Optimizing CNN Training Strategies with Limited Data

Optimization for machine learning - Group project

Henri Roniger Simon Deconihout Clément Froidevaux

Abstract

In this work, we explore strategies to improve the performance of convolutional neural networks (CNNs) under limited data conditions. Using a fixed CNN architecture trained on a small subset of 1,000 CIFAR-10 images, we evaluate the impact of various data augmentation techniques on generalization performance. Our experiments demonstrate that a carefully selected combination of augmentations yields a **14.11**% increase in accuracy compared to the baseline. Furthermore, by pretraining the network on the larger CIFAR-100 dataset and fine-tuning it on our small CIFAR-10 subset (1000 samples), we achieve an improvement of **37.91**%, highlighting the strong potential of transfer learning in low-data regimes.

1 Introduction

Deep learning has led to major advances in fields such as computer vision, natural language processing, and speech recognition [1]. These breakthroughs are largely fueled by the availability of large-scale annotated datasets, which allow neural networks to learn complex patterns from data. However, in many real-world applications, particularly in specialized scientific or medical contexts [2], labeled data is often rare, costly to acquire, or labor-intensive to annotate.

In such low-data regimes, standard deep learning models tend to overfit, resulting in poor generalization to unseen data. This limitation motivates the exploration of strategies that can improve model robustness when only a small amount of training data is available. In this work, we investigate methods for optimizing the training of convolutional neural networks (CNNs) under data-constrained conditions, using a reduced version of the CIFAR-10 dataset [3] as a benchmark.

Our focus is twofold: (1) evaluate the effectiveness of data augmentation techniques in increasing the diversity of the training distribution, and (2) explore transfer learning approaches as a means of leveraging knowledge from other larger datasets. Together, these methods aim to bridge the gap between performance and data availability while offering practical tools for developing efficient CNNs in real-world, low-resource settings. This joint approach has recently been shown to be particularly effective in improving model generalization under limited data [4].

2 Methodology

We work with the `CIFAR-10` dataset and define a convolutional neural network, `CIFARCNN`, specifically designed for image classification. To gain insight about the problem and select an appropriate optimizer, we experiment with three popular methods: SGD, Adam, and Lion. These are tested on three training subsets of sizes $n = 163$, 1000, and 20000, using their standard hyperparameters as established in [5], [6], and [7]. We also visualize their loss landscapes on a 2D plane defined by the principal components (via PCA, see Appendix A.2), which provides insights into the optimization dynamics.

To evaluate performance in low-data regimes, we train the same fixed architecture on randomly

sampled subsets of size $\{30, 100, 300, 500, 1000\}$ from `CIFAR-10`. Various data augmentation techniques are employed to artificially increase the diversity of the training set, including horizontal flipping, random cropping with padding, color jittering (brightness, contrast, saturation, and hue), and random rotations up to $\pm 15°$. All inputs are normalized using the mean and standard deviation computed from each respective subset to maintain consistency and prevent data leakage.

The `CIFARCNN()` model comprises three convolutional blocks, each followed by batch normalization, ReLU activation, and dropout, with two fully connected layers at the output. Training is conducted using the Adam optimizer with a learning rate of 10^{-3}, a batch size of 64, and for 50 epochs. To ensure statistical robustness, all experiments are repeated with 10 random seeds. Final performance is evaluated based on test accuracy on the complete `CIFAR-10` test set.

For transfer learning, we pre-train the same CNN model for 100 epochs on the full `CIFAR-100` dataset [3] (50,000 images and 100 classes) to learn general features. This pre-trained model is then fine-tuned on limited subsets of $\{30, 100, 300, 500, 1000\}$ images from `CIFAR-10`, adapting its weights to the target task. The convolutional weights are thus reused from the pre-trained model, while only the final classifier layer is randomly reinitialized. This leverages prior knowledge to improve generalization in data-scarce conditions.

3 Results and Discussion

The first exploratory results for three optimizers and three training subsets of $n = 163, 1000$ and 20000 are presented in figure 1. As shown by the yellow and red curves, it is clear that Adam yields the best performance for smaller training datasets (better accuracy and lower validation loss).

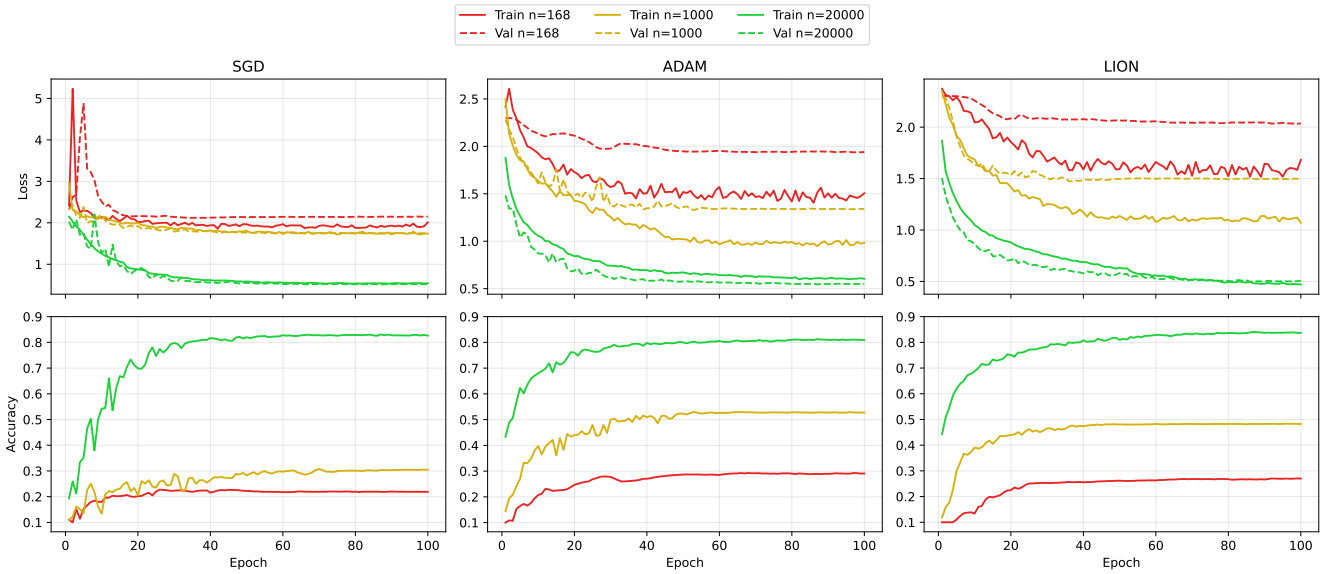

Figure 1: Progression of loss and accuracy during training and validation for SGD, Adam, and Lion across varying dataset sizes

After selecting Adam as the optimizer for all subsequent experiments, we next investigate the impact of data augmentation techniques on model generalization. When working with limited datasets, data augmentation can substantially improve model accuracy. As shown in Table 1, our best results for data augmentation were obtained by combining multiple augmentation techniques, yielding an improvement of 14.11%. Moreover, this suggests that augmentation

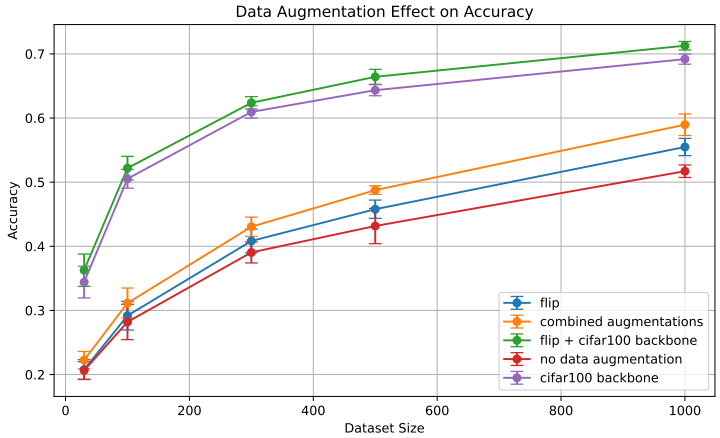

Figure 2: Impact of data augmentation and transfer learning on accuracy across dataset sizes

Dataset / Augmentation	**Mean Accuracy**	**Std Dev**	**Improvement (%)**
Original (no augmentation)	0.517	0.010	-
Horizontal Flipping	0.555	0.013	7.35
Random Cropping	0.539	0.012	4.25
Color Jittering	0.532	0.019	2.90
Rotation	0.524	0.013	1.35
Flip + Crop	**0.575**	0.016	**11.21**
Combined Augmentations	**0.590**	0.017	**14.11**
Trasfert learning	0.692	0.008	33.85
Trasfert learning + Flip	**0.713**	0.007	**37.91**

Table 1: Comparison of CNN performance (accuracy) trained on 1,000 CIFAR-10 images using different data augmentation strategies and transfert learning. Improvement percentages are relative to the baseline without augmentation. The *Combined Augmentations* setting corresponds to the application of all four individual augmentation techniques listed in the table.

strategies are most effective when they respect the underlying symmetries of the data. Horizontal flipping, for instance, is often harmless for objects like animals or vehicles, while rotations can distort class-defining features. In `CIFAR-10`, categories such as airplanes or trucks have a clear canonical orientation. Rotating them introduces ambiguity rather than helpful variation, which likely explains almost negligible improvement in performance. In contrast, combining augmentations like flipping and cropping seems to work well because they introduce spatial variability without breaking the visual semantics of the images. These transformations enrich the training set in a way that aligns with the structure of the task. Beyond augmentation, transfer learning proved particularly powerful. Pre-training the network on CIFAR-100 enables it to learn general-purpose visual features (edges, textures, shapes) that remain relevant for `CIFAR-10`. This is especially valuable in low-data regimes, where starting from random initialization would lead to poor generalization. Even when only fine-tuning the final layer, the benefits are substantial.

Dataset / Augmentation	**Mean Accuracy**	**Std Dev**	**Improvement (%)**
Original (no augmentation) ($n = 100$)	0.282	0.028	-
Trasfert learning + Flip ($n = 100$)	**0.505**	0.015	**79.08**

Table 2: Improvement from transfert learning for only 100 training samples

4 Bibliography

References

[1] Iqbal H. Sarker. Data science and analytics: An overview from data-driven smart computing, decision-making and applications perspective. *SN Computer Science*, 2(5):377, 2021.

[2] Xuxin Chen, Ximin Wang, Ke Zhang, Kar-Ming Fung, Theresa C. Thai, Kathleen Moore, Robert S. Mannel, Hong Liu, Bin Zheng, and Yuchen Qiu. Recent advances and clinical applications of deep learning in medical image analysis. *Medical Image Analysis*, 79:102444, 2022.

[3] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[4] Jianjun Su, Xuejiao Yu, Xiru Wang, Zhijin Wang, and Guoqing Chao. Enhanced transfer learning with data augmentation. *Engineering Applications of Artificial Intelligence*, 129:107602, 2024.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[6] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018.

[7] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms, 2023.

A Appendix

A.1 Additional data augmentation results

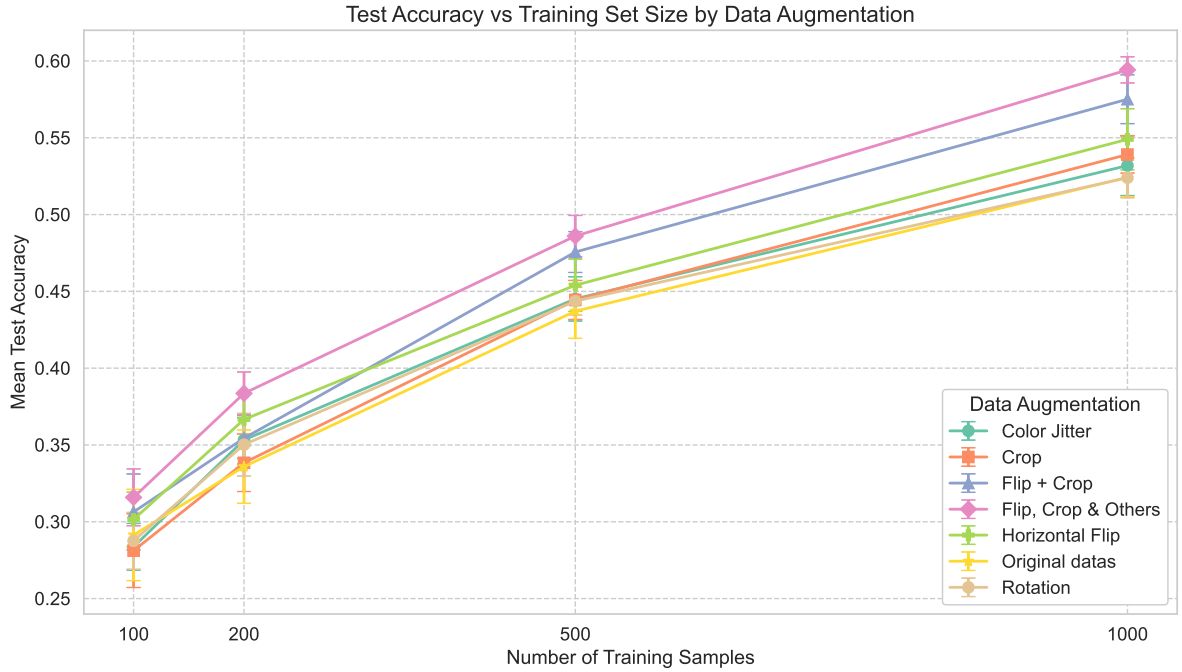

Figure 3: Test accuracy of all data augmentation techniques as a function of the number of samples

A.2 Loss landscape on the two principal vectors of PCA, and weights trajectory

To characterize the geometry of the optimization landscape and the implicit biases induced by optimizers, we analyze training trajectories projected into a low-dimensional subspace. Given a sequence of model parameters $\{\boldsymbol{\theta_t}\}_{t=0}^{T}$ collected during training with T epochs, we first vectorize the parameters $\boldsymbol{\theta_t} \in \mathbb{R}^d$ using PyTorch's internal flattening routine. Principal Component Analysis (PCA) is then applied to the stacked weight matrix W to extract the two dominant directions of variance:

$$W = \begin{bmatrix} \boldsymbol{\theta_0}^\top \\ \boldsymbol{\theta_1}^\top \\ \vdots \\ \boldsymbol{\theta_T}^\top \end{bmatrix} \in \mathbb{R}^{T \times d} \qquad U = \begin{bmatrix} \boldsymbol{u_1} \\ \boldsymbol{u_2} \end{bmatrix} = \mathrm{PCA}_2(W), \quad \text{with } U \in \mathbb{R}^{2 \times d}$$

These directions span the affine plane in which we evaluate the loss surface around the final iterate θ_T.
We define a grid in the (α, β) plane and compute the interpolated parameters as:

$$\boldsymbol{\theta}(\alpha, \beta) = \boldsymbol{\theta_T} + \alpha \boldsymbol{u_1} + \beta \boldsymbol{u_2}$$

where u_1, u_2 are the two principal vectors. At each grid point, we compute the loss $\mathcal{L}(\boldsymbol{\theta}(\alpha, \beta))$ on a held-out validation set.

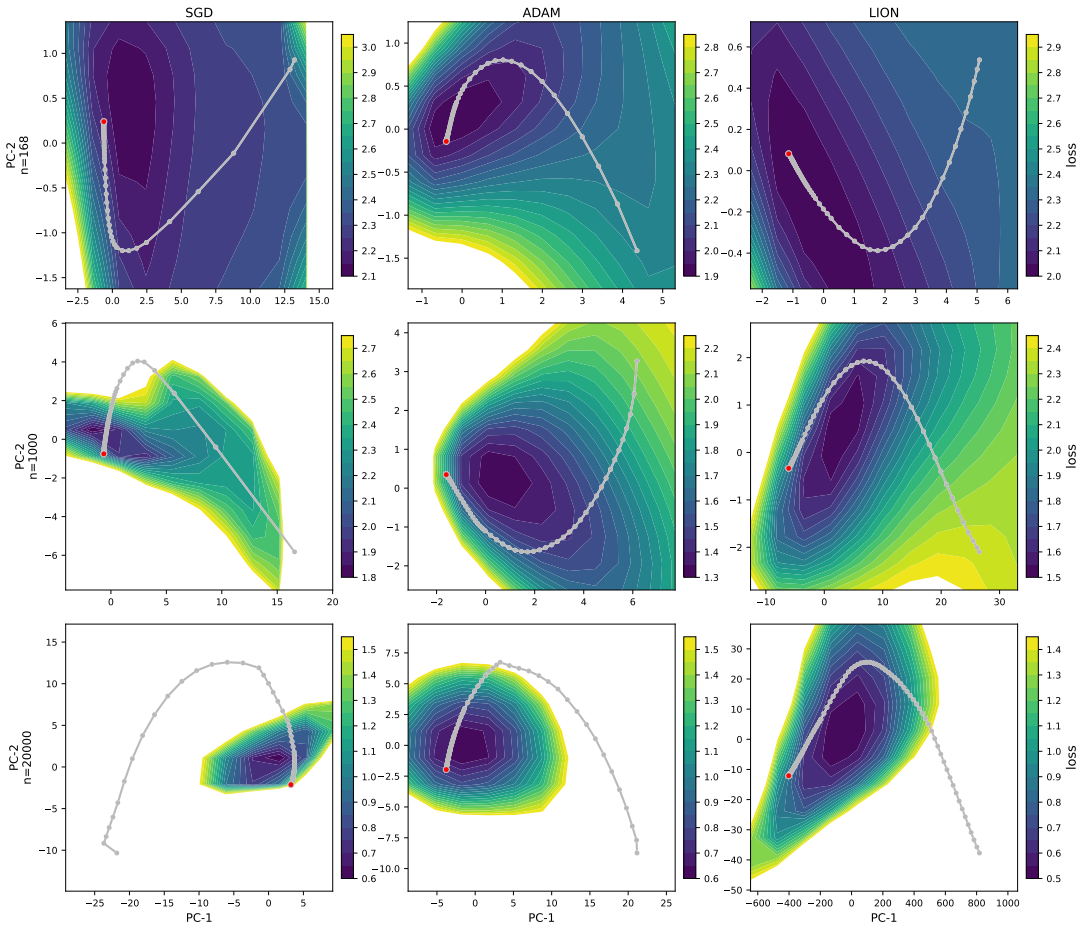

Figure 4: Convergence of the different optimizers in the PCA plane

To ensure interpretability across experiments and eliminate scale bias, we plot log-contours $\log \mathcal{L}(\theta)$ using a shared set of isocontour levels centered around the minimal loss observed:

$$\text{colorbar} \in \left[\min(\mathcal{L}),\ \min(\mathcal{L}) + \varepsilon\right] \qquad \text{with } \varepsilon = 1$$

This allows consistent comparison of sharpness and basin geometry across optimizers and dataset sizes. Trajectories are overlaid in the PCA plane with white markers at each epoch and the final iterate highlighted in red. Plots with a lot of white areas indicate narrower loss landscapes, while those with more blue regions represent flatter ones.
This method isolates the optimizer-induced dynamics from stochastic data variation, and provides geometric insights into optimization efficiency and generalization behavior.

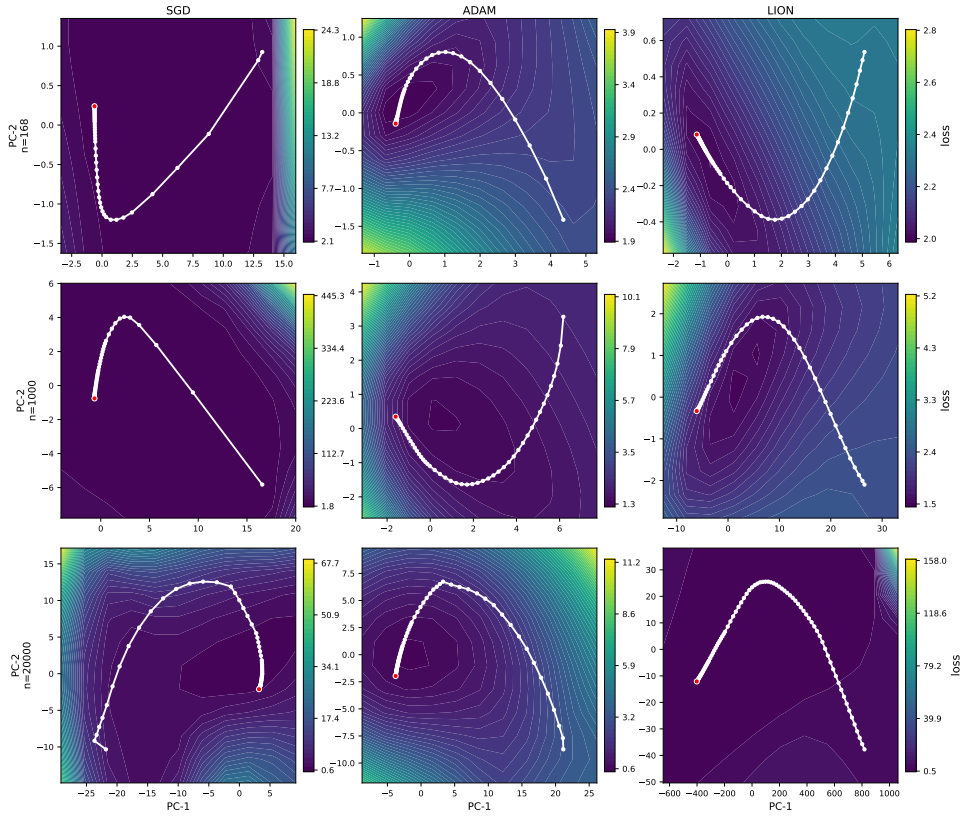

Figure 5: Total loss landscape: convergence of the different optimizers in the PCA plane

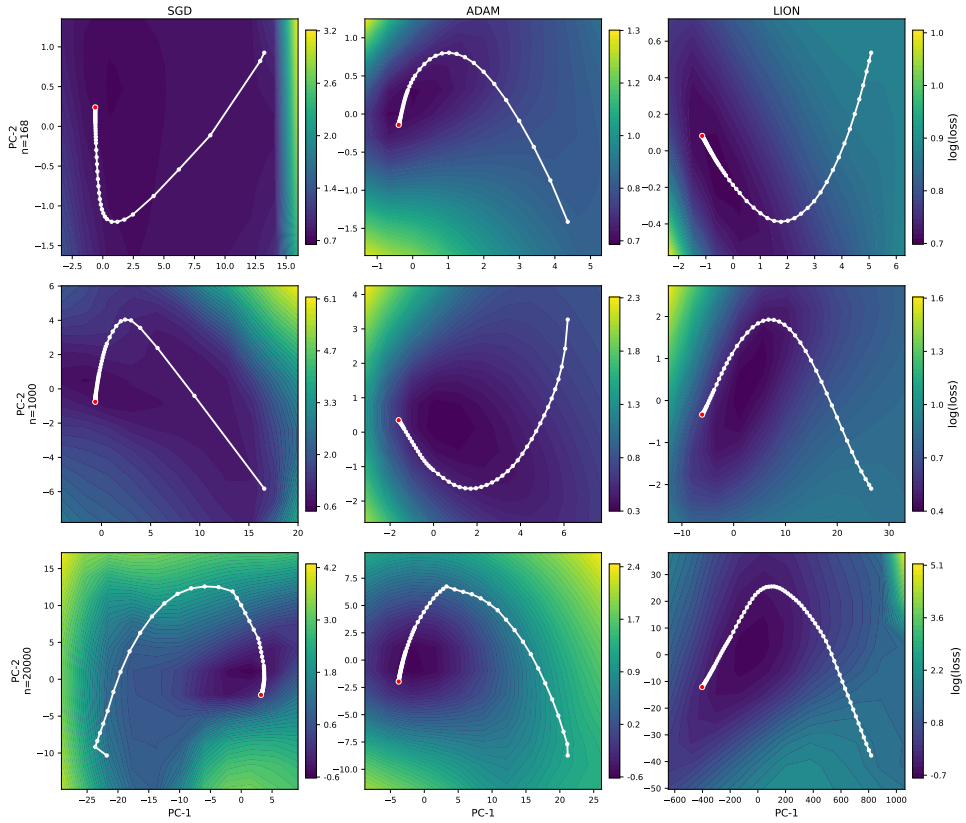

Figure 6: Logarithmic scale on the loss: convergence of the different optimizers in the PCA plane