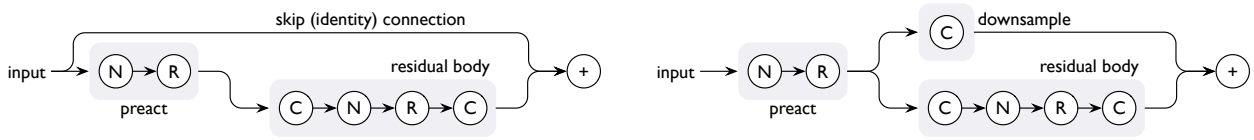
(a) ResNet18

(b) ResNet50

(c) ResNet101

(d) ResNet152

Figure 6: **Layer-wise robustness analysis on residual blocks of ResNets trained on ImageNet.**



(a) Residual block

(b) Residual block with downsampling

Figure 7: **Illustration of residual blocks (from ResNets V2) with and without a downsampling skip branch.** C, N and R stand for convolution, (batch) normalization and ReLU activation, respectively. Those are *basic* residual blocks used in ResNet18 and ResNet34; for ResNet50 and more layers, the *bottleneck* residual blocks are used, which are similar to the illustrations here except the residual body is now $\mathsf{C} \to \mathsf{N} \to \mathsf{R} \to \mathsf{C} \to \mathsf{N} \to \mathsf{R} \to \mathsf{C}$ with a 4× reduction of the convolution channels in the middle for a "bottlenecked" residual.

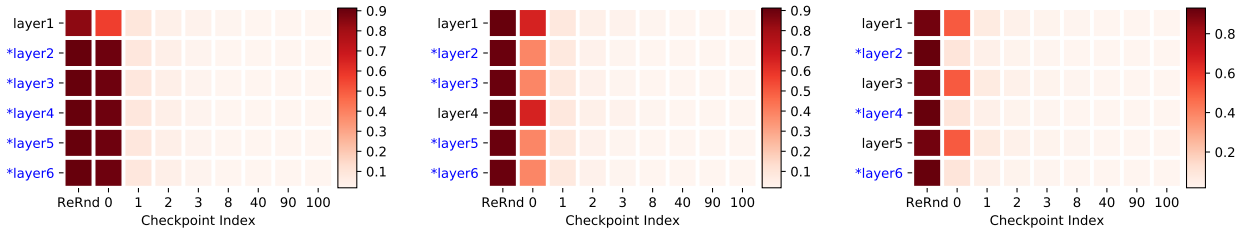
counting generalization bound. For example, if we have a generalization bound of the form

$$R(\hat{f}_n^m) \leq \hat{R}_n(\hat{f}_n^m) + \mathcal{B}(\mathcal{C}(m), n)$$

where $\hat{f}_n^m$ is a model with m parameters trained on n i.i.d. samples. $\mathcal{C}(\cdot)$ is some complexity measure based on counting the number of parameters, and $\mathcal{B}$ is the corresponding generalization bound. For example, Anthony and Bartlett [2] provides various bounds on VC-dimension based on the number of weights in neural networks, which could then be plugged into standard VC-dimension based generalization bounds for classification [36]. Now if we know that a fraction $\rho \in (0, 1)$ of the neural network weights will be robust to re-initialization after training, with a loss of the (empirical) risk of at most ε, then we get

$$R(\hat{f}_n^{(1-\rho)m}) \leq \hat{R}_n(\hat{f}_n^m) + \varepsilon + \mathcal{B}(\mathcal{C}((1-\rho)m), n)$$

where $\hat{f}_n^{(1-\rho)m}$ is a model obtained by re-initializing the ρ fraction of parameters of the trained model $\hat{f}_n^m$. Note that generalization bounds based on parameter counting generally does not work well for deep learning. Because of the



(a) layer2~6 (b) layer2,3,5,6 (c) layer2,4,6

Figure 8: **Joint robustness analysis of FCN** 5×256 **on MNIST.** The heatmap layout is the same as in Figure 1, but the layers are divided into two groups (indicated by the $\star$ mark on the layer names in each figure) and re-randomization and re-initialization are applied to all the layers in each group *jointly*. As a result, layers belonging to the same group have identical rows in the heatmap, but we still show all the layers to make the figures easier to read and compare with the previous *layer-wise* robustness results. The subfigures show the results from three different grouping schemes.

heavy over-parameterization, the resulting bounds are usually trivial. However, as noted in Arora et al. [3], most of the alternative generalization bounds proposed for deep neural network models recently are actually worse than naive parameter counting. Moreover, by tweaking the existing analysis with additional layerwise robustness condition, some PAC-Bayes based bounds can also be potentially improved [38, 3, 41].

Note that like the results in Arora et al. [3], Zhou et al. [41], the bounds provided by re-initialization robustness are for a different model (in our case the re-initialized one). Alternative approaches in the literature involve modifying the training algorithms to explicitly optimize the robustness or some derived generalization bounds [25, 8]. However, neither of the arguments provides guarantees for the model *directly* trained from SGD.

D Joint robustness

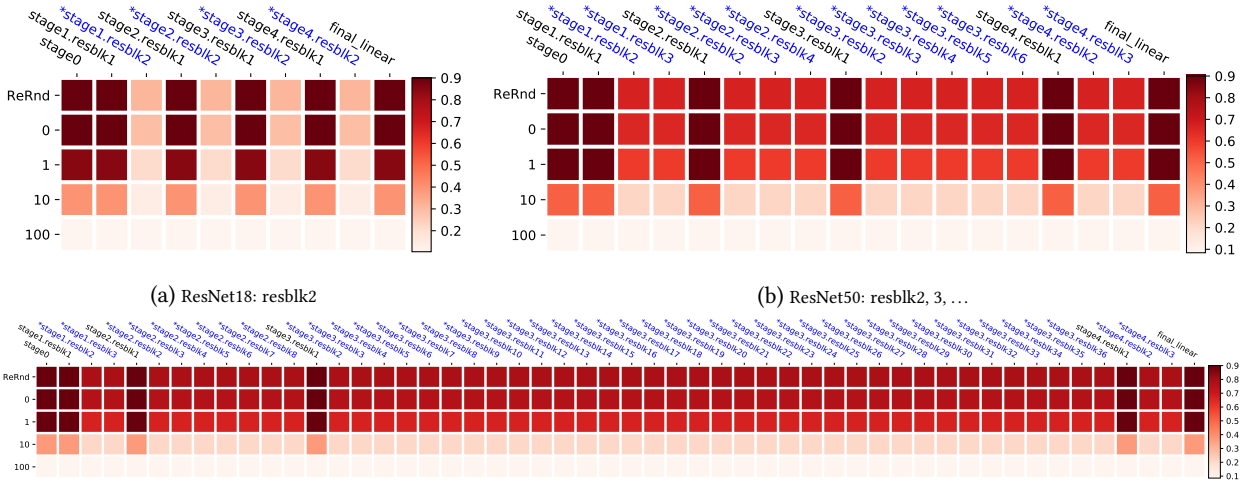
The theoretical analysis suggests that robustness to either re-initialization or re-randomization could imply better generalization. Combined with the experimental results in previous sections, it seems to suggest a good way to explain the empirical observations that hugely over-parameterized networks could still generalize well, as they are only using a small portion of their full capacity. However, there is a caveat: the re-initialization and re-randomization analysis in Section 3 study each layer independently. However, two or more layers being independently robust does not necessarily imply that they are robust *jointly*. If, for example, we want a generalization bound that uses only half of the capacity, we need to show that half of the layers are robust to re-initialization or re-randomization *simultaneously*.

D.1 Are robust layers jointly robust?

In this section, we do joint robustness analysis on groups of layers. From Section 3, we see that on MNIST, for wide enough FCNs, all the layers above `layer1` are robust to re-initialization. So we divide the layer into two groups: {`layer1`} and {`layer2`, `layer3`, ...}, and perform the robustness studies on the two groups. The results for FCN 5×256 are shown in Figure 8(a). For clarity and ease of comparison, the figure still spells out all the layers individually, but the values from `layer2` to `layer6` are simply repeated rows. The values show that the upper-layer-group is clearly *not* jointly robust to re-initialization (to checkpoint 0).

We also try some alternative grouping schemes: Figure 8(b) show the results when we group two in every three layers, which has slightly improved joint robustness; In Figure 8(c), the grouping scheme that include every other layer shows that with a clever grouping scheme, about half of the layers could be *jointly* robust.

Results on ResNets are similar. Figure 9 shows the joint robustness analysis on ResNets trained on CIFAR10. The grouping is based on the layer-wise robustness results from Figure 5: all the residual blocks in `stage1` to `stage4` are bundled and analyzed jointly. The results are similar to the FCNs: ResNet18 is relatively robust, but deeper ResNets are *not* jointly robust under this grouping. Two alternative grouping schemes are shown in Figure 10. By including

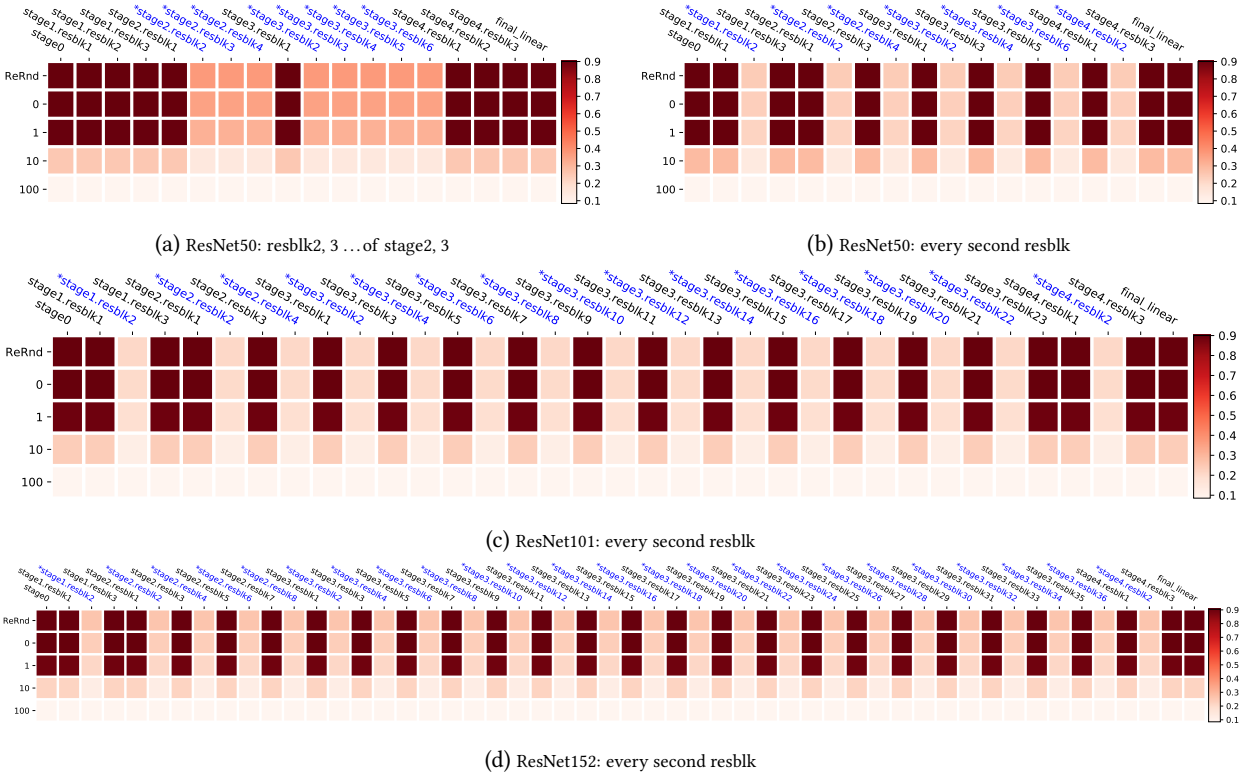


(a) ResNet18: resblk2

(b) ResNet50: resblk2, 3, …

(c) ResNet152: resblk2, 3, …

Figure 9: **Joint robustness analysis of ResNets on CIFAR10**, based on the scheme that group all but the first residual blocks in all the residual stages. Grouping is indicated by the ⋆ on the layer names.



(a) ResNet50: resblk2, 3 …of stage2, 3

(b) ResNet50: every second resblk

(c) ResNet101: every second resblk

(d) ResNet152: every second resblk

Figure 10: **Joint robustness analysis of ResNets on CIFAR10**, with alternative grouping schemes. Grouping is indicated by the ⋆ on the layer names.

Table 1: **Error rates (%) on CIFAR10 (top rows) and ImageNet (bottom rows), respectively.** Each row shows the performance of the full model, (the mean and std of) the layer-wise robustness to re-initialization, the performance when training with a subset of layers fixed at random initialization, and the performance when training with a subset of layers removed. In particular, the layer-wise robustness is averaged over all the residual blocks except the first one at each stage. The layer-freezing and layer-removal operations are also applied to those residual blocks (jointly).

	Arch	Full Model	Layer-wise Robustness	Layers Frozen	Layers Removed
CIFAR10	ResNet50	8.40	9.77±1.38	11.74	9.23
	ResNet101	8.53	8.87±0.50	9.21	9.23
	ResNet152	8.54	8.74±0.39	9.17	9.23
ImageNet	ResNet50	34.74	38.54±5.36	44.36	41.50
	ResNet101	32.78	33.84±2.10	36.03	41.50
	ResNet152	31.74	32.42±1.55	35.75	41.50

only layers from `stage1` and `stage4`, slightly improved robustness could be obtained on ResNet50. The scheme that groups every other residual block shows further improvements.

In summary, the individually robust layers are generally not jointly robust. But with some clever way of picking out a subset of the layers, joint robustness could still be achieved for up to half of the layers. In principle, one can enumerate all possible grouping schemes to find the best with a trade-off of the robustness and number of layers included.

D.2 Could robust layers be made jointly robust?

Results from the previous section show that there is a gap between the layer-wise robustness patterns and the the joint robustness. Here we try to see if we could close the gap by letting the training algorithm know that we are interested in the robustness of a subset of the layers. It is complicated to express this desire algorithmically, but we can make a stronger request by asking the learning algorithm to explicitly not "use" those layers. More specifically, we try two approaches to the layers in the group that is desired to be robust: 1) freeze them so that their parameters remain the same randomly initialized values; 2) remove the layers completely from the neural network architecture.

The results are shown in Table 1. When we explicitly freeze the layers, the test error rates are still higher than the average layer-wise robustness measured in a normally trained model. However, the gap is much smaller than directly measuring the joint robustness (see Figure 9 for comparison). Moreover, on CIFAR10, we found that similar performance can be achieved even if we completely remove those layers from the network. On the other hand, on ImageNet, the frozen random layers seem to be needed to achieve good performances, while the "layers-removed" variant under-perform by a big gap. In this case, the random projections (with non-linearity) in those frozen layers are helpful with the performance.

E Connections to other notions of robustness

The notion of layer-wise (and joint) robustness to re-initialization and re-randomization can be related to other notions of robustness in deep learning. For example, the *flatness* of the solution is a notion of robustness with respect to local perturbations to the network parameters (at convergence), and is extensively discussed in the context of generalization [17, 4, 20, 33, 28]. For a fixed layer, our notion of robustness to re-initialization is more restricted because the "perturbed values" can only be from the optimization trajectory; while the robustness to re-randomization could potentially allow larger perturbation variances. However, as our studies here show, the robustness or flatness at each layer could behave very differently, so analyzing each layer individually in the context of specific network architectures allow us to get more insights to the robustness behaviors.

On the other hand, *Adversarial robustness* [34] focus on the robustness with respect to perturbations to the inputs. In

Table 2: **Accuracies (%) of various model configurations on clean CIFAR10 test set and under a weak (FGSM) and a strong (PGD) adversarial attack, respectively.** The adversarial attacks are evaluated on a subset of 1000 test examples. Every experiment is repeated 5 times and the average performance is reported. The hyperparameters r and s in model configurations mean the number of random weights pre-created for each residual block, and the number of stages that are re-randomized during each inference. 4^2 means a ResNet architecture with two stages, where each stage contains four residual blocks; similarly 4^4 has four stages each with four residual blocks.

Model Configuration		Clean	FGSM	PGD
4^2	baseline	91.05 ± 0.00	12.75 ± 0.04	0.33 ± 0.16
	r=4,s=1	89.45 ± 0.13	69.85 ± 1.60	6.71 ± 0.37
	r=4,s=2	87.70 ± 0.25	71.18 ± 0.49	9.65 ± 0.27
4^4	baseline	90.08 ± 0.00	8.45 ± 0.00	0.00 ± 0.00
	r=4,s=1	89.64 ± 0.12	62.76 ± 1.09	2.60 ± 0.26
	r=4,s=2	89.13 ± 0.13	67.20 ± 0.63	3.56 ± 0.48
	r=4,s=4	88.24 ± 0.18	69.09 ± 1.59	5.60 ± 0.53

particular, it is found that trained deep neural network models are sensitive to input perturbations: small adversarially generated perturbations can usually change the prediction results to arbitrary different classes. A large number of defending and attacking algorithms have been proposed in recent years along this line. Here we briefly discuss the connection to adversarial robustness. In particular, take a normally trained ResNet[2], say with S stages and $(B_1, \dots, B_S)$ residual blocks in each stage. Given configuration $r > 0$ and $0 \leq s \leq S$, during each test evaluation, a subset of s stages are randomly chosen, and for each of the chosen stages, a random residual block is picked and replaced with one of the r pre-initialized weights for that layer. We keep r pre-allocated weights for each residual block instead of re-sampling random numbers on each evaluation call, primarily to reduce the computation burden during the test time.

From the previous robustness analysis, we expect the stochastic classifier to get only a small performance drop when averaged over the test set. However, at individual example level, the randomness of the network outputs will make it harder for the attacker to generate adversarial examples. We evaluate the adversarial robustness against a weak FGSM [10] attack and a strong PGD [22] attack. The results in Table 2 show that, compared to the baseline (the exact same trained model before being turned into a stochastic classifier), the randomness significantly increases the adversarial robustness against weak attacks. The performances under strong PGD attack drop to very low, but still with a non-trivial gap between the baseline.

In summary, the layer-wise robustness could improve the adversarial robustness of a trained model through injected stochasticity. However, it is not a good defense against strong attackers. If we work hard enough, more sophisticated attacks that explicitly deal with stochastic classifiers are likely to completely break this model.

F Details on experiment setup

Our empirical studies are based on the MNIST, CIFAR10 and the ILSVRC 2012 ImageNet datasets. Stochastic Gradient Descent (SGD) with a momentum of 0.9 is used to minimize the multi-class cross entropy loss. Each model is trained for 100 epochs, using a stage-wise constant learning rate scheduling with a multiplicative factor of 0.2 on epoch 30, 60 and 90. Batch size of 128 is used, except for ResNets with more than 50 layers on ImageNet, where batch size of 64 is used due to device memory constraints.

We mainly study three types of neural network architectures:

- FCNs: the multi-layer perceptrons consist of fully connected layers with equal output dimension and ReLU activation (except for the last layer, where the output dimension equals the number of classes and no ReLU is

[2]We use a slightly modified variant by explicitly having a downsample layer between stages, so that all the residual blocks are with real *identity* skip connections. See Figure 7.

Table 3: **Test performance (classification error rates %) of various models studied in this paper.** The table shows how much of the final performance is affected by training with or without weight decay (+wd) and batch normalization (+bn).

	Architecture	N/A	+wd	+bn	+wd+bn
CIFAR10	ResNet18	10.4	7.5	6.9	5.5
	ResNet34	10.2	6.9	6.6	5.1
	ResNet50	8.4	9.9	7.6	5.0
	ResNet101	8.5	9.8	6.9	5.3
	ResNet152	8.5	9.7	7.3	4.7
	VGG11	11.8	10.7	9.4	8.2
	VGG13	10.3	8.8	8.4	6.7
	VGG16	11.0	11.4	8.5	6.7
	VGG19	12.1		8.6	6.9
ImageNet	ResNet18	41.1	33.1	33.5	31.5
	ResNet34	39.9	30.6	30.1	27.2
	ResNet50	34.8	31.8	28.2	25.0
	ResNet101	32.9	29.9	26.9	22.9
	ResNet152	31.9	29.1	27.6	22.6

applied). For example, FCN 3×256 has three layers of fully connected layers with the output dimension 256, and an extra final (fully connected) classifier layer.

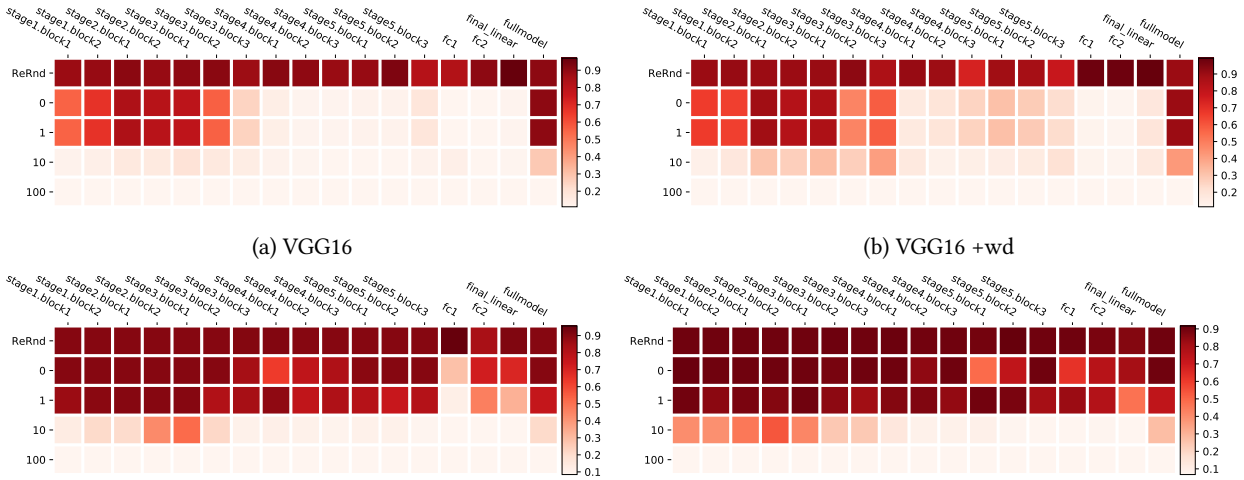
- VGGs: widely used network architectures from Simonyan and Zisserman [32].
- ResNets: the results from our analysis are similar for ResNets V1 [14] and V2 [15]. We report our results with ResNets V2 due to the slightly better performance in most of the cases. For large image sizes from ImageNet, the `stage0` contains a 7×7 convolution and a 3×3 max pooling (both with stride 2) to reduce the spatial dimension (from 224 to 56). On smaller image sizes like CIFAR10, we use a 3×3 convolution with stride 1 here to avoid reducing the spatial dimension.

During training, CIFAR10 images are padded with 4 pixels of zeros on all sides, then randomly flipped (horizontally) and cropped. ImageNet images are randomly cropped during training and center-cropped during testing. Global mean and standard deviation are computed on all the training pixels and applied to normalize the inputs on each dataset.

G Batch normalization and weight decay

The primary goal of this paper is to study the (co-)evolution of the representations at each layer during training and the robustness of this representation with respect to the rest of the network. We try to minimize the factors that explicitly encourage changing of the network weights or representations in the analysis. In particular, unless otherwise specified, weight decay and batch normalization are *not* used. This leads to some performance drop in the trained models. Especially for deep residual networks: even though we could successfully train a residual network with 100+ layers without batch normalization, the final generalization performance could be quite worse than the state-of-the-art. Therefore, in this section, we include studies on networks trained *with* weight decay and batch normalization for comparison.

In particular, Table 3 shows the final test error rates of models trained with or without weight decay and batch normalization. Note the original VGG models do not use batch normalization [32], we list +bn variants here for comparison, by applying batch normalization to the output of each convolutional layer. On CIFAR10, the performance gap varies from 3% to 5%, but on ImageNet, a performance gap as large as 10% could be seen when trained without weight decay and batch normalization. Figure 11 shows how different training configurations affect the layerwise robustness analysis patterns on VGG16 networks. We found that when batch normalization is used, none of the layers



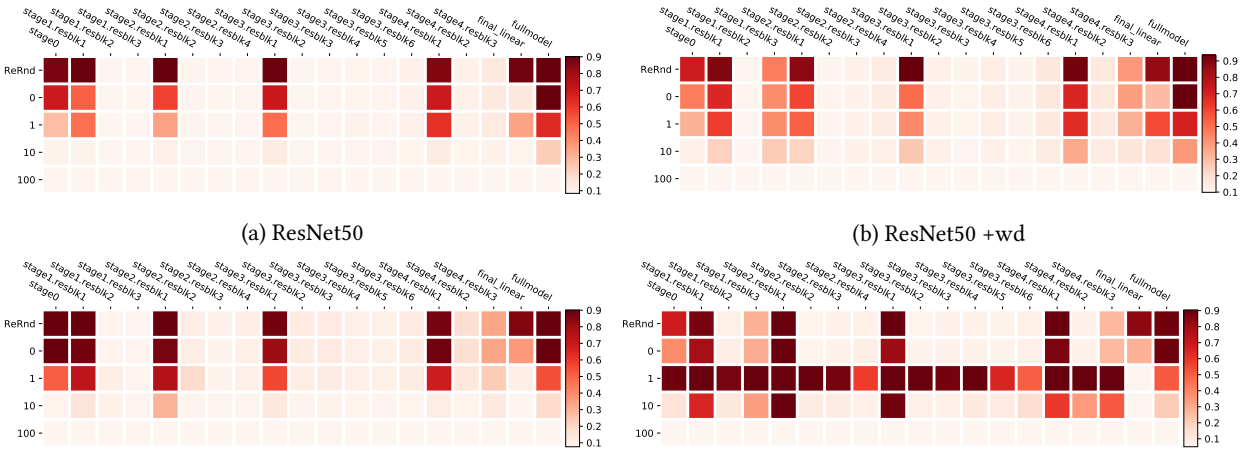
(a) VGG16

(b) VGG16 +wd

(c) VGG16 +bn

(d) VGG16 +wd +bn

Figure 11: **Layer-wise robustness analysis with VGG16 on CIFAR10.** The subfigures show how training with weight decay (+wd) and batch normalization (+bn) affects the layerwise robustness patterns.



(a) ResNet50

(b) ResNet50 +wd

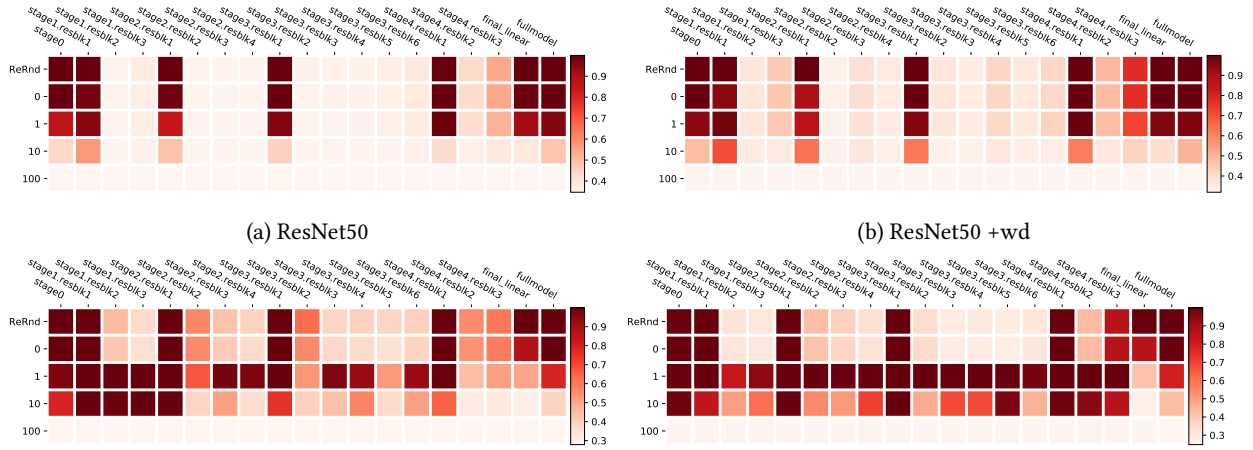
(c) ResNet50 +bn

(d) ResNet50 +wd +bn

Figure 12: **Layer-wise robustness analysis with ResNet50 on CIFAR10.** The subfigures show how training with weight decay (+wd) and batch normalization (+bn) affects the layerwise robustness patterns.

are robust any more.

Figure 12 and Figure 13 show similar comparisons for ResNet50 on CIFAR10 and ImageNet, respectively. Unlike VGGs, we found that the layerwise robustness patterns are still quite pronounced under various training conditions for ResNets. In Figure 12(d) and Figure 13(c,d), we see the mysterious phenomenon that re-initialing with checkpoint-1 is less robust than with checkpoint-0 for many layers. We do not know exactly why this is happening. It might be that during early stages, some aggressive learning is happening causing changes in the parameters or statistics with large magnitudes, but later on when most of the training samples are classified correctly, the network gradually re-balances the layers to a more robust state. Figure 15(d-f) in the next section shows supportive evidence that, in this case the distance of the parameters between checkpoint-0 and checkpoint-1 is larger than between checkpoint-0 and the final checkpoint. However, on ImageNet this correlation is no longer clear as seen in Figure 16(d-f). See the discussions in the next section for more details.



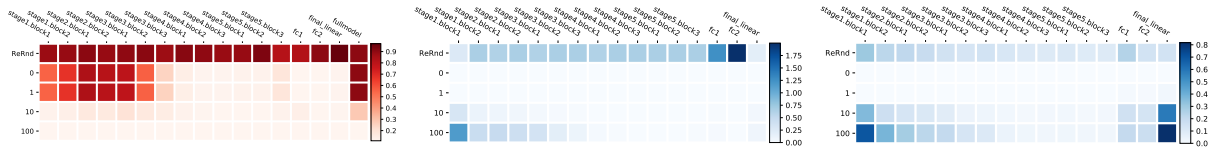
(a) ResNet50

(b) ResNet50 +wd

(c) ResNet50 +bn

(d) ResNet50 +wd +bn

Figure 13: **Layer-wise robustness analysis with ResNet50 on ImageNet.** The subfigures show how training with weight decay (+wd) and batch normalization (+bn) affects the layerwise robustness patterns.



(a) Test error

(b) $\|\theta_d^\tau - \theta_d^0\|$

(c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

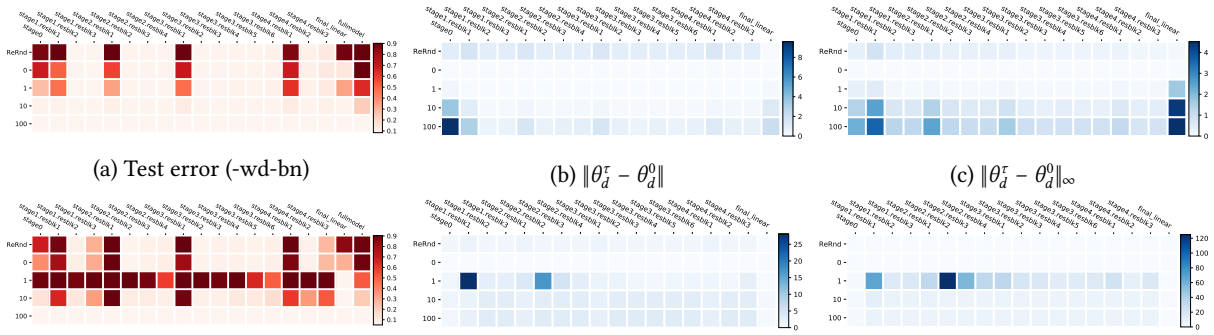
Figure 14: **Layer-wise robustness studies of VGG16 on CIFAR10.** (a) shows the robustness analysis measured by the test error rate. (b) shows the normalized ℓ_2 distance of the parameters at each layer to the version realized during the re-randomization and re-initialization analysis. (c) is the same as (b), except with the ℓ_∞ distance.

H Robustness and distances

In Figure 1 from Section 3, we compared the layerwise robustness pattern to the layerwise distances of the parameters to the values at initialization (checkpoint-0). We found that for FCNs on MNIST, there is no obvious correlation between the "amount of parameter updates received" at each layer and its robustness to re-initialization for the two distances (the normalized 2 and ∞ norms) we measured. In this appendix, we list results on other models and datasets studied in this paper for comparison.

Figure 14 shows the layerwise robustness plot along with the layerwise distance plots for VGG16 trained on CIFAR10. We found that the ℓ_∞ distance of the top layers are large, but the model is robust when we re-initialize those layers. However, the normalized ℓ_2 distance seem to be correlated with the layerwise robustness patterns: the lower layers that are less robust have larger distances to their initialized values.

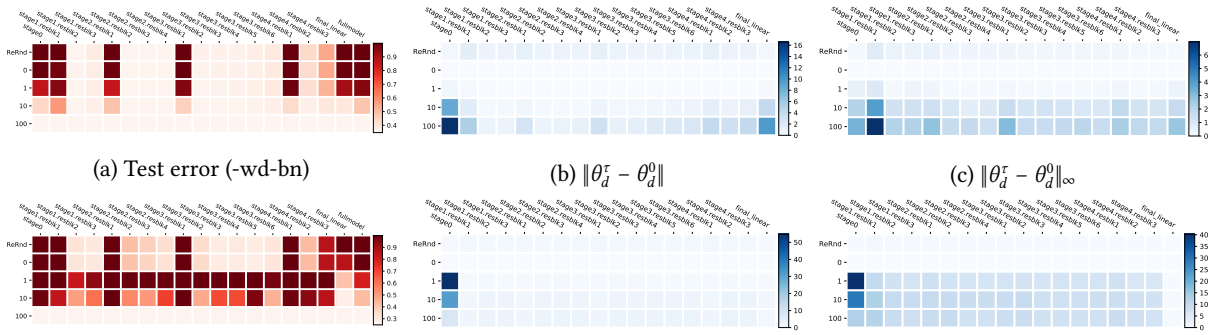
Similar plots for ResNet50 on CIFAR10 and ImageNet are shown in Figure 15 and Figure 16, respectively. In each of the figures, we also show extra results for models trained with weight decay and batch normalization. For the case without weight decay and batch normalization, we can see a weak correlation: the layers that are sensitive have slightly larger distances to their random initialization values. For the case with weight decay and batch normalization, the situation is less clear. First of all, in Figure 15(e-f), we see very large distances in a few layers at checkpoint-1. This provides a potential explanation to the mysterious pattern that re-initialization to checkpoint-1 is more sensitive than to checkpoint-0. Similar observations can be found in Figure 16(e-f) for ImageNet.



(a) Test error (-wd-bn) (b) $\|\theta_d^\tau - \theta_d^0\|$ (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

(d) Test error (+wd+bn) (e) $\|\theta_d^\tau - \theta_d^0\|$ (f) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 15: **Layer robustness for ResNet50 on CIFAR10.** Layouts are the same as in Figure 14. The first row (a-c) is for ResNet50 trained without weight decay and batch normalization. The second row (d-f) is with weight decay and batch normalization.



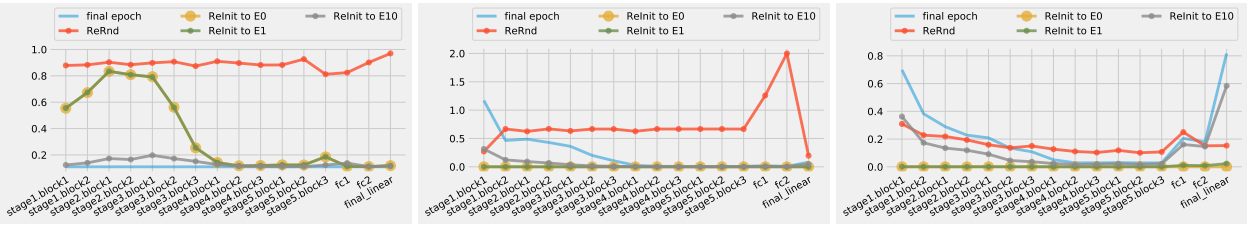
(a) Test error (-wd-bn) (b) $\|\theta_d^\tau - \theta_d^0\|$ (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

(d) Test error (+wd+bn) (e) $\|\theta_d^\tau - \theta_d^0\|$ (f) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 16: **Layer-wise robustness studies of ResNet50 on ImageNet.** Layouts are the same as in Figure 14. The first row (a-c) is for ResNet50 trained without weight decay and batch normalization. The second row (d-f) is with weight decay and batch normalization.

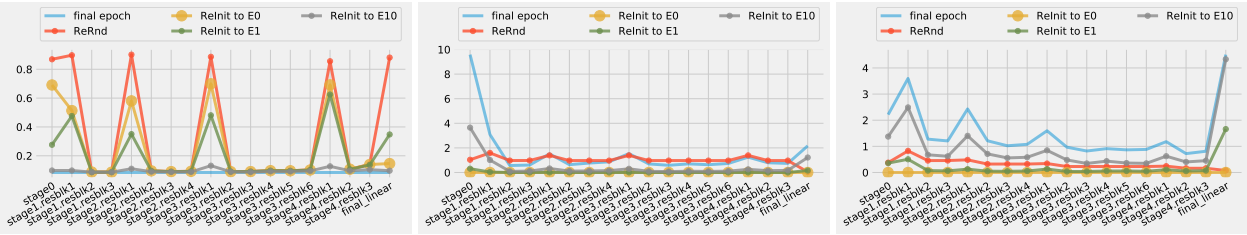
I Alternative visualizations

The empirical results on layer robustness are mainly visualized as heatmaps in the main text. The heatmaps allow uncluttered comparison of the results across layers and training epochs. However, it is not easy to tell the difference between numerical values that are close to each other from the color coding. In this section, we provide alternative visualizations that shows the same results with line plots. In particular, Figure 17 shows the layerwise robustness analysis for VGG16 on CIFAR10. Figure 18 and Figure 19 show the results for ResNet50 on CIFAR10 and ImageNet, respectively.



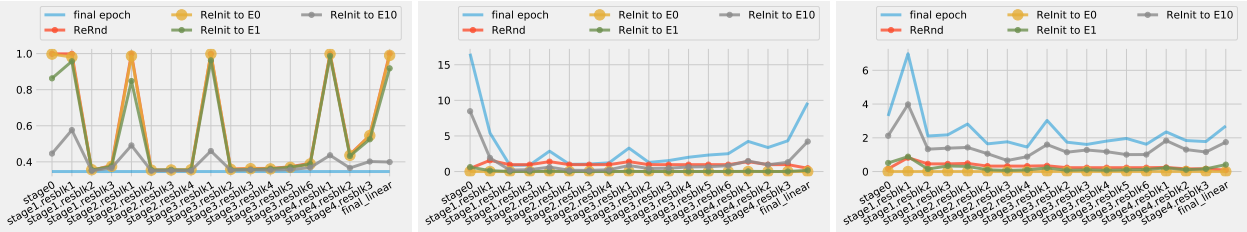
(a) Test error (b) $\|\theta_d^\tau - \theta_d^0\|$ (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 17: **Alternative visualization of layer robustness analysis for VGG16 models on CIFAR10**. This shows the same results as Figure 14, but shown as curves instead of heatmaps.



(a) Test error (b) $\|\theta_d^\tau - \theta_d^0\|$ (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 18: **Alternative visualization of layer robustness analysis for ResNet50 on CIFAR10**.



(a) Test error (b) $\|\theta_d^\tau - \theta_d^0\|$ (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 19: **Alternative visualization of layer robustness analysis for ResNet50 on ImageNet**.