A Appendix

Section B summarizes our experimental setup, while Section C, D, and E respectively detail the tangent Hessian computation method, the geodesic path generation algorithm, and the weak data augmentation strategy used for geodesic Monte Carlo integration. Finally, in section F we extend our discussion of related works. Additional experiments are reported in section G.

B Network Architectures and Training Setup

Network Architectures The ConvNets and ResNets used follow the experimental settings of Nakkiran et al. (2019b), with the only difference that we disable batch normalization in order to focus our study on implicit regularization. In summary, the ConvNets are composed of 4 convolutional stages (each with a single conv + ReLU block) with kernel size 3×3, stride 1, padding 1, each followed by maxpooling of stride 2 and kernel size 2×2. Finally, a max pooling layer of stride 2 and kernel size 2×2 is applied, followed by a linear layer. The Residual networks used in this study are ResNet18s (He et al., 2015) without batch normalization.

Both ConvNets and ResNets are formed by 4 convolutional stages at which the number of learned feature maps doubles, i.e. the base width of each stage follows the progression $[w, 2w, 4w, 8w]$, with $w = 64$ denoting a standard ResNet18. To control the number of parameters in each network, the base width w varies from 1 to 64.

Throughout our experiments, augmentations $\tilde{\mathbf{x}}_n$ of a sample $(\mathbf{x}_n, y_n)$ are labelled with their respective (potentially noisy) training target y_n.

Dataset Splits To tune the training hyperparameters of all networks, a validation split of 1000 samples was drawn uniformly at random from the training split of CIFAR-10 and CIFAR-100.

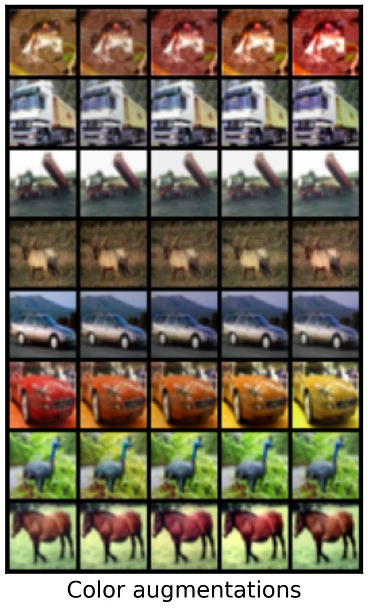
ConvNet Training Setup The training settings are the same for CIFAR-10 and CIFAR-100. All ConvNets are trained for 4k epochs with SGD with momentum 0.9, fixed learning rate $1\text{e}-3$, batch size 128, and no weight decay. All learned layers are initialized with Pytorch's default weight initialization (version 1.11.0). To stabilize prolonged training in the absence of batch normalization, we use learning rate warmup: starting from a base value of $1\text{e}-4$ the learning rate is linearly increased to $1\text{e}-3$ during the first 5 epochs of training, after which it remains constant at $1\text{e}-3$.

ResNet Training Setup All ResNets are trained for 4k epochs using Adam with base learning rate $1\text{e}-4$, batch size 128, and no weight decay. All learned layers are initialized with Pytorch's default initialization (version 1.11.0). All residual networks are trained with data augmentation, consisting of $4-pixel$ random shifts, and random horizontal flips.

Computational Resources Our experiments are conducted on a local cluster equipped with NVIDIA Tesla $A100$s with 40GB onboard memory. For each dataset and architecture, we train 64 different networks for 4000 epochs with 3 different seeds. The total time for computing our experiments, excluding training networks and hyperparameter finetuning, amounts to approximately 6 GPU years. Furthermore, computing our statistics requires evaluating per-sample Jacobians for each training point and corresponding augmentations, for increasing volumes around each point. For each training setting, this was performed for 72 model checkpoints collected during training, to produce the heatmaps in Figures 5, 11, 12, 13 and 14.

C Tangent Hessian Computation

To estimate the tangent Hessian norm at a point $\mathbf{x}_n$ through Equation 5, we approximate the tangent space to the data manifold local to $\mathbf{x}_n$ by using a set of random weak augmentations of $\mathbf{x}_n$. To guarantee that all augmentations $\mathbf{x}_n + \mathbf{u}_m$, as well as the displacements $\mathbf{x}_n + \delta\mathbf{u}_m$ lie on the data manifold, we use weak colour augmentations as follows.



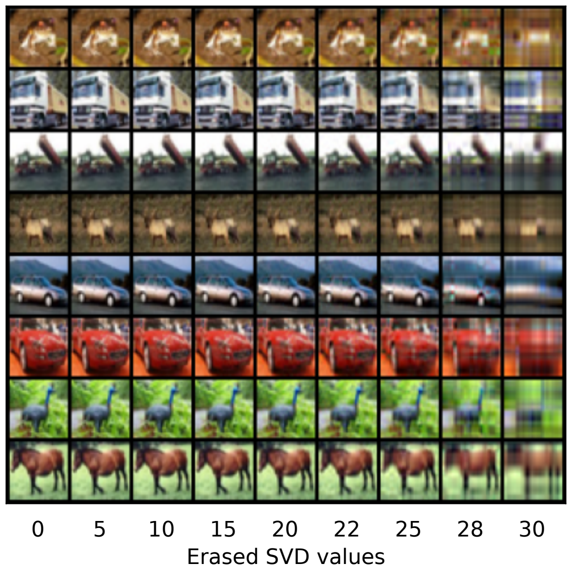


Figure 8: (Left) Visualization of random colour augmentations used to estimate the tangent Hessian norm. Each row represents a set of random augmentation, with the first image per-row showing the corresponding base sample. (Right) Each row represents SVD augmentations of increasing strength. Also in this case, the first column represents the base sample used to generate the corresponding augmentations in each row.

For each sample $\mathbf{x}_n$, we apply in random order the following photometric transformations:

- random brightness transformation in the range $[0.9, 1.1]$, with 1. denoting the identity transformation.
- random contrast transformation in $[0.9, 1.1]$, with 1. denoting the identity transformation.
- random saturation transformation in $[0.9, 1.1]$, with 1. denoting the identity transformation.
- random hue transformation in $[-0.05, 0.05]$, with 0. denoting the identity transformation.

Furthermore, a step size $\delta = 0.1$ is used for computing the finite differences in Equation 5. 4 augmentations $\mathbf{x}_n + \mathbf{u}_m$ are sampled for each point. All randomness is controlled to ensure reproducibility. Figure 8 (left) shows a visualization of the colour augmentations used.

D Geodesic Paths Generation

In this section, we provide pseudocode for the algorithm used for generating geodesic paths, used for Monte Carlo integration. Let $\mathbf{x}_0 \in \mathbb{R}^d$ denote a training point, which we use as the starting point of geodesic paths $\boldsymbol{\pi}_p$ emanating from $\mathbf{x}_0$. Let $\mathcal{T}_\mathbf{s} : \mathbb{R}^d \to \mathbb{R}^d$ denote a family of smooth transformations (data augmentation), dependent on a parameter $\mathbf{s}$ controlling the magnitude and direction of the transformation (e.g. radial direction and displacement for pixel shifts). Let $\mathcal{S} := \{\mathbf{s}^1, \dots, \mathbf{s}^K\}$ denote a sequence of parameters for the family $\mathcal{T}_\mathbf{s}$, each with strength $S^k = \|\mathbf{s}^k\|_2$ for $k = 1, \dots, K$, such that $S^1 < \dots < S^K$. Then, Algorithm 1 returns a geodesic path $\boldsymbol{\pi} : [0,1] \to \mathbb{R}^d$, based at $\mathbf{x}_0$, i.e. $\boldsymbol{\pi}(0) = \mathbf{x}_0$, which is anchored to the data manifold local to $\mathbf{x}_0$ by a sequence of augmentations of increasing strength, for $k = 1, \dots, K$.

Particularly, Algorithm 1 can be applied P times to generate paths $\boldsymbol{\pi}_p$ emanating from $\mathbf{x}_0$. Finally, by integrating metrics of interest (e.g. Jacobian and tangent Hessian norms) along each path $\boldsymbol{\pi}_p$, we obtain

Algorithm 1 Generate a geodesic path $\boldsymbol{\pi}$ emanating from a training point $\mathbf{x}_0$.

```
1: function GEODESIC PATH($\mathbf{x}_0$, $\mathcal{T}_\mathbf{s}$, $\mathcal{S} := \{\mathbf{s}^1, \dots, \mathbf{s}^K\}$)
2:     $\mathcal{P} \leftarrow \{\mathbf{x}_0\}$                              ▷ Set of on-manifold points.
3:     for $\mathbf{s}^k \in \mathcal{S}$ do
4:         sample $\mathbf{s} \sim \mathbf{s}^k$          ▷ Sample augmentation of strength $S^k = \|\mathbf{s}\|_2$.
5:         $\mathbf{x}^k = \mathcal{T}_\mathbf{s}(\mathbf{x}_0)$                ▷ Generate weak data augmentation.
6:         $\mathcal{P} \leftarrow \mathcal{P} \cup \{\mathbf{x}^k\}$
7:     end for
8:     return $\mathcal{P}$                 ▷ Set of data augmentations forming a path
                                             $\boldsymbol{\pi}$, with points sorted by distance from $\mathbf{x}_0$.
9: end function
```

estimates of sharpness of the loss along each path, which we use as Monte Carlo samples in Equation 7 for estimating volume-sharpness. We recall that the size of the volume considered is controlled by the maximum augmentation strength S^K used for generating weak augmentations, which is proportional to the distance travelled away from $\mathbf{x}_0$ in input space.

E SVD Augmentation

The SVD augmentation method presented in section 3.4 allows for generating images that lie in close proximity to the base sample $\mathbf{x}_n$. Figure 8 shows an illustration of the original image (first column) and several augmented images, as the augmentation strength (number of erased singular values) increases. Figure 9 shows the average (over the CIFAR-10 training set) Euclidean distance of augmented samples from their respective base sample, as well as the length of the polygonal path formed by connecting augmentations of increasing strength. We note that for $k < 30$, in expectation, augmentations lie in close proximity to the original base sample in Euclidean space.

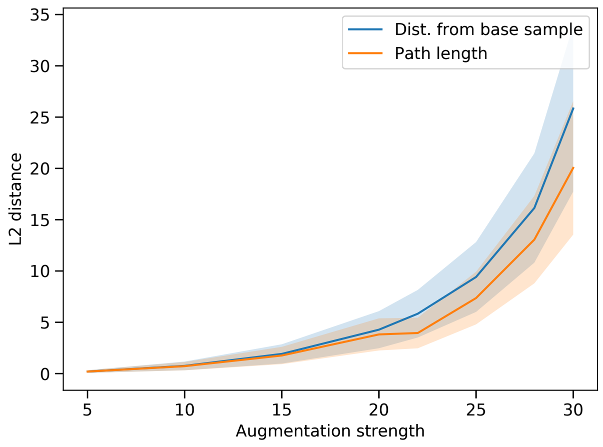


Figure 9: Average L2 distance from the base samples, for augmentations of increasing strength.

F Extended Related Works

In this section, we extend the related work discussion of section 2 to contextualize our findings in relationship to linear models.

In linear regression, the model-wise double descent phenomenon has been studied in terms of *harmless interpolation* (Muthukumar et al., 2020) or *benign overfitting* of noisy data (Bartlett et al., 2020), by controlling the number of input features $\boldsymbol{\theta}$ considered. Particularly, for the least squares solution to a noisy linear regression problem with random input and features, the impact of noise on generalization is mitigated by the abundance of weak features (Belkin et al., 2020). In this context, interpolation is studied for data whose population is described by noisy observations from a linear ground truth model. In the following, we delineate the main differences between linear regression and the experimental setting considered in our study.

We begin by noting that, since the model function of linear models has zero curvature (both w.r.t. model input $\mathbf{x}$ and parameters $\boldsymbol{\theta}$), the only source of nonlinearity and curvature in linear regression is the error function (MSE). To see this, let $f(\mathbf{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \mathbf{x}$ denote a linear regression model, estimated by minimizing the mean squared error $\mathcal{L}(\boldsymbol{\theta}, \mathbf{x}, y) = \frac{1}{2N} \sum_{n=1}^{N} (f(\mathbf{x}_n, \boldsymbol{\theta}) - y_n)^2$, where y_n is a noisy target $\forall n$. Then, the error function $\mathcal{L}$ has constant curvature $H = \| \frac{\partial^2 \mathcal{L}}{\partial \mathbf{x} \partial \mathbf{x}^T} \|_2 = \| \boldsymbol{\theta} \boldsymbol{\theta}^T \|_2$, independent of $\mathbf{x}$.

In contrast, we study the case of nonlinear classification problems and nonlinear models, which have notable differences from the linear case. First, there is no a priori closed form solution of the learning problem, thus providing relevance to empirical studies. Second, curvature of the model function is non-constant, and the function may oscillate arbitrarily outside of the training data (this is known as the Runge phenomenon). Third, studies that rely exclusively on the test error suggest that interpolation is harmless also in overparameterized nonlinear models. Finally, the model function of convolutional architectures is independent of input-data dimensionality, and the relationship between complexity of the model function and its underlying parameterization is therefore implicit.

In this setting, we experimentally show that, in the interpolating regime, (1) curvature at training points depends non-monotonically on model size; (2) oscillations occur especially for small interpolating models, which are worst affected by noise; (3) large models achieve low-curvature interpolation of both clean and noisy samples (in contrast with the polynomial intuition), and such property is observed over large volumes (non-zero measure) around each training point (in contrast with the Runge phenomenon, thus providing evidence of implicit regularization); (4) Interpolation of noise impacts generalization even for large models (contrary to the overparameterized linear regression case); (5) Double descent observed for input space curvature occurs even when fitting 100% noisy data, more clearly pinpointing properties that are consistently promoted by overparameterization in deep nonlinear networks.

Our methodology enables the study of sharpness of fit of training data for nonlinear models, providing a comparative study of the regularity with which different parameterizations achieve interpolation and (in some cases) generalization.

G Additional Experiments

G.1 Transformers

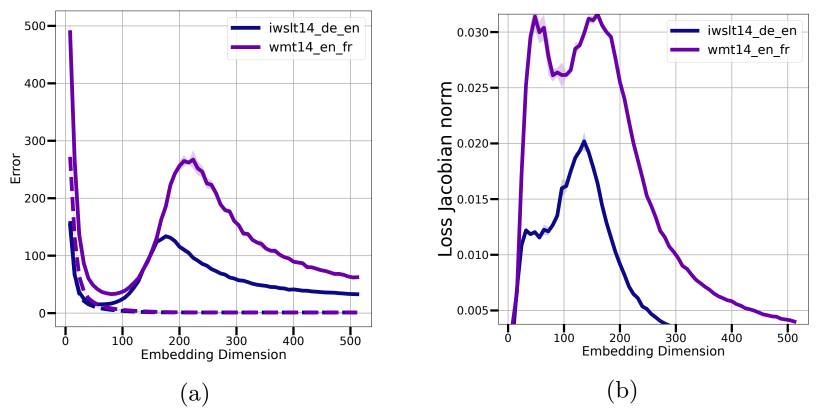


Figure 10: a) **Double descent** of the test error for transformers trained on translation tasks, as the embedding dimension and model width vary. b) **Average Jacobian norm**.

We consider multi-head attention-based Transformers (Vaswani et al., 2017) for neural machine translation tasks. We vary model size by controlling the embedding dimension d_e, as well as the width h of all fully connected layers, which we set to $h = 4d_e$ following the architecture described in Vaswani et al. (2017). We train the transformer networks on the WMT'14 En-Fr task (Macháček & Bojar, 2014), as well as ISWLT'14 De-En (Cettolo et al., 2012). The training set of WMT'14 is reduced by randomly sampling 200k sentences, fixed for all models. The networks are trained for 80k gradient steps, to optimize per-token perplexity, with 10% label smoothing, and no dropout, gradient clipping or weight decay.

For both datasets, Figure 10a shows the double descent curve for the test error for both datasets considered. Figure 10b extends our main result beyond vision models, showing that loss sharpness at each training point, as measured by the Jacobian norm, follows double descent for the test error.

G.2 ConvNets

Figures 11 and 13 summarize our main findings with heatmaps showing modelwise and epochwise trends for the test error, train loss, as well as our sharpness metrics, individually computed over the clean and noisy subsets of CIFAR-10.

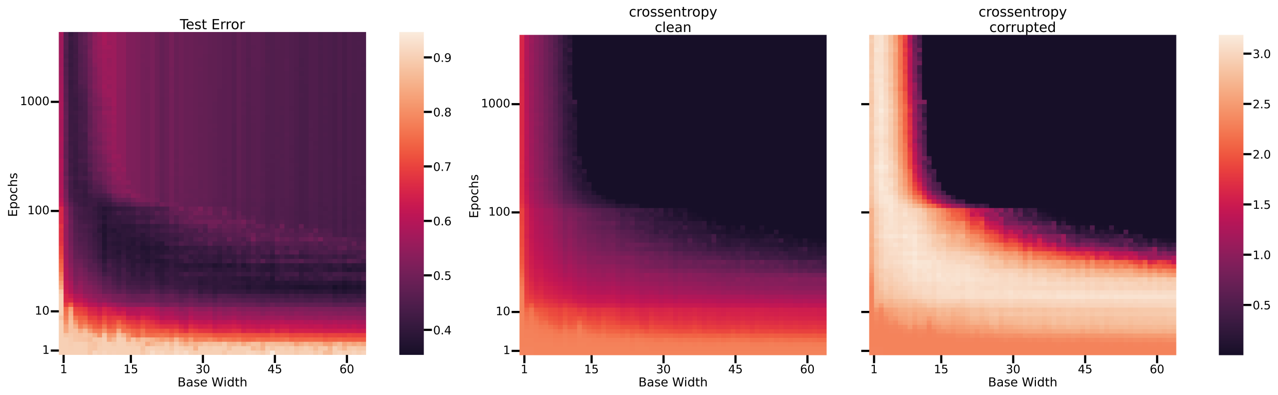


Figure 11: Test error (left), crossentropy loss over cleanly-labelled training samples (middle) and corrupted training samples (right) over epochs (y-axis) for different model sizes (x-axis), for ConvNets on CIFAR-10.

G.3 ResNets

Figures 12 and 14 present heatmaps showing modelwise and epochwise trends for the test error, train loss, as well as our sharpness metrics, individually computed over the clean and noisy subsets of CIFAR-10.

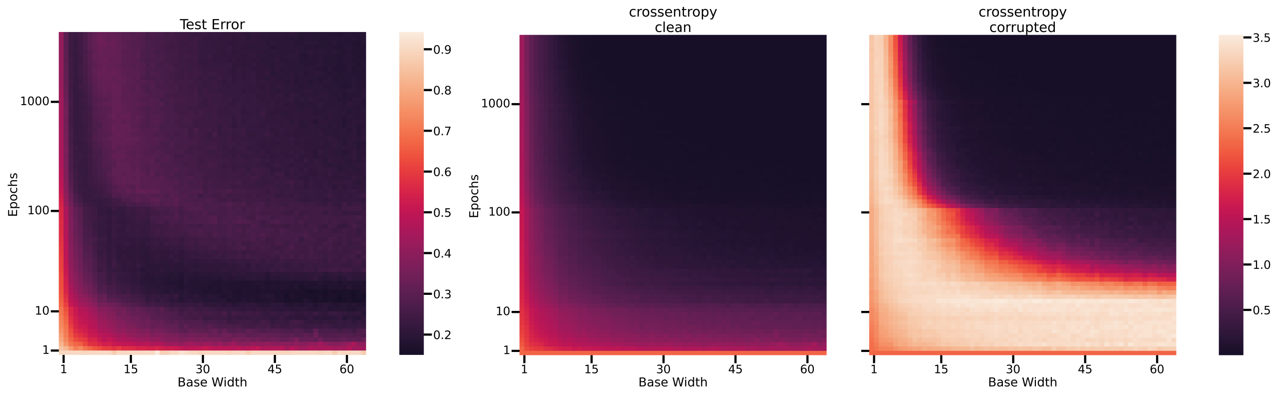


Figure 12: Test error (left), crossentropy loss over cleanly-labelled training samples (middle) and corrupted training samples (right) over epochs (y-axis) for different model sizes (x-axis), for ResNets on CIFAR-10.

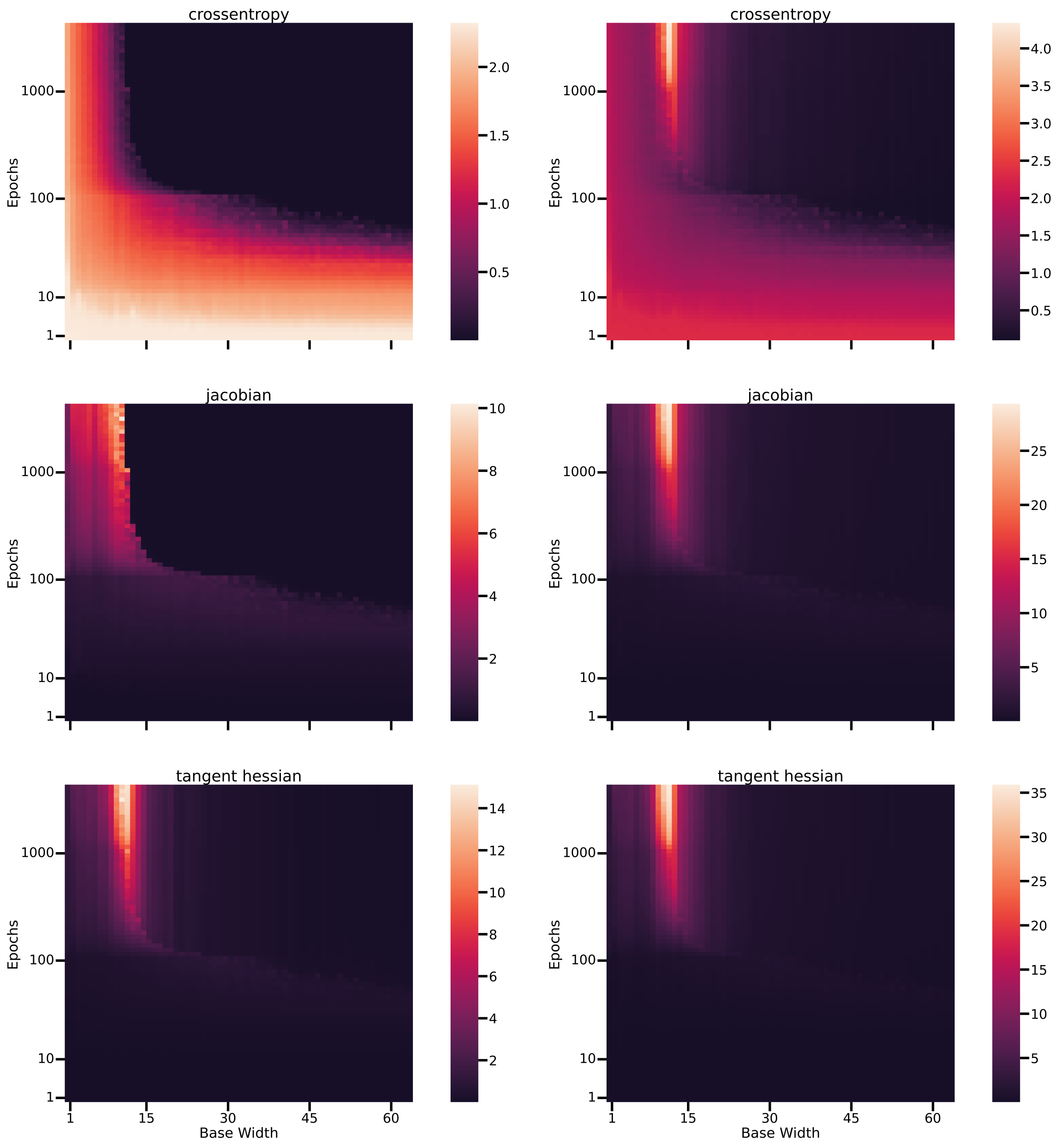


Figure 13: (Left column) Metrics evaluated on the training set without Monte Carlo integration on ConvNets. (Right column) Monte Carlo integration over a neighborhood with paths consisting of 7 augmentations.

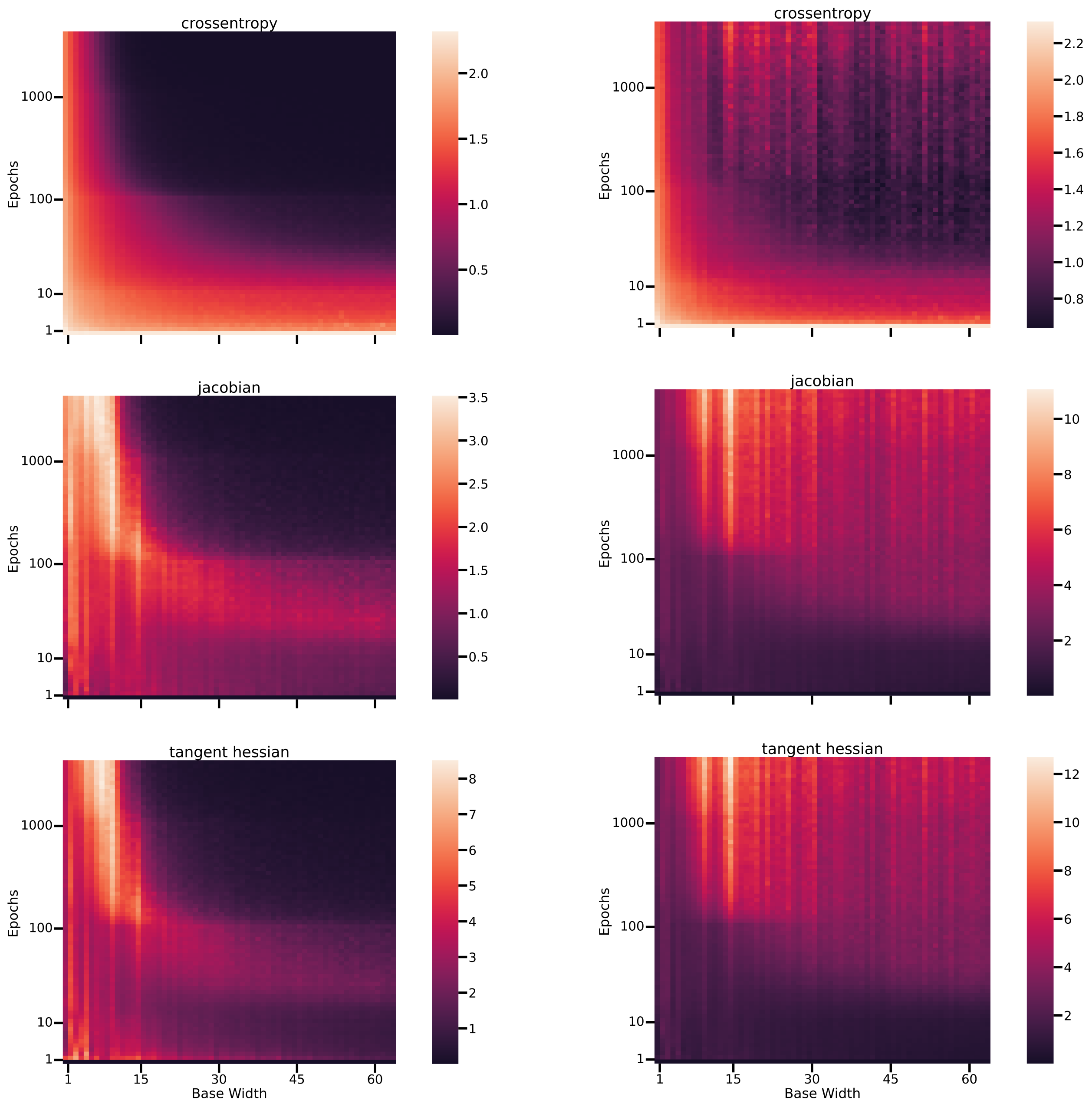


Figure 14: (Left column) Metrics evaluated on the training set without Monte Carlo integration on ResNets. (Right column) Monte Carlo integration over a neighborhood with paths consisting of 7 augmentations.