Final project - EE-411 - 2023-2024

Towards Learning Convolutions from Scratch

Yassin Alnuaimee, Danilo Cammarata, Mehrad Sahebi

February 3, 2024

***Abstract*—This project rigorously tests the claims of ”Towards Learning Convolutions from Scratch” by Behnam Neyshabur [1], focusing on the innovative use of optimizers in neural networks. We methodically recreate and critically evaluate key aspects of the research, specifically targeting Table 2, Fig. 3, and Fig. 4. Our endeavor not only seeks to affirm the reproducibility of the results but also to understand their broader impact, leading to a thoughtful critique and the identification of potential new directions in machine learning research.**

I. Introduction

In our final project, we confront the Reproducibility Challenge by delving into ”Towards Learning Convolutions from Scratch” by Behnam Neyshabur. This project transcends mere replication; it’s a profound inquiry into the core of convolutional neural networks (CNNs) and their learning mechanisms. We meticulously reproduce key experiments, comparing convolutional and fully-connected networks, and scrutinizing the innovative β-lasso training algorithm.

The project progresses through a series of structured tasks. Initially, we focus on numerically replicating the paper’s results, closely following the original methodologies or applying minor modifications when necessary. This step lays the foundation for a deeper analysis and understanding.

Next, we contextualize the importance of these results, discussing their relevance and placing them within the broader machine learning landscape. This reflection not only deepens our understanding but also highlights the practical and theoretical implications of the research.

A critical review follows, where we juxtapose our replication efforts with a thorough critique of the original paper. This phase encourages critical thinking and reflective analysis, prompting us to consider our findings in light of the original research.

The project concludes with us proposing new research directions, inspired by our engagement with the paper and the insights gained from the replication process. This final task emphasizes the project’s ultimate aim: to not just replicate but to understand, critique, and contribute to the field of machine learning, pushing its boundaries through innovative ideas and research proposals.

Overall, this Reproducibility Challenge is a journey of technical skill, intellectual inquiry, and creative thought, guiding us through the complexities of scientific validation and innovation in machine learning.

In section II we present a short summary of the paper to show our understanding of the main and relevent concepts mentioned in it.

In section III we discuss our implementation and challenges along the way as well as the results we reproduced.

In section IV we finally compare our results to those of the original paper, mention the shortcomings and try to justify them.

II. Summary of the original paper

Inductive biases of a learning algorithm help it perform better on the types of data that are structured in a way that aligns with the inductive bias. Since the beginning, machine learning has progressed towards learning inductive bias from the data itself instead of presuming one before hand.
Convolutional Layers are well-known inductive biases that perform incredibly on image data (or any data with local correlations). The goal of this paper is to learn these convolutions from the data. In other words we are going to replace the convolution layers with fully connected layers of the same expressivity and try to keep the performance intact. If such thing is achieved, it is a further step towards eliminating architecture inductive biases and learning everything from the data itself.

Next, we turn to introduce some of the architectures used in the paper. Note that in the original paper many architectures are used and discussed but they can be categorized in two categories: shallow and deep. For the purpose of this report and the graphs we are required to reproduce, only shalow networks are needed.

A. Architecture of Shallow Networks

There are four Shallow Networks we will discuss.

1) Shallow Convolutional Network S_{conv}: This network consists of 1 convolutional layer and two fully connected ones. The convolutional layer consists of $\alpha \times 9 \times 9 \times 3$ convolutional filters with a stride of 2. From what we understood from Table 4 of the paper, the output size of this layer should be $s/2 \times s/2 \times \alpha$ where s is the dimension of input image, we deduced that there should be a padding of 4 so that the output image dimension is exactly $s/2$. There are no max pooling layers.
The first fully connected layer has $s/2 \times s/2 \times \alpha$ input neurons and 24α output nodes resulting in $6s^2\alpha^2$ parameters.

The output layer is a fully connected layer with 24α input nodes and c output nodes where c is the number of classes. All layers have ReLU activation at the end except the output layer. A sketch of this architecture is shown in Figure 1 of the paper

2) Shallow Local Network S_{conv}: This network is the same as the previous one except the first layer. In its first layer, instead of one set of convolutional layer shared by all pixels it has different sets of $\alpha\times\ 9 \times 9 \times 3$ convolutional filters for all the relevant pixels ($s^2/4$ that are used for the next layer). It is worth mentioning that the structure of this architecture was not described in the paper and the locally connected layer was simulated by a grouped convolution layer using the groups option in nn.conv2d.

3) Shallow Fully Connected S_{FC}: This architecture is also the same as the convlutional one except the first layer where we use a fully connected layer of output size $s^2\alpha/4$.

4) 3 Layer Fully Connected 3_{FC}: This is a testbench network run alongside with the others. It has 3 fully connected where the middle one has the same number of input and outputs (hidden layers of same hidden units).

All of the above mentioned architectures have batch normalization at the end of each layer.

It is clear that for the same α, the number of parameters in these models and as a result their expressive powers, vary signifantly. It is then important that for each of them we choose α in a way that at the end all of them have the same numbe of parameters. In the paper all models have 256 million parameters and the training is done over 400 and 4000 epochs.

B. β-lasso algorithm

To train a fully connected layer and try to make it as sparse as a convolutional layer, we need more aggressive relaxations in the optimizer (even more aggressive as lasso (l_1 regulation). The proposed algorithm in the paper called β-lasso is shown in Algorithm1 of the paper. This algorithm does another step of shrinking after the normal update of lasso.

C. parameters and hyper parameters

Some of the parameters are already mentioned above. The training is done using batches of 512 training data points on 3 different datasets namely, CIFAR10, CIFAR100, SVHN.
The performance of S_{conv}, S_{local}, S_{FC}, 3_{FC} with Stochastic Gradient Descent (SGD) optimizer and learning rate of 0.1 and momentum of 0, 0.9 (hyper-parameter) and S_{FC} with β-lasso optimizer with three different β's 0,1 and 50 is compared.
λ is chosen from $10^{-6}, 2 \times 10^{-6}, 5 \times 10^{-6}, 10^{-5}, 2 \times 10^{-5}$.

III. Methods

Given the computational constraints compared to the original study, we opted to reproduce the results of the paper with slightly modified hyperparameters. Specifically:

- The batch sizes for training and testing were maintained at 512.
- The learning rate was preserved at 0.1.
- The values for β and the regularization parameter λ were kept unchanged.
- Only one value was used for momentum (which is 0) and no cross validation has been performed.
- To mitigate the extensive training duration, approximately two hours per class, we reduced the number of parameters by an order of magnitude, from 256×10^6 to 256×10^5.
- The epoch count was decreased by a factor of four, from 400 to 100 epochs.

These modifications necessitated a reduction in the input base channels, adjusted as follows:

- For classes S_{conv}, S_{local}, and 3_{FC}, the base channels (α) were reduced to 64.
- For the S_{FC} classes, the input base channels were reduced to 28.

This approach was aimed at replicating the study's outcomes while addressing our computational limitations.

In order to handle the task efficiently, we created four classes implementing the needed architectures:
S_{conv}, S_{local}, S_{FC}, 3_{FC}.
We decided to give as input arguments of these classes:

- the number of output channels, due to the different dataset requirements.
- the number of base channels α, so that it could be tuned once for all.

The main code follows the subsequent structure.

Each dataset has been split between train, validation and test sets.

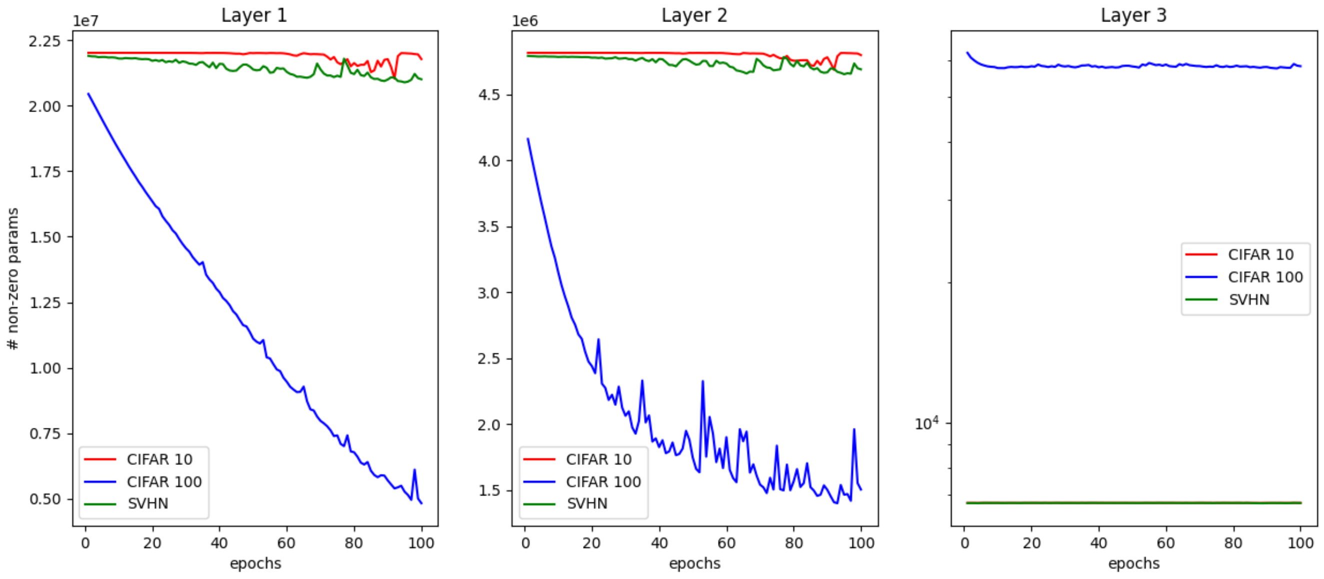

Figure 1: Number of non-zero parameters in different layers of S_{FC} trained with β-lasso

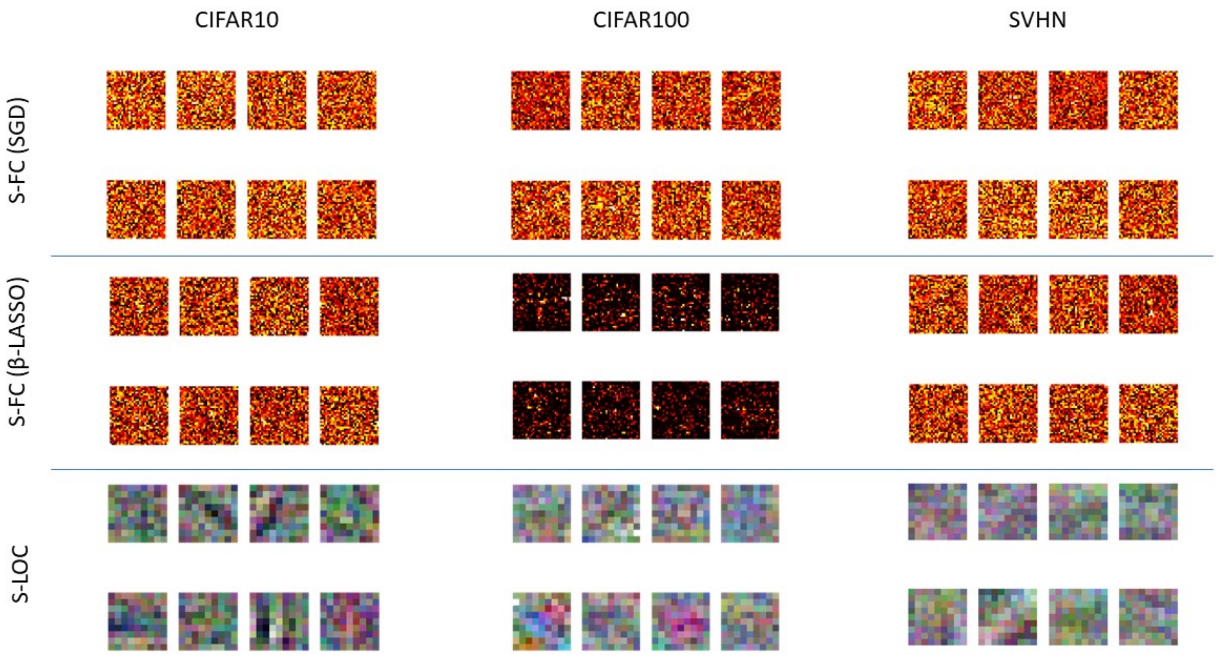

Figure 2: Comparing first layer of S_{FC} and S_{local} trained with SGD to that of compared β-lasso

Each class has been declared along with the optimizer we used in that simulation, and then we called the fit function to train the model: since Table I required seven cases for three datasets, we replicated these structures tewnty-one times.
For each epoch, the fit function:

- trains the Neural Networks through the train_epochs function, returning the train losses.
- counts the number of non-zero parameters for each layer with torch.count_nonzero.
- performs the validation through the predict function, returning the validation losses and the validation accuracy.

Moreover, it saves the computed parameters in a file to later be loaded and processed.
The train function performs the forward pass, computes the losses, computes the gradient and performs the step. Since some of our model needed to be modified by applying the β-lasso, we decided to assign the SGD as optimizer for all cases, and then apply this further step only in case the β is provided as argument to the function.

Model	Training Method	CIFAR 10	CIFAR 100	SVHN
S_{conv}	SGD	64%	34%	87%
S_{local}	SGD	64%	32%	87%
3_{FC}	SGD	36%	12%	58%
S_{FC}	SGD	58%	28%	80%
S_{FC}	β-lasso ($\beta = 0$)	57%	28%	84%
S_{FC}	β-lasso ($\beta = 1$)	59%	29%	82%
S_{FC}	β-lasso ($\beta = 50$)	55%	29%	81%

Table I: Comparing the performance of different models

IV. Results

Our results in Tab. I, Fig. 1, Fig. 2 mirror Figure 3,4 , Table 2 of the paper.
In Fig. 1, we have plotted the number of non-zero elements in different layers of fully connected net with β-lasso. Compared to the counterpart in the paper we do not see an abrupt change (in orders of magnitudes) and the reason is that we are only training for 100 epochs and not 4000. Had we continued our simulation, we would have obtained the same results. However, the results for the CIFAR 100 dataset are very close to the ones in the paper. Indeed, the plots suggest that β-lasso encourages the weights of the first and second layers to be much sparser but the last layer remains dense.

In Fig. 2, we plot the first layer filters of the S_{local} and S_{FC} trained with SGD, β-lasso. We see that training using S_{FC} with SGD lead to filters very dense and locally correlated whilst training with β-lasso lead to sparser and more locally connected filters in a similar fashion to S_{local}. This can be seen particularly using CIFAR100 data set. This promising result confirms that employing a learning algorithm with a stronger inductive bias allows for the development of an architecture with minimal initial bias, which can then be effectively learned from the data.

In Tab. I, we compare the performance of models on different models on data set. Although we expectedly do not reach the same performance as in the paper due to parameter modification, we can nonetheless conclude the results of the paper. That there is a significant improvement of performance of S_{FC} trained with β-lasso over fully-connected networks allowing fully connected networks to reach performances almost as good as the ones of locally connected and convolutionnaly learned architectures.

The saved models for our simulations are available but because of their immense size se did not include them in the final submission. They can be submitted upon request.

V. Proposals for future research

The paper advocates for the exploration of sparse matrices within fully connected layers.
Compared to Convolutional Neural Networks (CNNs) the multitude of parameters and weights in these layers produce computational challenges leading to the necessity for innovative approaches to alleviate their costs.
CNNs leverage convolutional operations to extract information by discerning similarities with neighboring elements, reducing the inter-neurons connectivity. Inspired by CNNs, future research should aim to extend the principles that underlie their success and efficiency.
Consequently, researchers can delve into alternative operations.

The application of β-lasso serves as an exemplification of this approach, showcasing potential performance improvements supported by experimental results.
Sparse matrices are an intriguing avenue for mitigating computational costs in machine learning algorithms. Researchers should focus on adapting models and architectures to support them to accelerate execution while maintaining or enhancing their efficiency.

VI. Summary

This work enabled a precise delineation of the effectiveness of sparse matrices in mitigating computational costs in machine learning algorithms, affirming the assertions made in the reference publication.
This study, conducted with constrained resources, adeptly replicated and evaluated the objectives of the original research in its entirety.

References

[1] B. Neyshabur, "Towards learning convolutions from scratch," 2020.