Course-ID:	MA-INF 4306
Course:	Lab Development and Application of Data Mining and Learning Systems: Machine Learning and Data Mining
Term:	Summer 2021
Supervisor(s):	Vanessa Toborek

How does training deep neural networks on a biased dataset affect the loss landscape of the network?

Ali Mohammadi Amit Kumar Rana

September, 2021

Abstract

Many studies show a positive correlation between the generalization ability of a deep neural network and the flatness of the minima in its loss landscape. Inspired by this statement, many studies investigate the effect of using different training parameters and network architecture on the loss landscape of the neural network. This study investigates the effect of training a deep neural network on a biased dataset on its loss landscape by visualizing the loss landscape of the trained model. We found that different types of biases in the training dataset can affect the geometry of the loss landscape around the minima.

Contents

1 Introduction

Convolutional Neural Networks (CNNs), especially Deep Neural Networks, have made significant progress in Machine Learning. They have become state-of-the-art methods in many areas, such as computer vision tasks [KSH12]. Many neural networks are trained by minimizing the loss function, such as cross-entropy loss or hinge loss, using some version of SGD (Stochastic Gradient Descent) and a backpropagation algorithm to update model weights in the direction of the gradients.

Compared to the traditional machine learning algorithms, deep networks have a significantly large number of trainable parameters compared to training data points. Still, they easily outperform these traditional algorithms by a considerable margin. The goal is to train such networks with heavy overparameterization to generalize well on an unseen test dataset (beyond the training dataset). Training neural loss functions is NP-hard in general as the goal is to minimize a high dimensional, highly non-convex loss function. Empirically, it has been observed that training methods find global minimizers (zero training error or significantly small training error) efficiently. It is not always the case that one can find global minimizers. It has been observed that the type of neural network architecture affects the ease of training. For example, adding skip connections makes the training easier for even very deep networks [Li+18]. The model's generalization ability depends on training parameters such as optimizers, learning rate, batch normalization, and dropout. These parameters affect the network in a certain way, but investigating the effect of each parameter choice in the architecture is very hard. The reason is that a deep neural network often has a considerably large number of parameters. It is a very high-dimensional function, and the evaluation requires all the data points to pass through the whole network. Given very high computation complexity, most of the work that has been done is theoretical.

A lot of prior works have shown a positive correlation between generalization and flatness around the minima [HS94; HS97; Cha+17; Kes+17; Nov+18; Wan+18]. Since a deep neural network can be reparameterization-invariant, a flatness measure around minima should also be reparameterization-invariant [Ran+19; Pet+19a]. For example, suppose we have a neural network with RELU activation functions, and we scale the weights of one layer by some value α and rescale the parameter of the next layer by $1/\alpha$. In that case, the resulting network produces the same prediction for an arbitrary input. Therefore, the network has the same ability of generalization with a different set of parameters.

Visualizing the loss landscape can provide insights into the loss curve and the training method's trajectory from initial weights to the convergence (global minima) in the parameters space. We can also visualize the behavior of loss landscape near maxima to check for the correlation between generalization and flatness around the minima. Visualizing the loss landscape is often a challenging task, given the very high dimensionality of the loss function. However, many studies try to provide methods for visualizing the loss landscape of a neural network. For example, Hao Li et al. [Li+18] suggest visualizing the loss landscape using filter normalized random directions. It compares different architectures and the training parameters by visualizing the loss landscape around the minima.

Only a few studies investigate the loss landscape of a neural network trained on a biased training dataset. The current study employs the method presented in [Li+18] to visualize the loss landscape and see whether there is a correlation between the generalization of a network trained on a biased dataset and the flatness around the minima.

2 Related Work

Many studies hypothesis that flat minima correlates positively with better generalization [HS94; HS97; Cha+17; Kes+17; Nov+18; Wan+18]. Sepp Hochreiter et al. [HS97] define flat minima as a large area in network parameters space where the loss value of a deep neural network remains low and relatively constant; the volume of such area is used as their measure for flatness of loss minima. On the other hand, N. Keskar et al. [Kes+17] use the maximum value of loss function around the minima to measure the flatness. Many studies suggest using the Hessian matrix of the loss function to measure the flatness [Kes+17; WZW17; Yao+18; Pet+19a; Ran+19].

Laurent Dinh et al. [Din+17] show that in some cases, arbitrarily sharper minima can be found through reparameterization without a loss in generalization ability and accuracy of a neural network; thus, they prove that flatness of minima does not necessarily correlate with generalization. Many studies try to address this problem by providing measurements for flatness invariant to such reparameterizations [Pet+19a; Ran+19]. Additionally, other topological properties of loss minima have been studied. For example, Haowei He et al. [HHY19] investigate asymmetric valleys in a loss landscape where the loss value increases fast in a direction and relatively slow in the opposite direction. Furthermore, many studies explore the loss landscape of neural networks concerning different network architecture [Li+18], different training parameters [Zha+17; Kes+17; Li+18], and even the training dataset [Chi+20].

3 Methods and Materials

3.1 Network Architectures

This study investigates the loss landscape of a neural network trained on a biased dataset. For that, we decided to ResNet [He+16] and a less advanced version of ShaResNet [Bou17] to train and conduct the study. In the following two sections, we briefly explain each architecture and our reasons for choosing them.

3.1.1 ResNet

We investigate the loss landscape of ResNet-20 architecture [He+16]. The reason for choosing 20 layered ResNet is that the training time increases notably with larger networks. The training time is even more critical considering that we have to train the network multiple times to investigate multiple biases in the training dataset with varying bias parameters. With deeper networks, the number of parameters also increases by a large margin. Since we have to visualize the loss landscape for each trained network (corresponding to different biases), it takes considerably high computational resources. On the other hand, the network performance deteriorates significantly with fewer layers. The parameter details are shown in Table 1.

3.1.2 ShaResNet

Even though ResNet-20 has relatively fewer parameters than the deeper networks, the number of parameters is still huge when visualizing and analyzing the network's loss landscape. Therefore, we decided also to train ShaResNet-20 with significantly fewer parameters in case the number of parameters becomes problematic. Inspired from [Bou17], we share the weight parameters between layers of a residual block if both layers operate at the same spatial scale. As shown in [Bou17], without compromising much on the accuracy, the number of trainable parameters reduced by around 30 percent compared to the original ResNet-20 network. The details of the parameters of ShaResNet-20, as well as sideways comparison with ResNet-20, are shown in Table 1.

Parameters details	**ResNet-20**	**ShaResNet-20**
Total parameters	274,442	175,130
Trainable parameters	273,066	173,754
Non-trainable parameters	1,376	1,376

Table 1: **Parameters details:** The number of total parameters along with the number of trainable and non-trainable parameters for ResNet-20 and ShaResNet-20.

3.2 Dataset

We use the CIFAR-10 dataset to conduct our experiments [Kri09]. The CIFAR-10 dataset consists of 60000, 32x32 color images in 10 classes, with 6000 samples per class. There are 50000 training images and 10000 test images. For visualisation purpose, 10 random samples per class are shown in Appendix C, Fig. 7.

3.3 Biasness

The current study studies three different kinds of biases in the training dataset with varying degrees of biasness. We do not add any bias to the test dataset, which is used for visualizing the loss landscape. The following three sections briefly explain each bias and how to add it to an unbiased training dataset.

3.3.1 Mislabel Bias

The first bias we investigated is the mislabeling bias, where some of the samples in the training dataset have an incorrect label. To obtain a training dataset with mislabeling bias, first, we select a fixed percent (error percentage) of the dataset uniformly distributed over all classes to mislabel. Next, we assign a new label chosen randomly (from a class other than the correct class) to each sample. Finally, we verify that the number of samples per class remains roughly the same, and therefore only the mislabeling bias is involved. We study mislabeling bias with three different error percentages of 30, 60, and 80 percent.

3.3.2 Size Bias (Skewness)

In a size-biased dataset, data samples are distributed non-uniformly over different classes. Therefore even though misinformation is not involved, the dataset becomes unbalanced concerning the number of samples per class. This study investigates the case where one particular class has significantly fewer samples than other classes. We add this bias by first selecting one class of data and then removing a fixed percent (error percentage) of its samples. We chose the 'Cat' class and varied the percentage of removed samples from 30 to 90 with an increment of 30. We also experimented with the 'Ship' class by removing 60 percent of its data points.

3.3.3 Gaussian Noise

Finally, we investigate the loss landscape of a neural network trained on a dataset consisting of samples corrupted by Gaussian noise. To obtain such a training dataset, first, we select a particular class to be corrupt. Next, we generate a Gaussian noise with the same mean as the mean of samples of the chosen class and a variance of 0.02. Finally, we add the generated noise to a fixed percent (error percentage) of the samples belong to the chosen class. We chose the 'Cat' class and varied the percentage of removed samples from 30 to 90 with an increment of 30.

3.4 Visualizing the Loss Landscape

We use the method introduced by Hao Li et al. [Li+18] to visualize the loss landscape of the trained models. This method plots the loss function along a random direction in the parameters space. Formally, $\boldsymbol{L(\theta)}$ can be defined as the loss value of a model with parameters $\boldsymbol{\theta}$. The set of parameters $\boldsymbol{\theta}$ with n parameters can be defined as a point in the n-dimensional parameters space. To provide a 1-dimensional visualization of a model's loss function along an n-dimensional direction vector $\boldsymbol{\delta}$, one can define a new function of form $\boldsymbol{f(\alpha) = L(\theta + \alpha\delta)}$ and instead plot $\boldsymbol{f(\alpha)}$ with $\boldsymbol{L(\theta)}$ at the center $(\boldsymbol{\alpha = 0})$. One can easily extend this approach for 2-dimensional visualization of the loss function by employing a second direction vector and plot $\boldsymbol{f(\alpha, \beta) = L(\theta + \alpha\delta + \beta\eta)}$.

Despite the simplicity of using the method of the random direction to visualize the loss landscape of a network, as shown by Hao Li et al. [Li+18], this approach is not invariant to the scale of parameters. It, therefore, is not suitable for the comparison of different networks. To address this problem, Hao Li et al. [Li+18] suggest using filter-wise normalized directions. To generate filter-wise normalized direction, initially, a direction $\boldsymbol{d}$ is used. Next, for each filter f with parameters $\boldsymbol{\theta_f}$, the corresponding parameters $\boldsymbol{d_f}$ are rescaled to have the same norm. In other words, each parameter in $\boldsymbol{d_f}$ is scaled by a scaling factor of $\frac{\|\boldsymbol{\theta_f}\|}{\|\boldsymbol{d_f}\|}$ where $\|\cdot\|$ denotes the Frobenius norm. This normalization is also applied to the fully connected layers treating them as convolutional layers with a 1x1 output feature map. Note that Hao Li et al. [Li+18] does not apply the random directions to the batch normalization parameters.

In the current study, we generate 20 randomly initialized direction vectors and perform filter-wise normalization on them. Next, we use "Mean Squared Error" and "Categorical Cross-Entropy" as loss function ($\boldsymbol{L(\theta)}$) and calculate $\boldsymbol{f(\alpha)}$ as defined above with $\boldsymbol{\alpha \in [-1, 1]}$. Additionally, for each point, we calculate the accuracy as well. Finally, we calculate the average of 20 values on each point to produce a single trajectory. For 2-dimensional visualizations, we use a similar approach. However, instead of using the average of multiple trials, we first produce 50 filter-wise normalized random directions and find 2 with the minimum dot product to assure maximum orthogonality between directions.

3.5 Implementation Details

We implement the neural networks and the training procedure in Python using Keras [Cho+15] and TensorFlow [Mar+15] libraries. The networks are trained in the "google colab" with the free GPUs available. We normalize the CIFAR-10 dataset by scaling pixel values from [0,255] to [0,1], then we demean the dataset by subtracting the mean of each sample from the sample. We train the network on 200 epochs over the whole training dataset, with a batch size of 128. We use Adam [KB15] optimizer with cross-entropy loss. Time taken per epoch for training is around 34 seconds for ResNet-20 and 22 seconds for ShaResNet on average. To stabilize the training, we use a learning rate decay. The learning rate is scheduled to be reduced after 80, 120, 160, 180 epochs.[1].

4 Results and Discussion

We first trained ResNet-20 on the CIFAR-10 training dataset without any bias. The classification report on validation data, is shown in Table 2. The classification report shows that class label 3 (Cat class) has the lowest precision, class label 8 (Ship class) has relatively high precision. Therefore as discussed in Sec. 3.3.2 and Sec. 3.3.1 we chose 'Cat' and 'Ship' classes to add size-biased and mislabeling bias.

Classfication Report, ResNet-20, unbiased			
Class label	**precision**	**recall**	**f1-score**
0	0.80	0.82	0.81
1	0.90	0.92	0.91
2	0.72	0.69	0.70
3	0.64	0.64	0.64
4	0.77	0.78	0.77
5	0.71	0.70	0.71
6	0.82	0.85	0.83
7	0.83	0.82	0.83
8	0.89	0.89	0.89
9	0.86	0.88	0.87

Table 2: **Classfication Report, ResNet-20, unbiased:** The classification report on validation data for the ResNet-20 architecture trained on the unbiased training dataset.

Deep neural networks can generalize so well even with notably more parameters than the training samples. To see this numerically, we use the equation and formulation for generalization error, also used in [Pet+19b]. The generalization error is given by the difference between the test error and training error (empirical error), given by

$$\varepsilon_{gen} = E_{(x,y)\sim D}[l((f(x),y)] - \frac{\sum_{(x,y)\in S} l((f(x),y)}{|S|} \tag{1}$$

which is the difference between the expected error on the target distribution D and the empirical(training) error on finite dataset S sampled from D. Here, f is the function denoted (learned) by the deep neural network.

Using the equation 1, we calculate the generalization error for both ResNet and ShaResNet trained with different biases. The result is shown in Table 3 and Table 4, for ResNet-20 and ShaResNet-20 respectively. We got the worst generalization for the mislabelled biases. The generalization declined with the increment in the mislabelled samples per class. The generalization errors are similar for both ResNet-20 and ShaResNet-20. However, as expected, the generalization error for the ShaResNet-20 is slightly higher than ResNet-20.

The generalization error for unbiased, gaussian-noise-bias-cat-60, size-bias-cat-30, size-bias-cat-60, and size-bias-ship-60 are comparable. It's evident from Table. 3, that the generalization error for size-bias-cat-30 and gaussian-noise-bias-cat-60 are slightly lower than

[1]Gihub repository: `https://github.com/ali-mohammadi-scrc/ML_Lab`

ResNet-20 after 200 epochs			
Bias Type	**Train accuracy**	**Val. accuracy**	**Gen. error**
unbiased	1.0	0.7968	0.2032
gaussian-noise-bias-cat-60	1.0	0.8111	0.1889
mislabeling-bias-30	0.9999	0.5081	0.4919
mislabeling-bias-60	1.0	0.2655	0.7345
mislabeling-bias-80	0.9978	0.1369	0.8609
size-bias-cat-30	1.0	0.8085	0.1915
size-bias-cat-60	1.0	0.7983	0.2017
size-bias-cat-90	1.0	0.7641	0.2359
size-bias-ship-60	1.0	0.7979	0.2021

Table 3: **Performance of ResNet-20:** Training accuracy, validation accuracy, and generalization error for the ResNet-20 architecture for different biases. The generalization error is calculated using the eq. 1.

the unbiased network. We observed in these cases, the accuracy for the corrupted class is declined while improving the accuracy of some of the other classes, balancing the overall accuracy. For example, when 'Cat' class samples were removed, the validation accuracy for the 'Cat' class went down, but at the same time, accuracy for the 'Dog' class increased. The mislabelling biases have the worst generalization. The reason is that a fixed percentage of the data from each class is mislabelled, meaning that we are feeding misinformation to the network, hence making the network unstable. This behavior gets even worse when we increase the samples of mislabelled samples in the training data. As seen in Table 3 and Table 4, When mislabelling is done to 80 percentage of the training dataset, the validation accuracy is around 13 percentage which is just slightly better than guessing the class labels by chance.

ShaResNet-20 after 200 epochs			
Bias Type	**Train accuracy**	**Val. accuracy**	**Gen. error**
unbiased	1.0	0.7846	0.2154
gaussian-noise-bias-cat-60	1.0	0.7774	0.2225
mislabeling-bias-30	0.9999	0.4801	0.5199
mislabeling-bias-60	0.9999	0.2551	0.7448
mislabeling-bias-80	0.9993	0.1334	0.8658
size-bias-cat-30	1.0	0.7795	0.2205
size-bias-cat-60	1.0	0.7753	0.2247
size-bias-cat-90	1.0	0.7549	0.2451
size-bias-ship-60	1.0	0.7721	0.2279

Table 4: **Performance of ShaResNet-20:** Training accuracy, validation accuracy, and generalization error for the ShaResNet-20 architecture for different biases. The generalization error is calculated using the equation [1].

To visualize the loss landscape of the trained networks, we follow the methodology from Sec. 3.4. 1-dimensional visualization of the loss landscape on the validation data for both ResNet and ShaResNet is shown in Fig. 1. It can be observed that the plots for mislabelling biases are significantly higher compared to other biases at the center location; this

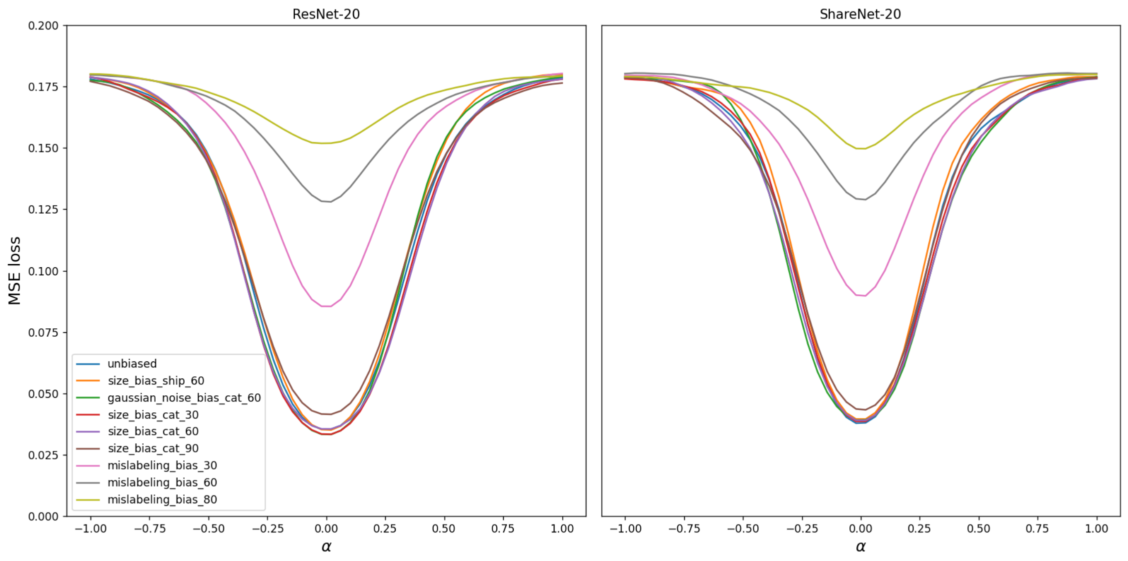

Figure 1: **1D visualization of the loss landscape of ResNet-20 and ShaResNet-20 with Mean-squared error loss**

implies that they have high validation loss around the minima. The plots for the rest of the biases are relatively similar, which also goes with the fact that the generalization behavior of the networks for these biases is almost similar, as shown in Table 3 and Table 4. Following our discussion in Sec. 1 and Sec. 2, a lot of previous studies show the correlation between generalization and flat minima. Nevertheless, most of those studies compare different architectures and network parameters. Here, we study the effect of using a biased training dataset, and 1-dimensional visualization is not strong or clear enough to give any correlation between flatter minima of loss landscape and better generalization.

To provide more insights into the loss landscape of the networks, we also generate the 2-dimensional visualization of the loss landscape, as described in Sec. 3.4. The resulting 2-dimensional visualizations for ResNet are shown in Fig. 2. It is evident from Fig. 2 that the shape of the loss landscape is pretty much similar for all the biases except for the mislabelling biases because all have similar generalizations. We can see that for mislabeling, especially for 60 and 80 percentage, we have a bit of sharpness around the minima compared to the unbiased network. We interpret this as a positive correlation between the generalization and flatness around minima. We also plotted the 2-dimensional contours for loss landscape around minima shown in Fig. 6. The visualizations with cross-entropy error loss are present in Appendix A.

We also observe that the fluctuations around the minima increase with the increase in biasness, adding chaotic behavior to the loss landscape around the minima. It is still tough to find any direct correlation from these geometric observations around minima and the networks' generalization. We also observe from Table. 3 that the networks' generalization when 30 and 60 percent of samples from the 'Cat' class are removed is better than the unbiased network. This behavior might be due to a correlation between the 'Cat' class and some other classes (mainly the 'Dog' class) because, with a slight decrease in the validation accuracy of the 'Cat' class, the validation accuracy for the 'Dog' class increases significantly.

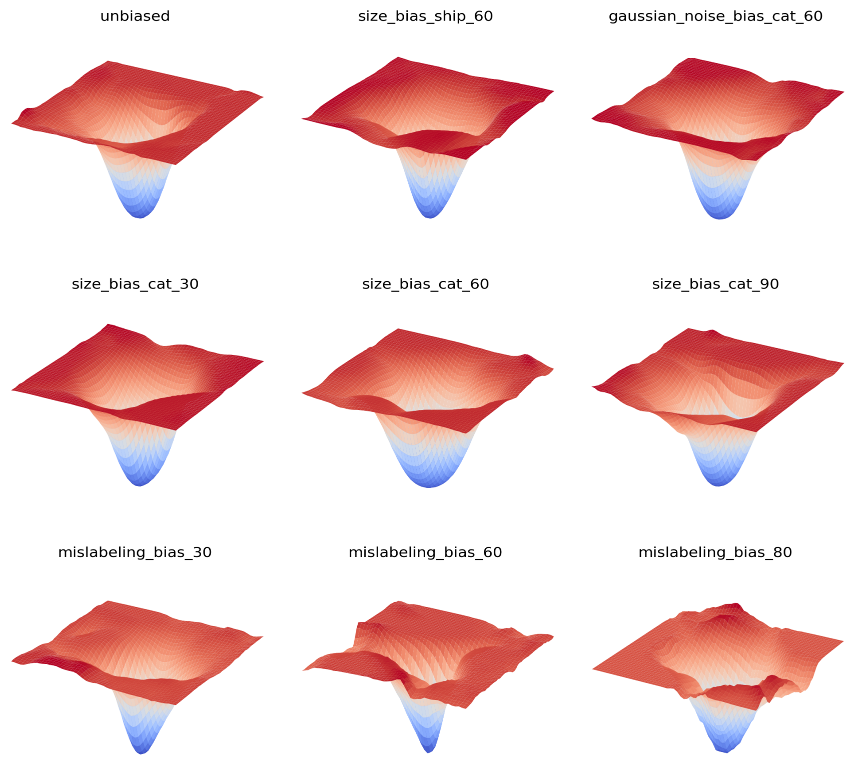

Figure 2: **2D Loss Landscape Visualization for ResNet-20 with Mean-squared error loss**

5 Conclusion and Further Remarks

Inspecting the loss landscape around the minima for neural networks can give us insights into the network's generalization ability. Many studies visualize the loss landscape for different network architectures and different training parameters. In this work, we inspect the loss landscape for networks trained on a biased training dataset. Following the discussion in Sec. 4, we experimented with multiple biases and found that when misinformation (mislabelled training dataset) is involved in the training dataset, the generalization ability of the network significantly drops. The increment in misinformation also leads to increment the generalization error. We found some implicit correlation between some of the classes in the training dataset when we used size bias(skewness). However, we are unsure whether it is generalizable to other datasets or specific to the CIFAR10 dataset.

1-dimensional visualization of the loss landscape does not provide much information about the network. 2-dimensional visualizations give us better insights into the geometry of the loss landscape around the minima. For the mislabeled biases, the minimas are sharper compared to the unbiased network. Expectedly their generalization error is relatively high. Since we use 2-dimensional vectors to visualize a high-dimensional loss landscape, it is hard to get the exact loss surface. Our work can be further extended using quantitative measures of flatness around the minima as an indicator of the generalization ability. Most of the methods [Pet+19b; Ran+19] are Hessian-based, which requires high computational resources, and therefore impossible to compute in this study.

References

[HS94] S. Hochreiter and J. Schmidhuber. “Simplifying Neural Nets by Discovering Flat Minima”. In: *NIPS*. 1994.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Flat Minima”. In: *Neural Computation* 9.1 (1997), pp. 1–42.

[Kri09] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[KSH12] A. Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60 (2012), pp. 84–90.

[Cho+15] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[Mar+15] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[He+16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*) (2016).

[Bou17] Alexandre Boulch. “ShaResNet: reducing residual network parameter number by sharing weights”. In: *ArXiv* abs/1702.08782 (2017).

[Cha+17] P. Chaudhari et al. “Entropy-SGD: Biasing Gradient Descent Into Wide Valleys”. In: *ArXiv* abs/1611.01838 (2017).

[Din+17] Laurent Dinh et al. “Sharp Minima Can Generalize For Deep Nets”. In: *ICML*. 2017.

[Kes+17] N. Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *ArXiv* abs/1609.04836 (2017).

[WZW17] Lei Wu, Zhanxing Zhu, and E. Weinan. “Towards Understanding Generalization of Deep Learning: Perspective of Loss Landscapes”. In: *ArXiv* abs/1706.10239 (2017).

[Zha+17] Chiyuan Zhang et al. “Understanding deep learning requires rethinking generalization”. In: *ArXiv* abs/1611.03530 (2017).

[Li+18] Hao Li et al. “Visualizing the Loss Landscape of Neural Nets”. In: *NeurIPS*. 2018.

[Nov+18] Roman Novak et al. “Sensitivity and Generalization in Neural Networks: an Empirical Study”. In: *ArXiv* abs/1802.08760 (2018).

[Wan+18] Huan Wang et al. “Identifying Generalization Properties in Neural Networks”. In: *ArXiv* abs/1809.07402 (2018).

[Yao+18] Z. Yao et al. “Hessian-based Analysis of Large Batch Training and Robustness to Adversaries”. In: *ArXiv* abs/1802.08241 (2018).

[HHY19] Haowei He, Gao Huang, and Yang Yuan. “Asymmetric Valleys: Beyond Sharp and Flat Local Minima”. In: *NeurIPS*. 2019.

[Pet+19a] Henning Petzka et al. “A Reparameterization-Invariant Flatness Measure for Deep Neural Networks”. In: *ArXiv* abs/1912.00058 (2019).

[Pet+19b] Henning Petzka et al. “A Reparameterization-Invariant Flatness Measure for Deep Neural Networks”. In: *ArXiv* (2019).

[Ran+19] Akshay Rangamani et al. “A Scale Invariant Flatness Measure for Deep Network Minima”. In: *ArXiv* abs/1902.02434 (2019).

[Chi+20] Sathya R Chitturi et al. “Perspective: new insights from loss function landscapes of neural networks”. In: *Machine Learning: Science and Technology* 1.2 (2020), p. 023002.

Appendix A Visualizations of the loss landscape with Cross-entropy error loss

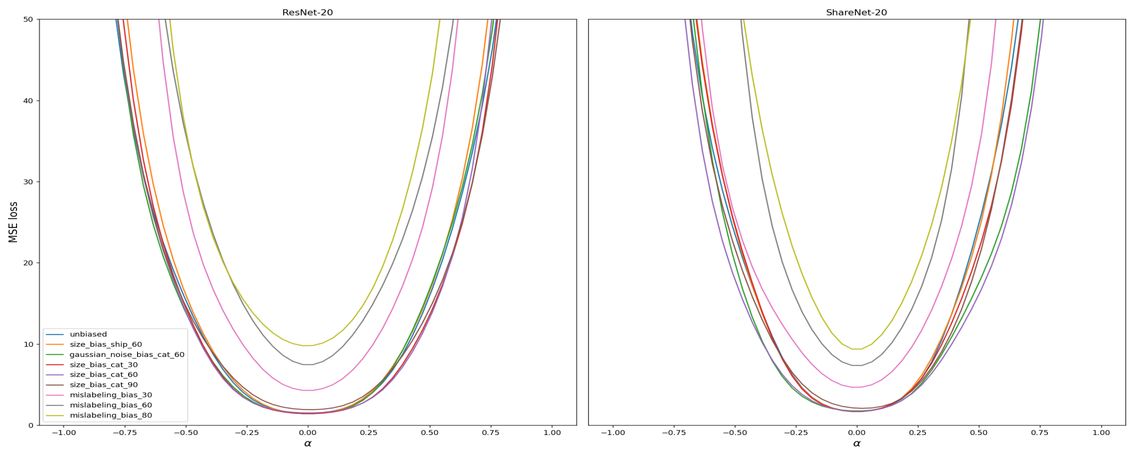

Figure 3: **1D visualization of the loss landscape for ResNet-20 and ShaResNet-20 with Cross-entropy error loss**

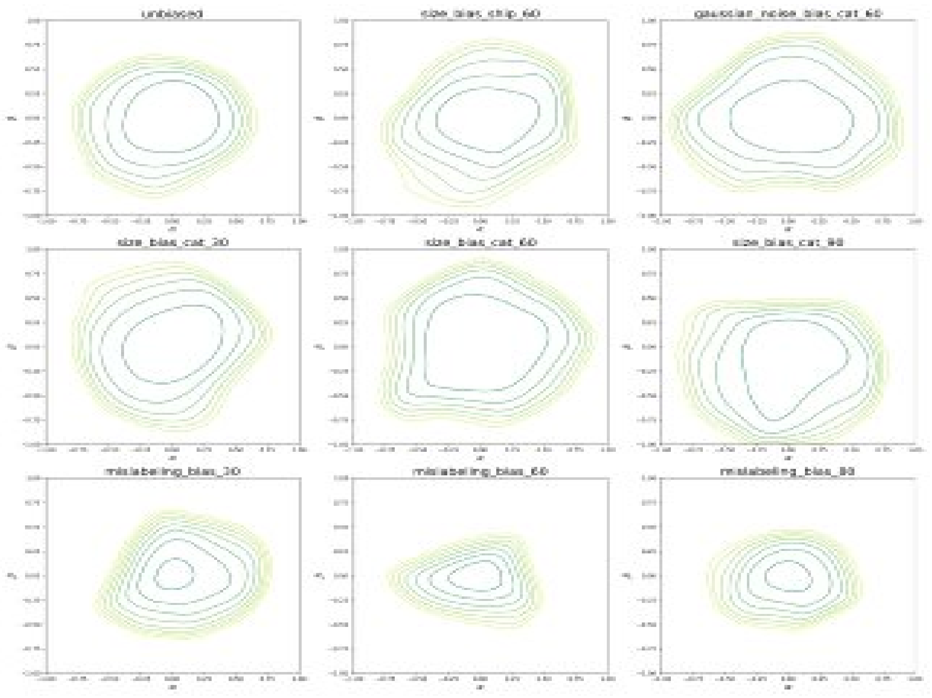

Figure 4: **2D Loss contour visualization for ResNet-20 withr Cross-entropy error loss**

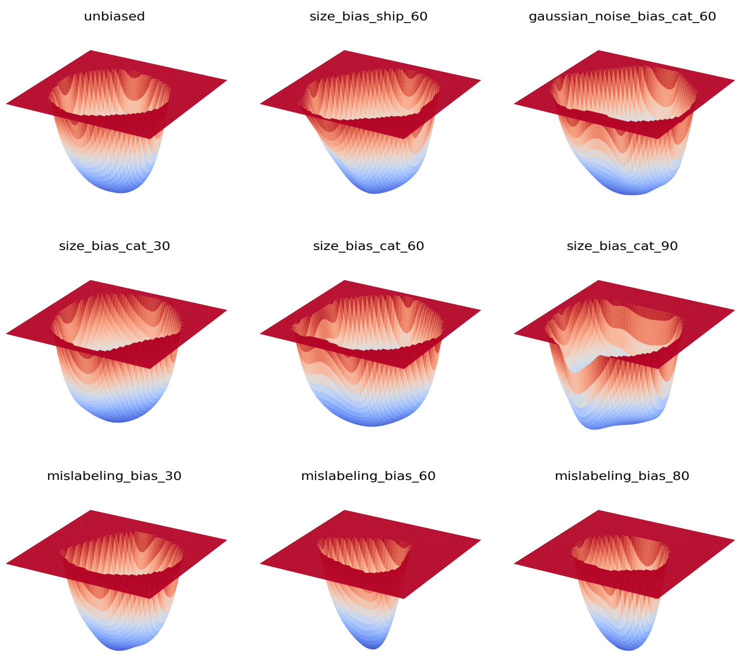

Figure 5: **2D loss landscape visualization for ResNet-20 with Cross-entropy error loss**

Appendix B 2D Contour Visualization of the loss landscape with Mean-squared Error

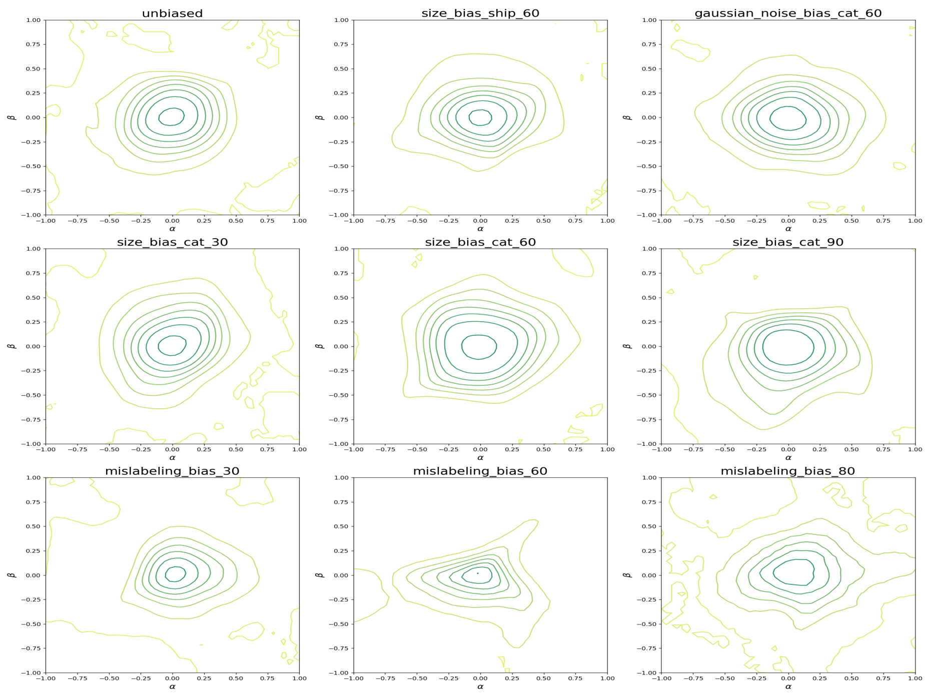

Figure 6: **2D Loss contour visualization for ResNet-20 with Mean-squared error loss**

Appendix C CIFAR10 Dataset

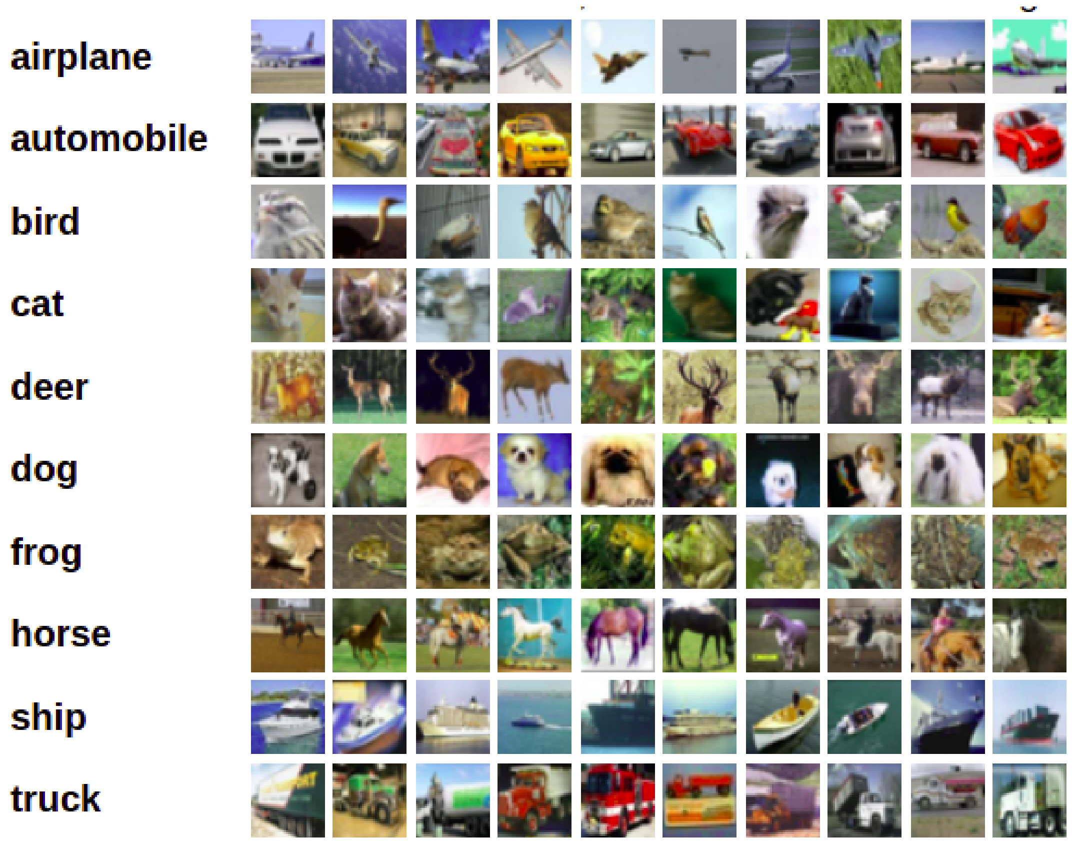

Figure 7: **CIFAR-10 dataset**: This image shows 10 random examples per class from CIFAR10 datset. Image from CIFAR10.