Article

Smooth Group $L_{1/2}$ Regularization for Pruning Convolutional Neural Networks

Yuan Bao [1], Zhaobin Liu [1], Zhongxuan Luo [2] and Sibo Yang [3,*]

[1] School of Information Science and Technology, Dalian Maritime University, Dalian 116026, China; yuanbao@dlmu.edu.cn (Y.B.); zhbliu@dlmu.edu.cn (Z.L.)
[2] School of Software, Dalian University of Technology, Dalian 116620, China; zxluo@dlut.edu.cn
[3] School of Science, Dalian Maritime University, Dalian 116026, China
* Correspondence: ysibo@dlmu.edu.cn

Abstract: In this paper, a novel smooth group $L_{1/2}$ ($SGL_{1/2}$) regularization method is proposed for pruning hidden nodes of the fully connected layer in convolution neural networks. Usually, the selection of nodes and weights is based on experience, and the convolution filter is symmetric in the convolution neural network. The main contribution of $SGL_{1/2}$ is to try to approximate the weights to 0 at the group level. Therefore, we will be able to prune the hidden node if the corresponding weights are all close to 0. Furthermore, the feasibility analysis of this new method is carried out under some reasonable assumptions due to the smooth function. The numerical results demonstrate the superiority of the $SGL_{1/2}$ method with respect to sparsity, without damaging the classification performance.

Keywords: convolutional neural network (CNN); fully connected layer; smooth group $L_{1/2}$ regularization; sparsity

Citation: Bao, Y.; Liu, Z.; Luo, Z.; Yang, S. Smooth Group $L_{1/2}$ Regularization for Pruning Convolutional Neural Networks. *Symmetry* **2022**, *14*, 154. https://doi.org/10.3390/sym14010154

Academic Editors: László T. Kóczy and Alexander Zaslavski

Received: 23 November 2021
Accepted: 9 January 2022
Published: 13 January 2022

1. Introduction

CNNs have been widely applied in many applications, such as intelligent information processing, pattern recognition, feature extraction [1–5], etc. As usual, slightly more hidden layer nodes were selected based on experience in neural networks. However, as is well known, too many nodes and weights in a deep network will increase the computational load, memory size and the risk of overfitting [6]. In fact, some hidden layer nodes and weights have little contribution to improving the performance of the network [7]. Therefore, choosing an appropriate number of hidden layer nodes and weights has become an important research topic in optimizing neural networks. Many algorithms have been proposed in order to optimize the number of nodes and weights in the neural network.

As one of the most effective methods to reduce the number of weights in the network, the regularization terms were introduced into the learning process. This is generally realized with L_p regularization, which penalizes the sum of the weight norm during training. The L_1 norm is the sum of the absolute values of the elements in a vector, so as to make the weight value close to zero [8]. In [9], the L_1-norm was combined with the capped L_1-norm to denote the amount of information extracted through the filter and control regularization. Gou et al. [10] proposed a discriminative collaborative representation-based classification (DCRC) method through L_2 regularizations to improve the classification capabilities. Xu et al. adopted the $L_{1/2}$ regularizer to transform a non-convex problem into a series of L_1 regularizer problems, and showed many superior properties, such as robustness, sparsity and oracle properties, compared to the L_1 and L_2 regularizers [11]. In [12], Xiao introduced sparse logistic regression (SLR) based on $L_{1/2}$ regularization to impose a sparsity constraint on logistic regression. The algorithms mentioned above successfully optimize the network only by pruning the weights.

Regularization methods have become more impressive than before, but all of them were designed mainly for pruning the superfluous weights, and the node can be deleted only if all its outgoing weights are close to zero. Then, researchers tried to prune the nodes to optimize the neural network. Simon et al. provided a group lasso method, which produced sparse effects both on and within the group, and showed the expected effect of group-wise and within-group sparsity [13–15]. Moreover, [16] considered a more general penalty and blended the lasso with the group lasso, which yielded solutions that are sparse at both the group and the individual feature level. For pruning the nodes of the network, the popular group lasso method (GL_2) imposes sparsity at the group level, so that either all the weights between nodes in the fully connected layer and all nodes of the output layer approach zero simultaneously, or none of them are close to zero. In other words, the group lasso regularization prunes the nodes of the fully connected layer, but does not prune redundant weights of surviving nodes.

It was shown that combining the $L_{1/2}$ regularization with the group lasso ($GL_{1/2}$) for feedforward neural networks can prune not only hidden nodes but also the redundant weights of the surviving hidden nodes, and can achieve better performance in terms of of sparsity [17]. However, $L_{1/2}$ regularization is not smooth at the origin, which results in oscillation during the numerical computation and causes difficulty in the feasibility analysis. To overcome these issues, the regularizer was approximated with a continuous function in our early work [18]. Furthermore, in [19], the smooth $L_{1/2}$ was applied to train the Sigma-Pi-Sigma neural network, and achieved better performance regarding both sparsity at the weight level and accuracy compared to the non-smooth $L_{1/2}$.

In this article, we combine the smooth $L_{1/2}$ regularization with the group lasso method, and propose a smooth group $L_{1/2}$ regularization algorithm. This novel algorithm inherits the advantages of the smooth function and $L_{1/2}$ regularization. As an application, the smooth group $L_{1/2}$ regularization algorithm is employed for the fully connected layer of CNNs. The main contribution of smooth group $L_{1/2}$ is to try to prune unnecessary nodes and control the magnitude of weights for the surviving nodes. In addition, due to the differentiability of the error function with smooth group $L_{1/2}$ regularization, it becomes easier to analyze the feasibility of the learning algorithm in theory. In the process of training the network, compared with GL_1, GL_2, and $GL_{1/2}$, smooth group $L_{1/2}$ regularization can not only prune the nodes and weights (improve the sparsity), but also overcome the oscillation in $GL_{1/2}$.

This paper is organized as follows. We first describe the simple process of the convolutional neural network and the smooth group $L_{1/2}$ regularization in the next section. Then, in Section 3, the feasibility analysis of the $SGL_{1/2}$ algorithm in CNNs is given, in which the training convergence with the $SGL_{1/2}$ term is proven theoretically. Numerical comparisons of several methods on four real-world datasets are carried out in Section 4. Finally, some conclusions are drawn in Section 5. In order to highlight the key points of this paper, the theorem proving process is included in the Appendix A.

2. Brief Description of CNNs and Smooth Group $L_{1/2}$ Regularization

In this section, we first demonstrate the simple calculation process of the convolutional neural network. After introducing the penalty term, the $SGL_{1/2}$ regularization is briefly described in Section 2.2.

2.1. Convolutional Neural Network

CNNs consist of three building blocks: convolution [20], pooling [21] and the fully connected layer [22]. Generally, the convolution filter is set as a symmetric matrix in CNNs. A filter in a convolution layer carries out a convolution operation on input images to obtain new feature maps, which can be expressed as:

$$x_j^l = \sum_{i \in P_j} x_i^{l-1} * k_j^l + b_j^l, \tag{1}$$

where x_j^l is the j-th feature map in the l layer, k_j^l denotes the convolution filter, the convolution operation is denoted by $*$, b_j^l is the bias, and P_j is a set of feature maps activated by filter k_j^l in the $l-1$ layer.

After x_j^l is activated by a function, such as ReLU [23,24], the pooling layer uses the max or mean approach to progressively reduce the spatial size of the representation, as shown in the following equation:

$$\mathrm{pool}(\mathrm{ReLU}(x_j^l)), \tag{2}$$

where the pool function can be selected as the maximum or average as needed and the ReLU function can be written as:

$$\mathrm{ReLU}(x) = \max\{0, x\}. \tag{3}$$

A CNN may include several convolution–ReLU–pooling parts. The output of the last pooling layer is flattened as one large vector $\mathrm{reshape}(\mathrm{pool}(\mathrm{ReLU}(x_j^l)))$ [25] and is fed to a fully connected layer for classification purposes. The final classification decision is driven by the following equation:

$$O = g(U \cdot \mathrm{reshape}(x_j^{l+1})), \tag{4}$$

where O is the actual output vector, U denotes the weight of the fully connected layer, $g(\cdot)$ represents the activation function, $\mathrm{reshape}(\cdot)$ denotes a function to transform a specified matrix into a matrix of specific dimensions. The image is classified to the i-th category if the i-th element of $g(\cdot)$ is the largest one. For a two-classification problem, U degenerates into a vector and the activation function $g(\cdot)$ is generally the sigmoid function.

2.2. *$SGL_{1/2}$ Regularization for Fully Connected Layer*

The error in the CNN is usually calculated by the following equation:

$$\widetilde{E} = \frac{1}{J}\sum_{j=1}^{J}(O^j - T^j)^2, \tag{5}$$

where J represents the number of samples, T^j and O^j are the target and actual output vectors of the j-th sample, respectively. Let r be the number of output nodes and q be the number of nodes in the fully connected layer. The error function with a penalty term is defined as

$$E = \widetilde{E} + \lambda \sum_{k=1}^{q} ||u_k||, \tag{6}$$

where the vector u_k is the weight vector connecting the k-th node of the fully connected layer and all output nodes, and λ is the penalty term coefficient. The norm could be the 1-norm, 2-norm or $\frac{1}{2}$-norm. When $||u_k|| = \sum_{i=1}^{r} |u_{ik}|$, it is the GL_1 method, while $||u_k|| = \sum_{i=1}^{r} u_{ik}^2$ is the GL_2 method.

Specifically, we take the $\frac{1}{2}$-norm [26]. Then,

$$E = \widetilde{E} + \lambda \sum_{k=1}^{q} \left(\sum_{i=1}^{r} |u_{ik}| \right)^{\frac{1}{2}}, \tag{7}$$

where $|\cdot|$ is the normal absolute value function. When the norm takes the $\frac{1}{2}$-norm, we call it the $GL_{1/2}$ method. Nevertheless, the partial derivative of E with respect to u_{ik} does not exist at the origin, which creates difficulties for the gradient descent method. Even though the partial derivative is expressed with a piecewise function, it still causes fluctuations in the process of training. In order to overcome this drawback, a $SGL_{1/2}$ regularization is proposed in this paper:

$$E = \widetilde{E} + \lambda \sum_{k=1}^{q} \left(\sum_{i=1}^{r} f(u_{ik}) \right)^{\frac{1}{2}}, \tag{8}$$

where $f(\cdot)$ represents a smooth function that approximates $|\cdot|$ (the absolute value function). Specifically, the following piecewise polynomial function is used:

$$f(t) = \begin{cases} |t|, \ |t| \geq m, \\ -\dfrac{1}{8m^3}t^4 + \dfrac{3}{4m}t^2 + \dfrac{3m}{8}, \ |t| < m, \end{cases} \tag{9}$$

where m is a small positive constant. This function $f(\cdot)$ has the following characteristics:

- $f(t) \to [\frac{3m}{8}, +\infty)$
- $f'(t) \to [-1, 1]$
- $f''(t) \to [0, \frac{3}{2m}]$

when $t \to 0^+$ is a small constant. The norm is taken as the $\frac{1}{2}$-norm, and the absolute value function in the $\frac{1}{2}$-norm is approximated by a smooth polynomial function near $x = 0$, which is called the $SGL_{1/2}$ method.

3. Feasibility Analysis of the $SGL_{1/2}$ Algorithm in CNNs

Now, it is enough to give the feasibility analysis of the $SGL_{1/2}$ algorithm. In order to obtain the convergence results, we first turn the CNN into mathematical formulae. Then, we proceed to give the convergence results.

3.1. Transform Convolution and Mean Pooling into Mathematical Equations

In regular neural networks, every layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the next layer before. This operation is easily expressed by multiplying matrices. However, in CNNs, the neurons in one layer do not connect to all the neurons in the next layer but only to a small part of it. The convolution operation is often described graphically. Thus, our first task is to transform the convolution operation into mathematical equations.

Although the convolution filter is usually symmetrical, for universal applicability in the proof, we choose a general matrix. Let an input array be filtered by a 2×2 filter, where the padding is 0 and the step is 1. As shown in Figure 1, when the filter slides over the input, a matrix multiplication of a submatrix of the input and the filter is performed and the sum of the convolution moves into the feature map, i.e., the output of this layer.

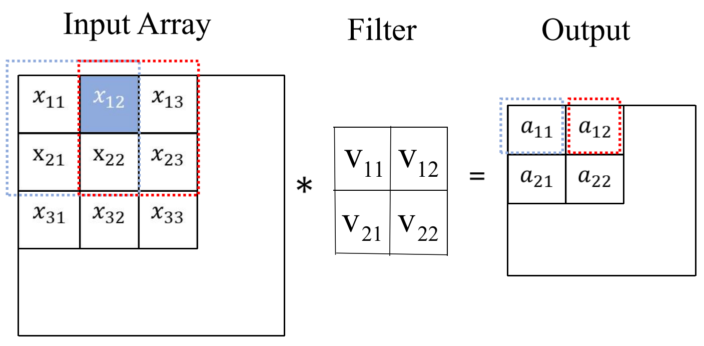


Figure 1. The convolution operation.

To express this operation with mathematical equations, we squash each submatrix that multiplies with the filter into a vector. More specifically, the red square of the input array in Figure 2 is squashed into the vector $(x_{11}, x_{12}, x_{21}, x_{22})$. Then, we put all squashed vectors into a matrix X in order of the filter sliding, as shown in Figure 2. The filter is also squashed with a vector $(v_{11}, v_{12}, v_{21}, v_{22})^T$ accordingly, and then is repeatedly put into the diagonal position of the matrix V, as shown in Figure 3. Other elements of V are 0. With X and V, the operation of the convolutional layer can be described with the matrix multiplication of X and V, as shown in Figure 4, i.e.,

$$\text{Conv} = X \cdot V \tag{10}$$

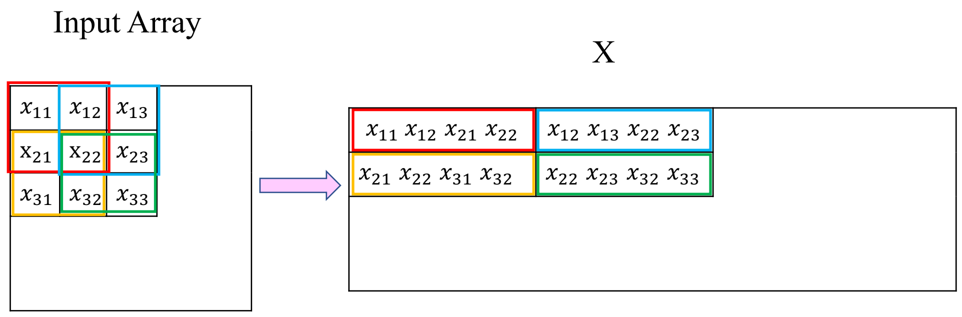

Input Array
X
x_{11} x_{12} x_{13}
x_{21} x_{22} x_{23}
x_{31} x_{32} x_{33}
x_{11} x_{12} x_{21} x_{22}
x_{12} x_{13} x_{22} x_{23}
x_{21} x_{22} x_{31} x_{32}
x_{22} x_{23} x_{32} x_{33}

Figure 2. Reshape the input array with a matrix X.

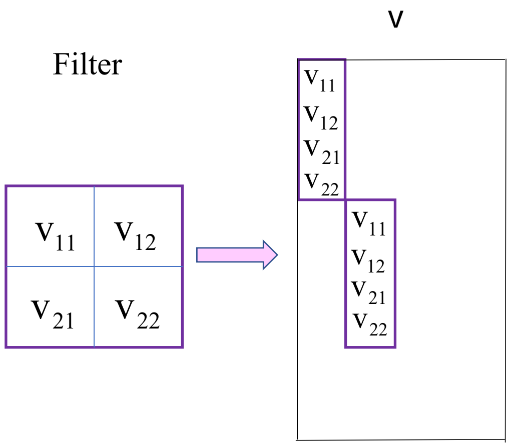

V
Filter
V_{11} V_{12}
V_{21} V_{22}
V_{11} V_{12} V_{21} V_{22}
V_{11} V_{12} V_{21} V_{22}

Figure 3. Reshape the filter with a matrix V.

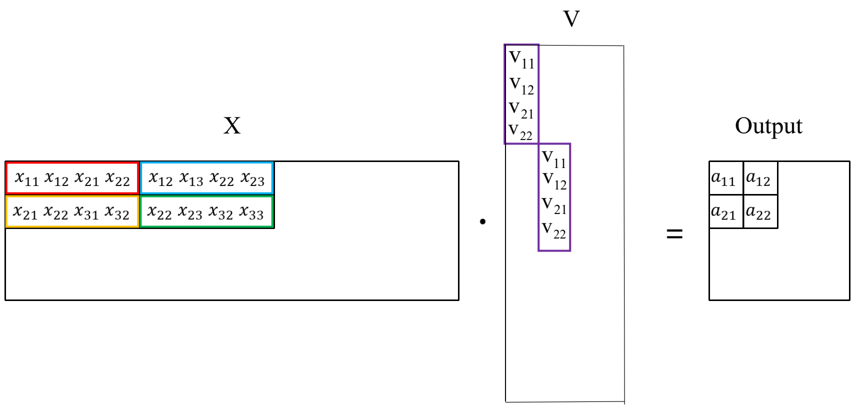
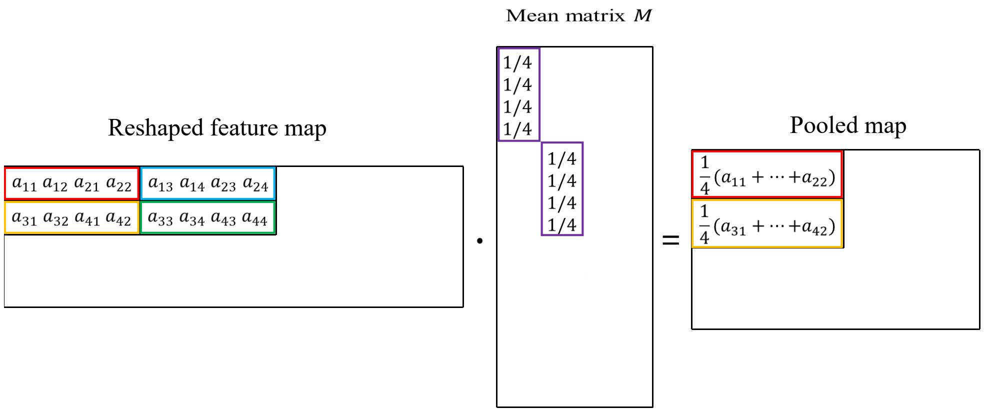

X
V
Output
x_{11} x_{12} x_{21} x_{22}
x_{12} x_{13} x_{22} x_{23}
x_{21} x_{22} x_{31} x_{32}
x_{22} x_{23} x_{32} x_{33}
V_{11} V_{12} V_{21} V_{22}
V_{11} V_{12} V_{21} V_{22}
a_{11} a_{12}
a_{21} a_{22}
·
=

Figure 4. The convolution operation is described with the matrix multiplication of the reshaped input array and filter.

The mean pooling is assumed to be applied in 2×2 patches of the feature map with a stride of $(2, 2)$. It can also be expressed with the matrix multiplication, as shown in Figure 5. Each patch of the feature map is flattened into a vector and all the vectors are merged into a matrix as a reshaped feature map, as shown in Figure 5. The sliding mean window is flattened as a vector $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})^T$ and is repeatedly put into the diagonal position of the mean matrix M, as shown in Figure 5. As in Equation (10), the mean pooling operation can be expressed with the matrix multiplication:

$$\text{mean pooling} = X \cdot M. \tag{11}$$



Mean matrix M
Reshaped feature map
$a_{11}\ a_{12}\ a_{21}\ a_{22}$
$a_{13}\ a_{14}\ a_{23}\ a_{24}$
$a_{31}\ a_{32}\ a_{41}\ a_{42}$
$a_{33}\ a_{34}\ a_{43}\ a_{44}$
1/4 1/4 1/4 1/4
1/4 1/4 1/4 1/4
Pooled map
$\frac{1}{4}(a_{11}+\cdots+a_{22})$
$\frac{1}{4}(a_{31}+\cdots+a_{42})$

Figure 5. The mean pooling is described with the matrix multiplication of the reshaped feature map and the mean matrix.

Now, given an input array X, the processing procedure from the convolution to the output layer can be expressed by mathematical equations. The output of the convolution layer is

$$A = G(X) \cdot V, \tag{12}$$

where the function G means the reshape operation shown in Figure 2. After the ReLU layer, the matrix $\text{ReLU}(A)$ is reshaped as $G(\text{ReLU}(A))$. In the pooling layer, the mean function is used

$$\text{mean pooling} = G(\text{ReLU}(A)) \cdot M. \tag{13}$$

Then, the output matrix of the pooling layer is vectorized by column scan and this process is denoted by

$$S = F(\text{mean pooling}), \tag{14}$$

where the function F denotes the layer vectorized by the column. Finally, the fully connected layer is

$$O = g(U \cdot S). \tag{15}$$

3.2. Convergence Results

To prove that our proposed method is feasible, here, we give the convergence results. For ease of understanding, we take the simplest single-layer CNN case as an example. This CNN includes one convolution, one pooling and one fully connected layer, where the convolution filter size is $(5,5)$ and the mean pooling size is $(2,2)$.

Given the training sample set $\{X^j, T^j\}_{j=1}^{J}$, each X^j is assumed to be the 28×28 input array and T^j is the 10×1 vector. According to Equations (12)–(15), the error function of Equation (8) can be expressed as

$$
\begin{aligned}
E &= \frac{1}{2J}\sum_{j=1}^{J}\sum_{i=1}^{10}(o_i^j - t_i^j)^2 + \lambda\sum_{k=1}^{144}\Big(\sum_{i=1}^{10} f(u_{ik})\Big)^{\frac{1}{2}} \\
&= \frac{1}{2J}\sum_{j=1}^{J}\sum_{i=1}^{10}\Big(g(U_i\cdot F(G(\mathrm{ReLU}(G(X^j)\cdot V))\cdot M)) - t_i^j\Big)^2 + \lambda\sum_{k=1}^{144}\Big(\sum_{i=1}^{10} f(u_{ik})\Big)^{\frac{1}{2}} \\
&= \sum_{j=1}^{J}\sum_{i=1}^{10} g_{ji}(U_i\cdot S) + \lambda\sum_{k=1}^{144}\Big(\sum_{i=1}^{10} f(u_{ik})\Big)^{\frac{1}{2}},
\end{aligned}
\tag{16}
$$

where $g_{ji}(t) = \frac{1}{2J}(g(t) - t_i^j)^2$, U_i is the i-th row vector of U.

Training a CNN involves finding a suitable V and U so that E reaches the minimum [27]. For this reason, the gradient descent method [28] is adopted. Notice that the mean matrix M does not need to be trained. In the backpropagation algorithm, V and U are changed according to the gradient descent direction of E. The partial derivative of E with respect to the element u_{ik} of U is as follows:

$$
E_{u_{ik}} = \sum_{j=1}^{J} g'_{ji}(U_i\cdot S)\cdot S_k + \lambda\frac{f'(u_{ik})}{2\sqrt{\sum\limits_{i=1}^{10} f(u_{ik})}},\ (k = 1,\cdots,144).
\tag{17}
$$

The partial derivative of E with respect to the element v_{ij} of the convolution filter V is the same as the original CNN because the partial derivative of the penalty term in Equation (16) with respect to v_{ij} is zero. That is,

$$
\begin{aligned}
E_{v_{pl}} &= \sum_{j=1}^{J}\sum_{i=1}^{10} g'_{ji}(U_i\cdot S)\sum_{k=1}^{144} u_{ik}\frac{\partial S_k}{\partial v_{pl}} \\
&= \sum_{j=1}^{J}\sum_{i=1}^{10} g'_{ji}(U_i\cdot S)\sum_{m=0}^{11}\sum_{h=1}^{12} u_{i,12m+h}\frac{\partial s_{m+1,h}}{\partial v_{pl}} \\
&= \frac{1}{4}\sum_{j=1}^{J}\sum_{i=1}^{10} g'_{ji}(U_i\cdot S)\sum_{m=0}^{11}\sum_{h=1}^{12} u_{i,12m+h} \\
&\quad\cdot\Big(\delta(a_{m+1,h})\frac{\partial a_{m+1,h}}{\partial v_{pl}} + \delta(a_{m+1,h+1})\frac{\partial a_{m+1,h+1}}{\partial v_{pl}} \\
&\quad+\delta(a_{m+2,h})\frac{\partial a_{m+2,h}}{\partial v_{pl}} + \delta(a_{m+2,h+1})\frac{\partial a_{m+2,h+1}}{\partial v_{pl}}\Big) \\
&= \frac{1}{4}\sum_{j=1}^{J}\sum_{i=1}^{10} g'_{ji}(U_i\cdot S)\sum_{m=0}^{11}\sum_{h=1}^{12} u_{i,12m+h} \\
&\quad\cdot(\delta(a_{m+1,h})x_{m+p,h+l-1} + \delta(a_{m+1,h+1})x_{m+p,h+l} \\
&\quad+\delta(a_{m+2,h})x_{m+p+1,h+l-1} + \delta(a_{m+2,h+2})x_{m+p+1,h+l}),
\end{aligned}
\tag{18}
$$

where δ is the derivative function of the rectified linear units function:

$$
\delta(t) = \begin{cases} 1, & t > 0, \\ 0, & t \le 0. \end{cases}
$$

Thus far, we have given the step direction of U and V by (17) and (18), respectively. Now, we proceed to give the step direction of the biases. The partial derivative of the biases can be computed similarly as shown in [29]; the reader can refer to this article for more details.

We combine all weights and biases into a large vector W. Then, the parameter updating algorithm of $SGL_{1/2}$ is defined as follows:

$$W^{n+1} = W^n + \triangle W^n = W^n - \eta E_{W^n}, \tag{19}$$

where η is the learning rate and n is the iteration step.

The convergence proof needs some assumptions as follows:

(1) $||W^n||$ $(n = 0, 1, 2, \dots)$ are uniformly bounded, where W^n is the error of the n-th step.

(2) λ and η are chosen to satisfy $\eta \leq \min\{\frac{1}{C_6}, \frac{1}{C_5+\lambda C_7}\}$, where C_5, C_6 and C_7 are constants defined below.

(3) There exists a compact set Φ such that $W^n \in \Phi$ and $\Phi_0 = \{W \in \Phi : E_W = 0\}$ contains finite points.

Theorem 1. *Let the error function be Equation (8) and the weight sequence $\{W^n\}$ be generated by Equation (19) with any initial value W^0. If Assumptions (1)–(2) are available, then*

(i) $E(W^n) \geq E(W^{n+1}),\ n = 0, 1, 2, \dots;$

(ii) *There exists $E^* \geq 0$, such that* $\lim_{n\to\infty} E(W^n) = E^*$;

(iii) *The weak convergence holds, i.e.,* $\lim_{n\to\infty} ||E_{W^n}||_2 = 0$.

In addition, if the Assumption (3) also holds, then the strong convergence result holds:

(iv) *There exists a point $W^* \in \Phi_0$ such that* $\lim_{n\to\infty} W^n = W^*$.

The proof process is not the focus of this article, so we include it in the Appendix A.

4. Numerical Experiments and Discussion

We evaluate $SGL_{1/2}$ in different ways, such as nodes [30] and weights sparsity [31], training and testing accuracy, the norm of weight gradient and the convergence speed, on four typical benchmark datasets: Mnist [32], Letter Recognition [33], Cifar 10 [34] and Crowded Mapping. For parameter sparsity, $SGL_{1/2}$ is compared with some conventional and sparse algorithms including GL_1, GL_2 and $GL_{1/2}$. Moreover, we investigate the test accuracy by comparing $SGL_{1/2}$ with the above regularization algorithms.

For the following numerical experiments, we refer to the arithmetic optimization algorithm [35] and adopt a five-fold cross-validation technique [36–38]. We randomly divide the dataset into five parts, where the sample size is equal (or almost equal). The network learning of these four algorithms is carried out five times. Each time, one of the five parts is selected in turn as the test sample set, and the other four parts are used as the training sample sets. Then, we rearrange the five-part samples and start the process again. This process is repeated twenty times. The experiment process is given in Algorithm 1.

Algorithm 1 The experiment process.

```
Step 1: Input the data and calculate the corresponding actual output;
Step 2: Calculate the difference between the actual output and the ideal output;
Step 3: Give the error function according to these four algorithms;
Step 4: Update the weights of the fully connected layer according to the error function of step 3;
Step 5: Keep iterating, repeat steps 2–4;
Step 6: Calculate the pruned nodes, pruned weights of surviving weights and classification
        accuracies under these four methods, respectively;
Step 7: Contrast.
```

Finally, for each dataset and algorithm, we obtain one hundred classification results. Each result contains the rate of pruned nodes (Rate of PN) (cf. Equation (20)), the rate of pruned weights of the remaining nodes (Rate of PW) (cf. Equation (21)), training accuracy (Training Acc.) and test accuracy (Test Acc.). The averages of these numerical results are listed in Tables 1–4 for these four datasets.

$$\text{Rate of PN} = \frac{\text{Pruned nodes}}{\text{All nodes}}, \tag{20}$$

$$\text{Rate of PW} = \frac{\text{Pruned weights of surviving nodes}}{\text{Surviving nodes} \times \text{Output nodes}}. \tag{21}$$

For an output node, the ideal output value is 1 or 0. When we evaluate the error between the ideal and real output values, we use the following "40-20-40" standard [39]: The actual output values of the output nodes between 0.00 and 0.40 are regarded as 0, values between 0.60 and 1.00 are regraded as 1, and values between 0.40 and 0.60 are regraded as uncertain and are considered incorrect.

Table 1. Mnist: Comparison of Rate of PN, Rate of PW, Training Acc. and Test Acc. by using GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ learning algorithms.

Algorithm	Rate of PN (%)	Rate of PW (%)	Training Acc. (%)	Test Acc. (%)
GL_1	15.50	2.60	99.75	99.13
GL_2	16.17	3.16	99.80	99.15
$GL_{1/2}$	12.33	**19.83**	99.67	99.06
$SGL_{1/2}$	**19.33**	17.79	**99.83**	**99.24**

Table 2. Letter: Comparison of Rate of PN, Rate of PW, Training Acc. and Test Acc. by using GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ learning algorithms.

Algorithm	Rate of PN (%)	Rate of PW (%)	Training Acc. (%)	Test Acc. (%)
GL_1	7.78	1.95	89.75	87.42
GL_2	8.89	2.86	89.81	87.60
$GL_{1/2}$	5.56	**18.37**	89.67	86.83
$SGL_{1/2}$	**15.56**	12.04	**90.43**	**87.93**

Table 3. Cifar 10: Comparison of Rate of PN, Rate of PW, Training Acc. and Test Acc. by using GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ learning algorithms.

Algorithm	Rate of PN (%)	Rate of PW (%)	Training Acc. (%)	Test Acc. (%)
GL_1	12.7	2.20	87.95	82.84
GL_2	13.4	2.63	88.07	83.26
$GL_{1/2}$	9.02	**18.05**	87.36	82.30
$SGL_{1/2}$	**20.61**	17.27	**88.39**	**83.86**

Table 4. Crowded Mapping: Comparison of Rate of PN, Rate of PW, Training Acc. and Test Acc. by using GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ learning algorithms.

Algorithm	Rate of PN (%)	Rate of PW (%)	Training Acc. (%)	Test Acc. (%)
GL_1	11.46	1.85	93.58	92.76
GL_2	12.85	3.07	94.39	93.58
$GL_{1/2}$	8.93	**19.65**	92.54	91.90
$SGL_{1/2}$	**22.60**	17.28	**95.21**	**94.63**

4.1. Mnist Problem

MNIST is a dataset for the study of handwritten numeral recognition, which contains 70,000 examples of 28×28 pixel images of the digits 0–9. For these four algorithms, we set the learning rate $\eta = 0.03$. The maximum iteration training step is 1000. In order to show the sparsity, we give the node sparsity and weight sparsity performances for $\lambda = 0.001, 0.002, 0.003, 0.004, 0.005$ of these four algorithms (see Figure 6; y-axis represents the percentage of the number of pruned nodes and pruned weights of the remaining nodes, respectively). The sparsity will become worse when $\lambda > 0.005$. Therefore, we choose $\lambda = 0.005$ to compare these algorithms. The performances of these four group lasso algorithms are compared in Table 1. We can see that, in terms of the sparsity, the performance of $SGL_{1/2}$ is better than GL_2, GL_1 and $GL_{1/2}$. In terms of of accuracy, $SGL_{1/2}$ is also the best. Figure 7a presents the loss functions of these four group lasso algorithms. Obviously, we can see that the $SGL_{1/2}$ approach has the lowest error after training, and $SGL_{1/2}$ has a large fluctuation during the training process.

We show the gradient norms of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ in Figure 7b, where the oscillation [40,41] of $GL_{1/2}$ is presented. From Figure 7, we find that the $SGL_{1/2}$ regularizer eliminates the oscillation and guarantees the convergence, as predicted in Theorem 1.

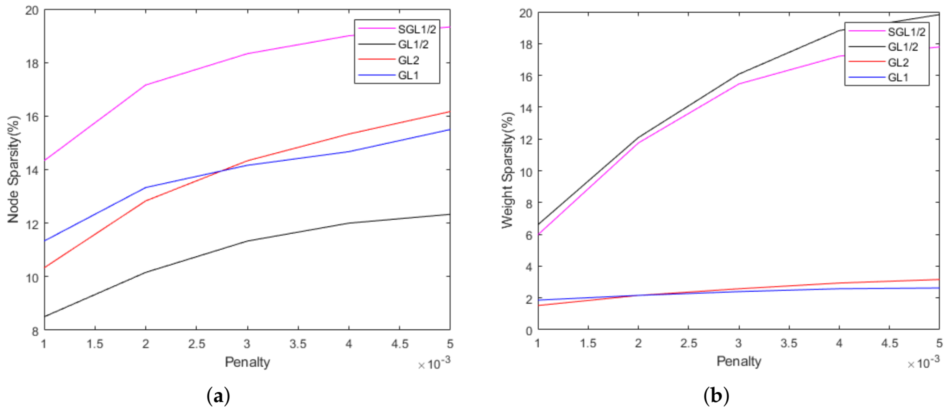


Figure 6. Sparsity of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ algorithms in Mnist dataset. (**a**) Node sparsity, (**b**) Weight sparsity.

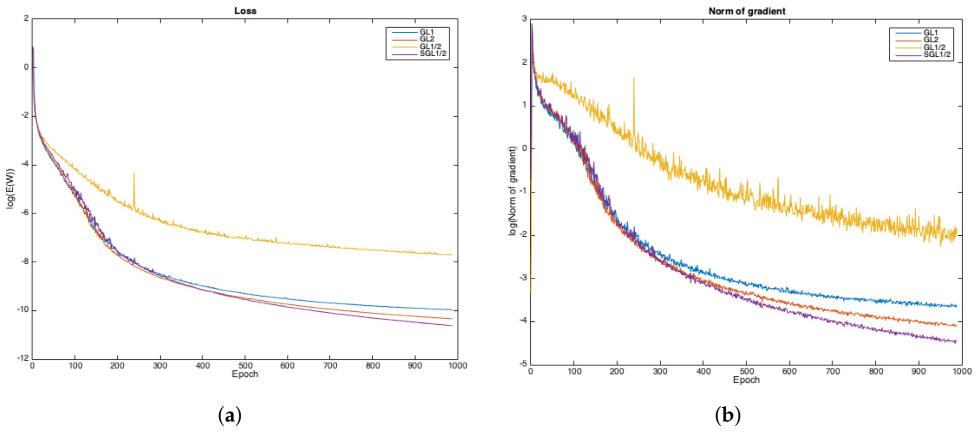


Figure 7. Loss and gradient of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ algorithms in Mnist dataset. (**a**) Loss function, (**b**) Norm of gradient.

4.2. Letter Recognition Problem

The Letter Recognition dataset consists of 20,000 samples with 16 attributes. Each 16-dimensional instance within this database represents a capital typewritten letter in one of twenty fonts. For these four algorithms, we set the learning rate $\eta = 0.05$. The maximum iteration training step is 1000. In order to show the sparsity, we give the node sparsity and weight sparsity performances for $\lambda = 0.002, 0.004, 0.006, 0.008$ of these four algorithms (see Figure 8). The sparsity will become worse when $\lambda > 0.008$. Therefore, we choose $\lambda = 0.008$ to compare these algorithms. The performances of GL_2, GL_1, $GL_{1/2}$ and $SGL_{1/2}$

are compared in Table 2. We see that, in terms of sparsity, the performance of $SGL_{1/2}$ is better than GL_2, GL_1 and $GL_{1/2}$. In terms of accuracy, $SGL_{1/2}$ is also the best among the above-mentioned four algorithms. Figure 9a presents the loss functions of these four group lasso algorithms. Obviously, we can see that the $SGL_{1/2}$ approach has the lowest error after training, and $SGL_{1/2}$ has a large fluctuation during the training process.

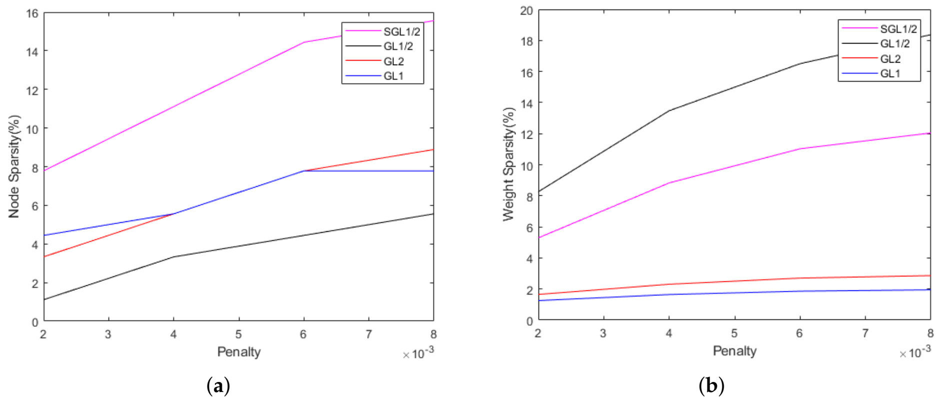


Figure 8. Sparsity of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ algorithms in Letter Recognition dataset. (**a**) Node sparsity, (**b**) Weight sparsity.

We show the gradient norms of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ in Figure 9b, where the oscillation of $GL_{1/2}$ is presented. From Figure 9, we find that the $SGL_{1/2}$ regularizer eliminates the oscillation and guarantees the convergence, as predicted in Theorem 1.

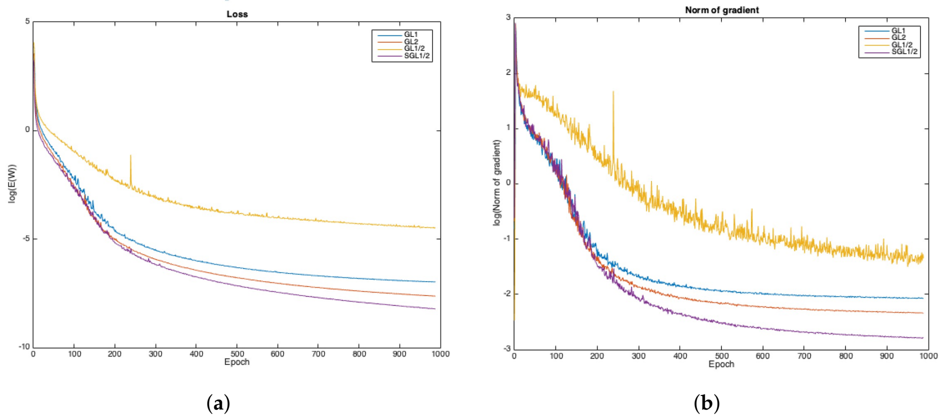
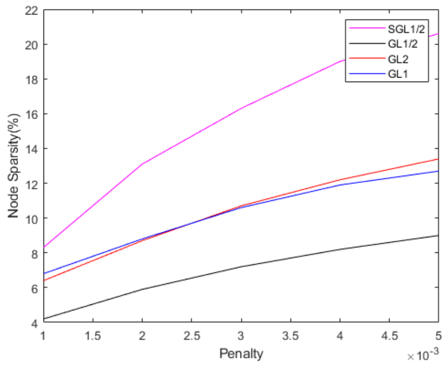


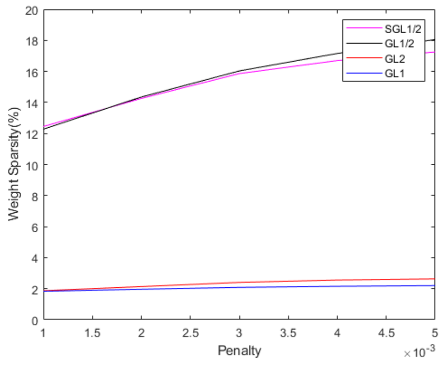
Figure 9. Loss and gradient of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ algorithms in Letter Recognition dataset. (**a**) Loss function, (**b**) Norm of gradient.

4.3. Cifar 10 Problem

The Cifar 10 dataset consists of 60,000 images, each of which is a 32×32 color map. This dataset contains 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck), with 6000 images per class. There are 50,000 training images and 10,000 test images. For these four algorithms, we set the learning rate $\eta = 0.03$. The maximum iteration training step is 1000. In order to show the sparsity, we give the node sparsity and weight sparsity performances for $\lambda = 0.001, 0.002, 0.003, 0.004, 0.005$ of these four algorithms (see Figure 10). The sparsity will become worse when $\lambda > 0.005$. Therefore, we choose $\lambda = 0.005$ to compare these algorithms. The performances of these four group lasso algorithms are compared in Table 3. We see that, in terms of of sparsity, the performance of $SGL_{1/2}$ is better than GL_2, GL_1 and $GL_{1/2}$. In terms of accuracy, $SGL_{1/2}$ is also the best. Figure 11a presents the loss functions of these four group lasso algorithms. Obviously, we can see that the $SGL_{1/2}$ approach has the lowest error after training, and $SGL_{1/2}$ has a large fluctuation during the training process.



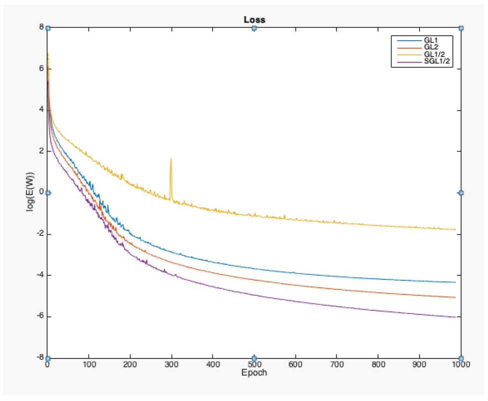
(**a**)



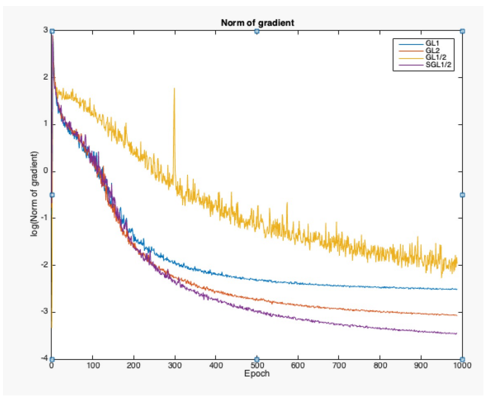
(**b**)

Figure 10. Sparsity of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ algorithms in Cifar 10 dataset. (**a**) Node sparsity, (**b**) Weight sparsity.

We show the gradient norms of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ in Figure 11b, where the oscillation of $GL_{1/2}$ is presented. From Figure 11, we find that the $SGL_{1/2}$ regularizer eliminates the oscillation and guarantees the convergence, as predicted in Theorem 1.



(**a**)

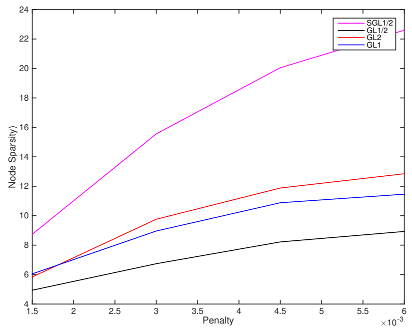


(**b**)

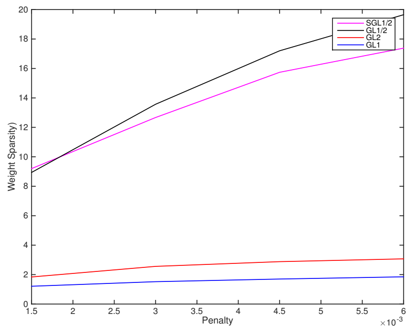
Figure 11. Loss and gradient of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ algorithms in Cifar 10 dataset. (**a**) Loss function, (**b**) Norm of gradient.

4.4. Crowded Mapping

The Crowded Mapping dataset consists of 10,546 samples with 28 attributes, and these samples are divided into six classes. For these four algorithms, we set the learning rate $\eta = 0.05$. The maximum iteration training step is 1000. In order to show the sparsity, we give the node sparsity and weight sparsity performances for $\lambda = 0.0015, 0.003, 0.045, 0.006$ of these four algorithms (see Figure 12). The sparsity will become worse when $\lambda > 0.006$. Therefore, we choose $\lambda = 0.006$ to compare these four algorithms. The performances of the GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ methods are compared in Table 4. We see that, in terms of sparsity, the performance of $SGL_{1/2}$ is better than GL_2, GL_1 and $GL_{1/2}$. In terms of accuracy, $SGL_{1/2}$ is also the best among the above-mentioned four algorithms. Figure 13a presents the loss functions of these four group lasso algorithms. Obviously, we can see that the $SGL_{1/2}$ algorithm has the lowest error after training, and $SGL_{1/2}$ has a large fluctuation during the training process.



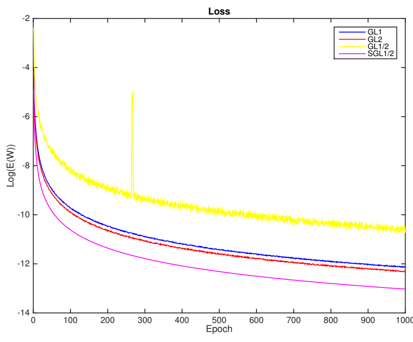
(**a**)



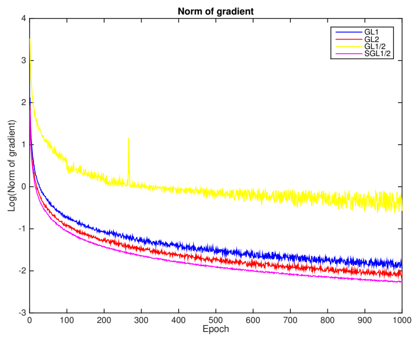
(**b**)

Figure 12. Sparsity of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ algorithms in Crowded Mapping dataset. (**a**) Node sparsity, (**b**) Weight sparsity.

We show the gradient norms of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ in Figure 13b, where the oscillation of $GL_{1/2}$ is presented. From Figure 13, we find that the $SGL_{1/2}$ regularizer eliminates the oscillation and guarantees the convergence, as predicted in Theorem 1.



(**a**)



(**b**)

Figure 13. Loss and gradient of GL_1, GL_2, $GL_{1/2}$ and $SGL_{1/2}$ algorithms in Crowded Mapping dataset. (**a**) Loss function, (**b**) Norm of gradient.

From the above experiments on the four datasets, it is easy to see that the GL_1 and GL_2 algorithms have better sparsity at the node level, and the $GL_{1/2}$ algorithm has better sparsity at the weight level. In some applications, the sparsity at the weight level is also of great significance. If the sparseness of the integrated node and weight level is better, the number of weights that need to be calculated and updated will be reduced in the process of training the CNNs. Furthermore, it also leads to a reduction in the amount of calculation and saves storage space. Compared with the GL_1 and GL_2 algorithms, the $SGL_{1/2}$ algorithm has better sparsity at the node level and the weight level, and can also improve the classification performance. Compared with the $GL_{1/2}$ algorithm, the theoretical analysis and numerical experiment are carried out to verify that the $SGL_{1/2}$ algorithm improves the sparsity at the node level, and at the same time improves the classification performance.

4.5. Discussion

Tables 1–4, respectively, show the performance comparison of PN, PW, training accuracies and test accuracies under these four methods. In terms of the sparsity, the PN calculation results of the $SGL_{1/2}$ method are much better than the other three methods, especially the $GL_{1/2}$ method. As for the PW, although the surviving node of the $GL_{1/2}$ has a higher rate of pruned weights of surviving weights, the rate of pruned nodes is too low, such that the sparsity of the $GL_{1/2}$ method is still far lower than that of the $SGL_{1/2}$ method. In terms of classification accuracy, the $SGL_{1/2}$

method is slightly higher than other methods, which means that this method can improve the sparsity without damaging the classification accuracy.

We can find that the specificity of CNNs is not actually used in the experiments, so the $SGL_{1/2}$ method can be widely applied to other neural network models.

5. Conclusions

Our main task was to introduce the $SGL_{1/2}$ algorithm. Based on the GL_1 and GL_2 algorithms, replacing 1-norm and 2-norm with $\frac{1}{2}$-norm can greatly improve the sparsity of the network weight level, but it does not help to achieve better sparsity of the node level. The non-smooth penalty term at the origin is the root cause of the poor sparsity of the $GL_{1/2}$ algorithm at the node level.

To this end, in this paper, a smooth group $L_{1/2}$ ($SGL_{1/2}$) regularization term is introduced into the batch gradient learning algorithm to prune the CNN. The feasibility analysis of the $SGL_{1/2}$ method for the fully connected layer of the CNN is performed. Numerical experiments show that the sparsity and convergence of $SGL_{1/2}$ give better results in terms of both the rate of pruned hidden nodes and weights of the remaining hidden nodes compared to $GL_{1/2}$, GL_1 and GL_2. In addition, the $SGL_{1/2}$ regularizer not only overcomes the oscillation phenomenon during the training process, but also achieves better classification performance.

In fact, the $SGL_{1/2}$ regularization algorithm provides a strategy to improve the sparsity of hidden layers of neural networks, not only for CNNs. Therefore, the performance of the $SGL_{1/2}$ regularization algorithm on other neural network models is also worthy of further verification. However, the $SGL_{1/2}$ algorithm is not particularly obvious in improving the classification accuracy. In future work, we will focus on continuing to improve the $SGL_{1/2}$ algorithm to achieve better classification performance of the neural network.

Author Contributions: Conceptualization, Y.B. and S.Y.; methodology, S.Y.; software, S.Y.; validation, Y.B., S.Y. and Z.L. (Zhaobin Liu); formal analysis, S.Y.; investigation, Z.L. (Zhongxuan Luo); resources, Z.L. (Zhongxuan Luo); data curation, Z.L. (Zhaobin Liu); writing—original draft preparation, Y.B. and S.Y.; writing—review and editing, Y.B., S.Y., Z.L. (Zhongxuan Luo) and Z.L. (Zhaobin Liu); funding acquisition, Z.L. (Zhongxuan Luo). All authors have read and agreed to the published version of the manuscript.

Funding: This project is supported by the National Natural Science Foundation of China (No. 61720106005).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets in this paper are both available at http://archive.ics.uci.edu/ml/datasets.php (accessed on 6 May 2020).

Acknowledgments: The authors would like to thank the referees for their careful reading and helpful comments.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Proof for (i) of Theorem 1. In $SGL_{1/2}$, each input array X^j is fed to the CNN to compute the corresponding output. Specifically, at the n-th iteration, when X^j is fed to the CNN, the flattened vector S is $S^{n,j}$. Let $\Delta S^{n,j} = S^{n+1,j} - S^{n,j}$ and $\Delta \mathbf{u}_{ij}^{n} = \mathbf{u}_{ij}^{n+1} - \mathbf{u}_{ij}^{n}$. By the error function (16) and Taylor's formula [42], we have

$$
\begin{aligned}
E(W^{n+1}) - E(W^n) &= \sum_{j=1}^{J}\sum_{k=1}^{10}\Big(g_{jk}(U_k^{n+1}S^{n+1,j}) - g_{jk}(U_k^nS^{n,j})\Big) \\
&\quad + \lambda\sum_{k=1}^{144}\Big[\Big(\sum_{i=1}^{10} f(u_{ik}^{n+1})\Big)^{\frac{1}{2}} - \Big(\sum_{i=1}^{10} f(u_{ik}^{n+1})\Big)^{\frac{1}{2}}\Big] \\
&= \sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^nS^{n,j})U_k^n\Delta S^{n,j} + \sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^nS^{n,j})\Delta U_k^nS^{n,j} + \delta_1 \\
&\quad + \lambda\sum_{k=1}^{144}\Big[\Big(\sum_{i=1}^{10} f(u_{ik}^{n+1})\Big)^{\frac{1}{2}} - \Big(\sum_{i=1}^{10} f(u_{ik}^{n})\Big)^{\frac{1}{2}}\Big],
\end{aligned}
\tag{A1}
$$

where

$$
\begin{aligned}
\delta_1 &= \frac{1}{2}\sum_{j=1}^{J}\sum_{k=1}^{10}\begin{bmatrix} (\Delta U_k^n)^T & (\Delta S^{n,j})^T \end{bmatrix} \\
&\begin{bmatrix} \frac{\partial^2 g_{jk}(\xi_k^n\nu^{n,j})}{\partial (U_k^n)^2} & \frac{\partial^2 g_{jk}(\xi_k^n\nu^{n,j})}{\partial U_k^n\partial S^{n,j}} \\ \frac{\partial^2 g_{jk}(\xi_k^n\nu^{n,j})}{\partial S^{n,j}\partial U_k^n} & \frac{\partial^2 g_{jk}(\xi_k^n\nu^{n,j})}{\partial (S^{n,j})^2} \end{bmatrix}\begin{bmatrix} \Delta U_k^n \\ \Delta S^{n,j} \end{bmatrix}
\end{aligned}
\tag{A2}
$$

for some real-valued vector $\xi_k^n \in (U_k^n, U_k^{n+1})$, and $\nu^{n,j} \in (S^{n,j}, S^{n+1,j})$. For the first term of Equation (A1), we obtain

$$
\begin{aligned}
&\sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^nS^{n,j})U_k^n\Delta S^{n,j} \\
&= \sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^nS^{n,j})U_k^n[F(G(\mathrm{ReLU}(A^{n+1}))\cdot M) - F(G(\mathrm{ReLU}(A^{n}))\cdot M)] \\
&= \frac{1}{4}\sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^nS^{n,j})\sum_{m=0}^{11}\sum_{h=1}^{12} u_{k,12m+h}^n\cdot\Big(\mathrm{ReLU}(a_{m+1,h}^{n+1}) + \mathrm{ReLU}(a_{m+1,h+1}^{n+1}) \\
&+ \mathrm{ReLU}(a_{m+2,h}^{n+1}) + \mathrm{ReLU}(a_{m+2,h+1}^{n+1}) - \mathrm{ReLU}(a_{m+1,h}^{n}) - \mathrm{ReLU}(a_{m+1,h+1}^{n}) \\
&- \mathrm{ReLU}(a_{m+2,h}^{n}) - \mathrm{ReLU}(a_{m+2,h+1}^{n})\Big) \\
&= \frac{1}{4}\sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^nS^{n,j})\sum_{m=0}^{11}\sum_{h=1}^{12} u_{k,12m+h}^n\cdot\Big(\mathrm{ReLU}(\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t,h+l-1}^j v_{tl}^{n+1}) \\
&+ \mathrm{ReLU}(\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t,h+l}^j v_{tl}^{n+1}) + \mathrm{ReLU}(\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t+1,h+l-1}^j v_{tl}^{n+1}) \\
&+ \mathrm{ReLU}(\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t+1,h+l}^j v_{tl}^{n+1}) - \mathrm{ReLU}(\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t,h+l-1}^j v_{tl}^{n}) \\
&- \mathrm{ReLU}(\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t,h+l}^j v_{tl}^{n}) - \mathrm{ReLU}(\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t+1,h+l-1}^j v_{tl}^{n}) \\
&- \mathrm{ReLU}(\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t+1,h+l}^j v_{tl}^{n})\Big) \\
&= \frac{1}{4}\sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^nS^{n,j})\sum_{m=0}^{11}\sum_{h=1}^{12} u_{k,12m+h}^n\cdot\Big(\delta(a_{m+1,h}^n)\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t,h+l-1}^j\Delta v_{tl}^n \\
&+ \delta(a_{m+1,h+1}^n)\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t+1,h+l-1}^j\Delta v_{tl}^n + \delta(a_{m+2,h}^n)\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t+1,h+l-1}^j\Delta v_{tl}^n \\
&+ \delta(a_{m+2,h+2}^n)\sum_{t=1}^{5}\sum_{l=1}^{5} x_{m+t+1,h+l}^j\Delta v_{tl}^n\Big).
\end{aligned}
\tag{A3}
$$

If we extract the first term of each sign $\sum_{t=1}^{5}\sum_{l=1}^{5}$ (the case of $t = 1$ and $i = 1$) and sum them together, we can obtain

$$\begin{aligned}
&\frac{1}{4}\sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^n S^{n,j}) \sum_{m=0}^{11}\sum_{h=1}^{12} u^n_{i,12m+h} \\
&\cdot \Big(\delta(a^n_{m+1,h})x^j_{m+1,h}\Delta v^n_{11} + \delta(a^n_{m+1,h+1})x^j_{m+2,h}\Delta v^n_{11} \\
&+ \delta(a^n_{m+2,h})x^j_{m+2,h}\Delta v^n_{11} + \delta(a^n_{m+2,h+2})x^j_{m+2,h+1}\Delta v^n_{11}\Big) \\
&= \frac{\Delta v^n_{11}}{4}\sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^n S^{n,j}) \sum_{m=0}^{11}\sum_{h=1}^{12} u^n_{i,12m+h} \\
&\cdot \Big(\delta(a^n_{m+1,h})x^j_{m+1,h} + \delta(a^n_{m+1,h+1})x^j_{m+2,h} \\
&+ \delta(a^n_{m+2,h})x^j_{m+2,h} + \delta(a^n_{m+2,h+2})x^j_{m+2,h+1}\Big) \\
&\overset{(18)}{=} E_{v^n_{11}}\Delta v^n_{11} \overset{(19)}{=} -\frac{1}{\eta}(\Delta v^n_{11})^2.
\end{aligned}$$

Similar processing can be applied to other terms of $\sum_{t=1}^{5}\sum_{l=1}^{5}$ (other 24 cases). Thus, it follows from Equation (A3) that

$$\sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^n S^{n,j})U_k^n \Delta S^{n,j} = -\frac{1}{\eta}\sum_{t=1}^{5}\sum_{l=1}^{5}(\Delta v^n_{tl})^2. \tag{A4}$$

For the second term of Equation (A1), we obtain

$$\begin{aligned}
&\sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^n S^{n,j})\Delta U_k^n S^{n,j} \\
&= \sum_{j=1}^{J}\sum_{k=1}^{10} g'_{jk}(U_k^n S^{n,j}) \sum_{i=1}^{144}\Delta u_{ki} S_i^{n,j} \\
&= \sum_{k=1}^{10}\sum_{i=1}^{144}\Delta u_{ki}\sum_{j=1}^{J} g'_{jk}(U_k^n S^{n,j}) S_i^{n,j} \\
&\overset{(17)}{=} \sum_{k=1}^{10}\sum_{i=1}^{144}\Delta u_{ki}\Big(E_{u^n_{ki}} - \lambda\frac{f'(u_{ki})\Delta u^n_{ki}}{2\sqrt{\sum\limits_{j=1}^{10} f(u_{ji})}}\Big) \\
&\overset{(19)}{=} -\frac{1}{\eta}\sum_{k=1}^{10}\sum_{i=1}^{144}(\Delta u^n_{ki})^2 - \lambda\sum_{k=1}^{10}\sum_{i=1}^{144}\frac{f'(u_{ki})\Delta u^n_{ki}}{2\sqrt{\sum\limits_{j=1}^{10} f(u_{ji})}}.
\end{aligned} \tag{A5}$$

For the third term δ_1 of Equation (A1), it is apparent that the second partial derivatives of the function g_{jk} are bounded. Thus, there exists a positive constant C_2 such that

$$
\begin{aligned}
|\delta_1| = \Bigg| \frac{1}{2} \sum_{j=1}^{J} \sum_{k=1}^{10} & \left[\ (\Delta U_k^n)^T \quad (\Delta S^{n,j})^T \ \right] \\
& \left[\begin{array}{cc} \frac{\partial^2 g_{jk}(\xi_k^n v^{n,j})}{\partial (U_k^n)^2} & \frac{\partial^2 g_{jk}(\xi_k^n v^{n,j})}{\partial U_k^n \partial S^{n,j}} \\ \frac{\partial^2 g_{jk}(\xi_k^n v^{n,j})}{\partial S^{n,j} \partial U_k^n} & \frac{\partial^2 g_{jk}(\xi_k^n v^{n,j})}{\partial (S^{n,j})^2} \end{array} \right] \left[\begin{array}{c} \Delta U_k^n \\ \Delta S^{n,j} \end{array} \right] \Bigg| \\
& \leq C_2 \sum_{j=1}^{J} \sum_{k=1}^{10} || \left[(\Delta U_k^n)^T \ (\Delta S^{n,j})^T \right] ||^2 \\
& \leq C_2 \sum_{j=1}^{J} \sum_{k=1}^{10} (||\Delta U_k^n||^2 + ||\Delta S^{n,j}||^2).
\end{aligned}
\tag{A6}
$$

Notice that

$$
\begin{aligned}
& ||\Delta S^{n,j}||^2 = ||S^{n+1,j} - S^{n,j}||^2 \\
& = ||F(G(\mathrm{ReLU}(G(X^j) \cdot V^{n+1})) \cdot M) - F(G(\mathrm{ReLU}(G(X^j) \cdot V^n)) \cdot M)||^2,
\end{aligned}
$$

and the operation F, G and the matrix M are linear. There exist two positive constants C_3 and C_4 such that

$$
||\Delta S^{n,j}||^2 \leq C_3 ||\mathrm{ReLU}(G(X^j) \cdot V^{n+1}) - \mathrm{ReLU}(G(X^j) \cdot V^n)||^2 \leq C_3 C_4 ||\triangle V^n)||^2.
$$

Then, from Equation (A6),

$$
\begin{aligned}
|\delta_1| & \leq C_2 \sum_{j=1}^{J} \sum_{k=1}^{10} (||\Delta U_k^n||^2 + C_3 C_4 ||\triangle V^n)||^2) \\
& = C_2 J (\sum_{k=1}^{10} \sum_{i=1}^{144} (\Delta u_{ki})^2 + 10 C_3 C_4 \sum_{t=1}^{5} \sum_{l=1}^{5} (\Delta v_{tl}^n)^2) \\
& = C_5 \sum_{k=1}^{10} \sum_{i=1}^{144} (\Delta u_{ki})^2 + C_6 \sum_{t=1}^{5} \sum_{l=1}^{5} (\Delta v_{tl}^n)^2,
\end{aligned}
\tag{A7}
$$

where $C_5 = C_2 J$ and $C_6 = 10 J C_2 C_3 C_4$.

With Taylor's formula, for the fourth term of Equation (A1), we have

$$
\begin{aligned}
& \lambda \sum_{k=1}^{144} \left[\Big(\sum_{i=1}^{10} f(u_{ik}^{n+1}) \Big)^{\frac{1}{2}} - \Big(\sum_{i=1}^{10} f(u_{ik}^n) \Big)^{\frac{1}{2}} \right] \\
& = \lambda \sum_{k=1}^{144} \left[\frac{1}{2} \sum_{i=1}^{10} \frac{f'(u_{ik}^n) \Delta u_{ik}^n}{\Big(\sum_{j=1}^{10} f(u_{ji}^n) \Big)^{\frac{1}{2}}} + \frac{1}{2} \sum_{t,l=1}^{10} \Delta u_{tk} \Delta u_{lk}^n \frac{\partial^2}{\partial u_{tk}^n \partial u_{lk}^n} (\sum_{i=1}^{10} f(u_{ik}^n))^{\frac{1}{2}} |_{U_k^n = \xi_k^n} \right] \\
& \leq \lambda \sum_{k=1}^{144} \sum_{i=1}^{10} \frac{f'(u_{ik}^n) \Delta u_{ik}^n}{2 \Big(\sum_{j=1}^{10} f(u_{ji}^n) \Big)^{\frac{1}{2}}} + \lambda C_7 \sum_{k=1}^{144} \sum_{i=1}^{10} (\Delta u_{ki}^n)^2,
\end{aligned}
\tag{A8}
$$

for some vector ξ_l^n. Because of the existence of the derivative of $f(\cdot)$ and the boundedness of $f''(\cdot)$, there exists some positive constant C_7 such that the inequality of Equation (A8) is true.

From Equations (A1), (A4), (A5), (A7) and (A8), we can obtain

$$\begin{aligned}
&E(W^{n+1}) - E(W^n) \\
&\leq -\frac{1}{\eta}\sum_{t=1}^{5}\sum_{l=1}^{5}(\Delta v_{tl}^n)^2 - \frac{1}{\eta}\sum_{k=1}^{10}\sum_{i=1}^{144}(\Delta u_{ki}^n)^2 - \lambda\sum_{k=1}^{10}\sum_{i=1}^{144}\frac{f'(u_{ki})\Delta u_{ki}^n}{2\sqrt{\sum_{j=1}^{10} f(u_{ji})}} \\
&+ C_5\sum_{k=1}^{10}\sum_{i=1}^{144}(\Delta u_{ki})^2 + C_6\sum_{t=1}^{5}\sum_{l=1}^{5}(\Delta v_{tl}^n)^2 \\
&+ \lambda\sum_{k=1}^{144}\sum_{i=1}^{10}\frac{f'(u_{ik}^n)\Delta u_{ik}^n}{2\Big(\sum_{j=1}^{10} f(u_{ji}^n)\Big)^{\frac{1}{2}}} + \lambda C_7\sum_{k=1}^{144}\sum_{i=1}^{10}(\Delta u_{ki}^n)^2 \\
&\leq (C_6 - \frac{1}{\eta})\sum_{t=1}^{5}\sum_{l=1}^{5}(\Delta v_{tl}^n)^2 + (C_5 + \lambda C_7 - \frac{1}{\eta})\sum_{k=1}^{10}\sum_{i=1}^{144}(\Delta u_{ki}^n)^2.
\end{aligned} \tag{A9}$$

As long as Assumption (2) is satisfied, it can yield $E(W^{n+1}) - E(W^n) \leq 0$. The proof for (i) of the theory is completed. □

Proof for (ii). From the conclusion of (i), we know that the nonnegative sequence $E(W^n)$ monotonically decreases. Hence, there must exist a $E^* \geq 0$ such that $\lim\limits_{n\to\infty} E(W^n) = E^*$. □

Proof for (iii). Let $C_8 = min\{\frac{1}{\eta} - C_6, \frac{1}{\eta} - C_5 - \lambda C_7\}\eta^2$. From Equation (A9) and (ii), we can obtain

$$\begin{aligned}
&\Delta E(W^n) = E(W^{n+1}) - E(W^n) \\
&\leq -min\{\frac{1}{\eta} - C_6, \frac{1}{\eta} - C_5 - \lambda C_7\}\left[\sum_{t=1}^{5}\sum_{l=1}^{5}(\Delta v_{tl}^n)^2 + \sum_{k=1}^{10}\sum_{i=1}^{144}(\Delta u_{ki}^n)^2\right] \\
&= -min\{\frac{1}{\eta} - C_6, \frac{1}{\eta} - C_5 - \lambda C_7\}\eta^2||E_{W^n}||^2 \\
&\leq -C_8||E_{W^n}||^2.
\end{aligned} \tag{A10}$$

That is,

$$C_8||E_{W^n}||^2 \leq -\Delta E(W^n). \tag{A11}$$

If $\lim\limits_{n\to\infty} \Delta E(W^n) = 0$, we have

$$\lim_{n\to\infty} ||E_{W^n}||^2 = 0. \tag{A12}$$

Before proving (iv), we need to review the following lemma [43]:

Lemma A1. *(Wu, Li, Yang, Liu, 2010, Lemma 1). On a bounded closed region $\Phi \subset \mathbb{R}^p$, let $F : \mathbb{R}^p \to \mathbb{R}$ be continuous and differentiable. If the set $\Phi_0 = \{x \in \Phi | \frac{\partial F(x)}{\partial x} = 0\}$ contains only finite points and the sequence $\{x_n\} \subset \Phi$ satisfies $\lim\limits_{n\to\infty} \|\frac{\partial F(x_n)}{\partial x}\| = 0$ and $\lim\limits_{n\to\infty} \|x_{n-1} - x_n\| = 0$, then there exists $x^* \in \Phi_0$ such that $\lim\limits_{n\to\infty} x_n = x^*$.*
□

Proof for (iv). Since the error function $E(W)$ is continuous and differentiable, from Equation (19), Assumption (3) and Lemma A1, we can easily achieve the desired result; there exists a point $W^* \in \Phi_0$ such that $\lim\limits_{n\to\infty} (W^n) = W^*$. □

References

1. Sharma, P.; Singh, A.; Singh, K.K.; Dhull, A. Vehicle identification using modified region based convolution network for intelligent transportation system. *Multimed. Tools Appl.* **2021**, 1–25. [CrossRef]
2. Nguyen, K.C.; Nguyen, C.T.; Nakagawa, M. Nom document digitalization by deep convolution neural networks. *Pattern Recognit. Lett.* **2020**, *133*, 8–16. [CrossRef]
3. Jogin, M.; Mohana; Madhulika, M.S.; Divya, G.D.; Meghana, R.K.; Apoorva, S. Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning. In Proceedings of the 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT), Bangalore, India, 18–19 May 2018; pp. 2319–2323.
4. Li, G.; Tang, H.; Sun, Y.; Kong, J.; Jiang, G.; Jiang, D.; Tao, B.; Xu, S.; Liu, H. Hand gesture recognition based on convolution neural network. *Clust. Comput.* **2019**, *22*, 2719–2729. [CrossRef]
5. Brachmann, A.; Redies, C. Using convolutional neural network filters to measure left-right mirror symmetry in images. *Symmetry* **2016**, *8*, 144. [CrossRef]
6. Yu, D. A new pose accuracy compensation method for parallel manipulators based on hybrid artificial neural network. *Neural Comput. Appl.* **2021**, *33*, 909–923. [CrossRef]
7. Wang, J.; Cai, Q.; Chang, Q.; Zurada, J.M. Convergence analyses on sparse feedforward neural networks via group lasso regularization. *Inf. Sci.* **2017**, *381*, 250–269. [CrossRef]
8. Ng, A.Y. Feature selection, L1 vs. L2 regularization, and rotational invariance. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; p. 78.
9. Bilal, H.; Kumar, A.; Yin, B. Pruning filters with L1-norm and capped L1-norm for CNN compression. *Appl. Intell.* **2021**, *51*, 1152–1160.
10. Gou, J.; Hou, B.; Yuan, Y.; Ou, W.; Zeng, S. A new discriminative collaborative representation-based classification method via L2 regularizations. *Neural Comput. Appl.* **2020**, *32*, 9479–9493. [CrossRef]
11. Xu, Z.; Chang, X.; Xu, F.; Zhang, H. $L_{1/2}$ regularization: A thresholding representation theory and a fast solver. *IEEE Trans. Neural Netw. Learn. Syst.* **2012**, *23*, 1013–1027.
12. Xiao, R.; Cui, X.; Qiao, H.; Zheng, X.; Zhang, Y. Early diagnosis model of Alzheimer's Disease based on sparse logistic regression. *Multimed. Tools Appl.* **2020**, *80*, 3969–3980. [CrossRef]
13. Goulart, J.; Oliveira, P.; Farias, R.C.; Zarzoso, V.; Comon, P. Alternating Group Lasso for Block-Term Tensor Decomposition and Application to ECG Source Separation. *IEEE Trans. Signal Process.* **2020**, *68*, 2682–2696. [CrossRef]
14. Diwu, Z.; Cao, H.; Wang, L.; Chen, X. Collaborative Double Sparse Period-Group Lasso for Bearing Fault Diagnosis. *IEEE Trans. Instrum. Meas.* **2020**, *70*, 1–10. [CrossRef]
15. Zheng, S.; Ding, C. A group lasso based sparse KNN classifier. *Pattern Recognit. Lett.* **2020**, *131*, 227–233. [CrossRef]
16. Friedman, J.; Hastie, T.; Tibshirani, R. A note on the group lasso and a sparse group lasso. *arXiv* **2010**, arXiv:1001.0736.
17. Alemu, H.Z.; Zhao, J.; Li, F.; Wu, W. Group $L_{1/2}$ regularization for pruning hidden layer nodes of feedforward neural networks. *IEEE Access* **2019**, *7*, 9540–9557. [CrossRef]
18. Wu, W.; Fan, Q.; Zurada, J.M.; Wang, J.; Yang, D.; Liu, Y. Batch gradient method with smoothing $L_{1/2}$ regularization for training of feedforward neural networks. *Neural Netw.* **2014**, *50*, 72–78. [CrossRef]
19. Liu, Y.; Li, Z.; Yang, D.; Mohamed, K.S.; Wang, J.; Wu, W. Convergence of batch gradient learning algorithm with smoothing $L_{1/2}$ regularization for Sigma–Pi–Sigma neural networks. *Neurocomputing* **2015**, *151*, 333–341. [CrossRef]
20. Kwon, H.; Go, B.H.; Park, J.; Lee, W.; Lee, J.H. Gated dynamic convolutions with deep layer fusion for abstractive document summarization. *Comput. Speech Lang.* **2021**, *66*, 101–159. [CrossRef]
21. Husain, S.S.; Bober, M. REMAP: Multi-layer entropy-guided pooling of dense CNN features for image retrieval. *IEEE Trans. Image Process.* **2019**, *28*, 5201–5213. [CrossRef]
22. Richter, O.; Wattenhofer, R. TreeConnect: A Sparse Alternative to Fully Connected Layers. In Proceedings of the 2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI), Volos, Greece, 5–7 November 2018.
23. Eckle, K.; Schmidt-Hieber, J. A comparison of deep networks with ReLU activation function and linear spline-type methods. *Neural Netw.* **2019**, *110*, 232–242. [CrossRef]
24. Guo, Z.Y.; Shu, X.; Liu, C.Y.; Lei, L.I. A Recognition Algorithm of Flower Based on Convolution Neural Network with ReLU Function. *Comput. Technol. Dev.* **2018**, 05. Available online: http://en.cnki.com.cn/Article_en/CJFDTotal-WJFZ201805035.htm (accessed on 1 November 2021).
25. Yang, S.C. A study on using deviation function method to reshape a rack cutter. *Int. J. Adv. Manuf. Technol.* **2006**, *30*, 385–394. [CrossRef]
26. Xu, Z.; Zhang, H.; Wang, Y.; Chang, X.; Liang, Y. $L_{1/2}$ regularization. *Sci. China Inf. Sci.* **2010**, *53*, 1159–1169. [CrossRef]
27. Haykin, S. *Neural Networks: A Comprehensive Foundation*, 3rd ed.; Prentice Hall: Hoboken, NJ, USA, 1998.
28. Baldi, P. Gradient descent learning algorithm overview: A general dynamical systems perspective. *IEEE Trans. Neural Netw.* **1995**, *6*, 182–195. [CrossRef]
29. Zhang, Z. *Derivation of Backpropagation in Convolutional Neural Network (CNN)*; University of Tennessee: Knoxville, TN, USA, 2016.
30. Wu, Y. Sparsity of Hidden Layer Nodes Based on Bayesian Extreme Learning Machine. *Control Eng. China* **2017**, *24*, 2539–2543.
31. Özgür, A.; Nar, F.; Erdem, H. Sparsity-driven weighted ensemble classifier. *Int. J. Comput. Intell. Syst.* **2018**, *11*, 962–978. [CrossRef]
32. Olshausen, B.A.; Field, D.J. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature* **1996**, *381*, 607–609. [CrossRef]

33. Bouma, H. Interaction effects in parafoveal letter recognition. *Nature* **1970**, *226*, 177–178. [CrossRef]
34. Carvalho, E.F.; Engel, P.M. Convolutional sparse feature descriptor for object recognition in cifar-10. In Proceedings of the 2013 Brazilian Conference on Intelligent Systems, Fortaleza, Brazil, 19–24 October 2013; pp. 131–135.
35. Abualigah, L.; Diabat, A.; Mirjalili, S.; Abd Elaziz, M.; Gandomi, A.H. The arithmetic optimization algorithm. *Comput. Methods Appl. Mech. Eng.* **2021**, *376*, 113609. [CrossRef]
36. Moreno-Torres, J.G.; Sáez, J.A.; Herrera, F. Study on the impact of partition-induced dataset shift on k-fold cross-validation. *IEEE Trans. Neural Networks Learn. Syst.* **2012**, *23*, 1304–1312. [CrossRef]
37. Burman, P. A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika* **1989**, *76*, 503–514. [CrossRef]
38. Wiens, T.S.; Dale, B.C.; Boyce, M.S.; Kershaw, G.P. Three way k-fold cross-validation of resource selection functions. *Ecol. Model.* **2008**, *212*, 244–255. [CrossRef]
39. Ampazis, N.; Perantonis, S.J. Two highly efficient second-order algorithms for training feedforward networks. *IEEE Trans. Neural Netw.* **2002**, *13*, 1064–1074. [CrossRef] [PubMed]
40. Zubic, S.; Wahlroos, A.; Altonen, J.; Balcerek, P.; Dawidowski, P. Managing Post-fault Oscillation Phenomenon in Compensated MV-networks. In Proceedings of the 13th IET International Conference on Developments in Power System Protection (DPSP 2016), Edinburgh, UK, 7–10 March 2016.
41. Yin, J.; Bian, L.; Fan, Q.; Fan, X.; Ai, H.; Tian, L. Oscillation phenomenon and its mechanism of an energy-saving and emission-reduction system. *Int. J. Energy Sect. Manag.* **2018**, *12*, 314–322. [CrossRef]
42. Dragomir, S.S. New estimation of the remainder in Taylor's formula using Grüss' type inequalities and applications. *Math. Inequalities Appl.* **1999**, *2*, 183–193. [CrossRef]
43. Wu, W.; Li, L.; Yang, J.; Liu, Y. A modified gradient-based neuro-fuzzy learning algorithm and its convergence. *Inf. Sci.* **2010**, *180*, 1630–1642. [CrossRef]