A Forward-backward Learning strategy for CNNs via Separation Index Maximizing at the First Convolutional Layer

Ali Karimi[1]
aliiikarimi@ut.ac.ir
Ahmad Kalhor[1]
akalhor@ut.ac.ir
Mona Ahmadian[2]
m.ahmadian@surrey.ac.uk

[1] University Of Tehran, Tehran, Iran
[2] University of Surrey, Guildford, UK

Abstract

In deep neural networks, involving a forward learning method for initial layers can mitigate the effect of the vanishing gradient problem and enhance the performance of the whole model. In this paper, based on the Separation Index (SI) concept, a forward-backward learning strategy is proposed for Convolutional Neural Networks (CNNs). At first, the concept of SI as a supervised complexity measure is explained, and then the learning strategy is introduced in two phases. In the first phase, as the forward learning part, the first layer of the CNN is learned by maximizing the SI; then in the second phase, the further layers are trained through the error backpropagation algorithm. To maximize the SI, a novel variant of triplet loss is introduced, and it is optimized by a quasi-least squares (QLS) error technique. The proposed learning strategy is applied to VGG, AlexNet, ResNet, Inception, and datasets such as CIFAR100, CIFAR10 and Fashion MNIST. A comparison of the proposed learning strategy with some state-of-the-art learning techniques shows that it is superior to all the models and datasets evaluated.

1 Introduction

In recent years, deep learning has advanced significantly. The use of deep learning has been demonstrated in a variety of fields, including video, image, audio, and data analysis. Deep learning applications and their ability to solve different problems have led to various studies to improve their performance. Accordingly, novel convolutional neural network architectures such as VGG [22] and EfficientNet [24] have been proposed. Also, new architectures such as transformers [6], improved loss functions [10], and novel learning strategies [9] introduced.

Although error backpropagation algorithms can achieve very high accuracy in image classification problems, they are constrained by the vanishing gradient issue. It is not possible for the first layer to be properly learned. The vanishing gradient occurs more when we have more layers and use deeper neural networks. With the development of deeper networks, this problem will become more visible. Due to this issue, a variety of approaches have been

proposed, including activation functions like Rectified linear unit (ReLU) [18] that avoid saturation in the early layers , use normalization techniques such as batch normalization or layer normalization to reduce the dependence of the gradients on the scale of the activation values [11] . But the issue still exists.

It is possible to solve the Vanshing gradient problem using forward learning, but error backpropagation is still more accurate and powerful [9] . We have proposed a method that combines forward learning and error backpropagation with a slight modification to improve accuracy. our novel learning is developed in two phases:

- At the first phase, as the forward learning part, After a batch normalization layer is applied to the data, the patches from the input data are extracted and principal component analysis (PCA) is carried out on the data based on these patches. The eigenvalues derived by PCA in the convolution layer are used as the values for the filter in the convolution layer. The output of the convolution layer is then received, and QLS is performed on them to calculate the updated values of the convolution layer filters. This continues until we reach maximum classification accuracy in the first layer.
- At the second phase, as the backward learning part, the output of the first layer is given as input to the second layer and the network is trained by an error backprogration algorithm.

2 Related Works

The related works section is organized into two sections. First, learning strategies are discussed, and then different complexity measures are reviewed.

2.1 Layerwise Learning

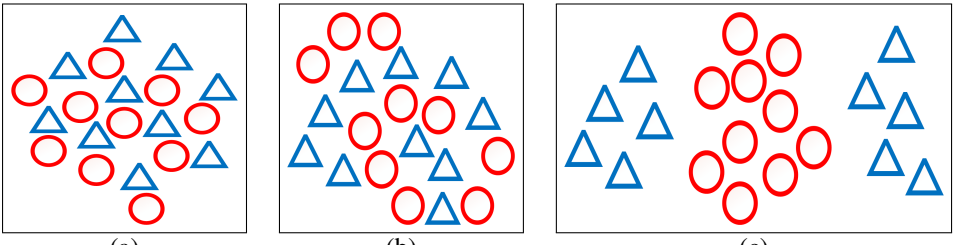
It is important to choose a learning strategy that determines how the network's weights are updated during training. A popular learning strategy is the error backpropagation algorithm based on stochastic gradient descent (SGD) or Adam [12] or RAdam [17]. Layerwise or forward training are also other innovative methods that can be used for neural network training. Hinton [9] has introduced a novel learning strategy for neural networks named the Forward-Forward Algorithm. By using this algorithm, forward and backward backpropagation passes are replaced by two forward passes. A greedy layer-wise learning approach was introduced by Bengio et al [3] . The main purpose of greedy layer-wise pretraining is to initialize the weights of a deep neural network layer by layer, beginning with the first layer. In each layer, a separate autoencoder is trained, using the previous layer's output as an input. After each layer has been trained, the entire network is fine-tuned using backpropagation.

2.2 Complexity measures

To evaluate the challenges of the dataset in a classification problem, several complexity measures have been introduced, [20]. Table 1 presents some introduced complexity measures and their overall evaluating approaches.

According to the given explanations about the Separation Index (SI) in [20], SI indicates how much the data points with different labels are separated from each other. In addition, it has been explained that while the SI increases layer by layer in a deep neural network

(DNN), the margin among different classes will increase and the generalization of the network increases. On the other hand, increasing the margin between different classes leads to boundary complexity reduction and hence one can consider SI as a variant of neighborhood measures. Fig 1 shows an illustrative example where data points of two classes (with circle and triangle indicators) have (a) low, (b) medium, or (c) high separation indices. As it is seen, when the SI increases in (c), the complexity boundaries between data points of two classes are less than the cases (a) and (b).



(a) (b) (c)

Figure 1: An illustrative example where data points of two classes (with the circle and triangle indicators) have (a) low ($SI \approx 0$) , (b) medium ($SI \approx 0.5$), or (c) high ($SI \approx 1$) separation indices.

Complexity measures	Overall evaluating approach
Feature-based	Discovering informative features by evaluating each feature independently [8, 15]
Linearity separation	Evaluating the linearly separation of different classes [4]
Neighborhood	Evaluating the shape of the decision boundary to distinguish different classes overlap [15, 16]
Network	Evaluating the data dataset structure and relationships by representing it as a graph [4]
Dimensionality	Evaluating the sparsity of the data and the average number of features at each dimension [8, 16]
Class imbalanced	Evaluating the proportion of dataset number between different classes [16]

Table 1: Some complexity measures and their evaluating approaches in a classification problem

The first order SI, in fact counts the number of all data points having the same labels with their nearest data points (neighbors with minimum distances), [20]:

$$\mathrm{SI}\left(\{\mathbf{z}^{\mathrm{q}}\}_{q=1}^{Q},\{l^q\}_{q=1}^{Q}\right)=\frac{1}{Q}\sum_{k=1}^{Q}\varphi\left(l^q-l^{q_{near}}\right)$$
$$\varphi(v)=\begin{cases}1 & v=0\\ 0 & v\neq 0\end{cases} \tag{1}$$

$$q_{\mathrm{near}}=\arg\min_{h}\left\|z^{\mathrm{q}}-z^{h}\right\|^2$$
$$h\in\{1,2,\ldots,\mathrm{Q}\}\quad\text{and}\quad h\neq q \tag{2}$$

Where $\{\mathbf{z}^q,l^q\}_{q=1}^{Q}$ denotes the dataset with their labels and $\varphi(.)$ denotes the Kronecker delta function. From (1) and (2), it is understood that the separation index is normalized between "zero" and "one".

Remark 1: In this paper, all applied distance norms as $\|x\|^2$ are L_2 norm.

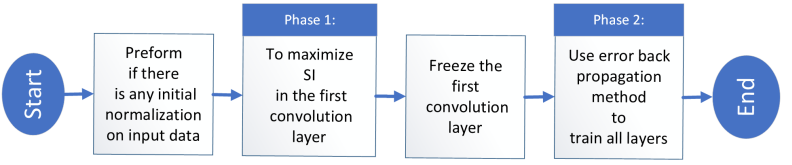


Figure 2: This flowchart indicates the learning method in which the quasi-LS is utilized to learn the first convolution layer (phase 1), and other layers are learned by an error backpropagation algorithm (phase 2).

3 Method

To increase the generalization of a DNN, it is required that SI increases along with the layers of a DNN, [20]. Using different error backpropagation algorithms indirectly cause increasing the SI. However, due to the vanishing gradient problem, the SI cannot significantly increase through initial layers, particularly at the first convolution layer as the base of the filtering process in a CNN. Here, to maximize the SI at the first layer of a CNN, QLS technique is proposed, [21]. Fig 2 shows the flowchart which is used to learn a CNN, where only the first convolution layer is learned by QLS, but the other layers are updated by the error backpropagation algorithm.

3.1 Quasi-Least Square Algorithm

Assume there are Q input training patterns at input layer: $\{x^q\}_{q=1}^{Q}$. Applying M convolutional filters and biases,$\{(\theta_l,b_l)\}_{l=1}^{M}$, to $\{x^q\}_{q=1}^{Q}$ and then activating the units of M feature maps by ReLU, $\{Z^q\}_{q=1}^{Q}$ are appeared, equation 3. In this paper, each x^q and z^q are reshaped as a vector.

$$\{x^q\}_{q=1}^{Q} \quad \text{Filters:} \xrightarrow{\{(\theta_l,b_l)\}_{l=1}^{M}} \xrightarrow{\text{ReLU}} \{z^q\}_{q=1}^{Q} \tag{3}$$

In order to maximize the SI at $\{z^q\}_{q=1}^{Q}$,it is aimed that at the first stage, filter parameters, $\{(\theta_l,b_l)\}_{l=1}^{M}$ are initialized by using PCA technique, and then through some iterations, $\{(\theta_l,b_l)\}_{l=1}^{M}$ are updated by following rules:

- **Rule 1:** each $\mathbf{z}^q$ becomes near to its nearest neighbor having the same label, $z^{q_{fr}}$
- **Rule 2:** each $\mathbf{z}^q$ becomes far from its nearest neighbor having the different label, $z^{q_{en}}$

According to above defined rules the following minmax loss function (4) is employed to maximize SI at first convolution layer:

$$J\left(\{(\theta_l,b_l)\}_{l=1}^{M}\right) =$$

$$\sum_{q=1}^{Q}\gamma_q\left\|\mathbf{z}^q-\mathbf{z}^{q_{fr}}\right\|^2-\sum_{q=1}^{Q}\gamma_q\left\|\mathbf{z}^q-\mathbf{z}^{q_{en}}\right\|^2 \tag{4}$$

$$\left\{\left(\theta_l^*,b_l^*\right)\right\}_{l=1}^{M}=\arg\min J\left(\{(\theta_l,b_l)\}_{l=1}^{M}\right)$$

where γ_q denotes the important weight of z^q, which is explained later in the presented quasi-LS algorithm. About the minimax loss function (4), it is required that the first part of the loss function is minimized and the second part is maximized. Ignoring the act of nonlinear activation function, the least-square (LS) technique is the best solution to minimize (4) but due to the nonlinearity and non-derivability of the activation function (Relu), gradient-based methods cannot be used for optimization purposes. However, the quasi-LS technique as a powerful hybrid optimization technique is employed. In a quasi-LS technique, at each iteration, by freezing the operation of the activation function, all positive units at the feature maps participate in an LS technique to update the parameters of the filters. However, since after each update, the operation of the activation function will change, the former LS technique should be repeated. It is expected that by repeating the LS technique for several iterations, the filter parameters converge to more optimal parameters, [21]. Table 2 presents the list of variables, which are used in the proposed algorithm.

Row	Variable	Explanation
1	K	Number of classes in the classification problem
2	Q	Number of all training patterns (each class has at least two patterns)
3	q	A counter variable for patterns $q \in \{1,2,\ldots,Q\}$
4	x^q	The vector of all units in qth pattern at the input layer
5	z^q	The vector of all units in qth pattern at the first convolutional layer
6	γ^q	The important weight of Z^q at each iteration.
7	$\mathbf{M}$	The number of feature maps at the first convolutional layer.
8	n	The number of units at each feature map.
9	l	A counter for number of units at each feature map.
10	n_F	The number of parameters at each filter without bias.
11	$\theta_l \in R^{n_F \times 1}$	the l^{th} filter parameters.
12	$b_l \in R$	The l^{th} bias corresponded to l^{th} filter.
13	$\sigma_\rho^q \in R^{\mathbf{1} \times n_F}$	The ρ^{th} chosen patch from pattern x_q .
14	$z_l^q \in R^{1 \times n}$	The l^{th} feature map from pattern z_q .
15	$z_{l\rho}^q \in R$	The ρ^{th} unit of l^{th} feature map from pattern z_q .
16	t	A counter for learning iterations in the Quasi-Least Square method.
17	t_{stop}	The number of iterations used in the Quasi-Least Square method.

Table 2: The list of variables used in the quasi-least square method

3.2 Forward Learning Phase

The forward learning phase of the proposed algorithm is based as follows:

3.2.1 Stage1

In this stage $\left\{\left(\widehat{\theta}_l^1, \hat{b}_l^1\right)\right\}_{l=1}^{M}$, as filter parameters and biases, are initialized by applying PCA on the input data: $\{x^q\}_{q=1}^{Q}$. It is assumed that M is an even number and $M \leq 2n_F$. Considering that there are Q patterns at the first layer where each one has n patches, the mean of all patches of all the training patterns is computed as equation (5):

$$\overline{\sigma} = \frac{1}{Q_B} \sum_{q=1}^{Q} \sum_{\rho=1}^{n} \sigma_p^q \quad Q_B = nQ \tag{5}$$

Now the covariance matrix of patches Σ is defined as equation (6) :

$$\Sigma = \frac{1}{Q_B} \tilde{X}^T \tilde{X} \quad \tilde{X}^T = \left[\tilde{\sigma}_1^1 \cdots \tilde{\sigma}_n^1 \tilde{\sigma}_1^2 \cdots \tilde{\sigma}_n^2 \cdots \tilde{\sigma}_1^Q \cdots \tilde{\sigma}_n^Q \right]$$

$$\sigma_p^q = \sigma_p^q - \bar{\sigma} \tag{6}$$

where σ_p^q denotes the vector form of p th patch of q th example. It is known that Σ is a symmetric and positive semi-definite matrix. By applying singular value decomposition, to Σ :

$$\begin{gathered} \Sigma = \mathrm{V}^{\mathrm{T}} \Lambda \mathrm{V} \quad \mathrm{V} = [v_1 v_2 \ldots v_{n_F}] \\ \Lambda = \mathrm{diag}\left(\lambda_1 \lambda_2 \cdots \lambda_{n_F}\right) \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{n_F} \end{gathered} \tag{7}$$

where V as matrix of eigenvectors includes unitary orthogonal columns and Λ as a diagonal matrix includes non-negative eigenvalues. Each eigenvector $v_\tau, \tau \in \{1,2,\ldots,n_F\}$ is corresponded with λ_τ and eigenvectors with bigger eigenvalues are more important because more data points are propagated along their determined directions. Now, all M vectors of filter parameters and biases are initialized as equation (8):

$$\begin{gathered} \widehat{\theta}_l^1 = \begin{cases} v_\tau & \text{if } l = 2\tau - 1 \\ -v_\tau & \text{if } l = 2\tau \end{cases} \quad \widehat{b}_l^1 = -\overline{\sigma}^T \widehat{\theta}_l^1 \\ \tau \in \{1,2,\ldots,n_F\} \quad l \in \{1,2,\ldots,M\} \end{gathered} \tag{8}$$

Remark 2: For when $M > 2n_F$, one can initialize the first $2n_F$ filters by (7) but other $M - 2n_F$ filters can be initialized by using a random initialization method.

3.2.2 Stage2

Repeat following steps for t=1 up to $t = t_{\text{stop}}$:

Step 1: Define the matrix of patches, X, from patterns in the first layer (9) :

$$\mathrm{X} = \begin{bmatrix} \mathrm{X}^1 \\ \vdots \\ \mathrm{X}^Q \end{bmatrix} \quad \mathrm{X}^{\mathrm{q}} = \begin{bmatrix} \sigma_1^q & 1 \\ \vdots & \vdots \\ \sigma_n^q & 1 \end{bmatrix} \tag{9}$$

Step 2: Update the patterns at layer L as equation (10):

$$\begin{gathered} \widehat{z}^q = \left[\hat{z}_1^q \hat{z}_2^q \cdots \hat{z}_M^q \right] \\ \hat{z}_l^q = \left[\hat{z}_{l,1}^q \hat{z}_{l,2}^q \cdots\cdots \hat{z}_{l,n}^q \right] \\ \hat{z}_{l,\rho}^q = \mathrm{ReLU}\left(\sigma_p^q \widehat{\theta}_l^t + \widehat{b}_l^t \right) \end{gathered} \tag{10}$$

Step 3: For $\hat{z}^q$, find the nearest neighbor pattern with the same label (friend pattern) as equation (11):

$$\widehat{z}^{q_{fr}} = \mathrm{near}_{fr}\left(\hat{z}^q\right) \tag{11}$$

where $\text{near}_{fr}(\hat{\mathbf{z}}^q)$ denotes a function which finds the nearest neighbor pattern from patterns which have the same label with $\hat{z}^q$. Now, decompose $\hat{z}^{q_{fr}}$ as equation (12) :

$$\begin{aligned} \hat{z}^{q_{fr}} \rightarrow \hat{z}^{q_{fr}} &= \left[\hat{z}_1^{q_{fr}} \hat{z}_2^{q_{fr}} \cdots \hat{z}_M^{q_{fr}}\right] \\ \hat{z}_l^{q_{fr}} \rightarrow \hat{z}_l^{q_{fr}} &= \left[\hat{z}_{l,1}^{q_{fr}} \hat{z}_{l,2}^{q_{fr}} \cdots \hat{z}_{l,n}^{q_{fr}}\right] \end{aligned} \tag{12}$$

Step 4: For $\hat{\mathbf{z}}_q^L$, find the nearest neighbor pattern from patterns which have different label with $\hat{z}^q$:

$$\hat{z}^{q_{en}} = \text{near}_{en}(\hat{z}^q) \tag{13}$$

where $\text{near}_{en}(\hat{\mathbf{z}}^q)$ denotes a function which finds the nearest neighbor pattern from patterns which have different label with $\hat{z}^q$. Now, decompose $\hat{z}^{q_{en}}$ as equation (14) :

$$\begin{aligned} \hat{\mathbf{z}}^{q_{en}} \rightarrow \hat{\mathbf{z}}^{q_{en}} &= \left[\hat{z}_1^{q_{en}} \hat{\mathbf{z}}_2^{q_{en}} \cdots \hat{\mathbf{z}}_M^{q_{en}}\right] \\ \hat{\mathbf{z}}_l^{q,en} \rightarrow \hat{\mathbf{z}}_l^{q_{en}} &= \left[\hat{z}_{l,1}^{q_{en}} \hat{z}_{l,2}^{q_{en}} \cdots \hat{z}_{l,n}^{q_{en}}\right] \end{aligned} \tag{14}$$

Step 5: Form following data matrices including

$$Z_l^{fr} = \begin{bmatrix} Z_l^1 fr \\ \vdots \\ Z_l^{Q_{fr}} \end{bmatrix} \quad Z_l^{q_{fr}} = \begin{bmatrix} z_{l,1}^{q_{fr}} \\ \vdots \\ z_{l,n} \end{bmatrix} \tag{15}$$

$$Z_l^{en} = \begin{bmatrix} Z^1 en \\ \vdots \\ Z^{Q_{en}} \end{bmatrix} \quad Z_l^{q_{en}} = \begin{bmatrix} z_{l,1}^{q_{en}} \\ \vdots \\ z_{l,n}^{q_{en}} \end{bmatrix} \tag{16}$$

Step 6: Define the weight matrix of all patches of all patterns as equation (17):

$$\begin{aligned} \Phi_l &= \begin{bmatrix} \Phi_l^1 & \mathbf{0} & 0 \\ \mathbf{0} & \ddots & \mathbf{0} \\ 0 & \mathbf{0} & \Phi_l^Q \end{bmatrix} \\ \Phi_l^q &= \gamma_q \begin{bmatrix} \text{sign}\left(\hat{z}_{l,1}^q\right) & \mathbf{0} & 0 \\ \mathbf{0} & \ddots & \mathbf{0} \\ 0 & \mathbf{0} & \text{sign}\left(\hat{z}_{l,n}^q\right) \end{bmatrix} \end{aligned} \tag{17}$$

where γ_q denotes the important weight of qth pattern. γ_q is defined as equation (18):

$$\gamma_q = \begin{cases} 2 & \text{if } \left\|\hat{Z}^q - \hat{z}^{q_{en}}\right\| \leq \left\|\hat{\mathbf{z}}^q - \hat{z}^{q_{fr}}\right\| \\ 1 & \text{otherwise} \end{cases} \tag{18}$$

With regard to equation (18), when the nearest neighbor of $\hat{z}^q$q does not have the same label with it, the important weight of $\hat{z}^q$ increases to 2, in quasi-LS technique.

Step 7: update the parameters of filters and biases. At first, compute following vector (equation 19):

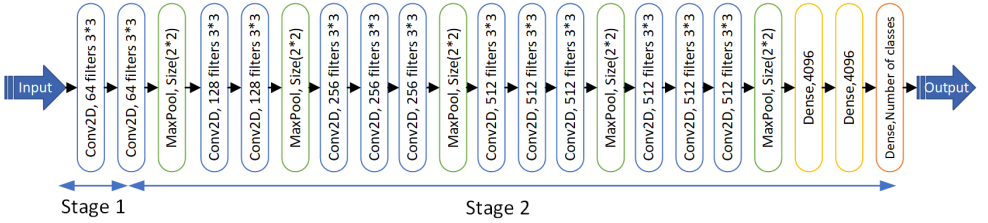


Figure 3: An example of our learning strategy for VGG16 [22] is demonstrated.

$$\begin{aligned} \psi^t = \psi^*_{fr} - \psi^*_{en} \quad \psi^*_{\mathrm{fr}} = \arg\min_{\psi_{\mathrm{fr}}} \left\| \mathrm{X}\psi_{\mathrm{fr}} - Z_l^{fr} \right\|^2 \\ \psi^*_{en} = \arg\min_{\psi_{en}} \|\mathrm{X}\psi_{en} - Z_l^{en}\|^2 \end{aligned} \tag{19}$$

where ψ^*_{fr} and ψ^*_{en} are optimized by least-square technique as equation (20):

$$\begin{aligned} \psi^*_{\mathrm{fr}} = \left(\mathrm{X}^{\mathrm{T}}\Phi_1^{\mathrm{t}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\Phi_l Z_l^{fr} \\ \psi^*_{en} = \left(\mathrm{X}^{\mathrm{T}}\Phi_1^{\mathrm{t}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\Phi_l Z_l^{en} \end{aligned} \tag{20}$$

Now, update the parameters of filters and biases as equation (21):

$$\begin{bmatrix} \widehat{\theta}_l^{t+1} \\ \hat{b}_l^{t+1} \end{bmatrix} = \begin{bmatrix} \widehat{\theta}_l^{t} \\ \hat{b}_l^{t} \end{bmatrix} + \psi^t \tag{21}$$

3.2.3 Satge3

Put $\theta_l = \widehat{\theta}_l^{t-1}$ and $b_l = \hat{b}_l^{t-1}$ as final opted filter parameters and biases at the first convolutional layer.

3.3 Backward Learning Phase

In the backward learning phase, first the first layer is freeze and then the other of the layers are trained using error backpropagation algorithm. In this phase, training is no different from the error backpropagation technique and training continues until the appropriate epoch.

4 Experimental Results

4.1 Implementation Details

To evaluate the proposed learning strategy, our proposed learning strategy compares to the error backpropagation algorithm and greedy layer-wise leaning on well-known architectures and datasets. Five CNN architectures including AlexNet [14], VGG16 [22], InceptionV3[23], SimCNN [7] and ResNet50[8], and three datasets including CIFAR10 [13], CIFAR100[13], and Fashion-MNIST [25] were used to evaluate the proposed learning strategy.

In the backward phase of our learning strategy and backpropagation error algorithm, we trained models 250 epoch with Adam as an optimizer and categorical cross entropy as a loss function, Also our batch size is 128 and the learning rate is 0.01.

Dataset	Learning Method	AlexNet	VGG16	ResNet50	InceptionV3
CIFAR10	Backpropagation algorithm	84.61	92.95	93.17	94.20
	Our proposed learning strategy	**85.84**	**94.81**	**95.02**	**95.43**
	Percentage of improvement	1.23	1.86	1.85	1.23
CIFAR10 - (plane,truck)	Backpropagation algorithm	96.45	97.18	97.64	97.09
	Our proposed learning strategy	**97.09**	**98.36**	**98.49**	**97.77**
	Percentage of improvement	0.64	1.18	0.85	0.68
CIFAR10 - (plane,cat,bird)	Backpropagation algorithm	96.21	96.80	96.88	96.81
	Our proposed learning strategy	**96.62**	**97.63**	**97.16**	**97.14**
	Percentage of improvement	0.41	0.83	0.28	0.33
CIFAR100	Backpropagation algorithm	62.22	70.98	75.30	76.31
	Our proposed learning strategy	**62.45**	**71.74**	**75.58**	**76.72**
	Percentage of improvement	0.23	0.76	0.28	0.41
Fashion-MNIST	Backpropagation algorithm	92.53	94.17	95.24	95.78
	Our proposed learning strategy	**92.65**	**94.73**	**95.38**	**95.91**
	Percentage of improvement	0.12	0.56	0.14	0.13

Table 3: Comparing the accuracy of our learning strategy in different architectures and datasets with error backpropagation algorithm

4.2 Results

Dataset	Learning Strategy	SimCNN (k=1) [0]	SimCNN (k=2) [0]	SimCNN (k=3) [0]
CIFAR10	Our Method	88.7	90.8	92.9
	Belilovsky et al [0]	88.3	90.4	91.7

Table 4: Comparison between networks and datasets (our method vs other layer-wise methods)

The results of the experiments are given in Table 3 and 4. As shown in Table 3, in all architectures and datasets the accuracy is increased by our proposed learning strategy. This improvement is more significant in datasets with fewer classes, as demonstrated in Table 3, The CIFAR10 dataset has the same number of features as the CIFAR100 dataset, but the CIFAR10 dataset has fewer classes. The accuracy of our learning strategy in CIFAR10 shows a greater improvement ratio to backpropagation than in CIFAR100. Furthermore, the improvement ratio in datasets with more features is much higher, for instance, the percentage of improvement in CIFAR10 and CIFAR100 was higher than that in Fashion-MNIST. In addition, the accuracy increase ratio in 2-class mode is higher than in 3-class mode in the CIFAR10 dataset. Also, our learning strategy as shown in Table 4 superior to greedy layerwise learning [0].

4.3 The Impact of Number of Layers In Forward Learning

We have applied the forward learning method to further layers and Table 5 summarizes the accuracy results. As it is seen, the best accuracy is for when we apply our forward learning method to only the first layer. In fact, providing a reliable basis layer by the forward learning method is enough to have better optimization on the further layers by error backpropagation method. In addition, by applying the forward learning method to further layers, the required memory to implement QLS algorithm increses significantly.

No. of Forward layers	Run Time(m:s)	ACC
0 (Backpropagation Method)	144 : 16	**92.95**
1 (Our Method)	127 : 44	**94.81**
2	131 : 18	94.76
3	125 : 45	94.32
4	124 : 58	93.64
5	120 : 06	93.73
6	114 : 05	93.66
...	...	...
11	87 : 03	93.29
12	85 : 51	93.25
13	81 : 14	93.13

Table 5: Comparison of Time Complexity and Accuracy of the Proposed Method in Different Layers with Backpropagation Learning Strategy for VGG16 on CIFAR10.

5 Conclusion

In this paper, a forward-backward learning strategy was proposed to enhance convolutional neural network learning. In the forward learning phase, the first layer of the CNN was learned by maximizing the SI which was known as a complexity measure. In fact, in this learning phase, by using a novel variant of triplet loss and applying an iterative quasi-least square optimization technique to input data, the resulting features with different labels were explicitly separated from each other. Then, in the backward learning phase, further layers were learned by the conventional error backpropagation algorithm. Next, the proposed method was applied to some well-known CNNs and datasets. The comparison results clearly demonstrated that such a straightforward learning strategy improved the learning of CNNs in all cases. For example in the classification of CIFAR10 images, the accuracy of VGG and ResNet50 increased near two percentages compared to the error backpropagation. Furthermore, in comparison with greedy layerwise learning, the proposed method was superior.

References

[1] Mitra Basu and Tin Kam Ho. *Data complexity in pattern recognition*. Springer Science & Business Media, 2006.

[2] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pages 583–593. PMLR, 2019.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19, 2006.

[4] Léon Bottou and Chih-Jen Lin. Support vector machine solvers. *Large scale kernel machines*, 3(1):301–320, 2007.

[5] Lisa Cummins. *Combining and choosing case base maintenance algorithms*. PhD thesis, University College Cork, 2013.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

[7] Luís PF Garcia, André CPLF de Carvalho, and Ana C Lorena. Effect of label noise in the complexity of classification problems. *Neurocomputing*, 160:108–119, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.

[10] Xiaolin Huang, Lei Shi, and Johan AK Suykens. Support vector machine classifier with pinball loss. *IEEE transactions on pattern analysis and machine intelligence*, 36 (5):984–997, 2013.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[15] Enrique Leyva, Antonio González, and Raul Perez. A set of complexity measures designed for applying meta-learning to instance selection. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):354–367, 2014.

[16] Ana C Lorena, Ivan G Costa, Newton Spolaôr, and Marcilio CP De Souto. Analysis of complexity indices for classification problems: Cancer gene expression data. *Neurocomputing*, 75(1):33–42, 2012.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

[18] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[19] Albert Orriols-Puig, Núria Macia, and Tin Kam Ho. Documentation for the data complexity library in c++. *Universitat Ramon Llull, La Salle*, 196:1–40, 2010.

[20] Mohsen Saffar and Ahmad Kalhor. Evaluation of dataflow through layers of convolutional neural networks in classification problems. *Expert Systems with Applications*, 224:119944, 2023.

[21] Justine Shults and Joseph M Hilbe. *Quasi-least squares regression*. CRC Press, 2014.

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[24] X TanM and V LeQ. Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning. New York: IEEE*, volume 97, pages 6105–6114, 2019.

[25] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.