A. Convex Setting

In this section we consider the case where the loss is convex, and we show the optimization problem remains convex when learning the subspace parameters.

Let $\boldsymbol{\omega} = (\omega_1, ..., \omega_m)$ denote the parameters used to construct the subspace. A simplified version of our objective is given by

$$h(\boldsymbol{\omega}) \triangleq \mathbb{E}_{\boldsymbol{\alpha} \sim \mathcal{U}(\Lambda)}[\ell(\mathsf{P}(\boldsymbol{\alpha}, \boldsymbol{\omega}))] \tag{6}$$

This objective is simplified from Equation 1 as we have removed the dependence on the training data and neural network—the loss ℓ is given parameters $\theta \in \mathbb{R}^n$ and returns a positive scalar.

We note that in each of the subspaces we learn—lines, curves, and simplexes—$\mathsf{P}(\boldsymbol{\alpha}, \boldsymbol{\omega})$ is linear with respect to $\boldsymbol{\omega}$.

Proposition A.1. *If $\ell : \mathbb{R}^n \to \mathbb{R}$ is convex and P is linear with respect to $\boldsymbol{\omega}$ then h is convex with respect to $\boldsymbol{\omega}$.*

Proof. For two sets of parameters $\boldsymbol{\omega}$ and $\overline{\boldsymbol{\omega}}$ and $\lambda \in [0, 1]$,

$$h((1-\lambda)\boldsymbol{\omega} + \lambda\overline{\boldsymbol{\omega}}) \tag{7}$$
$$= \mathbb{E}_{\boldsymbol{\alpha}}[\ell(\mathsf{P}(\boldsymbol{\alpha}, (1-\lambda)\boldsymbol{\omega} + \lambda\overline{\boldsymbol{\omega}}))] \tag{8}$$
$$= \mathbb{E}_{\boldsymbol{\alpha}}[\ell((1-\lambda)\mathsf{P}(\boldsymbol{\alpha}, \boldsymbol{\omega}) + \lambda\mathsf{P}(\boldsymbol{\alpha}, \overline{\boldsymbol{\omega}}))] \tag{9}$$
$$\le \mathbb{E}_{\boldsymbol{\alpha}}[(1-\lambda)\ell(\mathsf{P}(\boldsymbol{\alpha}, \boldsymbol{\omega})) + \lambda\ell(\mathsf{P}(\boldsymbol{\alpha}, \overline{\boldsymbol{\omega}}))] \tag{10}$$
$$= (1-\lambda)\mathbb{E}_{\boldsymbol{\alpha}}[\ell(\mathsf{P}(\boldsymbol{\alpha}, \boldsymbol{\omega}))] + \lambda\mathbb{E}_{\boldsymbol{\alpha}}[\ell(\mathsf{P}(\boldsymbol{\alpha}, \overline{\boldsymbol{\omega}}))] \tag{11}$$
$$= (1-\lambda)h(\boldsymbol{\omega}) + \lambda h(\overline{\boldsymbol{\omega}}), \tag{12}$$

where Equation 9 and Equation 10 respectively follow from the linearity of P (in $\boldsymbol{\omega}$) and convexity of ℓ. □

B. Additional Samples and Feature Similarity Regularization

In Algorithm 1 we approximate the inner expectation of Equation 1 using a single sample. In this section we approximate the expectation with multiple samples, leading to an improvement in accuracy along the subspace and of the ensemble. When approximating the expectation with s samples we split the batch of size b into s groups of size b/s and sample independent values of $\alpha \sim \mathcal{U}([0, 1])$ for each. Results for $s = \{1, 2, 4\}$ are shown in the first row of Figure 13.

Using multiple samples allows us to experiment with additional regularization to enable functional diversity. We can directly encourage models from different parts of the subspace to have orthogonal features. We experiment with regularization of this form, which we call feature similarity regularization, in the second row of Figure 13. For each batch we pick j, k randomly from $\{1, ..., s\}$, where s is the number of samples. Let α_j and α_k denote samples j and k from $\mathcal{U}([0, 1])$ and ϕ_j, ϕ_k denote the features obtained using models $\mathsf{P}(\alpha_j)$ and $\mathsf{P}(\alpha_k)$. The feature similarity regularization term is then given by

$$\lambda|\alpha_j - \alpha_k| \cos^2(\phi_j, \phi_k) \tag{13}$$

where the features ϕ are taken from the output of the penultimate layer and $\cos(\phi_j, \phi_k)$ is cosine similarity. The term $|\alpha_j - \alpha_k|$ allows for more feature similarity when models are close together on the subspace. Results for feature similarity regularization are shown in the bottom row of Figure 13.

C. Integrating over Subspaces

Is there a subspace from which you can efficiently ensemble *all* models? We believe this is not possible for the subspaces of general neural networks f we learn in this paper. However, this does become possible when considering a specific form for f.

Consider $\mathsf{P} : [0, 1] \to \mathbb{R}^n$ which defines a one-dimensional subspace of weights. In this section we investigate a mechanism for ensembling the output of all networks along the subspace—a closed-form expression for

$$\hat{y}(\mathbf{x}) = \int_0^1 f(\mathbf{x}, \mathsf{P}(\alpha)) d\alpha. \tag{14}$$

For a particular class of functions f, Equation 14 admits a straightforward solution. Consider

$$f(\mathbf{x}, \mathsf{P}(\alpha)) = g(\mathbf{x}, \mathsf{P}(0)) + \frac{dg(\mathbf{x}, \mathsf{P}(\alpha)))}{d\alpha}. \tag{15}$$

for which

$$\int_0^1 f(\mathbf{x}, \mathsf{P}(\alpha)) d\alpha = g(\mathbf{x}, \mathsf{P}(1)). \tag{16}$$

The function g can be any learned neural network. To train f (*i.e.* to learn g) we approximate the derivative by finite difference during training. For each training batch $(\mathbf{x}, \mathbf{y})$ we sample α uniformly from $[0, 1]$ and compute outputs

$$f(\mathbf{x}, \mathsf{P}(\alpha)) = g(\mathbf{x}, \mathsf{P}(0)) + \frac{g(\mathbf{x}, \mathsf{P}(\alpha + \epsilon)) - g(\mathbf{x}, \mathsf{P}(\alpha))}{\epsilon}. \tag{17}$$

During evaluation we then return $g(\mathbf{x}, \mathsf{P}(1))$ which corresponds to the ensemble of all networks $f(\mathbf{x}, \mathsf{P}(\alpha))$ (Equation 16). As shown in Figure 14, we experiment with this model on MNIST (LeCun, 1998) using $\epsilon = 0.1$. We use **Integral** to refer to the model described in this section. Recall that a label noise level of c denotes that a fraction c of the training data is assigned random and fixed labels before training. Since we are restricting the form of f, the accuracy does not differ significantly from standard training when there is no label noise. However, as label noise increases the integral solution outperforms other models.

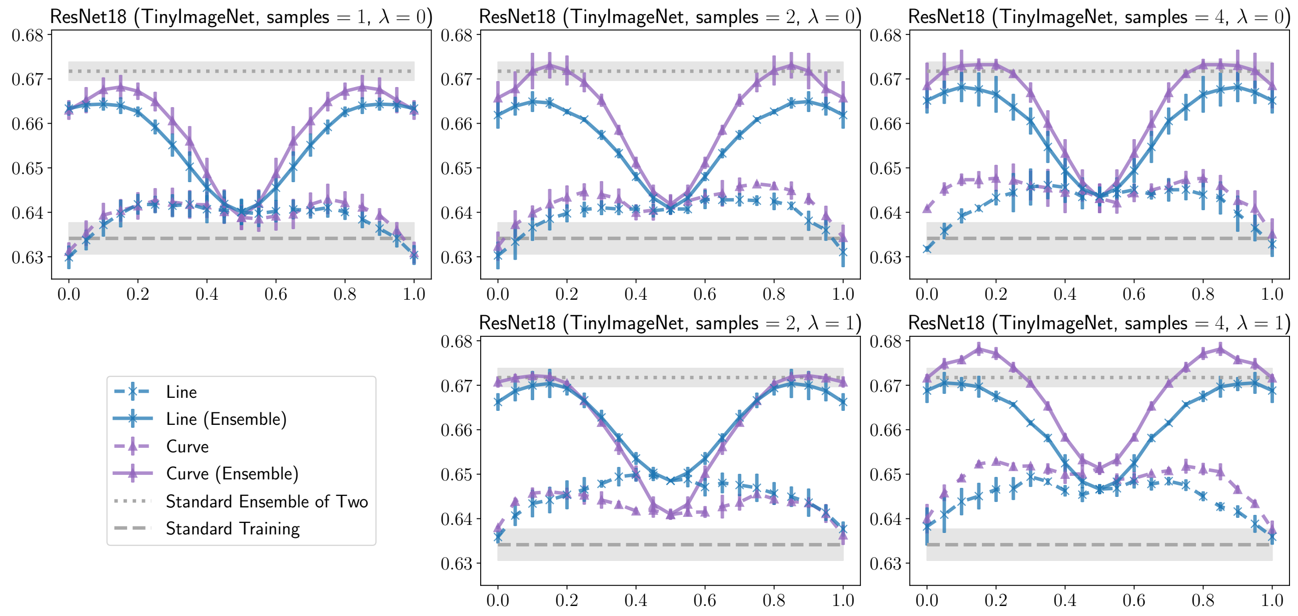

Figure 13. Model and ensemble accuracy along one-dimensional subspaces. For each subspace type, **(1)** accuracy of a model with weights $\mathsf{P}(\alpha)$ is shown with a dashed line and **(2)** accuracy when the output of models $\mathsf{P}(\alpha)$ and $\mathsf{P}(1-\alpha)$ are ensembled is shown with a solid line and denoted **(Ensemble)**. The number of samples of α used to approximate the inner expectation of Equation 1 is given by *samples* while λ denotes the strength of the feature similarity regularization (Appendix B). Both *samples* > 1 and $\lambda > 0$ tend to improve accuracy for both lines and curves.

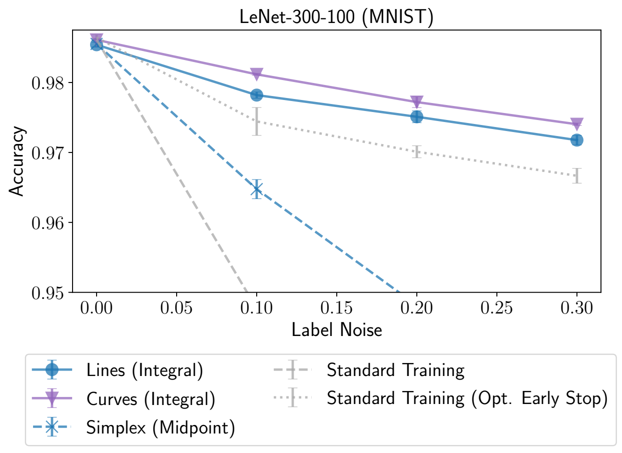

Figure 14. Learning subspaces of functions with efficient closed-form continuous ensembles (Equation 16). Since the functional form is restricted, these “Integral” solutions only provide an accuracy boost for nonzero label noise.

D. Additional Experimental Details

D.1. Models and Training Details.

For CIFAR10 experiments we use the ResNet20 model (referred to as cResNet20) which may be found at `https://github.com/facebookresearch/open_lth`. For TinyImageNet we use the ResNet$\{18, 50\}$ models which were used by Tanaka et al. (2020) in their TinyImageNet experiments. Finally, the ImageNet models are from PyTorch (Paszke et al., 2019). We use PyTorch 1.6 and Python 3.7. All models are trained on a single GPU except for the ImageNet models which are trained on 4 GPUs. Standard data augmentations are used—random crop and horizontal flip. To sample uniformly from the $m-1$ dimensional probability simplex we sample m random variables from the exponential distribution then normalize so that the sum is 1.

D.2. Computation

Consider a convolutional layer with kernel size $\kappa \times \kappa$, input size (b, c_1, w_1, h_1), and output size (b, c_2, w_2, h_2). The number of parameters is $p = c_1 c_2 \kappa^2$ while the number of FLOPs in standard training is $M = bpw_2h_2$. The algorithm we present requires $O(p(m-1))$ additional FLOPs to update the subspace's network weights, where m is the number of parameters used to construct the subspace. This overhead is minimal with respect to M (since b, w_2, h_2 tend

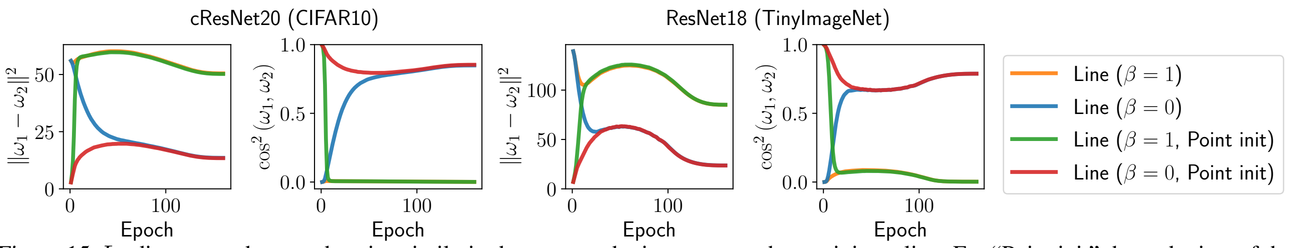

Figure 15. L_2 distance and squared cosine similarity between endpoints ω_1, ω_2 when training a line. For "Point init" the endpoints of the line were initialized with the same shared weight values.

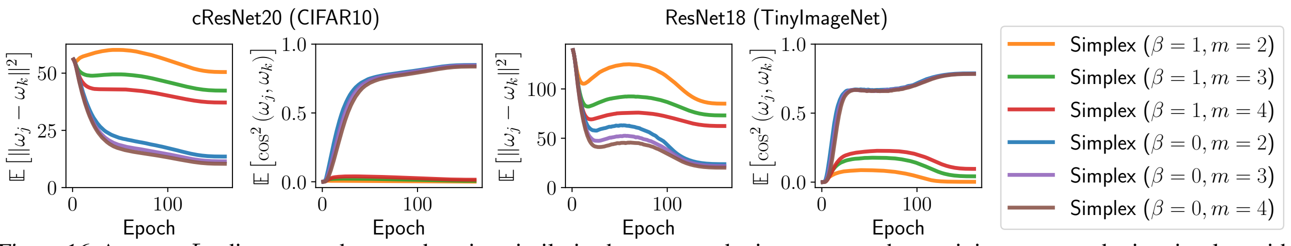

Figure 16. Average L_2 distance and squared cosine similarity between endpoints ω_j, ω_k when training an m endpoint simplex with regularization strength β (Equation 5).

to be large, b alone is over 100). The only storage overhead comes from storing multiple copies of the model parameters $O(p(m-1))$, which is not significant compared to buffers stored for the backward pass, of size $O(bc_2w_2h_2)$ (Chen et al., 2015). This is especially true for lines, curves, and low dimensional simplexes which constitute the majority of our experiments. No additional storage is required for computing the gradient, since the gradient updates to each endpoint are re-scaled versions of the same tensor (except the gradient of the regularization term, which has no dependence on the input data and can be computed after the initial buffers are freed).

D.3. Batch Normalization

In many cases batch norm (Ioffe & Szegedy, 2015) parameters require different treatment then other network weights. In standard training the batch norm scale parameter is initialized to be a vector of ones, and often remains mainly positive. Accordingly, cosine distance is likely the wrong distance metric to compare batch norm parameters. Moreover, the number of batch norm parameters is very small with respect to the total number of weights. Accordingly, in Figure 3 and Figure 15 we do not take batch norm parameters into account when considering cosine or L_2 distance. Moreover, in Algorithm 1 we do not take batch norm parameters into account when computing the regularization term (Equation 5).

Although we train batch norm parameters which lie on a line, curve, or simplex, batch norm layers also track a running mean and variance. Since these are not learned parameters, we follow Izmailov et al. (2018); Maddox et al. (2019) and recompute these statistics using training data. For instance, when evaluating the model at the midpoint of the simplex we first compute the running mean and variance with a pass through the training data before evaluating on the test set. For group norm (Wu & He, 2018) or layer norm (Ba et al., 2016) this would not be an issue, although these methods tend to achieve lower accuracy than batch norm in the settings we consider.

D.4. Baseline Hyperparameters

We implement all baselines with the same hyperparameters described in section 4 whenever possible. However, some baselines have additional hyperparameters. For SWA (Izmailov et al., 2018) we use the default values from `https://github.com/timgaripov/swa`—SWA LR of 0.05 and begin saving checkpoints 40 epochs before training ends (75% of the way through). For experiments with SWA throughout this wok we use either a cyclic (denoted *Cyclic LR*) or high constant (denoted *High Const. LR*) learning rate for the late phase of training and provide results for the best or both. For SWA-Guassian we construct the Gaussian using 6 saved SWA checkpoints.

Additionally, we tried using the regularization term (Equation 5) to encourage diversity among the SWA checkpoints but did not succeed in improving performance.

E. Further Subspace Dynamics

This section extends the results from subsection 4.1 which examine the shape of subspace throughout training. Figure 15 illustrates that initializing the endpoints of the line

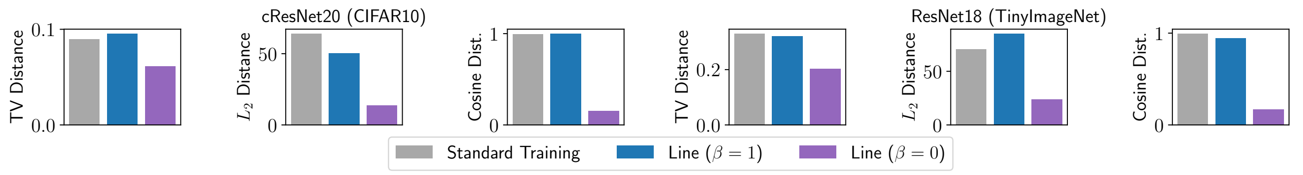

Figure 17. Comparing the statistics of the models which lie at the endpoints of a learned line with two independently trained models. We compare total variation (TV) distance between the outputs and $\{L_2, \text{Cosine}\}$ distance between the weights.

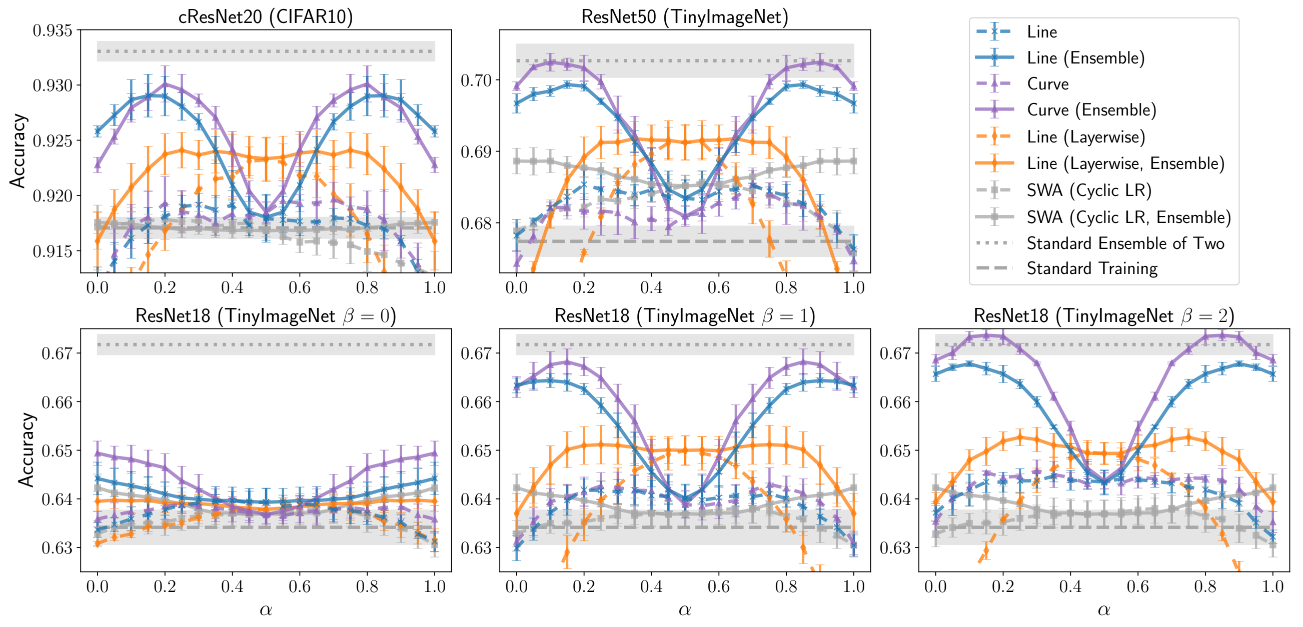

Figure 18. Model and ensemble accuracy along one-dimensional subspaces. For each subspace type, **(1)** accuracy of a model with weights $\mathsf{P}(\alpha)$ is shown with a dashed line and **(2)** accuracy when the output of models $\mathsf{P}(\alpha)$ and $\mathsf{P}(1-\alpha)$ are ensembled is shown with a solid line and denoted **(Ensemble)**. Note that quantity (2) is symmetric about 0.5 at which point it also intersects with quantity (1). "Standard Ensemble of Two" is the ensemble accuracy of two independently trained networks. For SWA we save only two checkpoints and consider the subspace formed by interpolating between them.

with the same shared initialization ("point init") has little effect on the dynamics. After a few epochs of training, any discrepency between "point init" and standard initialization nearly disappears. In Figure 16 we examine the average L_2 distance and squared cosine similarity between endpoints when training simplexes. The same general trends hold, but the average distance between endpoints decreases with the number of endpoints m. Since a new random pair of endpoints is sampled for each batch in Algorithm 1, closeness between each individual pair is penalized less for larger m. Finally, in Figure 17 we compare the endpoints of a line with two independently trained models in terms of L_2 distance, cosine distance, and total variation (TV) distance. For two networks with outputs $\mathbf{p}_1$ and $\mathbf{p}_2$ the TV distance is given by $\frac{1}{2}\|\mathbf{p}_1 - \mathbf{p}_2\|_1$ and is averaged over all examples in the test set. As expected, $\beta = 1$ produces lines with more distant and functionally diverse endpoints.

F. Additional Baselines for One-Dimensional Subspaces

In Figure 18 we augment the experiments from subsection 4.2 the additional baseline of SWA (described in section 2) with a cyclic learning rate scheduler. For experiments with SWA (Izmailov et al., 2018) throughout this wok we use either a cyclic (denoted *Cyclic LR*) or high constant (denoted *High Const. LR*) learning rate for the late phase of training and provide results for the best or both. In the case of Figure 18, where we save only two SWA checkpoints and interpolate between, cyclic performs better as the high constant scheduler does not find checkpoints which match standard training accuracy. Additional details on baseline hyperparameters are provided in Appendix D.

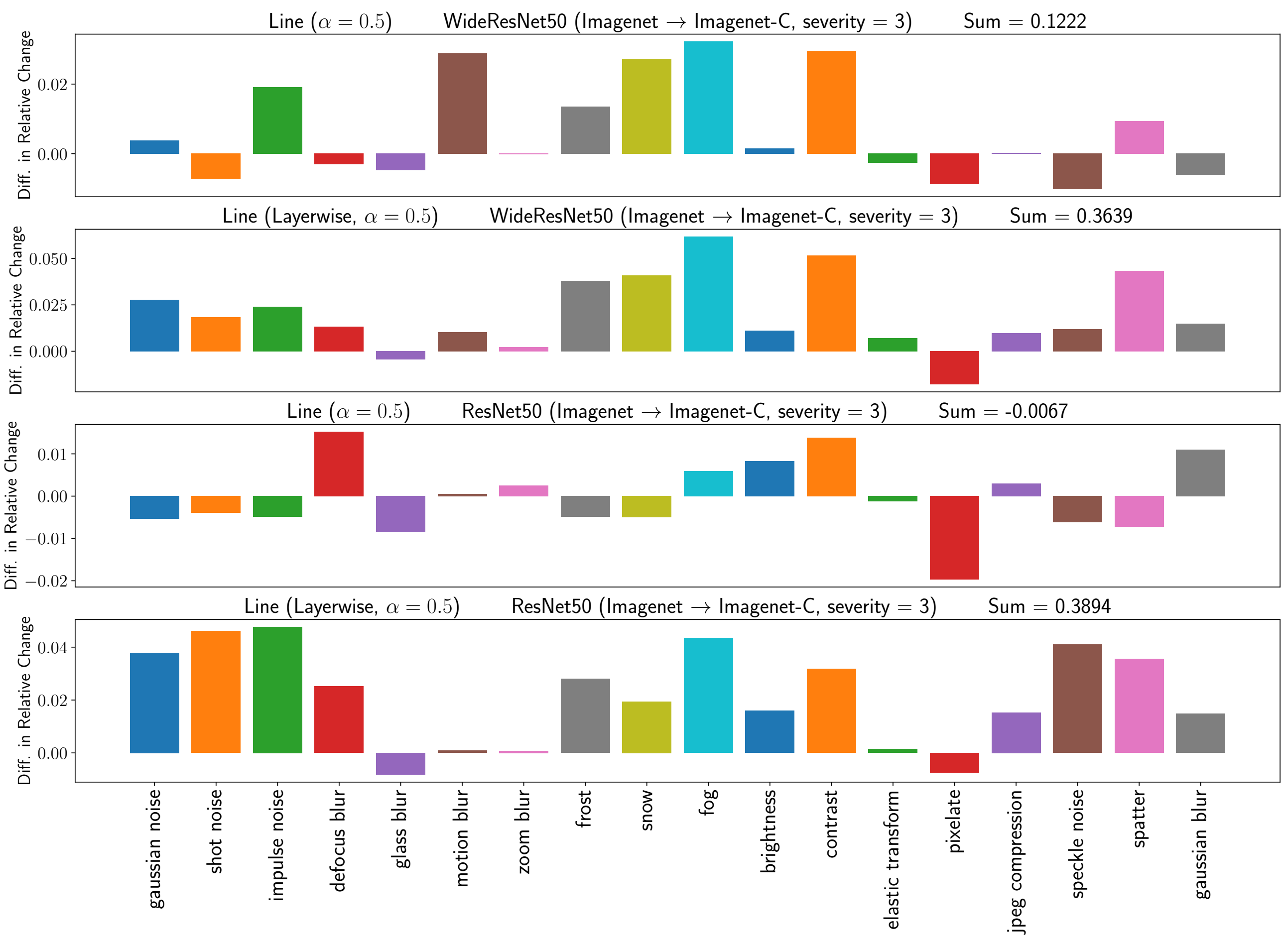

Figure 19. Comparing the relative change in accuracy when tested on corrupted data. Comparison is between the midpoint of a line and standard training. A positive bar indicates that the midpoint of the line has relatively less of a drop in accuracy from clean to corrupted data in ImageNet-C. (Hendrycks & Dietterich, 2019). See text (Appendix G) for details.

G. Additional ImageNet-C Robustness Experiments

In this section we test the models trained on ImageNet (Figure 9, subsection 4.4) across all image corruptions in the ImageNet-C dataset (Hendrycks & Dietterich, 2019). We consider the relative change in accuracy when models are evaluated on corrupted images. For a model with accuracy a on the clean set and b on the corrupted images, the relative change in accuracy is $(b - a)/a$. The relative change in accuracy (which we refer to as *relative change*) is chosen because performance on the clean test set can act as a confounder (Taori et al., 2020). The experiments are conducted with a corruption severity of 3.

Figure 19 illustrates the difference in *relative change* between the midpoint of the line and a model found through standard training. A positive value indicates that the midpoint of the line has a relatively less severe drop in accuracy when faced with corrupted data. Although performance on different corruption types is varied, the midpoint models we find tend to exhibit more robustness—WideResNet50 (layerwise) outperforms standard training on all but two corruption types.

However, this evaluation considers only the midpoint of the line, ignoring that we have trained family of models. In Figure 20, we compare the best-performing model on the line (over $\alpha \in \{0, 0.1, ..., 0.9, 1.0\}$, in terms of the relative change in accuracy) with standard training. This setting is not realistic as α is tuned on the test set, however it is a positive sign for the case when a validation set exists for the corrupted data of interest. A single training run can capture a family of models, and each can be tested on a validation set for the downstream domain.

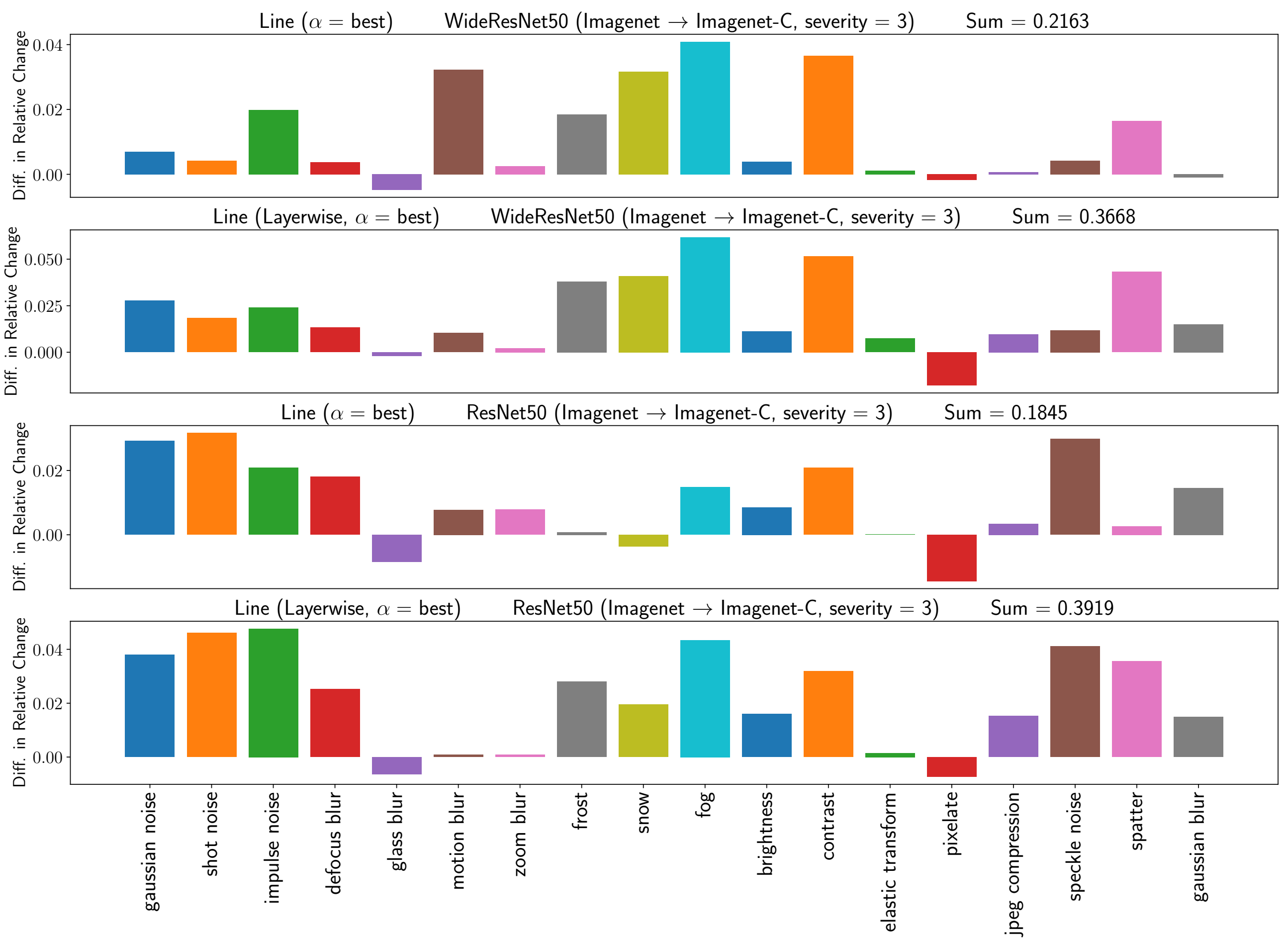

Figure 20. Comparing the relative change in accuracy when tested on corrupted data. Comparison is between the best model on the line and standard training. A positive bar indicates that there exists a model on the line with relatively less of a drop in accuracy from clean to corrupted data in ImageNet-C. This result demonstrates that there exists a model on the line which performs well, but does not indicate how to find this model. See text (Appendix G) for details.

H. Further Analysis of Frankle et al. (2020)

Recall from section 2, Observation 4 that Frankle et al. (2020) consider the scenario where two networks branch off after k epochs of the trajectory are shared. In other words, they consider $\theta_k = \text{Train}^{0\to k}(\theta_0, \xi)$ and let $\theta^i_{k\to T} = \text{Train}^{k\to T}(\theta_k, \xi_i)$ for $i \in \{1,2\}$. In Figure 21 (left) we reproduce results from Frankle et al. (2020), demonstrating that for very small k, the weight average of $\theta^1_{k\to T}$ and $\theta^2_{k\to T}$ matches the accuracy standard training accuracy. Note that the weight average refers to the accuracy of model $f\left(\cdot, \frac{1}{2}\left(\theta^1_{k\to T} + \theta^2_{k\to T}\right)\right)$ and the ensemble refers to the accuracy of model $\frac{1}{2}\left(f\left(\cdot, \theta^1_{k\to T}\right) + f\left(\cdot, \theta^2_{k\to T}\right)\right)$. For moderate k, the weight average exceeds standard training as a result of Observation 5 (section 2).

Figure 21 (right) demonstrates that these results hold when considering weight and output space ensembles of 5 models which all share k epochs of trajectory. Finally, in Figure 21 (middle) we consider random interpolations at different scales. Random Mixture (Global) is given by

$$\mathbb{E}_{\alpha\sim\mathcal{U}([0,1])}\left[\text{Acc}\left((1-\alpha)\theta^1_{k\to T} + \alpha\theta^2_{k\to T}\cdot\right)\right] \qquad (18)$$

For Random Mixture (Layerwise) we sample different coefficients α for each layer, and for Random Mixture (Per-weight) we sample different α for all weights in the network. The latter corresponds to a hyper-rectangle with corners $\theta^1_{k\to T}$ and $\theta^2_{k\to T}$. When k is at least half of thhe training epochs all models on this n dimensional hyper-rectangle match standard training accuracy.

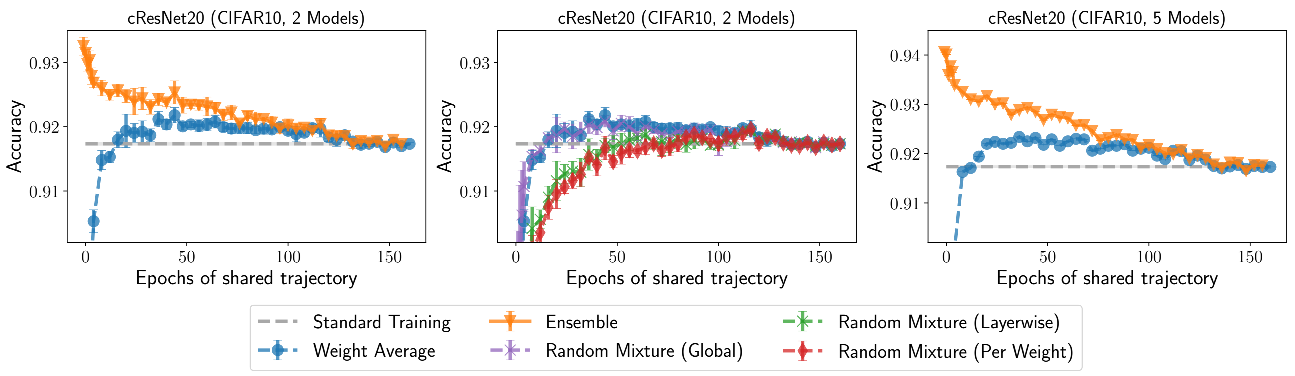

Figure 21. (left) Reproducing the instability analysis of (Frankle et al., 2020)—providing the accuracy of the weight space ensemble and output space ensemble of two models with k epochs of shared trajectory. (middle) Considering random interpolations between models with k epochs of shared trajectory. Interpolations are global, per-layer, and per-weight. (right) Extending the instability analysis result to 5 models.