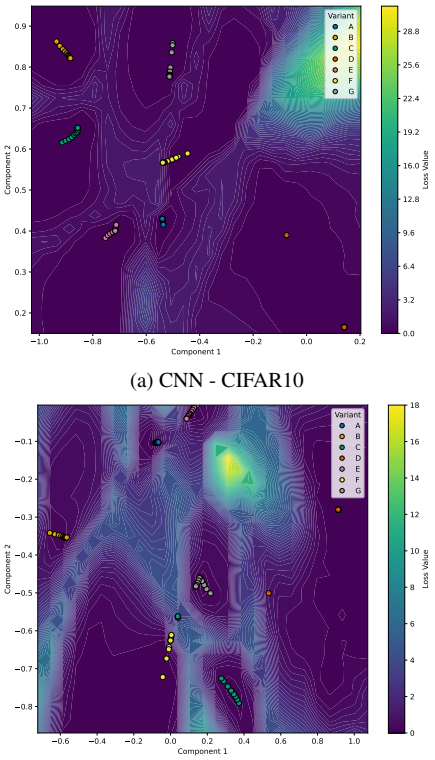
Globscope: Toward a Global View of the Loss Landscape

Supplementary Material



(a) CNN - CIFAR10

(b) CNN - CIFAR100

Figure A–1. Loss Landscape Visualization for a CNN model trained on different datasets

A. Appendix

A.1. Extended Results

Additional 2D visualizations for larger models (CNNs) and datasets (CIFAR10, CIFAR100) are shown in Figure A–1. The CNN model consists of 4 layers of convolutional blocks. Multiple training checkpoints (Variants A-G) of CNN have been created according to the guidelines of [12].

A.2. Computational Requirements

All experiments use a 2-dimensional latent space and a *Uniform AutoEncoder* with fixed hidden layers $[D, 128, 64, 16, 2],$, where D denotes the number of parameters of the target network. Since the hidden widths are fixed, the AutoEncoder parameter count scales linearly with D, dominated by the first encoder layer ($D \to 128$) and the final decoder layer ($128 \to D$). The total number of AutoEncoder parameters is approximately $P_{\text{AE}} \approx 256D$, corresponding to a weight memory of $M_{\text{weights}} \approx 2 \times 128 \times D \times 4$ bytes.

Model	# Params	Peak Mem.
MLP-MNIST	1.35×10^5	∼0.5 GB
ResNet56-CIFAR10	8.56×10^5	∼3.3 GB
CNN-CIFAR10	4.73×10^6	∼18.0 GB
CNN-CIFAR100	4.91×10^6	∼18.7 GB

Table 3. Memory requirements for different models

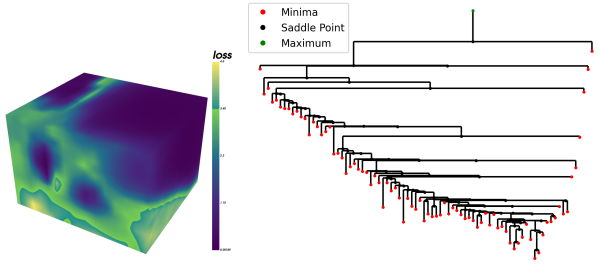


Figure A–2. 3D loss landscape and Merge Tree of ResNet56 trained on CIFAR-10

When accounting for optimizer states and gradients, the peak training memory is approximately $4\times$ the weight(and biases) memory footprint. As summarized in Table 3, the largest CNN models require up to ∼19 GB of GPU memory during training. Since all experiments are conducted on GPUs with 24 GB of VRAM, this imposes a practical limit on the size of target models, as larger models require a significantly larger autoencoder that may exceed the available memory. Training time also increases sublinearly with model size: the AutoEncoder for MLP-MNIST converges in approximately one hour, whereas CNN-based models require up to eight hours of training.

A fixed-size autoencoder does not scale well as target model sizes grow. While increasing the width or depth of the AutoEncoder could improve representational capacity, it would further exacerbate memory consumption and limit scalability. To enable the visualization of substantially larger models, a promising direction is to first apply low-rank compression techniques [29], model pruning [13] and sparsification [44] techniques to reduce the dimensionality of the model parameter space. The resulting compressed representations can then be used as inputs to our pipeline to generate low-dimensional loss landscapes.

A.3. 3D Loss Landscape Analysis

Results of the 3D loss landscape and its associated merge tree are presented in Figure A–2. Increasing the dimensionality of the latent space enables the recovery of richer

structural information regarding the relationships between local minima and their surrounding basins, as revealed by the merge tree. However, extending this analysis to higher dimensions is fundamentally limited by the exponential growth in sampling requirements. In particular, for a latent space of dimension d, a grid-based sampling strategy requires n^d evaluation points, where n denotes the number of samples per dimension, leading to computational and memory complexity that scales exponentially with d.

A.4. Experimental Guidelines

A.4.1. Training DEEP-ALIGN

The weight alignment algorithm,DEEP-ALIGN was trained on the MNIST MLP network for 50 epochs. The training was performed using the official Colab notebook provided in the authors' code repository [27][1].

A.4.2. Merge Tree Construction

Merge tree was constructed in *ParaView 6.0.1* using a CSV file containing spatial coordinates and the corresponding loss values of the loss landscape. The processing pipeline consisted of the following sequence of filters: *CSV* $\rightarrow$ *TableToPoints* $\rightarrow$ *Delaunay2D* $\rightarrow$ *Persistence Diagram* $\rightarrow$ *Threshold* $\rightarrow$ *Topological Simplification* $\rightarrow$ *TTK MergeTree*.

The CSV data were first converted into a point cloud using the *TableToPoints* filter. A triangulated 2D mesh was then generated via *Delaunay2D* to obtain a piecewise linear scalar field. The *Persistence Diagram* was computed to quantify the significance of topological features. To remove low-persistence noise, a persistence threshold was applied with a lower bound of 0.1. The scalar field was subsequently simplified using *Topological Simplification*, retaining only features above the specified persistence level. Finally, the *TTK MergeTree* filter was used to extract the resulting merge tree.

[1] https://github.com/AvivNavon/deep-align/tree/main