applied sciences

MDPI

Article

Training of an Extreme Learning Machine Autoencoder Based on an Iterative Shrinkage-Thresholding Optimization Algorithm

José A. Vásquez-Coronel [1,†], Marco Mora [2,3,*,†] and Karina Vilches [2,4,†]

[1] Doctorado en Modelamiento Matemático Aplicado, Universidad Católica del Maule, Talca 3480112, Chile
[2] Laboratory of Technological Research in Pattern Recognition (LITRP), Universidad Católica del Maule, Talca 3480112, Chile
[3] Departamento de Ciencias de la Computación e Industrias, Universidad Católica del Maule, Talca 3480112, Chile
[4] Departamento de Matemáticas, Física y Estadística, Universidad Católica del Maule, Talca 3480112, Chile
* Correspondence: mmora@ucm.cl
† These authors contributed equally to this work.

Abstract: Orthogonal transformations, proper decomposition, and the Moore–Penrose inverse are traditional methods of obtaining the output layer weights for an extreme learning machine autoencoder. However, an increase in the number of hidden neurons causes higher convergence times and computational complexity, whereas the generalization capability is low when the number of neurons is small. One way to address this issue is to use the fast iterative shrinkage-thresholding algorithm (FISTA) to minimize the output weights of the extreme learning machine. In this work, we aim to improve the convergence speed of FISTA by using two fast algorithms of the shrinkage-thresholding class, called greedy FISTA (G-FISTA) and linearly convergent FISTA (LC-FISTA). Our method is an exciting proposal for decision-making involving the resolution of many application problems, especially those requiring longer computational times. In our experiments, we adopt six public datasets that are frequently used in machine learning: MNIST, NORB, CIFAR10, UMist, Caltech256, and Stanford Cars. We apply several metrics to evaluate the performance of our method, and the object of comparison is the FISTA algorithm due to its popularity for neural network training. The experimental results show that G-FISTA and LC-FISTA achieve higher convergence speeds in the autoencoder training process; for example, in the Stanford Cars dataset, G-FISTA and LC-FISTA are faster than FISTA by 48.42% and 47.32%, respectively. Overall, all three algorithms maintain good values of the performance metrics on all databases.

Keywords: autoencoder; feature extraction; extreme learning machine; shrinkage-thresholding algorithms

Citation: Vásquez-Coronel, J.A.; Mora, M.; Vilches, K. Fast Training of Extreme Learning Machine Autoencoder based on Iterative Shrinkage-Thresholding Optimization Algorithm. *Appl. Sci.* **2022**, *12*, 9021. https://doi.org/10.3390/app12189021

Academic Editors: Krzysztof Ejsmont, Aamer Bilal Asghar, Yong Wang and Rodolfo Haber

Received: 3 August 2022
Accepted: 5 September 2022
Published: 8 September 2022

1. Introduction

In pattern recognition systems, efficient methods of feature selection and extraction can reduce the dimensionality problem, thus reducing both the computation time and the memory requirements of the training algorithms [1]. An autoencoder is a feed-forward neural network that builds a compact representation of the input data, and is mainly used for unsupervised learning [2]. It is composed of an encoder and a decoder: the encoder reads the input data and maps them to a lower-dimensionality space, while the decoder reads the compact representation and reconstructs the neural network input. In the same way as for all supervised learning neural networks [3,4], the core aspect of the training of an autoencoder is the backpropagation algorithm. This algorithm iteratively tunes the weights and biases of the neural network by applying the gradient descent method. Autoencoder networks are in great demand in multiple applications in modern society, for example, for dimensionality reduction, image retrieval, denoising, and data augmentation [2].

Several works in the literature have developed feature extraction methods using autoencoders and backpropagation. A comparative study between the performance of a

traditional autoencoder and a denoising sparse autoencoder is presented in [5]. Another widely used architecture is the stacked autoencoder, which includes the stacked denoising sparse autoencoder [6] and the symmetric stacked autoencoder with shared weights [7]. Semi-supervised learning algorithms have also been used as autoencoder training algorithms [8,9]. However, the backpropagation training technique limits the generalization ability of the autoencoder, which results in low computational efficiency and the presence of local optima.

An extreme learning machine (ELM) is an emerging training paradigm for neural networks which includes supervised [10], semi-supervised [11], and unsupervised training [12]. The fundamental principle of this architecture is the random assignment of weights and biases in the hidden layer and the analytical determination of the weights of the output layer, in which the least squares method is applied to a system of linear equations [13,14]. The simplicity of the model, the lower numbers of training parameters, the low convergence speed, and its high generalization capacity mean that ELM neural networks have advantages for classification, regression, and clustering problems. Variants such as the kernel-ELM (K-ELM) [15], due to the significant reduction in the number of parameters and the increase in the training speed, and the online sequential ELM [16], due to its capacity to process batches of data, can even deal efficiently with problems involving large volumes of data. In the field of rolling element fault diagnosis, ELM can also solve the problem of weak signals caused by long transmission paths [17]; for example, the novel method presented in [18] can help in frame feature selection for ELM. In computer vision, the researchers of [19] proposed a visual object tracking scheme with promising results.

On the other hand, to handle the problems that local minima can cause, some studies have used heuristic optimization techniques to estimate the network parameters, such as differential evolution, particle swarming (PSO), and genetic algorithms [20]. These heuristic algorithms always search for the global minimum of the objective function and are efficient in several research fields, as is the case for the PSO algorithm that adjusts the parameters of an underactuated surface search model [21]. An improved scheme of this method integrates the PSO with global optimization capability, a 3-Opt algorithm with local search capability, and a fuzzy system with fuzzy reasoning ability [22]. The probabilistic ant colony algorithm is another optimization technique integrated into the architecture of an ELM network [23]. The K-ELM method improves the hyperspectral image classification capacity by coupling the principal component analysis, local binary pattern, and gray wolf optimization algorithm with global search capability [24].

Due to the advantages of ELM over backpropagation, the authors of [25] proposed to train autoencoder networks using an extreme learning machine (ELM-AE). A sparse ELM-AE was presented in [26] by adding the L_1-norm in the quadratic term to give a more significant representation of data. Another approach for unlabeled feature extraction consisted of a generalized ELM-AE in which a manifold regularization factor was added to the objective function of the ELM-AE [27]. An unsupervised learning ELM-AE was proposed in [28], inspired by the use of embedding graphs to capture the structure of the input patterns, and was constructed using local Fisher discrimination analysis. A dense connection autoencoder was introduced into multi-layer learning, in which the output features of the previous layers were used by the following layers [29]. A double random hidden layer ELM autoencoder can achieve efficient extraction of features with dimension reduction capability for deep learning [30]. The L_{21}-norm regularization has been used for the ELM-AE optimization problem, creating a new unsupervised learning framework that included a disperse representation despite the influence of atypical values and noisy data [31]. In addition, the correntropy-based ELM autoencoder presented in [32] was developed to extract features from noisy input patterns. To lower the number of parameters to be tuned and for the efficient calculation of the pseudo-inverse matrix, an ELM-AE with a kernel function was proposed in [33,34]. From a different perspective, the need for unsupervised batch learning means that an online sequential ELM-AE [35,36] is an exciting approach to solving large-scale applications.

From the works described above, we see that the backpropagation method and the Moore–Penrose inverse are two classical methods of estimating the weights of an autoencoder network. However, using these methods may cause overfitting when the number of unknown variables is larger than the number of training data. In addition, the accuracy is low when the number of hidden neurons is small, and the training time increases significantly when solving higher-dimensional applications. To overcome these drawbacks, researchers have proposed the use of sparse regularization techniques [37], with FISTA being the most widely used algorithm [38]. Although its origin dates back to 2009, this algorithm is still under study, thanks to its convergence speed, accuracy, and straightforward implementation. Several state-of-the-art studies have demonstrated the successful use of this optimization tool in machine learning. For example, in multilayer network design, the minimization problem defined in the autoencoder uses shrinkage-thresholding optimization techniques for dimension reduction and sparse feature representation [26,32,39]. For noisy image and video processing, a mixed scheme was presented in [40], in which sparse coding was adopted with deep learning to solve supervised and unsupervised tasks. A visual dictionary classification scheme was addressed in [41] using a simple (deep) convolutional autoencoder, for which the source of inspiration was sparse optimization and the iterative shrinkage-thresholding algorithm [38]. More recent studies [42,43] have demonstrated this algorithm's development, continuity, and importance for solving convex minimization problems. In particular, they estimated the output weights of ELM and validated their results on classification problems.

Due to the importance of sparse regularization in the mathematical modeling of many real situations and the ability of shrinkage-thresholding algorithms to minimize these optimization problems, our work aims to study these optimization techniques in the architecture of an ELM-AE since there are few studies in the literature. Specifically, this paper proposes the use of fast algorithms from the shrinkage-thresholding family to improve the convergence speed achieved by FISTA during the training of an ELM-AE. In particular, two representative models of this class are considered: G-FISTA [44] and LC-FISTA [45,46]. To evaluate the efficiency of our scheme, six datasets that are widely used in the validation of ELM algorithms are adopted: MNIST, NORB, CIFAR10, UMist, Caltech256, and Stanford Cars. The main contributions of our article are as follows:

- We present a novel method for computing the output layer weights of the ELM-AE network that avoids the need for the Moore–Penrose inverse. Instead of solving the linear system analytically, we use first-order iterative algorithms from the class of shrinkage-thresholding algorithms to minimize the output weights of the ELM-AE.
- We demonstrate experimentally that the proposed method can effectively improve the computational time of the FISTA algorithm while maintaining its generalization capability. According to the theory presented in [47], the ELM-AE experiences a better performance when it achieves the lowest training error.
- Compared to FISTA, the G-FISTA, and LC-FISTA algorithms show a considerable improvement in the training time of the ELM-AE, for all of the databases, and a very competitive reconstruction capability. Thus, applying this mathematical tool in other contexts would constitute a significant scientific contribution.

The rest of this work is structured as follows: in Section 2, we briefly explain the ELM-AE, FISTA, G-FISTA, and LC-FISTA algorithms. Section 3 describes the method proposed to train the ELM-AEs. The datasets and the results are presented in Section 4. The discussion of results is presented in Section 5. Finally, Section 6 gives the conclusions of this work.

2. ELM-AE and Shrinkage-Thresholding Algorithms

2.1. Extreme Learning Machine Autoencoder

The ELM-AE [25] is trained with an ELM algorithm [10] in an unsupervised manner. It corresponds to a single hidden layer feedforward neural network, in which the outputs match the inputs. The ELM-AE has both a high training speed and a strong generalization

ability due to the pseudo-random creation of neurons in the hidden layer and the application of a pseudoinverse matrix to calculate the weights of the output layer. To improve the performance of this method, the authors of [25] assigned random orthogonal weights and biases to the hidden layer. This approach was shown to have good generalization, and to minimize $\|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{X}\|_2^2$ and the square norm of the coefficients $\|\boldsymbol{\beta}\|_2^2$. Given a set of N training samples $\{\boldsymbol{x}_i \mid i \leq N\}$, where $\boldsymbol{x}_i \in \mathbb{R}^m$ are both the input and output data, the output with L hidden neurons can be expressed as follows:

$$f_L(\boldsymbol{x}_j) = \sum_{i=1}^{L} \boldsymbol{\beta}_i g(\boldsymbol{w}_i \boldsymbol{x}_j + b_i), \quad j = 1, \ldots N \tag{1}$$

where $\boldsymbol{w}_i$ and b_i represent the pseudorandom weights and biases of the hidden layer, g is an activation function, and $\boldsymbol{\beta}_i$ represents the weights of the output layer. The previous overdetermined linear system can be compactly described as follows:

$$\boldsymbol{H}\boldsymbol{\beta} = \boldsymbol{X}, \tag{2}$$

where $\boldsymbol{H} \in \mathbb{R}^{N \times L}$ is the hidden layer output matrix, $\boldsymbol{\beta} \in \mathbb{R}^{L \times m}$ is the output layer weight matrix, and $\boldsymbol{X} \in \mathbb{R}^{N \times m}$ is the input data matrix.

Indeed, $\boldsymbol{\beta} = \boldsymbol{H}^{\dagger}\boldsymbol{X}$ is the standard solution to the overdetermined problem in (2), where $\boldsymbol{H}^{\dagger} = (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T$ is the Moore–Penrose inverse of matrix $\boldsymbol{H}$. To improve the performance of an ELM-AE, the authors of [25] determined the output weights $\boldsymbol{\beta}$ by means of the equation $\boldsymbol{\beta} = \left(\frac{\boldsymbol{I}}{\lambda} + \boldsymbol{H}^T\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{X}$, which is the result of solving the following regularized optimization problem:

$$\boldsymbol{\beta} = \text{argmin}\|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{X}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2, \tag{3}$$

where λ is a tuning regularization parameter.

2.2. The Shrinkage-Thresholding Class of Optimization Algorithms

A large number of applications can be formulated as a convex optimization problem [48], as shown by the following expression:

$$\min : h(\boldsymbol{\tau}) = f(\boldsymbol{\tau}) + \phi(\boldsymbol{\tau}), \tag{4}$$

where $f, \phi : \boldsymbol{\tau} \in \mathbb{R}^n \to \mathbb{R}$ are two convex functions,∇f is L_f-Lipschitz continuous and ϕ is a non-differentiable function that cannot be easily minimized in several machine learning applications. The solution to ϕ can be evaluated using the proximal operator [48], as shown in the following equation:

$$\text{prox}_{\rho\phi}(\boldsymbol{z}) = \arg\min_{\boldsymbol{\tau}} \tfrac{1}{2}\|\boldsymbol{\tau} - \boldsymbol{z}\|_2^2 + \rho\,\phi(\boldsymbol{\tau}), \tag{5}$$

where ρ is the stepsize and $\boldsymbol{z}$ is the initial estimate of the solution. As a consequence, the optimization of the function h uses a two-phase method: (i) performance of a forward gradient descent step on the smooth function f; and (ii) performance of the proximal operator or a backward gradient descent step on the non-smooth function ϕ.

In the following, a brief description of the FISTA, G-FISTA, and LC-FISTA algorithms is presented. These algorithms can be used to train ELM-AEs.

2.2.1. Fast Iterative Shrinkage Thresholding Algorithm

FISTA [38] is characterized by its convergence speed and high level of efficiency. When minimizing the convex problem in (4), the algorithm has a convergence rate on the order of $\mathcal{O}(1/k^2)$. The sequence of steps of the FISTA algorithm is as follows:

(1) Update: $\boldsymbol{\tau}_k = \text{prox}_{\rho\,\phi}(\boldsymbol{z}_k - \rho\nabla f(\boldsymbol{z}_k))$.

(2) Execute the intermediate step: $t_{k+1} = \frac{1+(1+4t_k^2)^{1/2}}{2}$.
(3) Update: $z_{k+1} = \tau_k + \gamma_k(\tau_k - \tau_{k-1})$.

The term $\rho \in (0, \frac{1}{L_f}]$ represents the step size, and $\gamma_k = \frac{t_k - 1}{t_{k+1}}$ represents the impulse factor. FISTA's convergence rate depends mainly on its proximal operator and the use of a weighted step z_k, which is expressed as a combination of both τ_{k-1} and τ_k. In addition, the configuration of the Lipschitz constant L_f of the gradient f plays an essential part in the convergence of the algorithm.

2.2.2. Greedy FISTA

In general terms, G-FISTA [44] is a variant of FISTA that is characterized by a higher speed. Given the conditions imposed by G-FISTA, its architecture can restart the FISTA algorithm with the fixed impulse factor $\gamma_k = 1$. This algorithm updates the solution to the optimization problem in (4), setting $\rho \in [\frac{1}{L_f}, \frac{2}{L_f}]$, $\xi < 1$ and $S > 1$ until the maximum number of iterations is reached. The updating scheme is as follows:

(1) Update: $\tau_k = \text{prox}_{\rho\phi}(z_k - \rho\nabla f(z_k))$.
(2) Update: $z_{k+1} = \tau_k + (\tau_k - \tau_{k-1})$.
(3) Restart: if $(z_k - \tau_k)^T(\tau_k - \tau_{k-1}) \geq 0$, then $z_{k+1} = \tau_k$.
(4) Safeguard: if $\|\tau_k - \tau_{k-1}\| \geq S\|\tau_1 - \tau_0\|$, then $\rho = \max\{\xi\rho, \frac{1}{L_f}\}$.

The G-FISTA algorithm improves the convergence rate and the reconstruction capability of FISTA, which uses a self-adaptive restart and adjustment scheme to obtain an additional acceleration and alleviate the oscillation problem in the reconstruction process. Based on the results discussed in [44,49], G-FISTA has efficient performance when $\rho \in [\frac{1}{L_f}, \frac{1.3}{L_f}]$. For this reason, all experiments performed in this work were configured with $\rho = 1.3/L_f$, $\xi = 0.96$, and $S = 1.1$.

2.2.3. Linearly Convergent FISTA

LC-FISTA is an accelerated version of FISTA derived from the research in [45,46]. Under specific error conditions, LC-FISTA works with global linear convergence for a large group of real applications, which minimizes the formulation of the convex problem of the equation in (4). The iterative steps of the algorithm are as follows:

(1) Set both $\tau_0 = z_0 = \mathbf{0}$, and α, θ, μ y L_f.
(2) Update: $y_k = \frac{1}{1+\theta}\tau_k + \frac{\theta}{1+\theta}z_k$.
(3) Update: $\tau_{k+1} = \text{prox}_{\rho\phi}(y_k - \rho\nabla f(y_k))$.
(4) Update: $z_{k+1} = (1-\theta)z_k + \theta y_k + \alpha(\tau_{k+1} - y_k)$.

With a mathematical analysis analogous to that presented in [45,46], the sequence $\{\tau_k\}_{k\in\mathbb{N}}$ generated by the LC-FISTA scheme exhibits a linear convergence rate, for fixed values of $\alpha = (\frac{L_f}{\mu})^{1/2}$ and $\theta = (\frac{\mu}{L_f})^{1/2}$. The practical use of LC-FISTA requires tuning parameters such as the strong convexity parameter μ for the differentiable function f. In addition, we need to set $\tau_0 = z_0$ to obtain convergence rates similar to the accelerated variant of FISTA presented in [50]. It is common to set $x_0 = 0$, since the location of the global minimum of the objective function h is not known.

Remark 1. *The proximal operator $prox_{\rho\phi}(u_k)$ is the step that is common to FISTA, G-FISTA, and LCFISTA. This value $\tau_{k+1} = prox_{\rho\phi}(u_k)$ is closer to the global minimum of h compared with the maximum descent step $u_k := z_k - \rho\nabla f(z_k)$. Since t_k causes oscillations in the FISTA scheme when it has a value of one, G-FISTA shortens the interval between two restarts by setting $t_k = 1$, which is an essential condition for G-FISTA to converge with fewer iterations. In LC-FISTA, the θ and α momentum terms incorporated in the two additional equations (see (1) and (3) for LC-FISTA) improve both the accuracy of the gradient and the speed of convergence of LC-FISTA via the proximal operator.*

3. Proposed Method of Training ELM-AEs Based on G-FISTA and LC-FISTA

ELM-AE has an extremely fast training speed, and good pattern reconstruction capability that is better than backpropagation-based learning. The formulation of this model can be expressed as a convex optimization problem which involves minimizing the weights of the $\boldsymbol{\beta}$ output layer, as shown in the following equation:

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{X}\|_u^{\kappa_1} + \lambda \|\boldsymbol{\beta}\|_v^{\kappa_2}, \tag{6}$$

where $\kappa_1, \kappa_2 > 0$, $u, v = 0, \frac{1}{2}, 1, 2, \ldots$ and λ is a control parameter between the training error and the generalization ability.

Several methods exist for computing the solution to (6), such as the Moore–Penrose inverse, orthogonal projection, and proper decomposition [51]. However, in a real situation, the number of independent variables L in the linear system $\boldsymbol{H}\boldsymbol{\beta} = \boldsymbol{X}$ is much larger than the number of data points N, which may cause overfitting. On the other hand, the reconstruction capacity is low whenever the number of hidden neurons L is small. As the training dataset increases, the computational cost of the inverse matrix $\boldsymbol{H}^{\dagger}$ increases significantly. Several regularization methods have been introduced in the literature to address the above problems. The two most commonly used classical techniques are ridge regression [52] and sparse regularization [37]. The sparse regularization plays an important role in feature selection, which has shown successful results in machine learning. This study presents an unsupervised feature selection framework, which integrates sparse optimization and signal reconstruction for pattern recognition models. The minimization problem takes the following form:

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{X}\|_2^2 + \lambda |\boldsymbol{\beta}|_1, \tag{7}$$

which is a special case of (4), in which we set $f(\boldsymbol{\beta}) = \|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{X}\|_2^2$ and $\phi(\boldsymbol{\beta}) = \lambda|\boldsymbol{\beta}|_1$.

From the description given above, we see that the estimation of $\boldsymbol{\beta}$ and the time required are the main challenges for training the ELM-AE. Our study proposes two novel iterative schemes for autoencoder training, G-FISTA, and LC-FISTA, which reduce the computational time of FISTA maintaining its accuracy, where FISTA is the classical algorithm for training neural networks. This optimization approach is also interesting because it can be extended to other regularization terms, which control the variables selection by calculating a derivative in the weakest sense through convex envelopes. In addition, the closed form of the proximal operator $\text{prox}_{\rho\phi}(\cdot)$ of ϕ is computed by the shrinkage operator $\text{shrink}(\boldsymbol{z}, \lambda\rho)$ [38], whose i-th element is calculated as follows:

$$\text{shrink}(\boldsymbol{z}, \lambda\rho)_i = \text{shrink}(\boldsymbol{z}_i, \lambda\rho) = \text{sign}(z_i)\max\{|z_i| - \lambda\rho, 0\}, \tag{8}$$

where $\boldsymbol{z}$ is the initial estimation of the weight of the $\boldsymbol{\beta}$ output layer.

The speed that can be achieved by the use of G-FISTA and LC-FISTA in the training of the ELM-AE is significantly higher than that obtained using FISTA. In addition, for poorly conditioned matrices, the two algorithms have the same ability as FISTA to reconstruct the input signals. This is realized by the omission of several steps of the gradient descent, and correction of the solution by means of the proximal gradient. Although the iterative scheme used by G-FISTA and LC-FISTA effectively has additional terms compared to FISTA, the computational cost can be similar, since they converge with fewer iterations. The following algorithm presents the steps that are followed in the training of the ELM-AE, where the parameters to be configured are the number of iterations I, the number of neurons L, the stopping criterion P and the regularization parameter λ. The variables represent features associated with the pixels of the images, ordered according to the columns of the matrix $\boldsymbol{X}$. In the reconstruction and variable selection process, the input data, expected target, network weights, and biases are matrix arrays that facilitate the computation of many vector operations.

4. Experimental Evaluation

This section presents the databases and evaluation metrics selected to evaluate the performance of the proposed method.

4.1. Hardware, Software and Databases

For this work, a server from the cluster of the Laboratory of Technological Research in Pattern Recognition (LITRP) of the Universidad Católica del Maule was used. The server was equipped with two CPUs, an Intel Xeon Gold 6238 252 CPU @ 2.20–4.00 GHz (56 physical cores), 126 GB RAM and a GPU NVIDIA Titan RTX. The algorithms were implemented in the MATLAB R2020a programming language.

To examine the performance of Algorithm 1, we used the CIFAR10, UMist, MNIST, NORB, Stanford Cars and Caltech256 databases, which were obtained from different free online repositories. These datasets are briefly described below.

(1) MNIST. This dataset contains 70,000 images of handwritten digits from zero to nine. These are grayscale images of size 28 × 28 pixels which generate a vector of 784 components.
(2) NORB. This consists of 48,600 grayscale stereo images of 50 toys from five generic classes: cars, four-legged animals, human figures, airplanes, and trucks. The size of the images is 96 × 96 pixels, and the output is a vector of 96 × 96 × 2 = 18,432 components.
(3) CIFAR10. This dataset consists of 60,000 color images distributed into 10 classes: airplanes, birds, automobiles, cats, dogs, frogs, deer, ships, horses, and trucks. The number of images per class is 6000, with 5000 used for training and the remainder for testing. In this experiment, the color images were converted to grayscale and had a size of 32 × 32 pixels.
(4) UMist. This contains 575 grayscale images of 20 people, showing individuals in different positions. The images are rectangular, with a size of 92 × 112 pixels.
(5) Caltech256. This dataset contains 30,607 color images grouped into 257 categories, each of which contain at least 80 images. For the purposes of this study, the images were converted to grayscale and had a size of 100 × 100 pixels.
(6) Stanford Cars (SCars). The Stanford Cars dataset contains 16,185 images of 196 categories of automobiles. For training, the color images were converted to grayscale images of size 96 × 97 pixels.

Algorithm 1 Training of the ELM-AE with FISTA, G-FISTA, and LC-FISTA.

Require: training set $\{\boldsymbol{x}_i\}_{i=1}^{N}$, maximum number of iterations I, stopping criterion P, number of neurons L, regularization parameter λ.

1: Random and orthogonal assignment of weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ in the hidden layer.
2: Calculation of the matrix $\boldsymbol{H}$ evaluating g in the terms $(\boldsymbol{x}_i, \boldsymbol{W}, \boldsymbol{b})$.
3: **while** P does not occur **do**
4: Calculate the optimal value of (4) using the G-FISTA or LC-FISTA algorithms for $f(\boldsymbol{\beta}) = \|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{X}\|_2^2$ and $\phi(\boldsymbol{\beta}) = \lambda|\boldsymbol{\beta}|_1$.
5: **end while**

Table 1 shows the numbers of training and test data, the numbers of categories, and the sizes of the input vectors for CIFAR10, UMist, MNIST, NORB, Stanford Cars, and Caltech256.

Table 1. Information from the databases to train the ELM-AE.

Dataset	# Training	# Testing	# Features	# Category
UMist	400	175	10,304	20
CIFAR10	50,000	10,000	1024	10
MNIST	60,000	10,000	784	10
NORB	24,300	24,300	18,432	5
SCars	8144	8041	9312	196
Caltech256	21,314	9293	10,000	257

4.2. Selection of Evaluation Metrics

To analyze the performance of the proposed algorithm, we used the root mean square error (RMSE), mean absolute error (MAE), R^2 score, index of agreement (AI), Theil inequality coefficient (TIC), minimum error (MIN), and maximum error (MAX), as defined in Table 2 [53]. We used the RMSE and MAE metrics to represent the average error between the estimated and actual values. The lower the values of these measures, the higher the efficiency of the autoencoder.

Table 2. Seven evaluation criteria to evaluate the performance of the proposed method.

Metric	Definition	Mathematical Expression
RMSE	Root mean square error	$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i^a - y_i^e)^2}{n}}$
MAE	Mean absolute error	$\text{MAE} = \frac{\sum_{i=1}^{n}\lvert y_i^a - y_i^e\rvert}{n}$
R^2	R^2 score	$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i^a - y_i^e)^2}{\sum_{i=1}^{n}(y_i^a - \overline{y}^a)^2}$
IA	Willmott's index of agreement	$\text{IA} = 1 - \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i^a - y_i^e)^2}}{\sum_{i=1}^{n}(\lvert y_i^e - \overline{y}^a\rvert + \lvert y_i^a - \overline{y}^a\rvert)}$
TIC	Theil inequality coefficient	$\text{TIC} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i^a - y_i^e)^2}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i^a)^2} + \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i^e)^2}}$
MIN	Minimum error	$\text{MIN} = \min\lvert y_i^a - y_i^e\rvert,\ i = 1, \ldots, n$
MAX	Maximum error	$\text{MAX} = \max\lvert y_i^a - y_i^e\rvert,\ i = 1, \ldots, n$

Note: $\boldsymbol{y}^a$ and $\boldsymbol{y}^e$ denote the vectors of actual and estimated values, respectively, $\overline{y}^a$ is the average of the actual values, and n is the number of pairs of y_i^a and y_i^e.

The R^2 score is a statistical measure that is defined on the domain $[0, 1]$, where a value of one indicates perfect prediction and zero an inefficient model [54]. In contrast, the TIC metric indicates the accuracy capability of the proposed system, with a value between zero and one. The closer the value is to zero, the higher the predictive efficiency of the ELM-AE method. Likewise, the IA prediction index can provide external statistical information on the predictive ability of the model. To verify the efficiency of the ELM-AE, each experiment was repeated 10 times, and the average values of the training times and the evaluation criteria represented in Table 2 were recorded.

4.3. Experimental Setup

In each experiment, the training and test data were standardized to the range $[0, 1]$ using the following equation:

$$y_{\text{norm}}^a = \frac{y_i^a - y_{\text{min}}^a}{y_{\text{max}}^a - y_{\text{min}}^a} \tag{9}$$

where y_i^a is the actual value to be standardized, y_{min}^a and y_{max}^a are the minimum and maximum of the actual values. In the hidden layer, the weights and biases are random values generated by means of a uniform distribution in the ranges $[-1, 1]$ and $[0, 1]$, and the sigmoid $g(x) = \frac{1}{1+e^{-x}}$ was used as the activation function. We chose this activation function due to its universal approximation capability in ELM networks.

The algorithm presented in Section 3 has the following parameters that need to be configured: the number of neurons L, the stopping criterion P, the maximum num-

ber of iterations I, and the regularization term λ. In the first scenario of the proposed method, the regularization parameter set to $\lambda = \frac{1}{2}\sqrt{\frac{2\log(L)}{N}}$, the stopping criterion P is expressed as $\frac{\|H\beta_k - X\|}{\|X\|} < \varepsilon$ and the maximum number of iterations is $I = 50$. The threshold ε is a user-specified value that is applied to interrupt the iteration of the algorithm. For the UMist, CIFAR10, MNIST, NORB, SCars, and Caltech256 databases, we set $\varepsilon = 0.30, 0.32, 0.60, 0.13, 0.40$ and 0.40, respectively. Moreover, the practical application of the FISTA, G-FISTA, and LC-FISTA algorithms for training of the ELM-AE requires tuning of the L_f Lipschitz constant, and the strong convexity constant μ must be tuned for the LC-FISTA algorithm. For the convex function $f(\boldsymbol{\beta}) = \|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{X}\|_2^2$, we chose values of $L_f = \lambda_{\max}(\boldsymbol{H}^T\boldsymbol{H})$ and $\mu = \lambda_{\min}(\boldsymbol{H}^T\boldsymbol{H})$, where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum proper values of $\boldsymbol{H}^T\boldsymbol{H}$. The additional control parameters in G-FISTA were described in Section 2.2.2.

In a second scenario, we evaluated the sensitivity of the regularization parameter λ on the performance of our method for the UMist, NORB, Stanford Cars, and Caltech256 datasets. This tuning term was chosen using cross-validation for each algorithm and database. Table 3 shows the optimal value of λ selected of $\{10^{-5}, 10^{-4}, \ldots, 10^4, 10^5\}$, obtained by comparing the RMSE between the generated models, chosen as the λ that gives a lower RMSE. The experiment only considered the case $L = 2000$ for the selected databases. The other parameters of the FISTA, G-FISTA, and LC-FISTA algorithms follow the same selection criteria of the first scenario. For example, the steplength for FISTA and LC-FISTA has been $\rho = \frac{1}{L_f}$, but for G-FISTA, $\rho = \frac{1.3}{L_f}$, $\xi = 0.96$, and $S = 1.1$. The Lipschitz constant L_f and the strong convexity parameter μ of f are taken as the maximum and minimum eigenvalue of the matrix $\boldsymbol{H}^T\boldsymbol{H}$. The mathematical analysis and numerical experiments presented in [42–45,49] motivated the choice of our parameters.

Table 3. Choice of λ for each algorithm and selected database.

Algorithm	Regularization Parameter λ			
	UMist	**NORB**	**SCars**	**Caltech256**
FISTA	0.001	10	0.001	1
G-FISTA	0.001	100	0.01	0.1
LC-FISTA	0.01	100	0.1	0.01

To avoid the problem of overfitting when the number of neurons is increased, we applied a 10-fold cross-validation technique. This control tool allowed us to restrict the training to a certain number of neurons. We then used the test set specified in Table 1 and the metrics defined in Table 2 to evaluate the performance of each algorithm. Figure 1 shows the values of the RMSE for the training, validation, and test sets for the G-FISTA algorithm. A graphical view is not shown for FISTA and LC-FISTA since all three algorithms had a comparable level of accuracy. According to the above analysis, $L = 900$ and $L = 1300$ were the maximum neuron thresholds that could be assigned in ELM-AE for the MNIST and CIFAR10 databases.

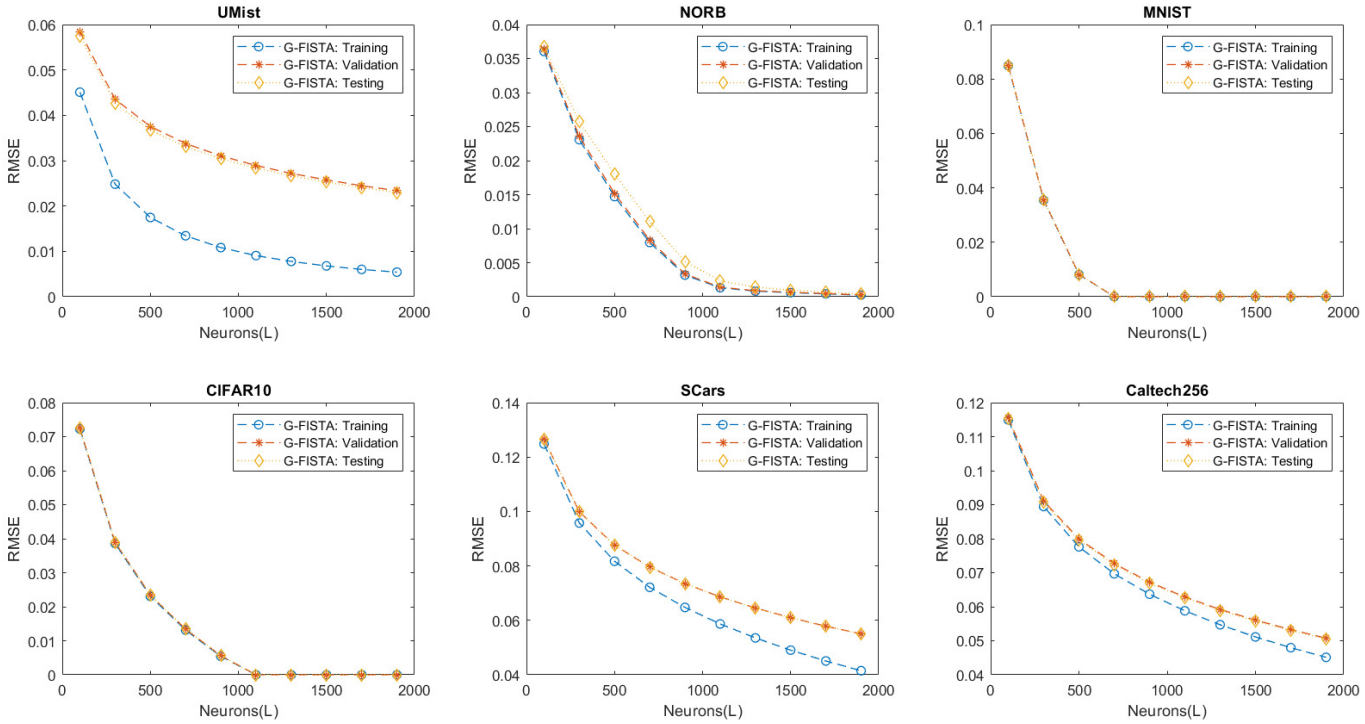

Figure 1. G-FISTA Algorithm: plot of RMSE for training, validation, and testing sets versus the number of neurons.

4.4. Representation of Features with Different Numbers of Neurons

The computational complexity and execution time of the proposed method are explored in this section. For this purpose, several cases are considered with varying numbers of neurons in the hidden layer of the ELM-AE. In this phase of the experiments, we selected the measures of training time, RMSE, MAE, R^2, IA, TIC, MIN, and MAX for comparison on the test set. These evaluation indicators were obtained as a result of training the ELM-AE with the shrinkage-thresholding class of algorithms introduced in Section 2.2.

Table 4 presents the training times and values of RMSE for the ELM-AE on each of the six datasets, for increasing numbers of neurons in the hidden layer. These experimental results show that the training speeds achieved by G-FISTA and LC-FISTA were lower than that obtained by FISTA. For example, for the CIFAR10 set, with 100 neurons in the hidden layer, the computation time fo rthe ELM trained by FISTA was 21.92 s, while the G-FISTA and LC-FISTA algorithms achieved average values of 12.31 s and 12.91 s. Likewise, when we increased the number of hidden neurons (L), the training time obtained by G-FISTA and LC-FISTA was approximately half that required by FISTA. At the test stage, the RMSE metric achieved by the G-FISTA and LC-FISTA algorithms proposed in this paper indicated a similar level of error as the current FISTA algorithm. This evaluation metric became smaller as the parameter L was increased. For MNIST and CIFAR10, we obtained evaluation metrics of up to $L = 500$ and $L = 1000$, respectively.

Table 4. Training times and RMSE values achieved by FISTA, G-FISTA, and LC-FISTA for training of the ELM-AE.

L	Algorithm	RMSE						Training Time (Seg)					
		CIFAR10	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**	**CIFAR10**	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**
100	FISTA	0.0751	0.0583	0.0856	0.0385	0.1284	0.1205	21.92	2.92	25.47	22.33	38.22	63.61
	G-FISTA	0.0753	0.0580	0.0856	0.0385	0.1285	0.1205	12.31	2.40	16.73	12.09	18.73	35.73
	LC-FISTA	0.0751	0.0579	0.0855	0.0385	0.1285	0.1205	12.91	2.34	18.97	11.17	20.61	33.03
500	FISTA	0.0242	0.0368	0.0096	0.0193	0.088	0.0836	48.46	11.42	51.24	48.40	83.23	122.15
	G-FISTA	0.0242	0.0373	0.0096	0.0193	0.088	0.0836	26.97	10.27	32.06	26.49	42.16	75.94
	LC-FISTA	0.0242	0.0370	0.0096	0.0193	0.088	0.0837	27.61	9.49	35.92	24.26	42.52	69.93
1000	FISTA	0.0019	0.0288	—	0.0044	0.0718	0.0681	78.80	29.78	—	80.18	153.88	211.51
	G-FISTA	0.0019	0.0298	—	0.0044	0.0718	0.0681	43.14	26.82	—	46.54	80.08	125.85
	LC-FISTA	0.0019	0.0294	—	0.0044	0.0718	0.0681	44.83	25.07	—	42.34	78.60	118.25
1500	FISTA	—	0.0244	—	0.0015	0.0617	0.0589	—	57.79	—	116.54	221.72	285.91
	G-FISTA	—	0.0257	—	0.0015	0.0617	0.0589	—	48.39	—	69.18	117.10	176.85
	LC-FISTA	—	0.0253	—	0.0015	0.0617	0.0589	—	46.48	—	63.09	116.60	162.03
2000	FISTA	—	0.0214	—	0.0003	0.0543	0.0522	—	96.09	—	161.86	297.65	379.08
	G-FISTA	—	0.0229	—	0.0003	0.0543	0.0522	—	78.74	—	95.27	158.85	239.25
	LC-FISTA	—	0.0225	—	0.0003	0.0543	0.0522	—	75.55	—	86.17	162.87	222.65

For each dataset, we show the execution time of the proposed method in Figure 2. A bar chart represents this evaluation measure as a function of the number of neurons. The blue, red, and orange bars indicate the training times obtained by FISTA, G-FISTA, and LC-FISTA, respectively. From the results illustrated in Figure 2, we infer that the proposed G-FISTA and LC-FISTA algorithms are faster than FISTA for training the ELM-AE.

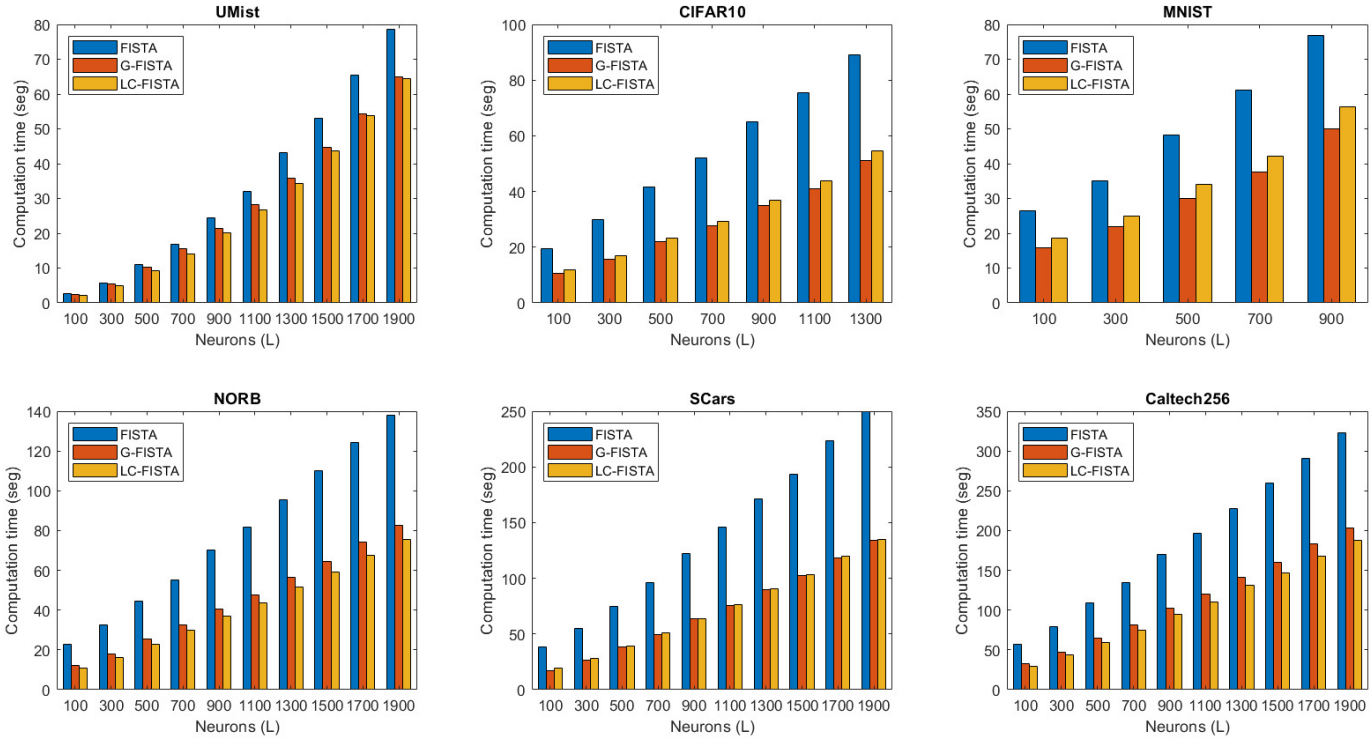

Figure 2. Bar graph representing the average training times required by FISTA, G-FISTA, and LC-FISTA for the ELM-AE.

In the following, we demonstrate the efficiency of the proposed method using four evaluation criteria: the MAE, R^2 score, IA, and TIC. A summary of the MAE and R^2 values is given in Table 5. The closer the value of MAE to zero, and the closer the value of R^2 to one, the better the performance of the autoencoder. From the values of the metrics given in Table 5, we can see that the order of magnitude is similar, and there is no significant difference between the shrinkage-thresholding methods, meaning that the proposed method maintains the same level of accuracy as the FISTA algorithm. Taking the UMist database as an example, we see that when $L = 1500$, the average MAE values obtained by FISTA, G-FISTA, and LC-FISTA are 0.0172, 0.0181, and 0.0178, while their R^2 scores are 0.9861, 0.9846, and 0.9851, respectively. The IA and TIA metrics are also important for evaluating the capability of the proposed learning system. These fit metrics are described in Table 6. As the average values of the IA and TIA approach one and zero, respectively, the feature representation improves.

Table 7 shows the MIN and MAX statistical indices for the proposed model on all the selected databases. This table indicates the minimum and maximum absolute errors for the ELM-AE in the testing stage. From the results of this experimental analysis, we can conclude that the FISTA algorithm has a similar generalization capability as G-FISTA and LC-FISTA.

Table 5. MAE and R^2 score values achieved by FISTA, G-FISTA and LC-FISTA for training of the ELM-AE.

L	Algorithm	MAE						R^2 Score					
		CIFAR10	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**	**CIFAR10**	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**
100	FISTA	0.0546	0.0415	0.0489	0.0184	0.0921	0.0837	0.9009	0.9209	0.9241	0.9533	0.7896	0.8426
	G-FISTA	0.0547	0.0413	0.0488	0.0185	0.0921	0.0837	0.9005	0.9217	0.9240	0.9533	0.7895	0.8426
	LC-FISTA	0.0547	0.0412	0.0488	0.0185	0.0921	0.0838	0.9005	0.9218	0.9241	0.9532	0.7896	0.8425
500	FISTA	0.0181	0.0256	0.0046	0.0101	0.0625	0.0566	0.9894	0.9684	0.9991	0.9883	0.8994	0.9241
	G-FISTA	0.0181	0.0258	0.0046	0.0101	0.0625	0.0566	0.9894	0.9677	0.9991	0.9883	0.8993	0.9241
	LC-FISTA	0.0181	0.0256	0.0047	0.0102	0.0625	0.0566	0.9894	0.9681	0.9991	0.9883	0.8994	0.9241
1000	FISTA	0.0014	0.0201	—	0.0018	0.0508	0.0461	0.9999	0.9807	—	0.9994	0.9342	0.9497
	G-FISTA	0.0014	0.0208	—	0.0018	0.0508	0.0461	0.9999	0.9793	—	0.9994	0.9342	0.9497
	LC-FISTA	0.0014	0.0205	—	0.0018	0.0508	0.0461	0.9999	0.9798	—	0.9994	0.9342	0.9497
1500	FISTA	—	0.0172	—	0.0005	0.0440	0.0401	—	0.9861	—	0.9999	0.9515	0.9624
	G-FISTA	—	0.0181	—	0.0005	0.0440	0.0401	—	0.9846	—	0.9999	0.9515	0.9624
	LC-FISTA	—	0.0178	—	0.0005	0.0440	0.0401	—	0.9851	—	0.9999	0.9515	0.9624
2000	FISTA	—	0.0152	—	0.0001	0.0391	0.0358	—	0.9893	—	1.0000	0.9624	0.9704
	G-FISTA	—	0.0163	—	0.0001	0.0391	0.0358	—	0.9878	—	1.0000	0.9624	0.9704
	LC-FISTA	—	0.0160	—	0.0001	0.0391	0.0358	—	0.9882	—	1.0000	0.9624	0.9704

Table 6. IA and TIA values achieved by FISTA, G-FISTA and LC-FISTA for training of the ELM-AE.

L	Algorithm	IA						TIC					
		CIFAR10	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**	**CIFAR10**	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**
100	FISTA	0.9734	0.9790	0.9799	0.9879	0.9386	0.9559	0.0700	0.0712	0.1288	0.0247	0.1207	0.0981
	G-FISTA	0.9732	0.9793	0.9799	0.9879	0.9386	0.9559	0.0701	0.0708	0.1288	0.0247	0.1208	0.0981
	LC-FISTA	0.9733	0.9793	0.9800	0.9879	0.9386	0.9559	0.0701	0.0708	0.1288	0.0247	0.1208	0.0981
500	FISTA	0.9973	0.9917	0.9998	0.9971	0.9729	0.9800	0.0228	0.0449	0.0121	0.0123	0.0829	0.0678
	G-FISTA	0.9973	0.9917	0.9998	0.9971	0.9729	0.9800	0.0228	0.0454	0.0121	0.0123	0.0829	0.0678
	LC-FISTA	0.9973	0.9918	0.9998	0.9971	0.9729	0.9800	0.0228	0.0451	0.0121	0.0123	0.0829	0.0678
1000	FISTA	1.0000	0.9951	—	0.9999	0.9827	0.9869	0.0018	0.0350	—	0.0028	0.0668	0.0551
	G-F ISTA	1.0000	0.9948	—	0.9999	0.9827	0.9870	0.0018	0.0363	—	0.0028	0.0668	0.0551
	LC-FISTA	1.0000	0.9949	—	0.9999	0.9827	0.9869	0.0018	0.0358	—	0.0028	0.0669	0.0551

Table 6. *Cont.*

L	Algorithm	IA						TIC					
		CIFAR10	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**	**CIFAR10**	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**
1500	FISTA	—	0.9965	—	1.0000	0.9874	0.9903	—	0.0297	—	9.81×10^{-4}	0.0573	0.0476
	G-FISTA	—	0.9961	—	1.0000	0.9874	0.9903	—	0.0313	—	9.79×10^{-4}	0.0573	0.0476
	LC-FISTA	—	0.9962	—	1.0000	0.9874	0.9903	—	0.0308	—	9.81×10^{-4}	0.0573	0.0476
2000	FISTA	—	0.9973	—	1.0000	0.9903	0.9924	—	0.0261	—	2.16×10^{-4}	0.0505	0.0422
	G-FISTA	—	0.9969	—	1.0000	0.9903	0.9924	—	0.0279	—	2.15×10^{-4}	0.0505	0.0422
	LC-FISTA	—	0.9970	—	1.0000	0.9903	0.9924	—	0.0273	—	2.16×10^{-4}	0.0505	0.0422

Table 7. MIN and MAX values achieved by FISTA, G-FISTA and LC-FISTA for training of the ELM-AE.

L	Algorithm	MIN						MAX					
		CIFAR10	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**	**CIFAR10**	**UMist**	**MNIST**	**NORB**	**SCars**	**Caltech256**
100	FISTA	0.0093	0.0251	0.022	0.066	0.0204	0.0059	0.1225	0.0743	0.933	0.0531	0.1843	0.2533
	G-FISTA	0.0095	0.0254	0.0224	0.0065	0.0206	0.0061	0.1211	0.0742	0.0953	0.0530	0.1818	0.2505
	LC-FISTA	0.0093	0.0250	0.0227	0.0066	0.0206	0.0060	0.1227	0.0742	0.0917	0.0533	0.1809	0.2528
500	FISTA	0.0028	0.0127	0.0010	0.0038	0.0143	0.0040	0.0563	0.0465	0.0277	0.0346	0.1292	0.1814
	G-FISTA	0.0027	0.0114	0.0009	0.0034	0.0143	0.0040	0.0564	0.0471	0.075	0.0348	0.1283	0.1811
	LC-FISTA	0.0028	0.0112	0.0009	0.0033	0.0144	0.0040	0.0563	0.0465	0.0278	0.0349	0.1290	0.1815
1000	FISTA	0.0006	0.0069	—	0.0005	0.0124	0.0028	0.0175	0.0357	—	0.0176	0.1141	0.1599
	G-FISTA	0.0006	0.0063	—	0.0005	0.0123	0.0028	0.0177	0.0373	—	0.0178	0.1138	0.1605
	LC-FISTA	0.0006	0.0058	—	0.0005	0.0123	0.0028	0.0176	0.0368	—	0.0177	0.1136	0.1602
1500	FISTA	—	0.0038	—	5.3×10^{-5}	0.0105	0.0020	—	0.0281	—	0.0084	0.1010	0.1382
	G-FISTA	—	0.0037	—	5.4×10^{-5}	0.0105	0.0020	—	0.0303	—	0.0082	0.1010	0.1380
	LC-FISTA	—	0.0033	—	5.4×10^{-5}	0.0105	0.0020	—	0.0295	—	0.0082	0.1012	0.1375
2000	FISTA	—	0.0028	—	1.7×10^{-5}	0.0095	0.0017	—	0.0252	—	0.0047	0.0950	0.1244
	G-FISTA	—	0.0029	—	1.7×10^{-5}	0.0095	0.0017	—	0.0274	—	0.0048	0.0948	0.1249
	LC-FISTA	—	0.0025	—	1.6×10^{-5}	0.0094	0.0017	—	0.0267	—	0.0047	0.0950	0.1244

The best tuning of the parameter λ in ELM-AE usually influences its generalization capacity. With the λ in Table 3 chosen using cross-validation, we show in Table 8 the performance of our method for the particular case $L = 2000$. The values of RMSE and the training time are the two indicators selected to compare the efficiency of FISTA, G-FISTA, and LC-FISTA in ELM-AE. The time shown in Table 8 is the execution time required to select the parameter λ. Note that G-FISTA and LC-FISTA maintain the reconstruction capability of FISTA but perform better in terms of computational time. In addition, the values of RMSE in Tables 4 and 8 for the first and second scenarios are comparable measures. Therefore, the λ chosen in the first scenario and the additional parameters may be adequate in large-scale application problems since cross-validation is not required.

Table 8. The performance of each algorithm for the λ chosen in Table 3.

Algorithm	RMSE				Training Time (Minutes)			
	UMist	NORB	SCars	Caltech256	UMist	NORB	SCars	Caltech256
FISTA	0.0211	0.0001	0.0537	0.0494	49	21	44	139
G-FISTA	0.0225	0.0001	0.0537	0.0494	40	12	27	85
LC-FISTA	0.0220	0.0001	0.0537	0.0494	37	11	26	70

5. Discussion

The main goal of an autoencoder is to learn a compressed representation of the input and then reconstruct it within a reasonable computation time. This study proposes a robust sparse optimization-based method for training ELM-AE networks. With the control parameters specified in the results section, we see that the training times for G-FISTA and LC-FISTA are less than for FISTA. Table 9 shows these important results in terms of percentages. For the CIFAR10, UMist, MNIST, NORB, SCars, and Caltech256 databases, the average computation times for G-FISTA are 44.48%, 14.42%, 35.87%, 42.97%, 48.42%, and 39.43% faster than for FISTA, respectively. The LC-FISTA algorithm achieved speedups of 42.41%, 18.71%, 27.70%, 47.93%, 47.32%, and 43.90%. In addition, the indicators recorded in Tables 4–8 are coherent, and show comparable performance for all databases. The proposed training scheme is an important automatic learning method since it can help make accurate decisions in shorter periods. Consequently, incorporating fast optimization techniques into the autoencoder architecture can help improve the performance of current models, and particularly the first-order methods of the shrinkage-thresholding class.

Table 9. Average speeds of the G-FISTA and LC-FISTA algorithms compared with FISTA for training of the ELM-AE.

L	Algorithm	Training Time (Seg)					
		CIFAR10	UMist	MNIST	NORB	SCars	Caltech256
100	G-FISTA	43.84	17.80	34.31	45.85	50.99	43.82
	LC-FISTA	41.10	19.86	25.52	49.97	46.07	48.07
500	G-FISTA	44.34	10.07	37.43	45.26	49.34	37.83
	LC-FISTA	43.02	16.90	29.89	49.87	48.91	42.75
1000	G-FISTA	45.25	9.93	—	41.95	47.95	40.49
	LC-FISTA	43.10	15.81	—	47.19	48.92	44.09
1500	G-FISTA	—	16.26	—	40.63	47.18	38.14
	LC-FISTA	—	19.57	—	45.86	47.41	43.32
2000	G-FISTA	—	18.05	—	41.14	46.63	36.88
	LC-FISTA	—	21.37	—	46.76	45.28	41.26
Av. time	G-FISTA	44.48%	14.42%	35.87%	42.97%	48.42%	39.43%
Av. time	LC-FISTA	42.41%	18.71%	27.70%	47.93%	47.32%	43.90%

Note: The formula $\% = (1 - A_1/A_2) \times 100$ compares the computation time of algorithms A_1 and A_2.

The proximal step of FISTA requires the main computational effort, and this also holds true for G-FISTA and LC-FISTA. The additional computation required in G-FISTA and LC-FISTA increases the convergence speed, but the execution cost is higher. However, this problem can be controlled, since G-FISTA and LC-FISTA converge with fewer iterations.

In the experimental evaluation described in Section 4, we raised the complexity of the problem by increasing the number of neurons in the hidden layer to get a clearer idea of the performance when solving applications involving higher dimensional datasets. Based on this evaluation criterion, we see that the proposed method can be configured to solve large-scale problems, since the vector operations involved in the ELM-AE architecture and the proposed mechanism are block-separable.

6. Conclusions

This paper has proposed the G-FISTA and LC-FISTA algorithms to reduce the training time of ELM-AEs. The performance of our methods was compared to FISTA, an algorithm of the shrinkage-thresholding class, which is a state-of-the-art method of training ELM-AEs.

Experiments were conducted on six databases: CIFAR10, UMist, MNIST, NORB, Stanford Cars, and Caltech256. The results showed that G-FISTA and LC-FISTA achieved similar computational times for the training of the autoencoder, both of which were significantly lower than FISTA. In numerical terms, G-FISTA and LC-FISTA were 37.60% and 37.99% faster than FISTA in training the ELM-AE (average value obtained from Table 9). Consequently, the advantage main of our study is to maintain the generalization capability of FISTA while the computational speed is improved. A possible disadvantage may be the hyperparameter settings, but many researchers set these values according to existing mathematical analysis in the literature to control runtime. Other advantages and disadvantages we will discuss in future perspectives.

Motivated by technological advances and the speed of convergence of shrinkage-thresholding algorithms, we intend to extend our methodology in further research to a parallel and distributed architecture. This approach will be of great interest for solving current large-scale applications in computer clusters and clouds. In addition, we aim to incorporate iterative schemes of the shrinkage-thresholding class into the architecture of a variational, convolutional deep autoencoder network to solve optimization problems with large datasets.

Author Contributions: Conceptualization, J.A.V.-C., M.M. and K.V.; methodology, J.A.V.-C. and M.M.; software, J.A.V.-C.; experimental execution and validation, J.A.V.-C.; research, J.A.V.-C., M.M. and K.V.; resources, M.M.; writing—review and editing, J.A.V.-C., M.M. and K.V.; scientific project coordination, M.M. and K.V.; funding acquisition, M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by Universidad Católica del Maule Doctoral Studies Scholarship 2019 (Beca Doctoral Universidad Católica del Maule 2019).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This paper is one of the scientific results of the project FONDECYT REGULAR 2020 N° 1200810 Very Large Fingerprint Classification based on a Fast and Distributed Extreme Learning Machine, National Research and Development Agency, Ministry of Science, Technology, Knowledge and Innovation, Government of Chile.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

ELM	Extreme learning machine
ELM-AE	Extreme learning machine autoencoder
K-ELM	Kernel-ELM
FISTA	Fast iterative shrinkage-thresholding algorithm
G-FISTA	Greedy FISTA
LC-FISTA	Linearly convergent FISTA
RMSE	Root mean square error
MAE	Mean absolute error
R^2	R^2 score
IA	Index of agreement
TIC	Theil inequality coefficient
MIN	Minimum error
MAX	Maximum error
PSO	Particle swarm optimization
CPU	Central processing unit
GPU	Graphics processing unit

References

1. He, X.; Ji, M.; Zhang, C.; Bao, H. A variance minimization criterion to feature selection using laplacian regularization. *IEEE Trans. Pattern Anal. Mach. Intell.* **2011**, *33*, 2013–2025. [PubMed]
2. Dong, G.; Liao, G.; Liu, H.; Kuang, G. A review of the autoencoder and its variants: A comparative perspective from target recognition in synthetic-aperture radar images. *IEEE Geosci. Remote Sens. Mag.* **2018**, *6*, 44–68. [CrossRef]
3. Leijnen, S.; Veen, F.V. The Neural Network Zoo. *Proceedings* **2020**, *47*, 9.
4. Baldi, P.; Sadowski, P.; Lu, Z. Learning in the machine: Random backpropagation and the deep learning channel. *Artif. Intell.* **2018**, *260*, 1–35. [CrossRef]
5. Meng, L.; Ding, S.; Xue, Y. Research on denoising sparse autoencoder. *Int. J. Mach. Learn. Cybern.* **2017**, *8*, 1719–1729. [CrossRef]
6. Meng, L.; Ding, S.; Zhang, N.; Zhang, J. Research of stacked denoising sparse autoencoder. *Neural Comput. Appl.* **2018**, *30*, 2083–2100. [CrossRef]
7. Li, R.; Li, S.; Xu, K.; Li, X.; Lu, J.; Zeng, M. A Novel Symmetric Stacked Autoencoder for Adversarial Domain Adaptation Under Variable Speed. *IEEE Access* **2022**, *10*, 24678–24689. [CrossRef]
8. Luo, X.; Li, X.; Wang, Z.; Liang, J. Discriminant autoencoder for feature extraction in fault diagnosis. *Chemom. Intell. Lab. Syst.* **2019**, *192*, 103814. [CrossRef]
9. Soydaner, D. Hyper Autoencoders. *Neural Process. Lett.* **2020**, *52*, 1395–1413. [CrossRef]
10. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [CrossRef]
11. Huang, G.; Song, S.; Gupta, J.N.D.; Wu, C. Semi-Supervised and Unsupervised Extreme Learning Machines. *IEEE Trans. Cybern.* **2014**, *44*, 2405–2417. [CrossRef] [PubMed]
12. Chen, J.; Zeng, Y.; Li, Y.; Huang, G.B. Unsupervised feature selection based extreme learning machine for clustering. *Neurocomputing* **2020**, *386*, 198–207. [CrossRef]
13. Huang, G.; Huang, G.B.; Song, S.; You, K. Trends in extreme learning machines: A review. *Neural Netw.* **2015**, *61*, 32–48. [CrossRef] [PubMed]
14. Huang, G.B.; Zhou, H.; Ding, X.; Zhang, R. Extreme learning machine for regression and multiclass classification. *IEEE Trans. Syst. Man Cybern. Part B-Cybern.* **2012**, *42*, 513–529. [CrossRef]
15. Bai, Z.; Huang, G.B.; Wang, D.; Wang, H.; Westover, M.B. Sparse extreme learning machine for classification. *IEEE Trans. Cybern.* **2014**, *44*, 1858–1870. [CrossRef]
16. Liang, N.Y.; Huang, G.B.; Saratchandran, P.; Sundararajan, N. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Trans. Neural Netw.* **2006**, *17*, 1411–1423. [CrossRef]
17. Ma, J.; Yu, S.; Cheng, W. Composite Fault Diagnosis of Rolling Bearing Based on Chaotic Honey Badger Algorithm Optimizing VMD and ELM. *Machines* **2022**, *10*, 469. [CrossRef]
18. Cui, H.; Guan, Y.; Chen, H. Rolling element fault diagnosis based on VMD and sensitivity MCKD. *IEEE Access* **2021**, *9*, 120297–120308. [CrossRef]
19. An, Z.; Wang, X.; Li, B.; Xiang, Z.; Zhang, B. Robust visual tracking for UAVs with dynamic feature weight selection. *Appl. Intell.* **2022**. [CrossRef]
20. Eshtay, M.; Faris, H.; Obeid, N. Metaheuristic-based extreme learning machines: a review of design formulations and applications. *Int. J. Mach. Learn. Cybern.* **2019**, *10*, 1543–1561. [CrossRef]

21. Li, G.; Li, Y.; Chen, H.; Deng, W. Fractional-order controller for course-keeping of underactuated surface vessels based on frequency domain specification and improved particle swarm optimization algorithm. *Appl. Sci.* **2022**, *12*, 3139. [CrossRef]
22. Zhou, X.; Ma, H.; Gu, J.; Chen, H.; Deng, W. Parameter adaptation-based ant colony optimization with dynamic hybrid mechanism. *Eng. Appl. Artif. Intell.* **2022**, *114*, 105–139. [CrossRef]
23. Ali, M.; Deo, R.C.; Xiang, Y.; Prasad, R.; Li, J.; Farooque, A.; Yaseen, Z.M. Coupled online sequential extreme learning machine model with ant colony optimization algorithm for wheat yield prediction. *Sci. Rep.* **2022**, *12*, 5488. [CrossRef]
24. Chen, H.; Miao, F.; Chen, Y.; Xiong, Y.; Chen, T. A hyperspectral image classification method using multifeature vectors and optimized KELM. *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.* **2021**, *14*, 2781–2795. [CrossRef]
25. Chamara, L.; Zhou, H.; Huang, G.B.; Vong, C.M. Representation learning with extreme learning machine for big data. *IEEE Intell. Syst.* **2013**, *28*, 31–34.
26. Tang, J.; Deng, C.; Huang, G.B. Extreme learning machine for multilayer perceptron. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *27*, 809–821. [CrossRef]
27. Sun, K.; Zhang, J.; Zhang, C.; Hu, J. Generalized extreme learning machine autoencoder and a new deep neural network. *Neurocomputing* **2017**, *230*, 374–381. [CrossRef]
28. Ge, H.; Sun, W.; Zhao, M.; Yao, Y. Stacked denoising extreme learning machine autoencoder based on graph embedding for feature representation. *IEEE Access* **2019**, *7*, 13433–13444. [CrossRef]
29. Wang, J.; Guo, P.; Li, Y. DensePILAE: a feature reuse pseudoinverse learning algorithm for deep stacked autoencoder. *Complex Intell. Syst.* **2021**, *8*, 2039–2049. [CrossRef]
30. Li, R.; Wang, X.; Lei, L.; Wu, C. Representation learning by hierarchical ELM auto-encoder with double random hidden layers. *IET Comput. Vis.* **2019**, *13*, 411–419. [CrossRef]
31. Li, R.; Wang, X.; Song, Y.; Lei, L. Hierarchical extreme learning machine with L21-norm loss and regularization. *Int. J. Mach. Learn. Cybern.* **2021**, *12*, 1297–1310. [CrossRef]
32. Liangjun, C.; Honeine, P.; Hua, Q.; Jihong, Z.; Xia, S. Correntropy-based robust multilayer extreme learning machines. *Pattern Recognit.* **2018**, *84*, 357–370. [CrossRef]
33. Wong, C.M.; Vong, C.M.; Wong, P.K.; Cao, J. Kernel-based multilayer extreme learning machines for representation learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 757–762. [CrossRef] [PubMed]
34. Vong, C.M.; Chen, C.; Wong, P.K. Empirical kernel map-based multilayer extreme learning machines for representation learning. *Neurocomputing* **2018**, *310*, 265–276. [CrossRef]
35. Paul, A.N.; Yan, P.; Yang, Y.; Zhang, H.; Du, S.; Wu, Q. Non-iterative online sequential learning strategy for autoencoder and classifier. *Neural Comput. Appl.* **2021**, *33*, 16345–16361. [CrossRef]
36. Mirza, B.; Kok, S.; Dong, F. Multi-layer online sequential extreme learning machine for image classification. In *Proceedings of ELM-2015 Volume 1*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 39–49.
37. Tibshirani, R. Regression shrinkage and selection via the Lasso. *J. R. Stat. Soc. B Methodol.* **1996**, *58*, 267–288. [CrossRef]
38. Beck, A.; Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.* **2009**, *2*, 183–202. [CrossRef]
39. Jiang, X.; Yan, T.; Zhu, J.; He, B.; Li, W.; Du, H.; Sun, S. Densely connected deep extreme learning machine algorithm. *Cogn. Comput.* **2020**, *12*, 979–990. [CrossRef]
40. Zhao, H.; Ding, S.; Li, X.; Huang, H. Deep neural network structured sparse coding for online processing. *IEEE Access* **2018**, *6*, 74778–74791. [CrossRef]
41. Liu, D.; Liang, C.; Chen, S.; Tie, Y.; Qi, L. Auto-encoder based structured dictionary learning for visual classification. *Neurocomputing* **2021**, *438*, 34–43. [CrossRef]
42. Janngam, K.; Wattanataweekul, R. A New Accelerated Fixed-Point Algorithm for Classification and Convex Minimization Problems in Hilbert Spaces with Directed Graphs. *Symmetry* **2022**, *14*, 1059. [CrossRef]
43. Chumpungam, D.; Sarnmeta, P.; Suantai, S. An Accelerated Convex Optimization Algorithm with Line Search and Applications in Machine Learning. *Mathematics* **2022**, *10*, 1491. [CrossRef]
44. Liang, J.; Luo, T.; Schönlieb, C.B. Improving "Fast Iterative Shrinkage-Thresholding Algorithm": Faster, Smarter, and Greedier. *SIAM J. Sci. Comput.* **2022**, *44*, A1069–A1091. [CrossRef]
45. Bussaban, L.; Kaewkhao, A.; Suantai, S. Inertial s-iteration forward-backward algorithm for a family of nonexpansive operators with applications to image restoration problems. *Filomat* **2021**, *35*, 771–782. [CrossRef]
46. Chambolle, A.; Dossal, C. On the convergence of the iterates of the "fast iterative shrinkage/thresholding algorithm". *J. Optim. Theory Appl.* **2015**, *166*, 968–982. [CrossRef]
47. Bartlett, P. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Trans. Inf. Theory* **1998**, *44*. [CrossRef]
48. Beck, A.; Teboulle, M. Gradient-based algorithms with applications to signal recovery. In *Convex Optimization in Signal Processing and Communications*; Cambridge University Press: Cambridge, UK, 2009; pp. 42–88.
49. Chen, L.; Xiao, Y.; Yang, T. Application of the improved fast iterative shrinkage-thresholding algorithms in sound source localization. *Appl. Acoust.* **2021**, *180*, 108101. [CrossRef]
50. Calatroni, L.; Chambolle, A. Backtracking strategies for accelerated descent methods with smooth composite objectives. *SIAM J. Optim.* **2019**, *29*, 1772–1798. [CrossRef]

51. Sun, P.; Yang, L. Generalized eigenvalue extreme learning machine for classification. *Appl. Intell.* **2022**, *52*, 6662–6691. [CrossRef]
52. Tikhonov, A.N.; Arsenin, V.Y. *Solutions of Ill-Posed Problems*; V.H. Winston: Washington, DC, USA, 1977.
53. Niu, X.; Wang, J.; Zhang, L. Carbon price forecasting system based on error correction and divide-conquer strategies. *Appl. Soft. Comput.* **2022**, *118*, 107935. [CrossRef]
54. Hao, Y.; Niu, X.; Wang, J. Impacts of haze pollution on China's tourism industry: A system of economic loss analysis. *J. Environ. Econ. Manag.* **2021**, *295*, 113051. [CrossRef] [PubMed]