EE-411: Fundamentals of inference and learning **2021-2022**

Paper 5: Towards Learning Convolutions from Scratch

Lecturer: Florent Krzakala *Students: Seyed Khashayar Najafi, Soroush Mehdi, Gojko Cutura*

DISCLAIMER: This is our second (**bonus**) project

Code: https://github.com/gojkoc54/convolutions_from_scratch

***Abstract*—In this paper, a new approach has been proposed to reduce the effect of experts bias. Since this goal requires learning convolution-like structures from scratch. This, however, means bigger models, higher computational loads and bigger networks. To achieve this goal, the author proposes the investigation of minimum description length as a guiding principle and shows that in some settings, it can be indicative of the performance of architectures. in order to identify architectures with small description length, they proposed β-LASSO, a simple variant of LASSO algorithm that, when applied on fully-connected networks for image classification tasks, learns architectures with local connections and bridges the gap between fully-connected and convolutional nets.**

I. Introduction

Expert biases has been one of the concerns of learning algorithms and many have proposed methods to reduce these biases and their effects. Many have introduced general-purpose models to remove this sort of bias. However, in certain cases, these general-purpose models learn similar biases to the ones present in the traditional, expert-designed tools.

In order to decrease this induced bias, we need more data, more computation, and larger models (like using fully-connected nets instead of convolutional netowrks). This highlights the need of a method to reduce the bias while maintaining the efficiency and just keeping only the core bias which enables high performance.

In this paper the author introduces shallow (S-CONV) and deep (D-CONV) all-convolutional networks with desirable properties for studying convolutions. Furthermore, Minimum Description Length (MDL) is studied as a guiding principle to what architectures generalize better. At last, he proposes a training algorithm β-LASSO, a variant of LASSO with a more aggressive soft-thresholding, to find architectures with few parameters and hence, a small description length.

In this project we tried recreating some of the findings of the author and have a more critical view of the proposed methods. However, since this project was done in addition to the previous one, we were not able to study it as thorough as the previous one.

II. Models and methods

A. Convolutional architectures

In the paper, D-CONV and S-CONV (deep and shallow) are two all-convolutional neural networks that perform relatively well on image classification applications. Moreover, it is possible to scale the network with respect to the number of channels in the convolutional network and input image size. As shown in the left panel of Figure. 1, D-CONV introduced by the paper, has 8 convolutional layers and two fully-connected layers in the tail. In contrast, S-CONV, the shallow sibling of D-CONV, has only one convolutional layer followed by two fully-connected layers.

B. Locally-connected architectures

Convolution is a designed bias which expresses local connectivity and weight sharing. A convolutional network, when it is locally-connected, features the same connectivity, but eliminates weight sharing. The corresponding fully-connected network then adds connections between all nodes in adjacent layers with weight sharing. When it comes to this paper, the corresponding locally-connected models (S-LOCAL and D-LOCAL) are designed completely analogously to their CNN counterparts, just without weight sharing, i.e. there is a separate learnable filter for each of the output positions/pixels, ultimately leading them to have more learnable parameters.

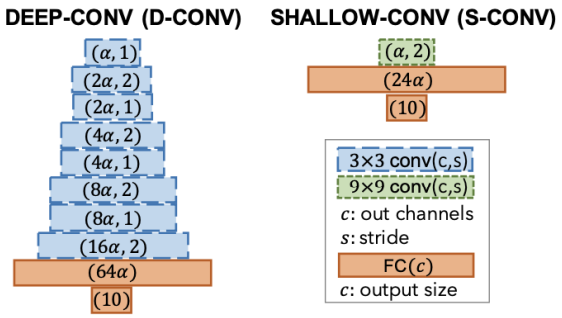

Figure 1: D-CONV and S-CONV architectures

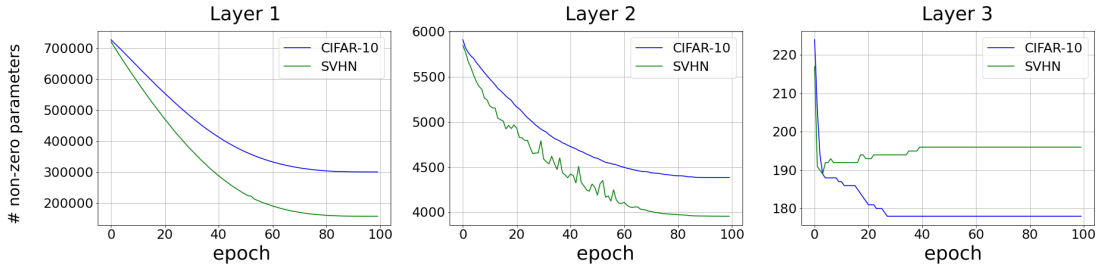

Figure 2: Number of non-zero parameters in each layer separately, for the S-FC model

C. Fully-connected architectures

The main focus of the paper are in fact the fully-connected versions of these baseline models. Namely, these are basic multi-layer perceptrons, that take in a flattened version of the input image and perform their classification. As stated before, the idea is to show that, when properly 'guided' by the optimization algorithms, these simple networks can perform as well as their convolutional and locally-connected counterparts.

D. β-LASSO Algorithm

Architectures with the least number of parameters are the ones that reveive most focus in this paper. The author the β-LASSO algorithm, which is a simple algorithm that encourages the sparsity by adding an $l1$ regularizer. The only difference compared to the conventional LASSO approach is that they add a β coefficient term, which is used to control the soft thresholding of the parameters during training, explicitly leading towards more sparse tensors and matrices. The algorithm can be seen in table I.

β-LASSO Algorithm
Parameters: f(θ): stochastic objective function with parameters θ, θ_0: the initial parameter vector, λ: coefficient of $l1$ regularizer, β: threshold coefficient, η: learning rate, τ: number of updates
1. for t=1 to τ do
2. $\theta_t \leftarrow \theta_{t-1} - \eta(\nabla_\theta f(\theta_{t-1}) + \lambda sign(\theta_{t-1}))$
3. $\theta_t \leftarrow \theta_t(\lvert\theta_t\rvert \geq \beta\lambda)$
4. Return θ_t

Table I: β-LASSO Algorithm

III. Recreating the results

In our efforts to reconstruct the results of this paper, we focused on 3 particular experiments, that analyze the following:

1) The effect of β-LASSO on the performance of S-FC for image classification (Table 2 in the original paper)
2) Number of non-zero parameters in different layers of S-FC trained with β-LASSO (Figure 3 in the original paper)
3) Spatial distribution of first layer filters of S-FC trained with and without β-LASSO (Figure 4 in the original paper)

In further subsections, we explain the training setup that we used for all experiments (hyperparameters, learning methods, ...) and finally the results of each experiment.

A. Training setup

We train all of our networks on an Nvidia Tesla V100 GPU, and since we didn't have too much time for this bonus project, and wanted to avoid a big electricity bill, we trained the models for 400 epochs (experiments reported in II and 3)and 100 epochs (2). Our goal wasn't to necessarily achieve the numbers reported in the original paper, but to analyze if the trend of these experiments can be captured even with less training.

First of all, the number of base channels α is fixed to 1. All models were trained using the Cosine Annealing learning rate scheduler with initial learning rate being 0.1. The batch size was 512 throughout the project, and we didn't use data augmentation (unlike the original paper). For experiments using the SGD optimizer, we didn't use momentum, weight decay, nor Nesterov acceleration, in order to keep it as raw as possible and analyze only the effect of the proposed algorithm. When it comes to β-LASSO, the regularization term λ was set to 2×10^{-5}, and β for recreating Figure 3 (2) was set to 50.

B. The effect of β-LASSO on the performance of S-FC

In this experiment, we try to recreate just a small part of Table 2 in the original paper. We only focus on the S-FC architecture trained on CIFAR-10 and neglect the others (the related work and S-LOCAL and S-CONV). The results are shown in table II. As we can see, changing the optimization algorithm from SGD to β-LASSO does in fact increase the performance of the S-FC model! However, we also notice that increasing β too much (to 50) starts to negatively affect the model, which is not the case in the original paper! More on that in the discussion section.

Model	Training Method	CIFAR-10
MLP (S-FC)	SGD	45.8
MLP (S-FC)	β-LASSO ($\beta = 0$)	49.6
MLP (S-FC)	β-LASSO ($\beta = 1$)	**51.2**
MLP (S-FC)	β-LASSO ($\beta = 50$)	48.9

Table II: table 2

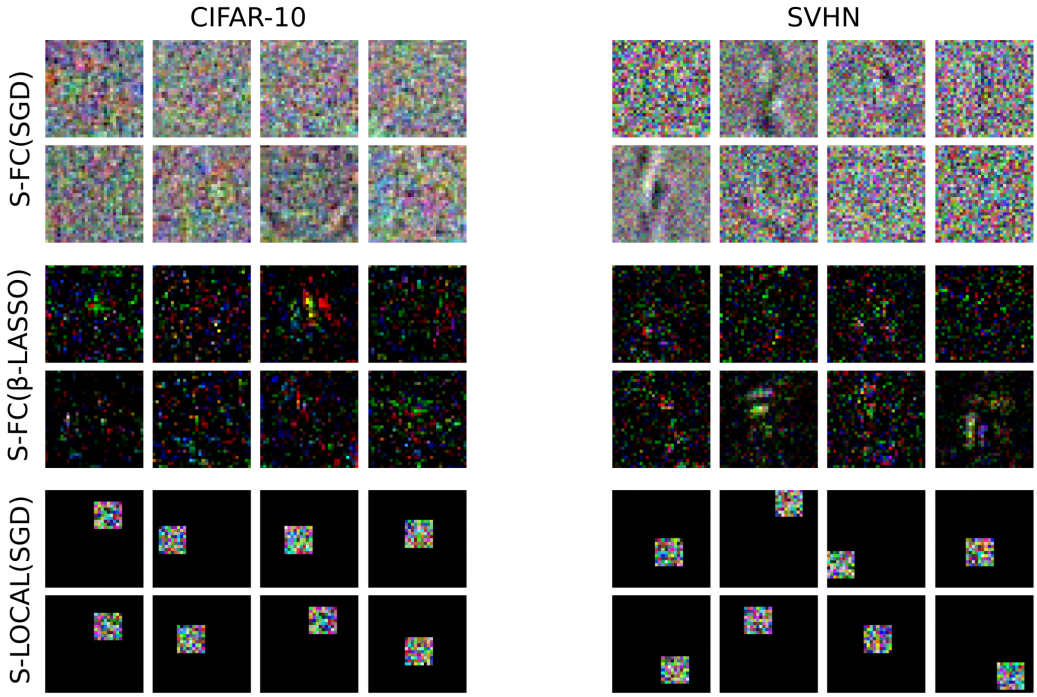

Figure 3: Comparison of first layer filters of S-FC trained with SGD and β-LASSO and S-LOCAL trained with SGD

C. Number of non-zero parameters in different layers

In this section, we try to investigate how do the parameters change over time (i.e. across epochs). More particularly, we measure the number of non-zero parameters in each layer of S-FC separately, over 100 epochs, for two datasets - CIFAR-10 and SVHN. The results are satisfying (reported in figure 2), since they imply that the first layer is constantly getting sparser and sparser, as well as the second layer (but a bit more mildly), whereas the third layer graph saturates pretty quickly and stays dense (relative to its size). However, what we don't manage to explain is this big difference in values on the y axis compared to the original paper. For example, their third layer values go all the way up to 10^6, even when we are certain that the layer has $24c\alpha = 240$ parameters, which is explicitly shown in the paper itself (Table 4).

D. Spatial distribution of first layer filters

We finally reach the experiment that is probably the most helpful in order to easily grasp and interpret the effect of β-LASSO. Namely, in figure 3 we visualize the learned filters of the first layer, for S-FC trained with SGD and β-LASSO, as well as S-LOCAL trained with SG. As we can see, the filters learned by S-FC using the SGD algorithm are completely dense, but sometimes showing a tendency to be locally-correlated (more noticeable in the plots corresponding to SVHN), but not as much as in the original paper (again, probably due to a smaller number of epochs). On the other hand, the same network, this time trained with β-LASSO, ends up learning completely different filters - they are sparse show more locality overall.

IV. Summary

In this bonus project of ours, we tried to cover as many interesting aspects of the original β-LASSO paper as possible. We did succeed in some of them, some were fails, but overall we were able the confirm what the author was proposing - with a proper optimization technique, such as β-LASSO, one can guide even a fully-connected network to learn sparse and locally-focused filters, leading them to perform well on image-classification tasks.

Some takeaways (or unanswered questions that could use additional focus) would be:

- In table II we noticed that $\beta = 50$ worsened the performance of the model, compared to smaller values of the parameter. Why does this happen? Can it be improved with more training and augmentations?
- When it comes to figure 2, we notice a big mismatch between our ranges of the y axis (number of non-zero parameters) of layers 2 and 3 compared to the ones in the original paper, but are (for now) confident that our ranges do make sense.
- The sparsity of filters in 3 was successfully recreated, but their locality was much more obvious in the original paper (even though it's noticeable in our project as well).