A Appendix

A.1 Loss Landscapes

To compute a loss landscape, we need to project the high-dimensional loss function into a two-dimensional space. To construct this space, we therefore need to sample two orthonormal vectors. We consider two different ways to sample these vectors (e.g., random directions, Hessian-based directions), but the rest of the procedure for sampling the loss function in the resulting spaces is the same regardless of how the directions were chosen.

The first (naive) way to define the lower-dimensional subspace is to sample two random vectors. In high dimensions, they will most likely be nearly orthogonal. Here, we use PyTorch to randomly sample two vectors having the same size as the number of parameters in the model.

The second way to define the lower-dimensional subspace is to use the top two Hessian eigenvectors. Since the eigenvectors associated with the largest eigenvalues correspond to the directions of most variation in the loss function, sampling along these directions can often reveal more interesting surfaces. Here, we use PyHessian [Yao et al., 2020], a PyTorch library for efficiently calculating Hessian information using (randomized) numerical linear algebra.

Given the two orthonormal directions, we follow the approach outlined by Li et al. [2018]. More formally, we sample the trained model at discrete positions in the subspace defined by the two directions and evaluate the loss $\mathscr{L}$ as follows:

$$f(\alpha_1, \alpha_2) = \mathscr{L}(\theta + \alpha_1 \delta_1 + \alpha_2 \delta_2), \tag{4}$$

where (α_1, α_2) are the coordinates in the two-dimensional subspace, δ_1 and δ_2 correspond to the first and second direction defining that subspace, and θ is the original model. As such, each coordinate corresponds to a perturbed model, and we store the loss for that model, such that the collection of loss values forms the two-dimensional loss landscape.

Given a two-dimensional loss landscape, we can represent the sampled points as *image data*, where each pixel represents the loss of a perturbed model, or as an *unstructured grid*, where each vertex in the grid is associated with a scalar loss value. For the unstructured grid, we need to define the spatial proximity (or connectivity) of vertices in the grid based on the similarity of their coordinates. This will allow us to characterize how the loss changes throughout the landscape (i.e., as parameters are perturbed from one vertex to the next).

Here we explored both triangulations (e.g., Delaunay, Gabriel graph) and neighbor-based graph approaches. We found that a scalable, approximate nearest neighbor algorithm [Dong et al., 2011] yielded good results while also being much faster. Specifically, we use a k-nearest neighbor graph, where each point is connected to the k most similar points, and we use $k = 8$, such that the connectivity is similar to the spatial proximity of pixels in an image (i.e., left, right, top, bottom, and all four corners).

A.2 Topological Data Analysis

We perform topological data analysis (TDA) to extract and summarize the most important features of the loss landscapes. We then further quantify the results from TDA to provide new metrics that can be related to things like model performance. Specifically, we compute the merge tree and the 0-dimensional persistence diagram. Recall that the branches in the merge tree are equivalent to the 0-dimensional features [Edelsbrunner and Harer, 2008]. However, compared to the persistence diagram which describes the life span of each component, the merge tree provides additional information including which component each component merges into. The merge tree also provides a nice intuitive visualization of the changes in the loss landscape. To quantify the merge tree, we count the number of saddle points and the number of minima. To quantify the persistence diagram, we measure the average persistence by computing the distance between each persistence pair and the diagonal and then taking the average.

Note, we compute the merge tree and persistence diagram for each loss landscape using the Topology ToolKit (TTK) [Bin Masood et al., 2021]. Further analysis and visualization of the resulting merge trees and persistence diagrams was performed in Python.

B Supplemental Figures

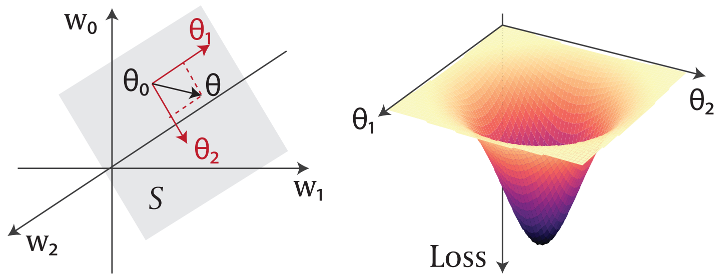

Figure 3: Illustration of generating the two-dimensional loss landscape. In this work, to compute a loss landscape, we first construct a subspace defined by two vectors, θ_1 and θ_2. Within this subspace, we perform model interpolation and evaluate the corresponding loss values.

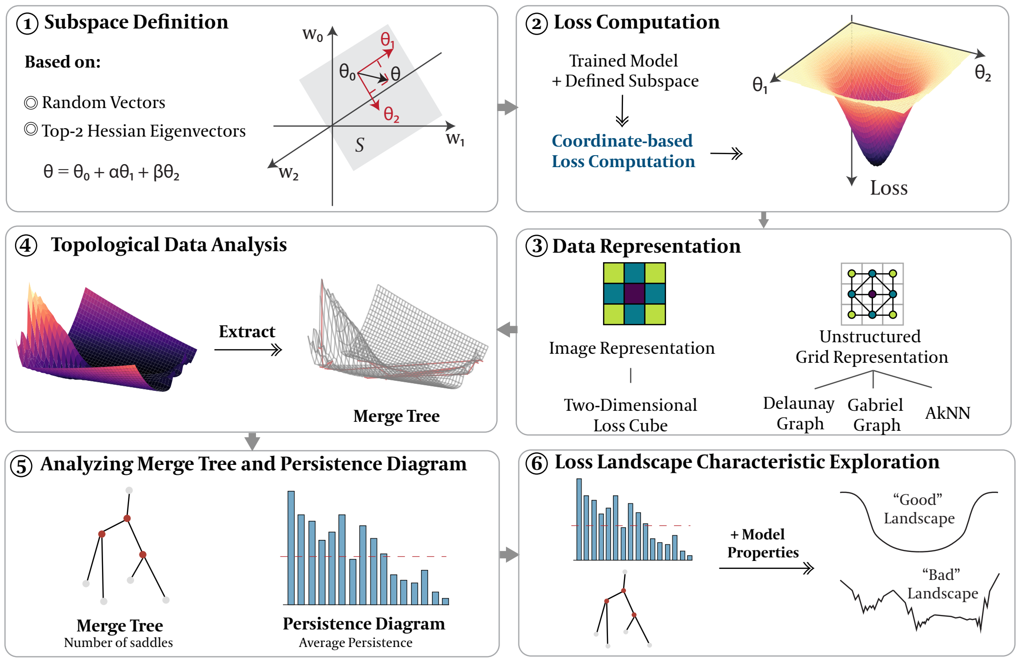

Figure 4: Our loss landscape analysis pipeline. The pipeline includes six stages: (1) *Subspace Definition*: Define the loss landscape subspace using random-based or Hessian-based directions. (2) *Loss Computation*: Calculate loss values for coordinate locations using our coordinate-based loss computation method. (3) *Data Representation*: Transform the loss landscape into suitable data structures for TDA. (4) *Topological Analysis*: Extract the merge tree and persistence diagram. (5) *Quantitative Evaluation*: Quantify the merge tree and persistence diagram. (6) *Loss Landscape Property Evaluation*: Relate the TDA-based metrics to loss landscape properties.

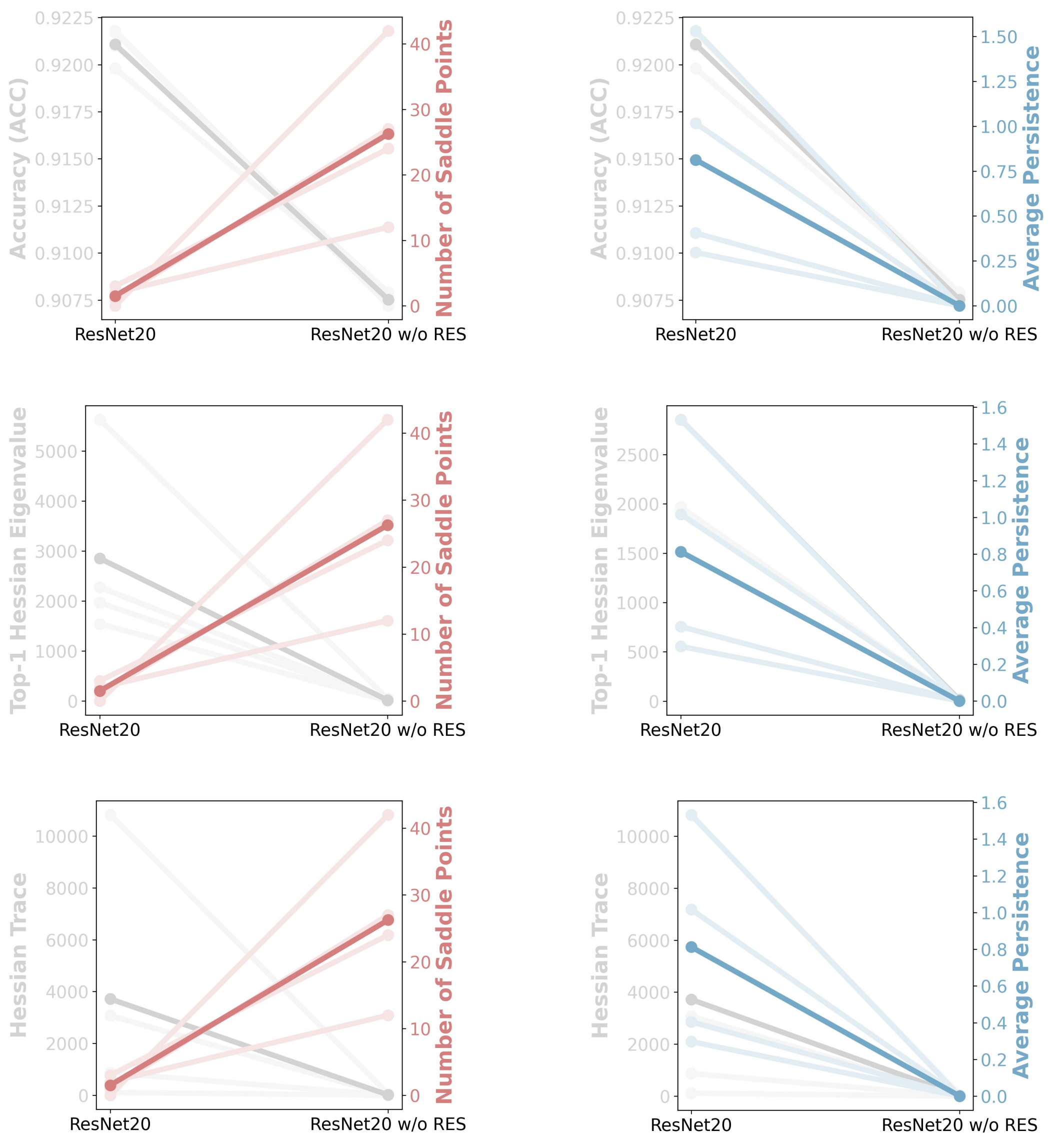

Figure 5: Quantifying the loss landscape for ResNet-20, with and without residual connections. We trained each version of the model four separate times, each time using a unique random seed (e.g., `0`, `123`, `123456`, and `2023`). Here, we numerically verify the observations we made in Fig. 1 and provide additional insights based on persistence diagrams (not shown). We quantified the merge tree and persistence diagram by counting the number of saddle points and computing the average persistence, respectively. We compare our results with traditional machine learning metrics, including the Accuracy, Top-1 Hessian Eigenvalue, and Hessian Trace. Here, we show the relationship between those ML-based metrics and our two TDA-based metrics. These plots provide additional insights beyond the qualitative differences in the loss landscapes we observed in Fig. 1, confirming that the landscapes for ResNet-20 models without residual connections correspond to merge trees with a higher number of saddle points (left column). In contrast, we see that these models (without residual connections) display a lower average persistence (right column). We observe an inverse relationship between the number of saddle points in the merge tree and the ML-based metrics, but a direct relationship between the average persistence and the same ML-based metrics. Together, these results provide insight into how changing the architecture of a neural network like ResNet-20 (i.e., by adding residual connections) can result in a "smoother" (and thereby easier to optimize) loss landscape.