ISSN 2281-4299

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Computational issues in Optimization for Deep networks

Corrado Coppola
Lorenzo Papa
Marco Boresta
Irene Amerini
Laura Palagi

Technical Report n. 06, 2022

Computational issues in Optimization for Deep networks

Corrado Coppola[1], Lorenzo Papa[1], Marco Boresta[2], Irene Amerini[1], and Laura Palagi[1]

[1] *Sapienza University of Rome, Department of Computer, Control, and Management Engineering Antonio Ruberti - Via Ariosto 25 - Roma*

[2] *Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti" - CNR - Via dei Taurini - Roma*

Abstract

In this paper, we aim at investigating some relevant computational issues, in particular the role of global vs local optimizers, the role of the choice of the starting point, as well as of the optimization hyper-parameters setting on classification performances of deep networks. We carry out extensive computational experiments using nine different open-source optimization algorithms to train a deep CNN on an image classification task. Eventually, we also assess the role of wideness and depth on the computational optimization performance, carrying out additional tests with wider and deeper neural architecture.

Keywords: optimization, deep network, machine learning performances

1 Introduction

Machine Learning is one of the most important fields of Artificial Intelligence, aiming at the development of complex systems capable of learning data-structures or behavioural patterns from a given set of samples. Machine Learning algorithms are mainly based on optimization techniques, i.e. the system is trained by solving a certain optimization problem. In most of the real-world applications, as well as in the case we will study in this paper, the system is a neural network with real weights trained by minimizing a given differentiable loss function, i.e. a function measuring, in a single dataset, the dissimilarity between the given target values and the values returned by the network. The training process consists, indeed, in minimizing the loss function with respect to the network weights, which can result in a complex large-scale problem.

The crucial role played by optimization algorithms and theories in Machine Learning, acknowledged since the birth of this research field, has been widely and deeply discussed in literature, both from an Operations Research and from a Computer Science perspective. While the underlying idea of stochastic gradient-like methods, proposed by Robbins et al. [1], dates back to 1950s and its theoretical and computational properties have been deeply investigated by the research community, an incredibly vast amount of new algorithms has been developed in the past two decades. Several attempts have been made in the scientific literature to assess the efficiency, effectiveness, theoretical guarantees, and scaling capability of optimization algorithms applied to Machine Learning problems, which are frequently characterized by highly nonlinear and non-convex objective functions. For instance, concerning classification tasks, more than 20 years ago Lim et al. [2] carried out a deep comparative analysis of almost all the algorithms available at the time to determine how different types of data and tuning of algorithms parameters influenced performances. As Machine Learning, in particular Deep Learning, gained a permanently increasing interest in the community, this type of comparative analysis scheme has been adopted more directly and exclusively to specific methods and to different type of neural architectures. While more recently Pouyanfar et al. [3] and Brayek et al. [4] provided methodological surveys on different approaches to ML problems, in the same years optimization methods have been widely studied from an Operations Research perspective. Bottou et al. [5] studied different first-order optimization algorithms applied to large-scale Machine Learning problems, while Baumann et al. [6] produced a thorough comparative analysis of first-order methods in a Machine Learning framework and traditional combinatorial methods on the same classification tasks. Following the increased need for a high-level overview, some other works on first-order methods have been published by Lan [7], which provides a detailed survey on stochastic optimization algorithms, and by Sun et al. [8], which compare from a theoretical perspective the main advantages and drawbacks of some of the most used methods in Machine Learning. Recent studies have also been focused on the Generative Adversarial Network (GAN) paradigm, such as the surveys Aggarwal et al. [9], Creswell et al. [10], Wang et al. [11]. Eventually, following the increase, both in dimension and in quantity, of available data being processed by neural architectures, sorting methods gained an increasingly relevant role in a variety of data science applications, as reported in [12, 13].

However, in spite of all the studies that have already been carried out, to the best of our knowledge only few tried to answer to some relevant computational questions when using optimization methods to train DNs. As reported in the recent survey [14], optimization of DN is quite complicated and can be decomposed into several steps, which includes convergence to stationary points, rate of convergence, achieving low local solution (aka global minimizers). Further, in ML the quality of the solution is evaluated on the basis of the test accuracy rather than on the quality of the solution obtained by the optimization. The overfitting phenomenon leads to favouring solutions when the training accuracy is away from zero, using early stopping rules and with the implicit claim that reaching a global solution is not worthwhile.

In this paper we aim at assessing the influence of structural, algorithmic and architectural choices on the test performances. In particular, we consider optimization algorithms as implemented in standard libraries and we address the following computational issues:

- convergence to local versus global minimizers
- role of the starting point
- role of optimization hyper-parameters setting on the test performance
- role of wideness and depth on the computational optimization performance

We are not interested in the theoretical aspects but in the impact of the settings of the optimization open-source algorithms both on the computational aspects when tackling large scale training problems and on the

effectiveness of final classification model. Most of the papers in the literature are devoted to the theoretical aspects. Sun [14], for instance, investigates the problem of choosing the best initialization of parameters and the best performing algorithms for a given dataset, naming this issue Global Optimization of the Network. He is also one of the few, jointly with Palagi [15], to have remarked the importance of reaching a globally optimal solution when training a neural network, highlighting how finding a global optimum leads also to better test performances. Im et al. [16] use a loss function projection mechanism to analyze how optimization algorithms can find very different solutions to the same problem; they investigate algorithms behaviour in the neighborhood of saddle points and they find that most of the first-order methods produce the same loss function shape regardless the choice of hyper-parameters (momentum coefficient, learning rate etc.). We consider their work particularly valuable for our research since their computational experience appears to confirm that algorithms may react in a significantly different manner to the presence of saddle points, leading to the most disparate performances, depending on each problem peculiarities. Xu et al. [17] are amongst the first to point out the problem of the robustness to hyper-parameters; they discuss how traditional first-order methods can get stuck in bad local minima or saddle points when tackling non-convex ML problems, and how computational results can also depend on the hyper-parameters setting. They propose a novel approach to approximated second order methods, which are proved to perform better in terms of robustness to hyper-parameters. The importance of an accurate choice of the optimization hyper-parameters is also highlighted by Chauhan et al. [18] in their research on neural models for classification of COVID-19 cases. Furthermore, selecting a robust hyper-parameters setting can prevent or reduce the performance degradation issue, which has been widely investigated by Young et al. [19]. Jais et al. [20] carry out a thorough analysis of Adam algorithm performance on a classification problem, focusing on optimizing the network structure as well as Adam parameters; however, their work is limited to Adam algorithm and focused more on the algorithm response to architectural changes than on its inherent properties influencing the behavior. Even more recently, Ding et al. [21] provided a detailed mathematical proof on the existence of suboptimal local minima for deep neural networks with smooth activation, showing how it is practically and theoretically impossible to avoid the problem of bad local minima in Deep Learning, which points at our main objective, that is analyzing how the quality of the convergence point is influenced by structural choices.

In this paper, we consider an open-source dataset to train a convolutional neural network (CNN) for an image classification task. Developed and formalized by Le Cun et al. [22], CNNs are one of the most widespread type of neural network for image processing, e.g. image recognition and classification [23], monocular depth estimation [24], semantic segmentation [25], video recognition [26]. We train the network using several optimization algorithms as implemented in standard libraries for optimization and ML.

We show that not all the algorithms reach a neighborhood of the global optimum, getting stuck in local minima. We also found that test performance, i.e. the classification test accuracy, is remarkably higher when a good approximation of the global solution is reached. Achieving a better solution can be obtained by carefully choosing the optimization hyper-parameters setting. The paper is organized as follows. In section 2 we describe the network architecture, while in section 3 we formalize the optimization problem behind the image classification task, mathematically describing the convolution operation performed by the network layers. In section 4 we briefly describe each of the nine different algorithms we have tested on our task and in section 5 we describe the composition of our open-source dataset. Eventually, in section 6, we explain in details which kind of tests have been carried out on the network and why our computational experience appears to confirm our initial hypothesis concerning the importance of global optimization and the role of hyper-parameters setting. Eventually, we present our conclusions in section 7.

2 The Network Architecture

Convolutional neural networks (CNNs), a special type of Deep Neural network (DNN) architecture, achieved excellent results in most vision, speech, and image processing tasks [27, 28, 29]. The proposed work is focused on a multi-class classification task, a predictive modelling problem where a class, among the set of classes, is predicted for a given input data. To tackle the problem, on two-dimensional input-image data, well-known Deep CNN models such as ResNet [30] MobileNet [31] and DenseNet [32] have been developed. Those architectures are designed to progressively downsample the input image, extracting features at multiple different scales and providing a compact set of high-level features as output.

In this paper, to compare the different optimizers' performance over CNN architectures, we design a lightweight low-complexity baseline model composed of five cascaded Convolutional Downsampling Blocks

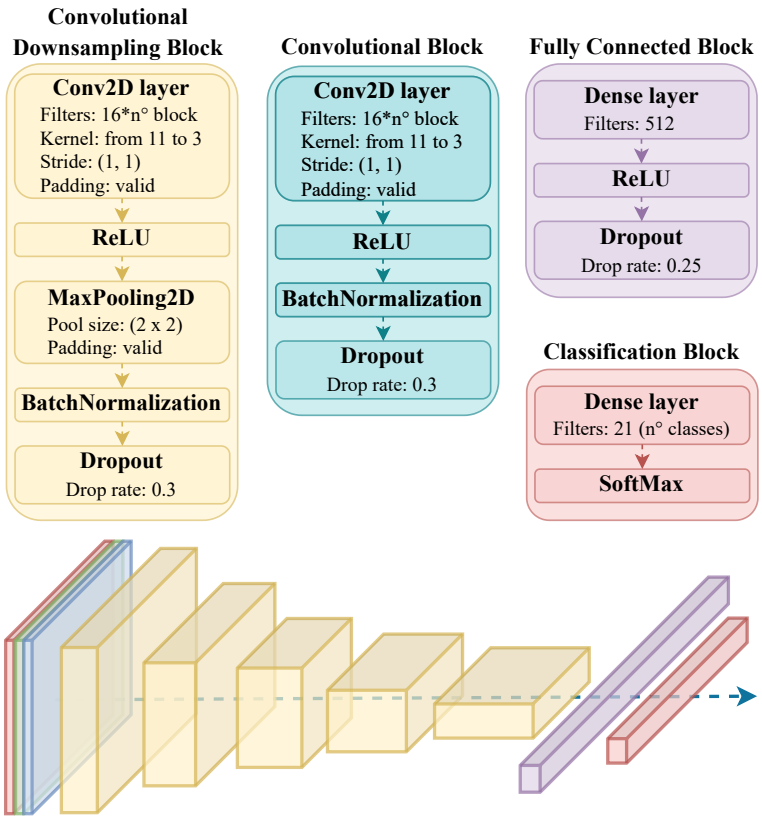


Figure 1: Overview of the baseline model and corresponding blocks used for the study.

(CDBs) followed by a Fully Connected Block (FCB) and a Classification (CB) one used for the final prediction. A graphical representation of the developed reference model and a detailed block diagram representation, with layers operations and respective parameters, are reported in Figure 1 while an overview of the input-output shapes of the reference model and respective number of trainable parameters is reported in Table 1.

The CDB block is composed of five operations. The first three are used for the feature extraction, and they are: a standard 2D-convolution, where its filters dimension and kernel window are respectively increased and reduced by a factor of 2 each time the spatial feature dimensions (height and width) decrease, i.e. from 16 filters with an 11×11 window applied to the input image to 32 filters with a 9×9 window after the pooling operation, followed by the ReLU as activation function, and a 2D-maxpooling operation to downsample the extracted features along its spatial dimensions by taking the maximum value over the 2×2 input window. Differently, the other two operations, i.e. a batch normalization and a dropout with a drop rate equal to 0.3, have been applied to improve both the training process and the model generalization capability while avoiding the overfitting. The proposed architecture, in its simple design and with a limited number of computed operations ($2.76M$ of trainable parameters), is ideal for the proposed analysis.

Operation	Input Shape [H, W, C]	Output Shape [H, W, C]	N° Param. [K]
CDB_1	(256, 256, 3)	(123, 123, 16)	5.9
CDB_2	(123, 123, 16)	(57, 57, 32)	42.6
CDB_3	(57, 57, 32)	(25, 25, 64)	100.7
CDB_4	(25, 25, 64)	(10, 10, 128)	205.4
CDB_5	(10, 10, 128)	(4, 4, 256)	296.2
Flatten	(4, 4, 256)	(1, 1, 4096)	0
FCB	(1, 1, 4096)	(1, 1, 512)	2097.6
CB	(1, 1, 512)	(1, 1, 21)	10.8

Table 1: Structure of the proposed baseline architecture with respective input-output spatial dimensions, reported in the [H, W, C] format, and number of trainable parameters (N° Param.).

Moreover, we designed three variants of the baseline architecture called WIDE, DEEP and WIDE-DEEP needed for the respective analysis, i.e. the WIDE-test, DEEP-test and WIDE&DEEP-test, with the following variations:

- WIDE: it is designed doubling the number of convolutional filters of the baseline model. The final architecture has $6.78M$ of trainable parameters.
- DEEP: it is designed doubling the number of convolutional operations, adding at each CDB a Convolutional Block reported in Figure 1. The latter block has been developed with the same behaviour as the CDB by removing the downsampling step computed by the 2D-maxpooling layer. The final architecture has $10.36M$ of trainable parameters.
- WIDE&DEEP: it is designed merging the previous WIDE and DEEP structures, i.e. doubling both convolutional filters and operations. The final architecture has $24.63M$ of trainable parameters.

3 The optimization problem

As mentioned above, a single CNN layer performs the steps of convolution, non-linear activation and pooling. In the CNN framework, the CNN layers are stacked together until the high-level features are obtained; the output of the final convolutional layer is then flattened to form a one-dimensional feature vector, which is then fed into the fully-connected network, which estimates the likelihood of each output class appearing in the image. The CNN takes in input a level representation of image $x^c \in \mathbb{R}^{m \times m}$ for each channel c is the number of channels ($c = 1$ for B/W image and $c = 3$ for RGB images).

The convolutional layer is made of k feature maps, f, that are calculated by taking the dot product between the k-th filter w_c^k of size $n \times n$ for each channel c.

We denote with $f \in \mathbb{R}^{(m-n-1)\times(m-n-1)}$ the feature map associated to the k-th filter, that can be computed as:

$$f_{ij}^k = \sigma\Big(\sum_c w_c^k * x_c\Big) \tag{1}$$

where σ indicates a nonlinear activation function (ReLU in our experiments) and the $*$ operator denotes the discrete convolution operation, as reported in [33] Chapter 9, between the filter and the input feature x. The max-pooling operation has the aim of reducing the dimensionality of the convolutional layer. To achieve this, the maximum of the feature maps over a limited non-overlapping spatial region of dimension $p \times p$ is computed. We denote by $g \in \mathbb{R}^{[(m-n-1)/p]\times[(m-n-1)/p]}$ the pooling layer of the k-th filter. We call $P_{(i,j)}$ the set of positions (r, c), i.e. rows and columns of the input feature, that are located in this region, centered in (i, j). Thus, the output of the Max-Pool layer is computed as:

$$g_{ij}^k = \max_{(r,c)\in P_{(i,j)}} f_{r,c}^k \tag{2}$$

To carry out our computational tests, we have set $p = 2$.

Another potential and common pooling operation is the *average-pooling*, the difference from (1) lying in the *mean* operation instead of the *max*.

The output of the CNN is then sent into a Feed-forward Neural Network made up of L fully connected layers, where we call N_l and σ_l the number of neural units and the activation function of the l-th layer. Given the input of the l-th layer x_{l-1}, the weights matrix $W_l \in \mathbb{R}^{N_{l-1}\times N_l}$ and tha bias vector $b_l \in \mathbb{R}^{N_l}$, the output x_l is defined by the following relation:

$$x_l = \sigma_l(W^T x_{l-1} + b) \tag{3}$$

We implemented the proposed study using TensorFlow 2[1] deep learning high-level API. We set the environment seed (also for the normal initializer of the convolutional kernels) at a randomly chosen value equal to 1699806 or to a specific list[2] of values in the multi-start analysis.

For the given task, we chose the Categorical cross entropy (CCE) as loss function, as implemented in TensorFlow[3]. Its mathematical formulation is reported in Equation 4 where N is the number of possible classes

[1] https://www.tensorflow.org/

[2] $[0, 100, 500, 1000, 1500, 10000, 15000, 100000, 150000, 1000000, 1500000, 1699806]$

[3] `https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy`

(i.e. 21), P the number of samples, ω the weight vector of the whole network, y_i^j the target class returned by the network fro sample j and $\hat{y}_i^j(\omega)$ the predicted one. The minus sign ensures that the loss gets smaller when the distributions get closer to each other.

$$f(\omega) = -\sum_{j=1}^{P}\sum_{i=1}^{N} y_i^j \log(\hat{y}_i^j(\omega)) \tag{4}$$

For the purpose of simplicity, we will further omit the dependency of $\hat{y}_i^j$ from all the network parameters ω, and we will thus refer to the predicted class as $\hat{y}_i^j$.

4 Optimization Algorithms

In this work, L-BFGS and eight optimization algorithms with multiple hyper-parameters setups are compared. Precisely, those are: Adam [34], Adamax [34], Nadam [35], RMSprop[4], SGD [1, 5, 36, 37], Ftrl [38], Adagrad [39], and Adadelta [40]. However, we have focused on five of these optimizers since, as we will see in section 6, they achieved the highest accuracy prediction. We further report a detailed list of all of them with the respective hyper-parameters values, where the default values are taken from the TensorFlow (respective paper) documentation. For the sake of completeness, we also succinctly describe the updating rule of each algorithm, assuming it is applied to the objective function $f(\omega) : \mathbf{R}^h \to \mathbf{R}$, which either is continuously differentiable or has a finite number of non-differentiable points, that are not in the sequence of points returned by the algorithm. Furthermore, provided that these algorithms are used to optimize a loss function on a given dataset, we also assume $f(\omega) = \sum_{p=1}^{P} f_p(\omega)$, being P the number of samples in the dataset.

4.1 L-BFGS

Being one of the best known first-order method with strong convergence properties (see [41]), L-BFGS belongs to the limited memory quasi-Newton methods class. This algorithm is purely deterministic and, at every iteration k, exploits an approximate of the inverse Hessian of the objective function. The updating rule implemented in TensorFlow has been formalized by Nocedal et al. [42]. Given $s_k = \omega_{k-1} - \omega_k$, $y_k = \nabla f(\omega_{k+1}) - \nabla f(\omega_k)$, we define $\rho_k = \frac{1}{y_k^T s_k}$ and $V_k = I - \rho_k y_k s_k^T$, where I is the identity matrix. Given an initial approximate Hessian $H_0 \sim \left[\nabla^2 f(\omega_0)\right]^{-1}$, the algorithm uses the rule:

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T$$

And performs the following update scheme:

$$\omega_{k+1} = \omega_k - \eta_k H_k \nabla f(\omega_k)$$

Where η_k is either the learning rate or, more generally, the step size obtained via some linesearch method.

Differently from the other eight algorithms, L-BFGS is not amongst the most widespread optimizers used in Machine Learning. However, in force of its strong convergence properties, this algorithm is recently gaining an increasing interest in the research community. Some multi-batch versions of L-BFGS have been proposed in the past years in [43, 44, 45], in particular for image processing tasks in medicine in [46, 47].

Since L-BFGS is not available in TensorFlow library, we have used the SciPy[5] version implemented with an open-source wrapper available online[6].

4.2 SGD

The Stochastic Gradient Descend is the basic algorithm to perform the minimization of the objective function using an estimate of its gradient. The update rule is:

$$\omega_{k+1} = \omega_k + \eta_k d_k + \beta_k(\omega_k - \omega_{k+1})$$

[4]RMSprop is an adaptive learning rate method devised by Geoff Hinton in one of his Coursera Class (`http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`) that is still unpublished

[5]https://scipy.org/

[6]https://gist.github.com/piyueh/tf_keras_tfp_lbfgs.py

Where d_k is the descent direction, in particular an approximate of $-\nabla f(\omega_k)$. In the TensorFlow implementation the following mini-batch approximation is used:

$$d_k = -\frac{1}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla f_i(\omega_k) \text{ for some } \mathcal{B}_k \subset \{1, \dots, P\}$$

Furthermore, β_k is called momentum term and represents an extrapolation along the difference of the two past iterations. TensorFlow also allows to tune a boolean parameter, called Nesterov, which enables the Nesterov acceleration step (see [48]), i.e. the computation of $z_k = \omega_k + \beta_k(\omega_k - \omega_{k+1})$ is performed in a first sub-step, and then it is set $\omega_{k+1} = z_k + d_k$.

4.3 Adam

Adam is one of the first SGD extensions, where the gradient estimate is enhanced with the use of an exponential moving average according to two coefficients: β_1 and β_2. The index $i \in \{1, 2\}$ is referred to the moment of the stochastic gradient, i.e., the first moment (expected value) and the second moment (non-centered variance). Being g_k the same approximate $\nabla f(\omega_k)$ used in SGD, i.e., $g_k = -\frac{1}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla f_i(\omega_k)$, we define the following first and second moment estimators at iterate k:

$$m_k \sim \mathbb{E}\left[\nabla f(\omega_k)\right] = (1 - \beta_1) \sum_{i=1}^{k} \beta_1^{k-i} g_i$$

$$v_k \sim \mathbb{E}^2\left[\nabla f(\omega_k)\right] = (1 - \beta_2) \sum_{i=1}^{k} \beta_2^{k-i} g_i^2$$

Being $\hat{m}_k = \frac{m_k}{1-\beta_1^k}$ and $\hat{v}_k = \frac{v_k}{1-\beta_2^k}$, and given $\epsilon > 0$ the updating rule is the following:

$$\omega_{k+1} = \omega_k - \eta_k \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}$$

4.4 Adamax

Adamax performs mainly the same operations described in Adam, but it does not make use of the parameter ϵ, and the algorithm exploits the infinite norm to average the gradient. Recalling the same notation explained above, we can define $u_k = \lim_{p \to \infty} (v_t)^{\frac{1}{p}} = \max_{i=1,\dots,k} \beta_2^{k-i} |g_i|$. Thus, the updating rule is:

$$\omega_{k+1} = \omega_k - \eta_k \frac{m_k}{u_k(1 - \beta_1^k)}$$

4.5 Nadam

Nadam, also known as Nesterov-Adam, performs the same updating rule of Adam, but employs the Nesterov acceleration step, meaning that there are no mathematical differences with Adam. Nadam is expected to be more efficient, but Nesterov trick involves only the order with which operation are carried out and not the updating formula.

4.6 Adagrad

Adagrad is the first Adam extension which makes use of adaptive learning rates to discriminate more informative and rare features. The general update rule of $\omega_k \in \mathbf{R}^h$ involves complex matrix operations, for which we need to introduce some other notation.

Given a vector $a \in \mathbf{R}^h$, we define the matrix $A = \text{diag}(a) \in \mathbf{R}^{h \times h}$, such that $A_{ii} = a_i$ for $i = 1, \dots, h$ and $A_{ij} = 0 \ \forall i \neq j$. Furthermore, given two vectors $a, b \in \mathbf{R}^h$, we refer to their component-wise product as $a \odot b \in \mathbf{R}^h$, such that $[a \odot b]_i = a_i b_i$ for $i = 1, \dots, h$.

At iteration k we introduce the vector $G_k = \sum_{t=0}^{k-1} (\nabla f(\omega_t) \odot \nabla f(\omega_t))$. Given $\varepsilon > 0$ and the identity matrix I, we define the following matrix:

$$H_k(\varepsilon) = I\varepsilon + \text{diag}(G_k)^{\frac{1}{2}}$$

Thus, the updating rule resulting after the minimization of a specific proximal function (see [39]) is the following:

$$\omega_{k+1} = \omega_k - \eta H_k(\varepsilon)^{-1} \nabla f(\omega_k)$$

4.7 RMSProp

Proposed by Hinton et al. in the unpublished lecture [49], RMSProp performs basically the same operations of Adagrad, but the update rule is modified in order to slow down the learning rate decrease. In particular, the second raw moment of the gradient is averaged using an exponential rule according to a given parameter $\beta \in (0, 1)$. Following the notation introduced in the last subsection, at iteration k we define the matrix $\tilde{G}_k = \sum_{t=0}^{k-1} \beta^{k-t}(1-\beta) (\nabla f(\omega_t) \odot \nabla f(\omega_t))$. Hence, given the following matrix:

$$\tilde{H}_k(\varepsilon) = I\varepsilon + \text{diag}(\tilde{G}_k)^{\frac{1}{2}}$$

The update rule is:

$$\omega_{k+1} = \omega_k - \eta \nabla f(\omega_k) \tilde{H}_k(\varepsilon)^{-1}$$

4.8 Adadelta

Adadelta can be considered as an extension of Adagrad, which allows a less rapid decrease of learning rate. Firstly, given a stochastic variable X, we introduce the following function:

$$RMS[X] = \sqrt{\hat{\mathbb{E}}[X^2] + \epsilon}$$

Where the hat is to indicate that the expected value is somehow approximated. Recalling that, given a mini-batch of variables $\mathcal{B}_k$, $g_k = \frac{1}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla f_i(\omega_k)$, we define $\Delta\omega = -\eta g$. Thus, we can write the Adadelta updating rule as follows:

$$\omega_{k+1} = \omega_k - \frac{RMS[\Delta\omega_{k-1}]}{RMS[g_k]} g_k$$

Where the multiplication is performed component-wise.

4.9 FTRL

FTRL (Follow The Regularized Leader) is a regularized version of SGD, which uses the L1 norm to perform the variables update. Given, at every iteration k, $d_k = \sum_{t=1}^{k} g_t$, and fixed the quantity σ_k such that $\sum_{t=0}^{k} \sigma_t = \frac{1}{\eta_k}$, the update rule is the following:

$$\omega_{k+1} = \arg\min_{\omega} \{ d_k^T \omega + \sum_{t=1}^{k} \sigma_t ||\omega - \omega_t||^2 + \lambda ||\omega||_1 \}$$

5 The data set

The benchmark dataset used to show the performance of the different optimization methods over the introduced convolutional architecture is the UC Merced [50]. It is a balanced dataset that comprises a total of 2100 land samples divided into 21 classes, i.e. 100 images per class, such as harbour, beach, and buildings. The dataset images have a resolution of 256×256 pixels, the same used in our study. The high number of classes and the limited number of samples for each class makes the multi-class classification a non-trivial task. Moreover, for the proposed study, we compare each optimization algorithm with two different setups: a standard one and an augmented one, where we apply both horizontal and vertical flips to the input images. This choice is due to understanding the different optimization behaviours and optimal hyper-parameters values at the increase of the training data.

Table 2: Default values of the TensorFlow built-in optimizers hyper-parameters

Algorithm	η	β	β_1	β_2	ϵ	Amsgrad	ρ	Centered	Nesterov
SGD	10^{-2}	0	-	-	-	-	-	-	False
Adam	10^{-3}	-	0.9	0.999	10^{-7}	False	-	-	-
Adamax	10^{-3}	-	0.9	0.999	10^{-7}	-	-	-	-
Nadam	10^{-3}	-	0.9	0.999	10^{-7}	-	-	-	-
RMSProp	10^{-2}	0	-	-	10^{-7}	-	0.9	False	-
Adadelta	10^{-3}	-	-	-	10^{-7}	-	0.95	-	-
Adagrad	10^{-3}	-	-	-	10^{-7}	-	0.95	-	-
FTRL	10^{-3}	0.1	0	0	-	-	-	-	-

Table 3: Test accuracy obtained with default values of the optimizers hyper-parameters with and without data augmentation. We remind the reader that data augmentation has not been with the three worst performing algorithms.

Algorithm	No Aug	Aug
Adadelta	17.6 %	-
Adagrad	32.7 %	-
Adam	60.0 %	72.4 %
Adamax	61.3 %	72.5 %
FTRL	4.6 %	-
Nadam	61.3%	72.1 %
RMSProp	60.2%	74 %
SGD	59.4 %	65.1 %

6 Computational Results

We have first tested L-BFGS and the eight optimizers described in section 4 as implemented in TensorFlow, setting the algorithm hyper-parameters to their default value. The results of this first testing phase are reported in subsection 6.1, jointly with multi-start test on the three worst-performing algorithm: Adadelta, Adagrad, FTLR. Furthermore, we carried out a grid search to find the optimal setting of the algorithms hyper-parameters and, in subsection 6.2, we show how this approach lead to significant improvements in terms of accuracy. Eventually, we modified the network architecture to assess the algorithms performances under three different configuration and in subsection 6.3 we report the results of this last testing phase.

Provided that L-BFGS is the only batch method, computational tests with the other eight optimizers have been conducted using a mini-batch size $bs = 128$. For the first experiments with default hyper-parameters in subsection 6.1 the network was trained setting to 100 the number of epochs over the whole dataset. We remark that a single epoch consists in $\frac{P}{bs}$ update steps, being P the number of samples in the dataset. For the other experiments, once the optimal hyper-parameters setting was found, we halved the number of epochs.

We eventually underline that the TensorFlow implementation of the eight built-in optimizers, as well as the SciPy version of L-BFGS, is based on finite differences approximate gradient.

6.1 Training with default hyper-parameters

The first study we carried out was aimed at defining the optimizers performances using their default hyper-parameters values, which we report in Table 2 for the purpose of completeness. In Figure 2 we report the trends of the losses with and without data augmentation and in Table 3 we report the accuracy values obtained. We noticed that, while most of the algorithms were converging to points in the neighborhood of the globally optimal solution, i.e. the loss was almost zero, Adadelta, Adagrad and FTRL got stuck in some local minima, as it can clearly be seen in Figure 2. This produced quite poor performances in terms of accuracy, therefore we did not even carry out the test with data augmentation for these algorithms. The observed behaviour seems to confirm what has been already pointed out in [51, 46]: neural networks can be affected by the local minima issue, which is not a merely mathematical problem, but has a direct influence on the performance metrics. In fact, in our case we see how the lack of convergence to the globally optimal solution implies an accuracy level that makes the entire network useless for the purpose of correctly classifying images. Furthermore, our loss shapes do not seem to depend on the dataset: data augmentation has no substantial effect in modifying the convergence endpoint. Adadelta, Adagrad and FTRL are somehow stable in returning the same locally optimal point, as well as SGD, Adam, Adamax, Nadam and RMSProp always converge to good solutions, leading to similar values of accuracy, as we can clearly see again in Table 3. Nonetheless, data augmentation seems to have

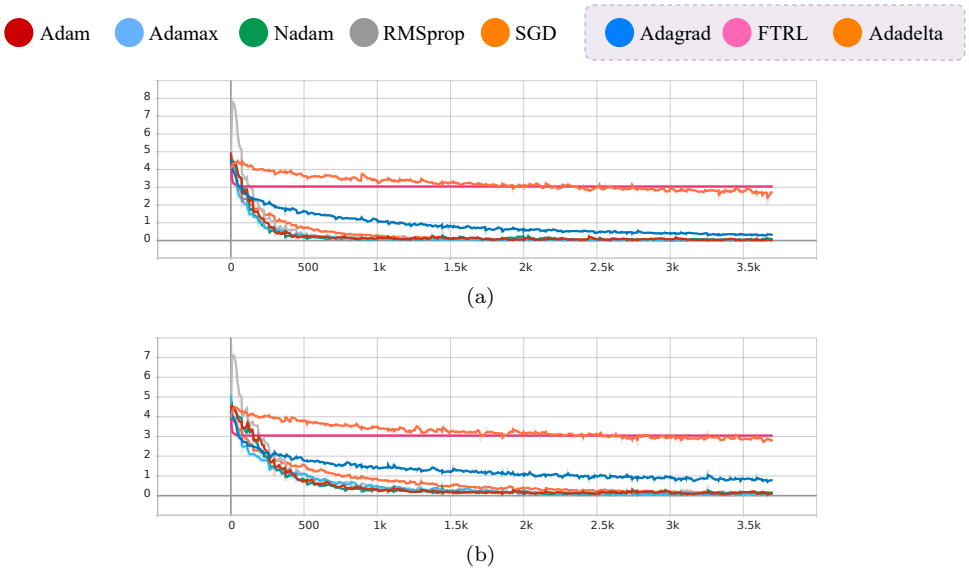


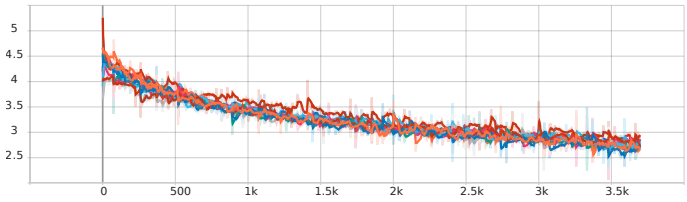
Figure 2: Optimizers comparison with default hyper-parameters values. One the left the losses trend without augmentation and on the right the one with the augmentation. We report the number of iterations on the X-axis against the loss function values on the Y-axis. A color is assigned to each algorithm according to the labels above the figure, and the three worst performing algorithms are inside the dashed box.

a boosting effect on accuracy for all the five working algorithms; we suppose that this improvement is obtained because, thanks to the data augmentation, the network collects more experience during the training process and this enhances its generalization capability.

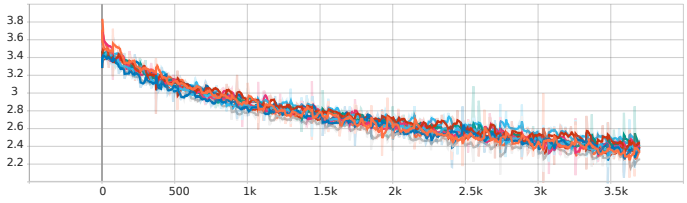
For the purpose of investigating the behaviour on Adadelta, Adagrad and FTRL and assessing the real stability of this phenomenon, we carried out another test on these algorithms, employing a multi-start framework. We used two different wights initialization distributions (Glorot Uniform and Lecun Normal) and we ran the algorithm with different seed values. However, as we can see in Figure 3, Figure 4,Figure 5, the loss shape is more or less always the same, even with different seeds. The three algorithms always get stuck in a locally optimal point. We do not report the test accuracy, as the values obtained are far below 20%. This suggest that Adagrad, Adadelta and FTRL do not fit with our task and that their behaviour is absolutely stable, not influenced by some other factors, such as the dataset dimension, the seed or the starting point. These algorithms converge only to local minima, and local minima are clearly not enough for our task.

Concerning L-BFGS, we have first trained the network with the same architecture reported in Figure 1 and we have obtained the loss trend reported in Figure 6(b). We observe that the algorithm often fails before achieving convergence, i.e. with some random initializations the line stops because at a given iteration the returned loss was infinite. We argue that this is caused by a non-differentiability issue. In fact, L-BFGS convergence is guaranteed only when the objective function is continuously differentiable (see [41]) and the ReLU as well as the MaxPooling operation (2) both cause the occurrence of non-differentiable points, i.e. points where the gradient is not defined. Hence, we also trained the network removing MaxPooling layers and using the continuously differentiable activation Swish proposed in [52, 53], and we obtained the loss trend in Figure 6(a). Although we did not encounter the non-differentiability issue anymore when using Swish, the loss did not tend to zero and, instead, L-BFGS generally converged to local minima with a far worse loss value compared to the other five working algorithms. Furthermore, L-BFGS performed quite poorly in terms of test accuracy, achieving only 18.6% when using the MaxPooling layers and the ReLu activation functions and 28.4% when removing the MaxPooling layers and replacing ReLu with Swish. We argue that this behaviour is motivated by the inherent nature of the algorithm, which is a batch method employing the whole gradient, i.e. all then samples in the dataset. We eventually remark that L-BFGS was significantly less efficient with respect to the other built-in algorithms. This may be due to the finite-differences evaluation of the gradient in the L-BGFS Scipy implementation.

We will further refer to the results in Figure 2 and Table 3 as our baseline, according to which we evaluated other performances.

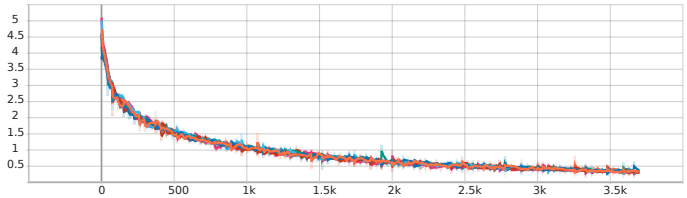


(a) Glorot Uniform

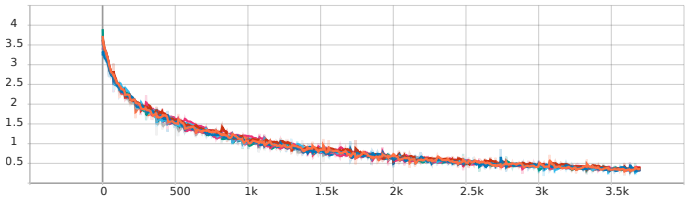


(b) Lecun Normal

Figure 3: Multi-start method with Adadelta. We report the number of iterations on the X-axis against the loss function values on the Y-axis.



(a) Glorot Uniform



(b) Lecun Normal

Figure 4: Multi-start method with Adagrad. We report the number of iterations on the X-axis against the loss function values on the Y-axis.

Table 4: Range of the grid search for the different hyper-parameters

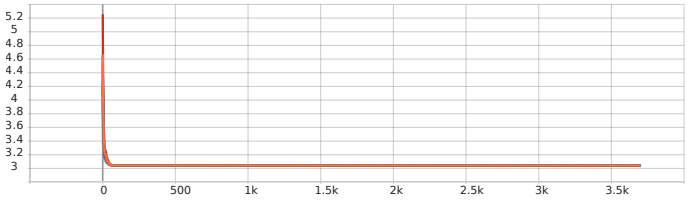
Algorithm	η	β	β_1	β_2	ϵ	Amsgrad	ρ	Centered	Nesterov
SGD	$10^{-i}, i = 1, 2, 3, 4$	$\{0, 0.5, 0.9\}$	-	-	-	-	-	-	True, False
Adam	$10^{-i}, i = 2, 3, 4$	-	$\{0.6, 0.9, 0.99\}$	$\{0.99, 0.999, 0.9999\}$	$[10^{-i}, i = 6, 7, 8]$	True, False	-	-	-
Adamax	$10^{-i}, i = 2, 3, 4$	-	$\{0.6, 0.9, 0.99\}$	$\{0.99, 0.999, 0.9999\}$	$[10^{-i}, i = 6, 7, 8]$	-	-	-	-
Nadam	$10^{-i}, i = 2, 3, 4$	-	$\{0.6, 0.9, 0.99\}$	$\{0.99, 0.999, 0.9999\}$	$[10^{-i}, i = 6, 7, 8]$	-	-	-	-
RMSProp	$10^{-i}, i = 2, 3, 4$	$\{0, 0.5, 0.9\}$	-	-	$[10^{-i}, i = 6, 7, 8]$	-	$\{0.6, 0.9, 0.99\}$	True, False	-

6.2 Grid search

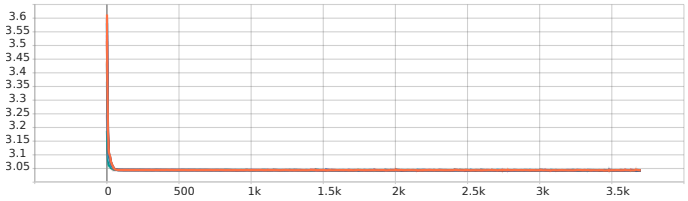
Once the baseline for the multiple optimizers has been set, in this subsection, we compare multiple hyper-parameters values to determine their optimal setting by performing a grid search. We carried out the grid search only for the five built-in algorithms that were actually achieving the global solution, i.e. we excluded from this test Adagrad, Adadelta, and FTRL. The range of our grid search is reported in Table 4 for all the hyper-parameters involved.

After carrying out the grid search and setting the hyper-parameters to their optimal value, which is reported for all the algorithms in Table 5, we trained again the network performing half time the epochs in section 4 and we obtained the loss shapes reported in Figure 7, Figure 8, Figure 9, Figure 10, Figure 11.

All the five algorithms achieve the global solution but, despite the loss always tends to zero, data augmen-

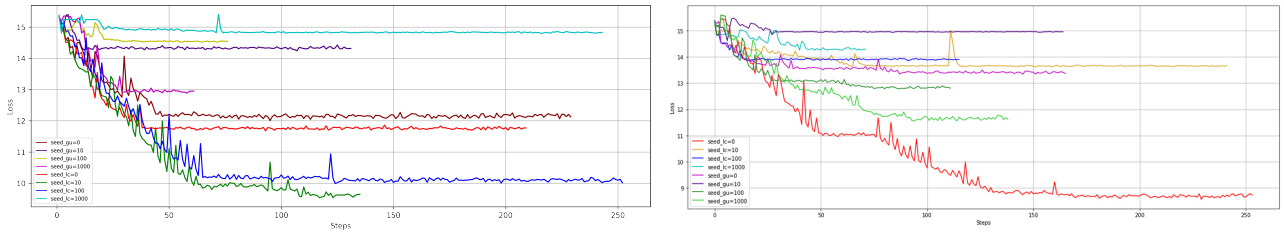


(a) Glorot Uniform



(b) Lecun Normal

Figure 5: Multi-start method with FTRL. We report the number of iterations on the X-axis against the loss function values on the Y-axis.



(a) Without MaxPooling layers and using Swish activation function (b) With MaxPooling layers and using ReLU activation function

Figure 6: L-BFGS training with different initialization seeds: (a) and (b) loss trends

tation appears to speed up the process, in particular for Adam and Adamax, as it can be seen in Figure 7 and Figure 8. Hyper-parameters optimization does not have a direct influence on the convergence point, enforcing the supposition that every algorithm has its proper performance on a given task. Concerning our image classification tasks, looking only at the loss shapes, one could argue that Nadam, RMSProp, and SGD are the best performing algorithms when hyper-parameters setting is optimized. Nonetheless, we should also consider the accuracy we manage to obtain, which in our case highlights the importance of carrying out the grid search. In fact, the optimal hyper-parameters setting produced a significant increase in terms of accuracy with respect to the values in Table 3. We report the achieved increase in Table 6. We observe that, in Adam and SGD, this gain is much more significant when data augmentation is used. Recalling section 4, we think that this happens because the averaged gradient in SGD and Adam is computed using more sample, which leads to better test perfomances.

Eventually, we remark the fact that these tests have been conducted using only half the epochs used in subsection 6.1. We did not only obtain an improvement in terms of accuracy, but we also achieved this result halving the computational effort, which confirms what stated in [17] concerning the dependency of the final performance metrics on the hyper-parameters.

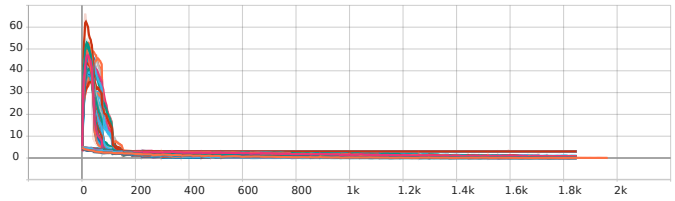
6.3 Modifying the neural architecture

In this last test, we study the behaviour of the different optimizers, comparing the default hyper-parameters and the optimal ones identified in subsection 6.2 while changing the network structure. Specifically, we have identified three configurations called: WIDE, DEEP, DEEP & WIDE. Each configuration is obtained by modifying the baseline network as shown in the following list:

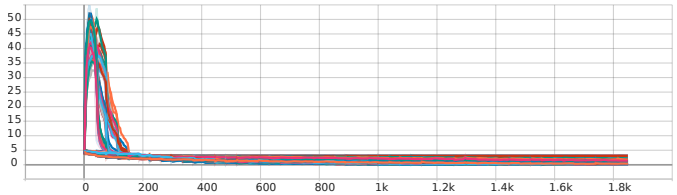
- WIDE: Here, the number of filters of the convolutional layers has been increased by a factor of 2.

Table 5: Optimal values of the optimizers hyper-parameters

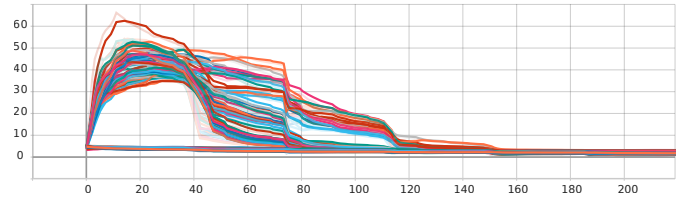
Algorithm	η	β	β_1	β_2	ϵ	Amsgrad	ρ	Centered	Nesterov
SGD	10^{-2}	0	-	-	-	-	-	-	False
Adam	10^{-3}	-	0.9	0.9999	10^{-8}	True	-	-	-
Adamax	10^{-3}	-	0.6	0.99	10^{-6}	-	-	-	-
Nadam	10^{-3}	-	0.99	0.99	10^{-6}	-	-	-	-
RMSProp	10^{-3}	0.0	-	-	10^{-6}	-	0.9	False	-



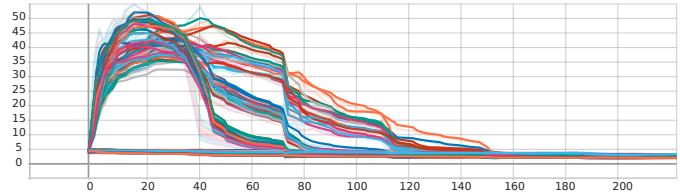
(a) 50 epochs with augmentation



(b) 50 epochs without augmentation



(c) Zoom of the first epochs



(d) Zoom of the first epochs

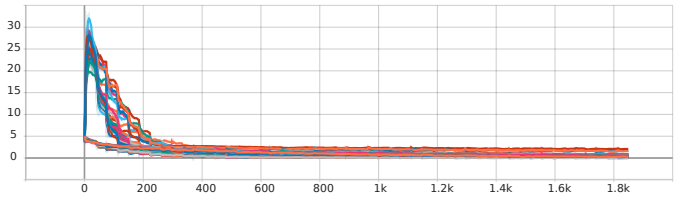
Figure 7: Adam loss trends with different hyperparameters setting: (a)(c) with augmentation (b)(d) without augmentation. We report the number of iterations on the X-axis against the loss function values on the Y-axis.

Table 6: Test accuracy increase when hyper-parameters are optimized with respect to the case when they are set to their default values.

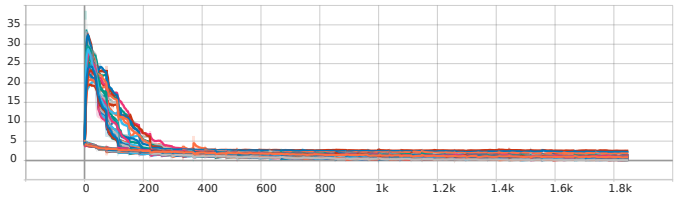
Algorithm	No Aug	Aug
Adam	2.1 %	2.5 %
Adamax	1.7 %	0.0 %
Nadam	3.0 %	2.6 %
RMSProp	3.1 %	-1.9 %
SGD	1.8 %	4.4 %

- DEEP: Here, all convolutional layers have been doubled.
- DEEP & WIDE: It is obtained as a merging between the previously described DEEP and WIDE networks.

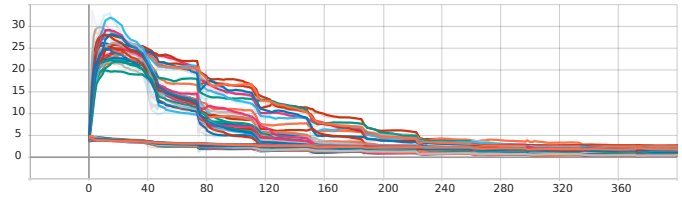
This test was aimed at assessing whether the optimal hyper-parameters setting is robust to architectural changes



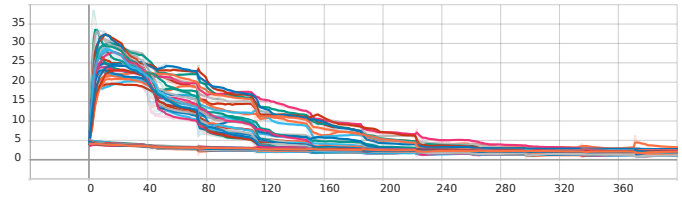
(a) 50 epochs with augmentation



(b) 50 epochs without augmentation



(c) Zoom of the first epochs



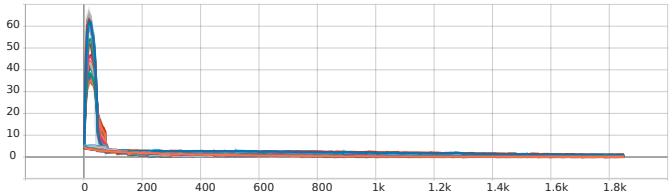
(d) Zoom of the first epochs

Figure 8: Adamax loss trends with different hyperparameters setting: (a)(c) with augmentation (b)(d) without augmentation. We report the number of iterations on the X-axis against the loss function values on the Y-axis.

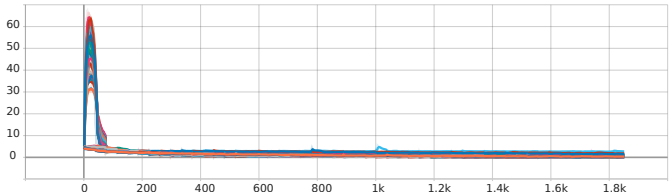
of the network. Since in the former experiments data augmentation lead to better performances in terms of accuracy, for the purpose of simplicity we have carried out this test only using data augmentation. In Figure 12 we see that the loss tends very rapidly to zero both using default hyper-parameters and using the optimal setting, while in Figure 13 and Figure 14 loss decrease is less expeditious, which makes us suppose that the task becomes harder when increasing the depth of the network. While in terms of loss decrease we do not observe significant differences between the optimal hyper-parameters setting and the default one, test accuracy results in Table 7 confirm the robustness of our optimal configuration. Adam, Adamax and SGD almost always achieve far better accuracy with the optimal setting found after the grid search than with the default one. Furthermore, the robustness of the optimal configuration is particularly evident when using SGD and Adamax, which produce extremely poor performances in DEEP and DEEP & WIDE configurations with default hyper-parameters. Finally, in Table 7 we also notice that the average accuracy performances are quite lower when dealing with deeper architectures (DEEP and DEEP & WIDE), confirming what we had already noticed concerning the loss decrease in those configurations.

7 Conclusions

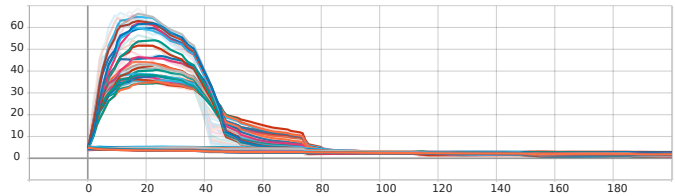
In this paper, nine optimization open-source algorithms have been extensively tested on training a deep CNN network on a hard classification problem. In the first set of tests, convergence to global minimizers is analyzed embedding the local algorithm in a multi-start framework. Computational experience shows that not all the algorithms reach a neighborhood of a global solution, and some of them get stuck in local minima, independently



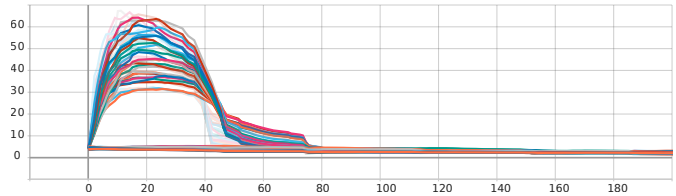
(a) 50 epochs with augmentation



(b) 50 epochs without augmentation



(c) Zoom of the first epochs



(d) Zoom of the first epochs

Figure 9: Nadam loss trends with different hyperparameters setting: (a)(c) with augmentation (b)(d) without augmentation.

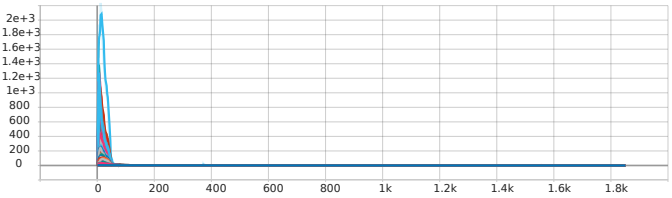
of the choice of the starting point. Further, test accuracy performances depend on the quality of the solution reached. Algorithms reaching a local non global solution have KPIs far below the minimal required threshold for such a task. This confirms the initial claim that reaching a neighborhood of the global optimum is extremely important for the test performance.

Furthermore, a fine grid search on the optimization hyper-parameters leads to hyper-parameters choices that give remarkable improvements in terms of test accuracy, without any change in the network structure. Thus using of standard setting may be not the better choice.

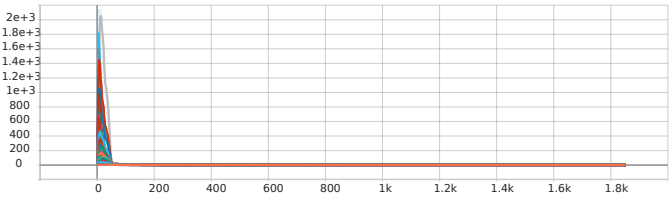
Eventually, the tests on different architectures, increasing the depth and the number of neurons, suggest that increasing the number of neurons seem to have a minor impact on the computational performance of the optimization algorithm. Instead, increasing the depth of the network seems to make the optimization task more difficult both with standard and refined hyper-parameters setting.

Acknowledgements

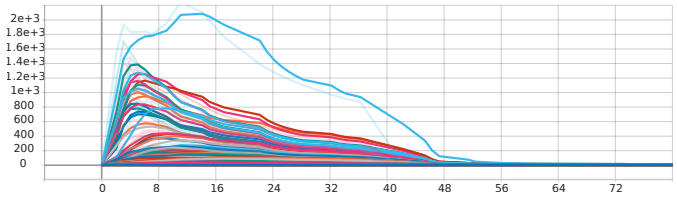
We thank Nicolas Zaccaria for the extensive editing performed on the graphs. We also thank ALCOR laboratory for having made available the workstations for the tests.



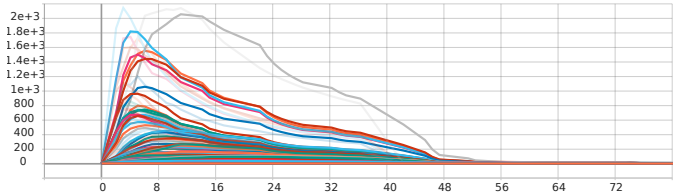
(a) 50 epochs with augmentation



(b) 50 epochs without augmentation



(c) Zoom of the first epochs



(d) Zoom of the first epochs

Figure 10: RMSprop loss trends with different hyperparameters setting: (a)(c) with augmentation (b)(d) without augmentation. We report the number of iterations on the X-axis against the loss function values on the Y-axis.

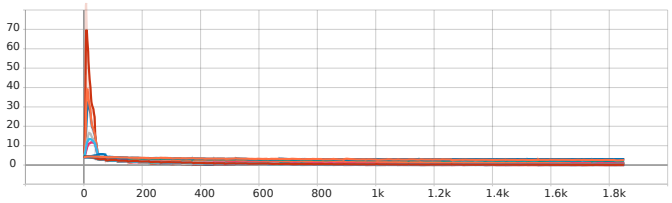
References

[1] H. Robbins, S. Monro, A stochastic approximation method, The annals of mathematical statistics (1951) 400–407.

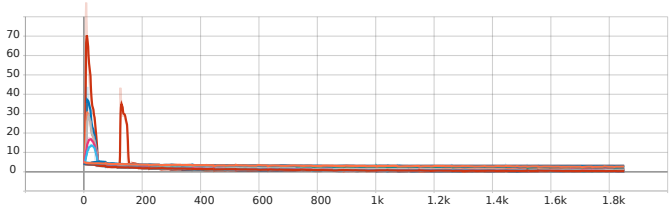
[2] T.-S. Lim, W.-Y. Loh, Y.-S. Shih, A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, Machine learning 40 (3) (2000) 203–228.

[3] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, S. S. Iyengar, A survey on deep learning: Algorithms, techniques, and applications, ACM Computing Surveys (CSUR) 51 (5) (2018) 1–36.

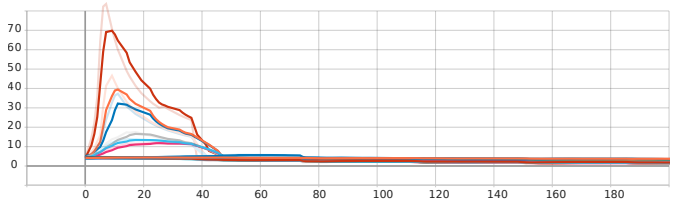
[4] H. B. Braiek, F. Khomh, On testing machine learning programs, Journal of Systems and Software 164 (2020) 110542. doi:https://doi.org/10.1016/j.jss.2020.110542.
URL https://www.sciencedirect.com/science/article/pii/S0164121220300248



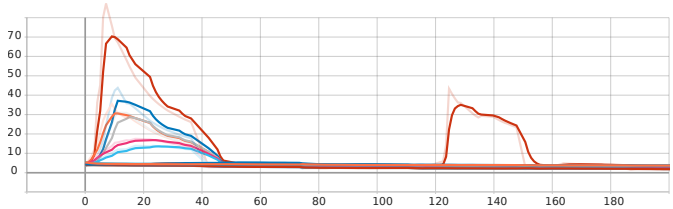
(a) 50 epochs with augmentation



(b) 50 epochs without augmentation



(c) Zoom of the first epochs



(d) Zoom of the first epochs

Figure 11: SGD loss trends with different hyperparameters setting: (a)(c) with augmentation (b)(d) without augmentation. We report the number of iterations on the X-axis against the loss function values on the Y-axis.

[5] L. Bottou, F. E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, Siam Review 60 (2) (2018) 223–311.

[6] P. Baumann, D. S. Hochbaum, Y. T. Yang, A comparative study of the leading machine learning techniques and two new optimization algorithms, European journal of operational research 272 (3) (2019) 1041–1057.

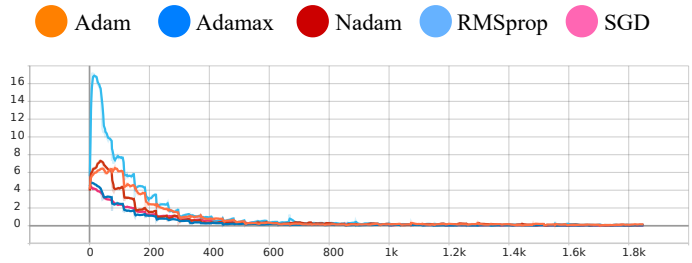
[7] G. Lan, First-order and stochastic optimization methods for machine learning, Springer, 2020.

[8] S. Sun, Z. Cao, H. Zhu, J. Zhao, A survey of optimization methods from a machine learning perspective, IEEE transactions on cybernetics 50 (8) (2019) 3668–3681.

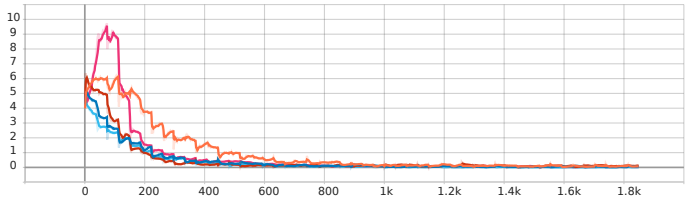
[9] A. Aggarwal, M. Mittal, G. Battineni, Generative adversarial network: An overview of theory and applications, International Journal of Information Management Data Insights 1 (1) (2021) 100004. doi:https://doi.org/10.1016/j.jjimei.2020.100004.
URL https://www.sciencedirect.com/science/article/pii/S2667096820300045

[10] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, A. A. Bharath, Generative adversarial networks: An overview, IEEE signal processing magazine 35 (1) (2018) 53–65.

[11] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, F.-Y. Wang, Generative adversarial networks: introduction and outlook, IEEE/CAA Journal of Automatica Sinica 4 (4) (2017) 588–598.

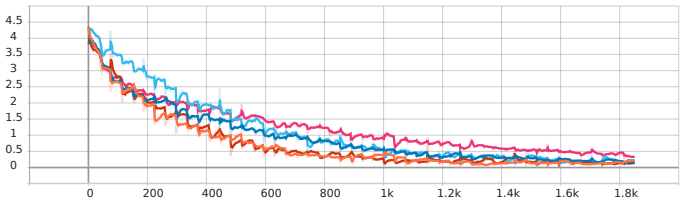


(a) Default HP

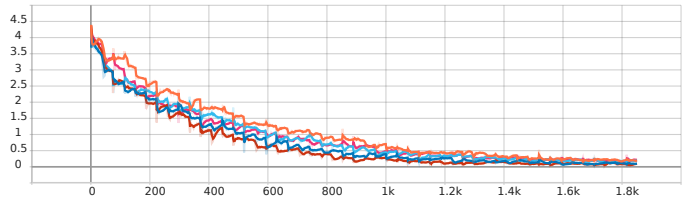


(b) Optimal HP

Figure 12: Loss trends in the WIDE configuration. Colors are assigned to optimizers according to the labels above.



(a) Default HP



(b) Optimal HP

Figure 13: Loss trends in the DEEP configuration. Colors are assigned to optimizers according to the labels in Figure 12.

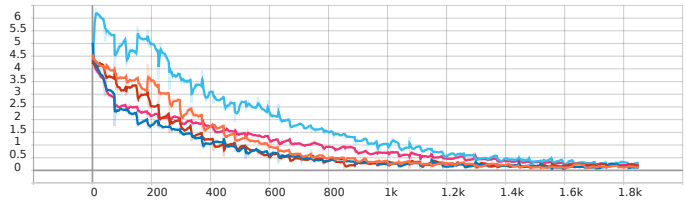
[12] A. Zutshi, D. Goswami, Systematic review and exploration of new avenues for sorting algorithm, International Journal of Information Management Data Insights 1 (2) (2021) 100042. doi:https://doi.org/10.1016/j.jjimei.2021.100042.
URL https://www.sciencedirect.com/science/article/pii/S2667096821000355

[13] X. Li, M. J. Garzaran, D. Padua, Optimizing sorting with machine learning algorithms, in: 2007 IEEE International Parallel and Distributed Processing Symposium, IEEE, 2007, pp. 1–6.

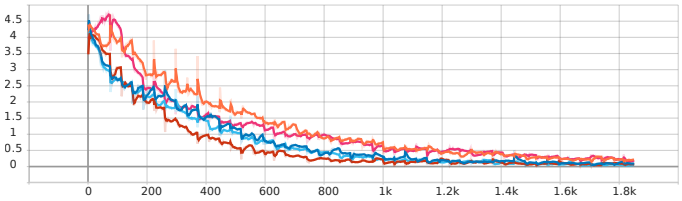
[14] R. Sun, Optimization for deep learning: theory and algorithms, arXiv preprint arXiv:1912.08957.

[15] L. Palagi, Global optimization issues in deep network regression: an overview, Journal of Global Optimization 73 (2) (2019) 239–277.

[16] D. J. Im, M. Tao, K. Branson, An empirical analysis of the optimization of deep network loss surfaces, arXiv preprint arXiv:1612.04010.



(a) Default HP



(b) Optimal HP

Figure 14: Loss trends in the DEEP & WIDE configuration. Colors are assigned to optimizers according to the labels in Figure 12.

[17] P. Xu, F. Roosta, M. W. Mahoney, Second-order optimization for non-convex machine learning: An empirical study, in: Proceedings of the 2020 SIAM International Conference on Data Mining, SIAM, 2020, pp. 199–207.

[18] T. Chauhan, H. Palivela, S. Tiwari, Optimization and fine-tuning of densenet model for classification of covid-19 cases in medical imaging, International Journal of Information Management Data Insights 1 (2) (2021) 100020. doi:https://doi.org/10.1016/j.jjimei.2021.100020.
URL https://www.sciencedirect.com/science/article/pii/S2667096821000136

[19] Z. Young, R. Steele, Empirical evaluation of performance degradation of machine learning-based predictive models – a case study in healthcare information systems, International Journal of Information Management Data Insights 2 (1) (2022) 100070. doi:https://doi.org/10.1016/j.jjimei.2022.100070.
URL https://www.sciencedirect.com/science/article/pii/S2667096822000143

[20] I. K. M. Jais, A. R. Ismail, S. Q. Nisa, Adam optimization algorithm for wide and deep neural network, Knowledge Engineering and Data Science 2 (1) (2019) 41–46.

[21] T. Ding, D. Li, R. Sun, Suboptimal local minima exist for wide neural networks with smooth activations, Mathematics of Operations Research.

[22] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series, The handbook of brain theory and neural networks 3361 (10) (1995) 1995.

[23] S. Hijazi, R. Kumar, C. Rowen, et al., Using convolutional neural networks for image recognition, Cadence Design Systems Inc.: San Jose, CA, USA 9.

[24] L. Papa, E. Alati, P. Russo, I. Amerini, Speed: Separable pyramidal pooling encoder-decoder for real-time monocular depth estimation on low-resource settings, IEEE Access 10 (2022) 44881–44890. doi:10.1109/ACCESS.2022.3170425.

Table 7: Test accuracy obtained in the different network architectures with optimal and with default hyper-parameters values.

Algorithm	WIDE		DEEP		DEEP & WIDE	
	Opt HP	Def HP	Opt HP	Def HP	Opt HP	Def HP
Adam	70.3 %	65.1 %	51.7 %	56.0 %	48.6 %	61.9 %
Adamax	67.8 %	68.2 %	58.2 %	18.3 %	64.1 %	30.5 %
Nadam	73.8 %	70.8 %	39.7 %	51.3 %	59.0 %	60.1 %
RMSProp	64.2 %	65.2 %	28.1 %	58.9 %	44.3 %	52.5 %
SGD	63.2 %	63.2 %	51.4 %	24.3 %	56.8 %	23.3 %

[25] Y. Guo, Y. Liu, T. Georgiou, M. S. Lew, A review of semantic segmentation using deep neural networks, International journal of multimedia information retrieval 7 (2) (2018) 87–93.

[26] C. Ding, D. Tao, Trunk-branch ensemble convolutional neural networks for video-based face recognition, IEEE transactions on pattern analysis and machine intelligence 40 (4) (2017) 1002–1014.

[27] B. J. Abbaschian, D. Sierra-Sosa, A. S. Elmaghraby, Deep learning techniques for speech emotion recognition, from databases to models, Sensors (Basel, Switzerland) 21.

[28] S. Kuutti, R. Bowden, Y. Jin, P. Barber, S. Fallah, A survey of deep learning applications to autonomous vehicle control, IEEE Transactions on Intelligent Transportation Systems 22 (2021) 712–733.

[29] C. Shorten, T. M. Khoshgoftaar, B. Furht, Deep learning applications for covid-19, Journal of Big Data 8.

[30] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 770–778.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, ArXiv abs/1704.04861.

[32] G. Huang, Z. Liu, K. Q. Weinberger, Densely connected convolutional networks, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 2261–2269.

[33] Y. Bengio, Practical recommendations for gradient-based training of deep architectures, in: Neural networks: Tricks of the trade, Springer, 2012, pp. 437–478.

[34] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980.

[35] T. Dozat, Incorporating nesterov momentum into adam, in: ICLR Workshop, 2016.

[36] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747.

[37] I. Sutskever, J. Martens, G. E. Dahl, G. E. Hinton, On the importance of initialization and momentum in deep learning, in: ICML, 2013.

[38] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, J. Kubica, Ad click prediction: a view from the trenches, Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining.

[39] J. C. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, in: J. Mach. Learn. Res., 2010.

[40] M. D. Zeiler, Adadelta: An adaptive learning rate method, ArXiv abs/1212.5701.

[41] D. C. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Mathematical programming 45 (1) (1989) 503–528.

[42] J. Nocedal, S. J. Wright, Numerical optimization, Springer, 1999.

[43] A. S. Berahas, J. Nocedal, M. Takác, A multi-batch l-bfgs method for machine learning, Advances in Neural Information Processing Systems 29.

[44] R. Bollapragada, J. Nocedal, D. Mudigere, H.-J. Shi, P. T. P. Tang, A progressive batching l-bfgs method for machine learning, in: International Conference on Machine Learning, PMLR, 2018, pp. 620–629.

[45] A. S. Berahas, M. Takáč, A robust multi-batch l-bfgs method for machine learning, Optimization Methods and Software 35 (1) (2020) 191–219.

[46] C. Yun, S. Sra, A. Jadbabaie, Small nonlinearities in activation functions create bad local minima in neural networks, arXiv preprint arXiv:1802.03487.

[47] H. Wang, H. Gemmeke, T. Hopp, J. Hesser, Accelerating image reconstruction in ultrasound transmission tomography using l-bfgs algorithm, in: Medical Imaging 2019: Ultrasonic Imaging and Tomography, Vol. 10955, SPIE, 2019, pp. 67–76.

[48] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in: International conference on machine learning, PMLR, 2013, pp. 1139–1147.

[49] G. Hinton, N. Srivastava, K. Swersky, Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, Cited on 14 (8) (2012) 2.

[50] Y. Yang, S. Newsam, Bag-of-visual-words and spatial extensions for land-use classification, in: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 270–279. doi:10.1145/1869790.1869829.
URL https://doi.org/10.1145/1869790.1869829

[51] G. Swirszcz, W. M. Czarnecki, R. Pascanu, Local minima in training of neural networks, arXiv preprint arXiv:1611.06310.

[52] M. A. Mercioni, S. Holban, P-swish: Activation function with learnable parameters based on swish activation function in deep learning, in: 2020 International Symposium on Electronics and Telecommunications (ISETC), IEEE, 2020, pp. 1–4.

[53] P. Ramachandran, B. Zoph, Q. V. Le, Searching for activation functions, arXiv preprint arXiv:1710.05941.