11-20-2018

Wide and Deep Neural Networks in Remote Sensing: A Review

okan ersoy
Purdue University, ersoy@purdue.edu

Follow this and additional works at: https://docs.lib.purdue.edu/ecetr
WIDE AND DEEP NEURAL NETWORKS  
FOR CLASSIFICATION IN REMOTE SENSING: A REVIEW

O. K. Ersoy  
Purdue University  
School of Electrical and Computer Engineering  
West Lafayette, IN  47907-1285
ABSTRACT
Wide and deep neural networks in multispectral and hyperspectral image classification are discussed. Wide versus deep networks have always been a topic of intense interest. Deep networks mean large number of layers in the depth direction. Wide networks can be defined as networks growing in the vertical direction. Then, wide and deep networks are networks which have growth in both vertical and horizontal directions. In this report, several directions in order to achieve such networks are described.

We first review a methodology called Parallel, Self-Organizing, Hierarchical Neural Networks (PSHNN’s) which have stages growing in the vertical direction, and each stage can be a deep network as well. In turn, each layer of a deep network can be a PSHNN. The second methodology involves making each layer of a deep network wide, and this has been discussed especially with deep residual networks. The third methodology is wide and deep residual neural networks which grow both in horizontal and vertical directions, and include residual learning principles for improving learning. The fourth methodology is wide and deep neural networks in parallel. Here wide and deep networks are two parallel branches, the wide network specializing in memorization while the deep network specializing in generalization. In leading to these methods, we also review various types of PSHNN’s, deep neural networks including convolutional neural networks, autoencoders, and residual learning.

Partially due to moderate sizes of current multispectral and hyperspectral image sets, design and implementation of wide and deep neural networks hold the potential to yield most effective solutions. These conclusions are expected to be valid in other areas with similar data structures as well.

Keywords: wide and deep neural networks, remote sensing, multispectral, hyperspectral, classification
1. INTRODUCTION
Deep neural networks have recently led to a number of breakthroughs in a number of areas, notably in computer vision and image classification including remote sensing hyperspectral images. There are two major reasons leading to these results. One is new ways of increasing the number of layers of deep networks, and the second one is the availability of very large datasets. In spite of these developments, deep neural networks still have some difficulties such as eventual degradation as the number of layers increases, and exploding/vanishing gradients.

Wide versus deep networks have always been a topic of intense interest. Deep networks mean large number of layers in the depth direction. Wide networks can be defined as networks growing in the vertical direction. Then, wide and deep networks are networks which have growth in both vertical and horizontal directions. In this paper, we review several directions in order to achieve such networks, especially in the area of remote sensing. Thus, the paper is focused upon the interplay between wide and deep neural networks. For a more complete reviews of deep neural networks in remote sensing, some excellent references would be [31] thru [35].

We will first review a methodology called Parallel, Self-Organizing, Hierarchical Neural Networks (PSHNN’s) which have stages growing in the vertical direction, and each stage can be a deep network as well. In turn, each layer of a deep network can be a PSHNN. The second methodology involves making each layer of a deep network wide, and this has been discussed especially with deep residual networks. The third methodology is wide and deep residual neural networks which grow both in horizontal and vertical directions, and include residual learning principles for improving learning. The fourth methodology is wide and deep neural networks in parallel. Here wide and deep networks are two parallel branches, the wide network specializing in memorization while the deep network specializing in generalization. In leading to these methods, we also review various types of PSHNN’s, deep neural networks including convolutional neural networks, autoencoders, and residual learning.

All the papers cited below cover applications of the methods in classification of remote sensing images, especially hyperspectral images. They have many experimental results, showing that they achieve better results than previous statistical methods, such as the maximum likelihood method, and previous neural networks such as the backpropagation network. PHSNN's have time complexity similar to previous networks, but they are much faster during testing because of their inherent nature for parallel processing of stages. Deep neural networks are slower and more difficult to learn due to many deep stages, but they make up by achieving high accuracies.

The paper consists of 9 sections. Section 2 summarizes most commonly used hyperspectral datasets for classification. Sections 3 thru 8 include classification results of remote sensing datasets, especially hyperspectral images. Section 3 discusses various types of PSHNN’s. Sections 4, 5, and 6 discuss consensual neural networks, statistical self-organizing neural networks, and PSHNN’s with continuous inputs and outputs. Section 7 is a review of deep neural networks. Section 8 covers new existing wide and deep neural networks. Section 9 is conclusions.
2. MOST COMMONLY USED HYPERSPECTRAL DATASETS

Even though there are considerably more number of publicly available hyperspectral datasets used in classification, the most common ones with ground truth are Indian Pines, Salinas, University of Pavia, Kennedy Space Center (KSC), and Washington, DC datasets [1]. The Indian Pines dataset was gathered by the NASA AVIRIS spectrometer in northwestern Indiana. The number of spectral bands in the 0.4-2.45 \( \mu m \) region is 220, and the spectral resolution is 20 m. Some classes with few labeled training and testing samples are sometimes discarded, leaving 8 classes for training and testing. The Salinas dataset was also collected by the AVIRIS spectrometer over Salinas Valley, California. It has a spatial resolution of 3.7 m. The images are of size 512x512x220. There are 16 classes, mostly consisting of vegetables, bare soils and vineyard fields. The University of Pavia dataset was collected by ROSIS spectrometer, covering the city of Pavia in Italy. It has a spatial resolution of 1.3 m. The spectral coverage is in the region 0.43–0.86 \( \mu m \). The images are of size 610x340x103. The number of classes with ground truth map is 9. The KSC dataset was acquired by the AVIRIS spectrometer in 224 bands of 10 nm width with center wavelengths from 400 - 2500 nm. It has a spatial resolution of 18 m. After removing water absorption and low SNR bands, 176 bands are usually used for analysis. Training data were selected using land cover maps derived from color infrared photography provided by the Kennedy Space Center and Landsat Thematic Mapper (TM) imagery. Discrimination of land cover for this environment is difficult due to the similarity of spectral signatures for certain vegetation types. For classification purposes, 13 classes representing the various land cover types that occur in this environment were defined for the site. The Washington, DC dataset was obtained with a 191-band airborne hyperspectral data flightline over the Washington, DC Mall area [2]. The sensor system measured pixel response in 210 bands in the 0.4–2.4- m region of the visible and infrared spectrum. Bands in the 0.9–1.4- m region where the atmosphere is opaque is often omitted from the dataset, leaving 191 bands. The dataset contains 1280 scan lines with 307 pixels in each scan line. There are seven ground cover types: Roofs, Street, Path, Grass, Trees, Water, and Shadow.
3. PARALLEL, SELF-ORGANIZING, HIERARCHICAL NEURAL NETWORKS

There are several types of PSHNN. In the first type of PSHNN, there is an error detection scheme at the output of each stage neural network (SNN). If an input vector is accepted for the next SNN, it goes through a nonlinear transformation before system (SSOLS), there are two stages, a mapping stage and a learning stage, corresponding being input to the next SNN. In the second type of PSHNN called statistical, self-organizing learning system, there are three layers, namely, input layer, enhancement layer, and output layer [Hoiming]. The input nodes and the enhancement nodes are cross-connected together by a randomly generated enhancement weight matrix, while the output nodes are connected to both the input nodes and the enhancement nodes by another weight matrix. The enhancement nodes are generated and learned incrementally. In the second stage, the weight matrix is learned by least squares. In the third type of PSHNN, there are continuous-valued inputs and outputs, the succeeding stages are trained to learn remaining output errors. This is similar to projection pursuit. In the fourth type of PSHNN called consensual neural networks, the input data is transformed several times as input to SNN’s as if they were independent inputs. The independent inputs are first classified using the stage neural networks. The output responses from the stage networks are then weighted and combined to make a consensual decision.

3.1 PSHNN of the First Type

Parallel, self-organizing, hierarchical neural networks (PSHNN’s) were introduced in References [3] and [4]. The original PSHNN involves a self-organizing number of stages, similar to a multilayer network. Each stage can be a particular neural network, to be referred to as the stage neural network (SNN). For example, it can also be a deep network. Each SNN is essentially independent of the other SNN’s in the sense that each SNN does not receive its input directly from the previous SNN.

The general comparison of the PSHNN and a cascaded multistage network such as a deep network in the first type of PSHNN is shown in Figure 1. Learning with the PSHNN is similar to learning with a multilayer network except that error-detection is carried out at the output of each SNN, and the procedure is stopped without further propagation into the succeeding SNN’s if negligible errors are detected. Testing (recall) with the PSHNN can be done in parallel with all the SNN’s simultaneously rather than each SNN waiting for data from the previous SNN, as seen in Fig. 1(A).

In the original experimental studies with the PSHNN, each SNN was chosen as a single stage network learned by the delta rule. Experimental results in applications such as classification with satellite remote-sensing data [3], [4], [5] indicated that it can perform as well or better than multistage networks with backpropagation learning [6]. The PSHNN was found to be about 25 times faster in training than the backpropagation network, in addition to parallel implementation of stages during testing. This conclusion is believed to be valid no matter what technique is used for the computation of each stage. For example, PSHNN’s can also be generated with SNN’s based on competitive learning or a deep network.
3.2 Nonlinear Transformations of Input Vectors

A variety of schemes can be used to nonlinearily transform input data vectors. Two major categories of data to consider are binary data, and analog data. Some techniques initially used with both types of data are described below.

3.2.1 Binary Input Data

The first method for the desired transformation was achieved by using a fast transform followed by the bipolar thresholding (sign) function given by

\[
S'(n) = \begin{cases} 
1 & S(n) \geq 0 \\
-1 & \text{otherwise}
\end{cases}
\]  

(1)

There are a number of fast transforms such as the real discrete Fourier transform (RDFT) [7] which can be utilized.
The simplest approach is complementing the input vector if it is represented in a binary code. Another simple approach which can be used together with complementing is the scrambling of the binary components of the input vector. The binary input vectors can also be represented by a Gray code [3]. One simple possibility for input nonlinear transformation that worked well in practice is to use this scheme successively for the succeeding stages. This is done by using the Gray-coded input of the previous SNN and then determining the Gray code of the Gray code.

3.2.2 Analog Input Data
A general approach used for the transformation of analog input data was based on the wavelet packet transform (WPT) followed by the backpropagation algorithm [8]. The wavelet packet transform provides a transformation of a signal from the time domain to the frequency domain and is a generalized version of the wavelet transform [9]. The WPT is computed on several levels with different time/frequency resolutions.

3.2.3 Other Transformations
There are many other ways to conceive nonlinear transformations of input data vectors. For example, the revised backpropagation algorithm discussed in Section 9.1 can be used.

3.3 Training, Testing and Error Detection Bounds
Below we summarize the training and testing procedures with the original PSHNN algorithm. In both cases, error detection is crucial. How this is done is discussed in Section 3.3.1.

3.3.1 Training
In order to speed up learning, the upper limit of the number of iterations in each SNN during learning is restricted to an integer k. Let us assume that the ith SNN is denoted by \( SNN(i) \).

Below we describe its training procedure:

Assume that the number of iterations is upper bounded by \( k \) for each SNN.

Initialize: \( i = 1 \)

1. Train \( SNN(i) \) by a chosen learning algorithm in at most \( k \) iterations.
2. Check the output for each input vector.
   1) If no error, stop the training
   2) If errors, get the error detection bounds and go to step 3.
3. Select the input data which are detected to give output errors.
   1) If all the chosen data are in one class, then assign the final class number (FCV) as indicating that class. Stop the training.
   2) If not, go to step 4.
4. Compute the nonlinear transform (NLT) of the chosen data set. Increase \( i \) by 1. Go to step 1.

3.3.2 Testing
Testing (recall) with the PSHNN is similar to testing with a multilayer network except that error detection is carried out at the output of each SNN, and the procedure is stopped without further propagation into the succeeding SNN's if no errors are detected. The following describes the testing procedure:
Initialize: $i = 1$

1. Input the test vector to SNN $(i)$.
2. Check whether the output indicates an error-causing input data vector. If so, then,
   a) If it is the last SNN, then classify with the FCV
   b) if it is not, non-linearly transform the input test vector and go to step 1,
      else classify the output vector.

An interesting observation is that the testing with the PSHNN can be done in parallel with all the SNN's simultaneously rather than each SNN waiting for data from the previous SNN [1].

### 3.3.3 Detection of Potential Errors

The output neurons yield 1, 0 (or -1) as their final value. The decision of which binary value to choose involves thresholding. It is possible to come up with a number of decision strategies. Below a particular algorithm is described.

The value $x$ obtained after the weighted summation at the $i$-th output neuron is first passed through the sigmoid function defined by (2) to give a value $y(i)$ between 0 and 1:

$$y(i) = f(x) = \text{sigmoid}(x) = (1 + e^{-x})^{-1}$$

The value $x$ actually equals the weighted summation plus a threshold term $\theta$ which is trained by using an extra input neuron whose input is 1. The final output value $z$ is obtained by the hard limiter:

$$z(i) = \begin{cases} 
1 & \text{if } y(i) \geq 0.5 \\
0 & \text{if } y(i) \leq 0.5 
\end{cases}$$

After training the SNN by a maximum of $k$ iterations, the output vector $Z$ is compared with the desired output vector. If they are different from each other, the input vector is counted as an "error-causing" vector of the SNN.

The set of error-causing vectors are the input to the next SNN after being processed by one of the nonlinear transformation techniques discussed in Section 3.2.

Now an algorithm to detect potential errors during testing is needed. For this, error bounds and no-error bounds are defined. The following is the original algorithm for estimating the error bounds:

### Error Bounds

Assume: number of data vectors = $I$

length of input vectors = $n$

$y_j^i = j$th component of the $i$th vector $Y$.

Initialize the error bounds as

$$\begin{align*}
    y_j^0(\text{upper}) &= 0.5 \\
    y_j^0(\text{lower}) &= 0.5 \\
\end{align*}$$

where $j = 1, 2, \ldots, n$

Initialize: $i = 1$.

1. Check whether the $i$-th data vector is an error-causing vector. If so,
   (1) If $y_j^i \geq 0.5$, then
\[ y'_j(\text{upper}) = \text{MAX}\left[y'_j^{i-1}(\text{upper}), y'_j\right] \]

(2) If \( y'_j < 0.5 \), then
\[ y'_j(\text{lower}) = \text{MIN}\left[y'_j^{i-1}(\text{lower}), y'_j\right] \]

2. If \( i = 1 \), the final error bounds are
\[ r_j(\text{upper}) = y'_j(\text{upper}) \]
\[ r_j(\text{lower}) = y'_j(\text{lower}) \]
else \( i = i + 1 \) and go to step 1
end

The output classes can be denoted by binary vectors. For example, the desired output of each class can be represented by one-of-m representation. Then, an input vector is classified as an error-causing vector if the correct "1" bit at the output is 0 and vice versa.

The simplest rejection procedure during testing is to check whether any of the components \( y'_i \) of the vector \( Y \) is within the error bounds or not. If it is, the corresponding input data vector is rejected. During testing, some misclassified data may not be rejected since no \( y'_i \) is within the error bounds. Simultaneously some correctly classified data may also be rejected since some \( y'_i \)'s are within the error bounds. These sources of error can be further reduced by simultaneously utilizing no-error bounds. The following is the original procedure for estimating the no-error bounds:

**No-Error Bounds**

*Initialize the no-error bounds as*
\[
\begin{cases}
   y'_j^0(\text{upper}) = 0.5 \\
   y'_j^0(\text{lower}) = 0.5
\end{cases}
\text{ where } j = 1, 2, \ldots, n
\]

*Initialize \( i = 1 \).*
1. Check whether the \( i \)-th data vector is not an error-causing vector.
   * If so, then \( i = i + 1 \), and go to step 1,
   * else go to step 2.
2. Update the no-error bounds \( r'_j \) for \( j = 1, 2, \ldots, n \)
   as follows:
   (1) If \( y'_j \geq 0.5 \), then
   \[ y'_j(\text{upper}) = \text{MIN}\left[y'_j^{i-1}(\text{upper}), y'_j\right] \]
   (2) If \( y'_j < 0.5 \), then
   \[ y'_j(\text{lower}) = \text{MAX}\left[y'_j^{i-1}(\text{lower}), y'_j\right] \]
3. If \( i = 1 \), the final no-error bounds are
   \[ s'_j(\text{upper}) = y'_j(\text{upper}) \]
   \[ s'_j(\text{lower}) = y'_j(\text{lower}) \]
else
  \( i = i + 1 \) and go to step 1
end

With the no-error bounds, the rejection procedure can be to check whether the vector \( Y \) is not in the correct region determined by the no-error bounds. If it is not, then the corresponding input data vector is rejected.

A procedure which gave best results experimentally is to utilize both the error and no-error bounds [1]. For this purpose, three intervals \( I_1(j), I_2(j), I_e(j), j = 1, 2, \ldots, n \) are defined as

\[
I_1(j) = [r_j(\text{lower}), r_j(\text{upper})] \\
I_2(j) = [s_j(\text{lower}), s_j(\text{upper})] \\
I_e(j) = I_1(j) \cap I_2(j)
\]

(4)

Then, an input vector is classified as an error-causing vector if any \( y_j \) belongs to \( I_e(j) \). With this procedure, better accuracy is achieved because correctly classified data vectors are not rejected even if some \( y_j \)'s are within the error bounds. However, some error-causing data vectors can still be among those not rejected since no \( y_j \) belongs to \( I_e(j) \).

### 3.4 PSHNN with Competitive Learning and Safe Rejection Schemes

Multiple safe rejection schemes and competitive learning can be used as the learning algorithm of the PSHNN to get around the disadvantages of both supervised learning and competitive learning algorithms [10]. In this approach, the reference vectors are first computed in parallel for all the classes using competitive learning. Then, safe rejection boundaries are constructed in the training procedure so that there are no misclassified training vectors. The experimental results show that the proposed neural network has more speed and accuracy than the multilayer neural network trained by backpropagation and the PSHNN trained by the delta rule.

One typical competitive learning algorithm can be described as follows:

\[
W_k(t + 1) = W_k(t) + C(t)(X(t) - W_k(t)) \quad \text{if } k \text{ wins,}
\]

\[
W_k(t + 1) = W_k(t) \quad \text{if } k \text{ loses,}
\]

(5)

where \( W_k(t + 1) \) represents the value of the k-th reference vector after adjustment; \( W_k(t) \) is the value of the k-th reference vector before adjustment; \( X(t) \) is the training vector at time \( t \). \( C(t) \) is the learning rate coefficient. Usually slowly decreasing scalar time functions are used as the learning rates. At each instant of time, the winning reference vector is the one which has the minimum Euclidean distance between the reference vector and \( X(t) \).

If neural networks are trained using only competitive learning algorithms, the reference vectors are used for classification by the nearest neighbor principle, namely by the comparison of the testing vector \( X \) with the reference vector \( W \) in the nearest neighbor sense. The classification accuracy relies on how correctly the reference vectors are computed. However, it is difficult to compute the reference vectors which produce globally minimum errors because reference vectors depends on initial reference vectors, learning rate, the order of training samples, and so on.

To overcome the limitations of competitive learning algorithms, the proposed system incorporates the rejection schemes. The purpose of the rejection scheme is to reject the hard vectors, which are difficult to classify, and to accept the correctly classified vectors as much as possible. The next SNN is trained with only those training vectors that are rejected in the
previous SNN. Thus, some criteria are needed to reject error-causing vectors during both the training procedure and the testing procedure. For this purpose, rejection boundaries are constructed for the reference vectors during the training procedure, and are used during both the training procedure and the testing procedure.

3.4.1. Safe Rejection Schemes

Two possible definitions for safe rejection schemes are as follows:

Definition 1: A rejection scheme is said to be safe if every training vector is classified correctly and rejected otherwise by each SNN so that there are no misclassified training vectors if enough SNN’s are utilized.

Definition 2. A rejection scheme is said to be unsafe if there exists a misclassified training vector at the output of the total network.

Two safe rejection schemes to construct the safe rejection boundaries for the reference vectors belonging to the j-th class were developed. The procedure for the first scheme called RADPN is described below.

**RADPN (RADP and RADN):**

**Initialise.** $k = 1$, RADP$_{ni}$ = $w_{ni}$ and RADN$_{ni}$ = $w_{ni}$ for $n = 1, 2, \ldots, I$ and $i = 1, 2, \ldots, L$. The variable $w_{ni}$ represents the n-th element of a reference vector $W_i$ where $I$ is the dimension of the training vectors and $L$ is the number of reference vectors that belong to the j-th class.

**Step 1.** For a training vector $X_j(k)$ belonging to the j-th class, find the nearest reference vector $W_i$ using Euclidean distance measure.

**Step 2.** Compare $x_{nj}(k)$, the n-th element of $X_j(k)$, with $w_{ni}$.

1) If $x_{nj}(k)$ is bigger than $w_{ni}$, check whether $x_{nj}(k)$ is outside the previous rejection boundary RADP$_{ni}$.

a) If $x_{nj}(k) >$ RADP$_{ni}$, RADP$_{ni}$ is modified to $\text{RADP}_{ni} = x_{nj}(k)$.

b) If $x_{nj}(k) <$ RADP$_{ni}$, RADP$_{ni}$ is not changed.

2) If $x_{nj}(k)$ is smaller than $w_{ni}$, check whether $x_{nj}(k)$ is outside the previous rejection boundary RADN$_{ni}$.

a) If $x_{nj}(k) <$ RADN$_{ni}$, RADN$_{ni}$ is modified to $\text{RADN}_{ni} = x_{nj}(k)$.

b) If $x_{nj}(k) \geq$ RADN$_{ni}$, RADN$_{ni}$ is not changed.

**Step 3.** Check whether $X_j(k)$ is the last training vector belonging to the j-th class.

1) If $k = M_j$, where $M_j$ is the number of training vectors belonging to the j-th class, stop the procedure and save the current RADP$_{ni}$ and RADN$_{ni}$.

2) If $k < M_j$, $k = k + 1$ and go to Step 1.

The above procedure can be executed in parallel for all classes ($j = 1, 2, C$ where $C$ is the number of possible classes) or can be executed serially.

Each reference vector generates the interconnection weights between the input nodes and a particular output node identified with the reference vector. The output of an output node is set to 1 when a training vector is inside or on its rejection boundary. It has output 0 when a training vector is outside its rejection boundary. For RADPN, a training vector $X(k)$ is judged to be inside or on the rejection boundary if it satisfies the following condition for every $n = 1, 2, \ldots, I$:

$$\text{RADN}_n \leq x_n(k) \leq \text{RADP}_n$$

(6)
RADN_n and RADP_n represent the n-th elements of RADN and RADP of the reference vector identified with the output node, respectively. The variable x_n(k) is the n-th element of X(k). If at least one element of X(k) does not satisfy (9), X(k) is said to be outside the rejection boundary.

If one or more reference vectors belonging to a class has output 1, the class output is set to 1. If none of the reference vectors belonging to a class has output 1, the class output is set to 0. A training vector is rejected by the rejection scheme if more than one class have output 1. A training vector is not rejected if only one class has output 1.

### 3.4.2 Training

Assume that a training set of vectors with known classification is utilized. Each sample in the training set represents an observed case of an input-output relationship and can be interpreted as consisting of attribute values of an object with a known class. The training procedure is described as follows:

**Initialize:** m = 1.

**Step 1.** For SNN_m (the m-th stage neural network), compute the reference vectors using a competitive learning method.

**Step 2.** With the training vectors belonging to each class, construct safe rejection boundaries for reference vectors belonging to each class, as discussed in Section 3.A.

**Step 3.** Determine the input vectors rejected by all safe rejection schemes. If there is no rejected training vector or the predetermined maximum number of SNN’s are exceeded, stop the training procedure. Otherwise, go to Step 4.

**Step 4 (optional).** Transform nonlinearly the rejected data set.

**Step 5.** m = m + 1. Go to Step 1.

Assume a predetermined number of processing elements, each one provided with a reference vector W_k. Their number may be a multiple L (say, ten times) of the number of classes considered. The variable L is determined by the total number of output processing elements and the number of classes:

\[
L = \frac{\text{the total number of elements}}{\text{the number of classes}}
\]  

(7)

In Step 1 of the training procedure, we investigated two possible methods for the computation of the reference vectors. In Method I, all the reference vectors are computed together using the whole training data set. This is the way the reference vectors are computed in conventional competitive learning characterized by Eq. (5).

In Method II, competitive learning is performed in parallel for all the classes as follows: For the j-th class,

\[
W_j(t+1) = \begin{cases} 
W_j^i(t) + C^i(t)[X^i(t) - W_j^i(t)] & \text{if } i \text{ wins} \\
W_j^i(t) & \text{if } i \text{ loses}
\end{cases}
\]

(8)

where W_j^i(t+1) represents the value of the i-th reference vector of class j after adjustment-, W_j^i(t) is the value of the i-th reference vector before adjustment; X^i(t) is the training vector belonging to the j-th class used for updating the reference vectors at time t; C^i(t) is the learning rate coefficient for the computational of the reference vectors of the j-th class.
When the reference vectors are computed separately for each class and in parallel for all the classes, the learning speed is improved by a factor approximately equal to the number of classes, in comparison to conventional competitive learning. Method I is obviously more optimal when traditional competitive learning algorithms are used without rejection schemes. Interestingly, Method II gives better performance in terms of classification accuracy when rejection schemes are used [10].

3.4.3 Testing

The output of an output node is set to 1 when the testing vector is inside or on its rejection boundary. It has output 0 when the testing vector is outside its rejection boundary. For RADPN, the testing vector \( X(k) \) is judged to be inside or on the rejection boundary if it satisfies (6) for every \( n = 1, 2, \ldots, I \). Otherwise, \( X(k) \) is said to be outside the rejection boundary.

If one or more output nodes belonging to a class has output 1, the class output is set to 1. If none of the output nodes belonging to a class has output 1, the class output is set to 0. A testing vector is not rejected by the rejection scheme if only one class has output 1. A testing vector is rejected if more than one class have output 1 or no class has output 1.

Every training vector exists inside or on at least one rejection boundary. However, this is not necessarily true for the testing vectors. It is logical to class such vectors to reduce the burden of the next SNN instead of just rejecting them.

One promising way for this purpose is as follows: among the rejection boundaries of the rejection scheme by which no class has output 1, we find \( N \) nearest rejection boundaries. Then, we check whether they all belong to one class. If they do, we classify the testing vector to that class. Otherwise, the vector is rejected. Usually, \( 1 \leq N \leq L \) where \( L \) is the number of reference vectors of each class. The greater \( N \) is, the harder it is for the testing vector to be classified to a class. If all the testing vectors are required to be classified, the last SNN involves classifying the rejected testing vector to the class of the nearest reference vector.

The following describes the complete testing procedure:

*Initialize*: \( m = 1 \).

*Step 1*. Input the testing vector to SNN.

*Step 2*. Check whether the testing vector is rejected by every rejection scheme.

1) If it is rejected by all rejections schemes, find \( N \) nearest reference boundaries and perform the steps (a) and (b) below for every rejection scheme by which all class outputs are O's.
   (a) If \( N \) nearest reference boundaries belong to one class, classify the input as belonging to that class.
   (b) If \( N \) nearest reference boundaries come from more than one class, do not classify.
   (c) If a and b are done for all rejection schemes, go to Step 3.

2) If it is rejected by all rejection schemes and there is no rejection scheme by which all class outputs are 0's, go to Step 4.

3) If it is not rejected by at least one rejection scheme, classify the input as belonging to the class whose output is 1. Stop the testing procedure.

*Step 3*. Count the number of classes to which the input is classified.

1) If there is only one such class, assign the testing vector to that class. Stop the testing procedure.

2) If more than one class are chosen, do not classify the testing vector. Go to Step 4.
Step 4. Check whether or not the current SNN is the last.

1) If it is the last SNN, then classify the testing vector to the class of the nearest reference vector, stop the testing procedure.
2) If it is not, go to Step 5.

Step 5 (optional) Take the nonlinear transform of the input vector.

Step 6. \( m = m + 1 \). Go to Step 1.

Step 2 in the testing procedure can be executed in parallel or serially for all safe rejection schemes since every rejection scheme works independently.

Two or more rejection schemes can be used in parallel rather than serially. In the case of serial use of \( X \) and \( Y \), \( X \) can be used after \( Y \) or vice versa. During the training step, the ordering of \( X \) and \( Y \) is immaterial since there are no misclassified training vectors. However, during testing, the actual ordering of \( X \) and \( Y \) may affect the classification performance. In the case of parallel use of more than one rejection scheme, all the rejection schemes are used simultaneously, and each rejection scheme decides which input vectors to reject. During testing, if an input vector accepted by one or more rejection schemes is classified to different classes by more than two rejection schemes, it is rejected.

3.5 Experimental Results

Two particular sets of remote sensing data were used in the experiments. The first data set was based on Multispectral Earth Observational remote sensing data called Flight Line C1 (FLC1). The geographic location of the FLC1 is the southern part of Tippecanoe County, Indiana. The second data set called Colorado dataset is a multispectral earth observation remotely sensed data covering a mountainous area in Colorado [11]. The classification performance of the new algorithms was compared with those of backpropagation and PSHNN trained by the delta rule.

The PSHNN with competitive learning and safe rejection schemes produced higher classification accuracy than the backpropagation network and the PSHNN with the delta rule [10]. In the case of simple competitive learning without rejection schemes characterized by (8), the training and testing accuracies were considerably lower than the present method.

The learning speed of the proposed system is improved by a factor approximately equal to 57 (\(=7.15 \times 8\)), in comparison to the PSHNN with the delta rule when the reference vectors are computed in parallel for each class. Ersoy and Hong [3] estimated the learning speed of PSHNN and backpropagation networks. The 4NN requires about 25 times longer training time than the PSHNN. Thus, the training time for the PSHNN with competitive learning and safe rejection schemes is about 1425 (\(=57 \times 25\)) times shorter than the time for the backpropagation network.

4. STATISTICAL SELF-ORGANIZING LEARNING SYSTEM (SSOLS)

The second type of PSHNN is SSOLS [12-14]. It consists of two stages, a mapping stage and a learning stage, corresponding to three layers, namely, input layer, enhancement layer, and output layer. Figure 1 illustrates the structure of a SSOLS. The input nodes and the enhancement nodes are cross-connected together by a randomly generated enhancement weight matrix, while the output nodes are connected to both the input nodes and the enhancement nodes by another weight matrix. In the second stage, the weight matrix is learned by least squares. Cross-validation, especially leave-one-out
cross validation is used to determine the optimum number of enhancement nodes and to prevent overfitting.

![SSOLS architecture diagram](image)

Figure 2. Schematic of SSOLS architecture.

More specifically, the proposed structure uses a self-organizing algorithm in building the network: starting with only one enhancement node, an extra node being appended at each iteration. The leave-one-out cross-validation error is then computed and recorded in each iteration. The self-organizing algorithm stops adding enhancement nodes when the cross-validation error reaches the minimum value. Furthermore, an inclusion to the mapping stage, the t-test algorithm, mitigates the effect of overfitting and further reduces the number of enhancement nodes required, resulting in a more compact network. Higher accuracy is obtained if several SSOLSs are used as stage neural networks in a consensual neural network to reduce prediction variance and improve prediction accuracy.

Using the Sherman–Woodbury formula in linear algebra [15], a recursive update algorithm for the least squares solution is derived whenever a new enhancement node is added, without computing the solution all over again [12-14].

Two remote sensing datasets were used in the experiments. The first one was a 191-band airborne hyperspectral data flightline over the Washington, DC Mall area [2]. The second dataset was a seven-band Landsat Thematic Mapper acquired over an agricultural area in Tippecanoe County, IN [11]. It contains seven ground cover types: background, corn, soybean, wheat, alfalfa/oats, pastures, and sensor distortion. With both datasets, the SSOLS network had higher accuracy than the backpropagation, quadratic maximum likelihood, Fisher linear, correlation and matched filter classifiers.

### 5. PSHNN WITH CONTINUOUS INPUTS AND OUTPUTS

PSHNN’s discussed above assume quantized, say, binary outputs. PSHNN’s with continuous inputs and outputs were discussed in references [16]. The resulting architecture is similar to neural networks with projection pursuit learning [17], [18]. Three types of networks in which the stages are learned by the delta rule, sequential least-squares (SLS), and the backpropagation (BP) algorithm, respectively, were investigated. In all cases, the new networks achieve better performance than linear prediction.

A revised BP algorithm was also developed for learning input nonlinearities. When the BP algorithm is to be used, better performance is achieved when a single BP network is replaced by
a PSHNN of equal complexity in which each stage is a BP network of smaller complexity than the single BP network. This algorithm is further discussed below.

5.1 Learning of Input Nonlinearities by Revised Backpropagation (RBP)

In the preceding sections, it became clear that how to choose the input nonlinearities for optimal performance is an important issue. The RBP algorithm can be used for this purpose. It consists of linear input and output units and nonlinear hidden units. One hidden layer is often sufficient. The hidden layers represent the nonlinear transformation of the input vector.

The RBP algorithm consists of two training steps, denoted as step I and step II, respectively. During step I, the RBP is the same as the usual BP algorithm [6]. During step II, the weights between the input layer and the hidden layers are fixed, and the weights between the last hidden and the output layers are retrained.

Each stage of the PSHNN now consists of a RBP network, except possibly the first stage with NLT1 equal to the identity operator. In this way, the first stage can be considered as the linear part of the system.

There are a number of reasons why the two-step training described above may be preferable over the usual training with the BP algorithm. The first reason is that it is possible to use the PSHNN with RBP stages together with the SLS algorithm or the delta rule. For this purpose, we assume that the signal is reasonably stationary for N data points. Thus, the weights between the input and hidden layers of the RBP stages can be kept constant during such a time window. Only the last stage of the RBP network is then made adaptive by the SLS algorithm or the delta rule, which is much faster in learning speed than the BP algorithm requiring many sweeps over a data block. While the block of N data points is being processed with the SLS algorithm or the delta rule, the first $M << N$ data points of the block can be used to train the stages of the PSHNN by the BP algorithm. At the start of the next time window of $N$ data points, the RBP stages are renewed with the new weights between the input and hidden layer. This process is repeated periodically every $N$ data points. In this way, nonstationary signals which can be assumed to be stationary over short time intervals can be effectively processed.

The second reason is that the two-step algorithm allows faster learning. During the first step, the gain factor is chosen rather large for fast learning. During the second step, the gain factor is reduced for fine training. The end result is considerably faster learning than with the regular BP algorithm. It can be argued that the final error vector may not be as optimal as the error vector with the regular BP algorithm. We believe that this is not a problem since successive RBP stages compensate for the error. As a matter of fact, considerably larger errors, for example, due to imperfect implementation of the interconnection weights and nonlinearities can be tolerated due to error compensation [3].

6. PARALLEL CONSENSUAL NEURAL NETWORKS

The parallel consensual neural network (PCNN) was developed as another type of PSHNN and mainly applied in classification of multisource remote sensing and geographic data [8],[19]. The PCNN architecture involves statistical consensus theory [20]. The input data transformed several times as input to SNN’s are used as if they were independent inputs. The independent inputs are first classified using the stage neural networks. The output responses from the stage networks are then weighted and combined to make a consensual decision.

Two approaches used to compute the data transforms for the PCNN were the Gray code of Gray code method for binary data and the WPT technique for analog data. The experimental results
obtained with the proposed approach show that the PCNN outperforms both a conjugate-gradient backpropagation neural network and conventional statistical methods in terms of overall classification accuracy of test data [8].

In multisource classification, different types of information from several data sources are used for classification in order to improve the classification accuracy as compared to the accuracy achieved by single-source classification. Conventional statistical pattern recognition methods are not appropriate in classification of multisource data since such data cannot, in most cases, be modeled by a convenient multivariate statistical model. In [20], it was shown that neural networks performed well in classification of multisource remote sensing and geographic data. The neural network models were superior to the statistical methods in terms of overall classification accuracy of training data. However, statistical approaches based on consensus from several data sources outperformed the neural networks in terms of overall classification accuracy of test data. The PCNN gets over this disadvantage and actually performs better than the statistical approaches.

6.1 Implementation
Implementing consensus theory in PSNN involves using a collection of SNN’s. When the training of all the stages has finished, the consensus for the SNN’s is computed. The consensus is obtained by taking class-specific weighted averages of the output responses of the SNN’s. Thus, the PCNN attempts to improve its classification accuracy by weighted averaging of the SNN responses from several different input representations. Among several techniques to do the weighted averaging, the most practical solution was

\[ Y = AX \]  

(9)

where \( X \) is a matrix containing the output of all the SNN’s, and \( A \) is a weight matrix. Assuming that \( X \) has full column rank, the above equation can be solved for \( A \) by using the pseudo-inverse of \( X \) or a simple delta rule.

6.2 Experimental Results
Two experiments were conducted with the PCNN on multisource remote sensing and geographic data.

The WPT was used for input data transformations followed by the BP network with conjugate gradient training. Each level of the full WPT consists of data for the different stage networks. Therefore, the stages will have the same original input data with different time-frequency resolutions. Thus, the PCNN attempts to find the consensus for these different representations of the input data.

The experimental results obtained showed the PCNN to perform very well in the experiments in terms of overall classification accuracy [8]. In fact, the PCNN with the optimal weights outperformed both conjugate-gradient backpropagation and the best statistical methods in classification of multisource remote sensing and geographic data in terms of overall classification accuracy of test data.

7. DEEP NEURAL NETWORKS
Deep neural networks (DNN’s) have been fast expanding in the applications of remote sensing, covering a large number of topics such as image processing, spectral classification, spatial-spectral classification, target recognition, scene understanding and
high level feature extraction. In the case of spectral classification the spectral feature vector at each pixel is utilized as the input vector of the DNN. Remote sensing data often include not only spectral data but also include other types of data such as forest maps, ground cover maps, radar data, and topographic information such as elevation and slope data. These are collectively called multisource data which can be efficiently processed by DNN’s.

7.1 Convolutional Neural Networks
Among the various DNN architectures, the deep convolutional neural networks (CNN’s) and their various offsprings such as the residual networks have so far found most use in remote sensing applications [22]. Supervised classification has been the most active research area in hyperspectral image classification. CNN’s are reported to be efficient in extracting high-level, hierarchical and abstract features. The simplest CNN architecture is one-dimensional (1-D). For example, they have been used in the classification of Indian Pines, Salinas and University of Pavia hyperspectral images [1]. Higher accuracies were reported as compared to other machine learning methods such as support vector machines (SVM’s). Figure 3 shows a possible schematic for the CNN architecture used in such studies [Hu]. In the max pooling layer, the maximum element in a small window of size $k_1$ is chosen. Some other variants of the maximum function is averaging, subsampling and computing the mean.

Figure 3. CNN schematic with a convolutional layer, a max pooling layer, and a fully connected layer [22].
Figure 4 shows the actual architecture used [22]. Each pixel in the HIS image is of length $n_1$. The value of $k_1$ is chosen as $n_1/9$. There are 20 convolution kernels used. Hence the size in the C1 layer is $20 \times n_2$ where $n_2 = n_1 - k_1 + 1$ if no zero padding is used. In the max pooling layer, a kernel size of $k_2$ is used, reducing the hidden layer size to $n_3 = n_2 / k_2$. This is followed by two fully connected layers. The numbers of nodes in these layers are chosen as $n_4$ equal to 100 and $n_5$ equal to the number of classes, respectively.

All the weights are chosen randomly with values in the range [-0.5, 0.5]. The backpropagation algorithm with stochastic gradient descent is used for training. The activation function in the hidden layers was chosen as the hyperbolic tangent function. Currently, the RELU (rectified linear unit) function has also been popular. The output layer activation function was chosen as the softmax function given by

$$f(u_i) = e^{u_i} / \sum_{j=1}^{K} e^{u_j}$$

where $K$ is the number of nodes (usually classes), and $u_i$ is the output of the $i$th node.

The performance of such a network can be further improved by using a deeper network, joint use of spatial and spectral information and residual learning discussed below [23]. The residual network allows building deeper and wider networks, and is also more effective with a relatively small number training samples, which is the current scenario with available hyperspectral data [24].
A more general approach to include both spectral and spatial information is to use a 4-dimensional convolution kernel of size $N \times N \times M \times L$ where $N \times N$ is the spatial kernel size, $M$ is the number of spectral bands, and $L$ is the number of kernels in each convolutional layer. For example, with an input size of $5 \times 5 \times 200$, a convolutional kernel size might be $2 \times 2 \times 200 \times 50$. Then, the next layer input would be of size $4 \times 4 \times 50$ [23].

### 7.2 Residual Learning

In residual learning, instead of learning the mapping $x \rightarrow y$ with a function $H(x)$, the residual function $F(x) = H(x) - x$ is learned. This is afterwards corrected by adding $x$ to $F(x)$. This is based on the observation that learning $F(x)$ is easier than learning $H(x)$. Residual learning is usually incorporated into a CNN network as shown in Figure 5 [He].

![Figure 5. A building block for residual learning](image)

The identity connection shown in Figure 5 is usually called identity shortcut connection. Such connections do not add extra computational complexity. The entire network can be learned, for example, by gradient descent as before. Further details on residual learning are given in Sections 8.1 and 8.2.
7.2 Autoencoders

As the name implies, an autoencoder (AE) maps the input $X$ back to itself. In the simplest case, it has one hidden layer of $h$ units with an activation function $f$. Since the desired output is also $X$, the output is called reconstruction. Figure 6 shows the schematic of the simple autoencoder.

![Figure 6. A simple autoencoder](image)

The equations for the autoencoder can be written as

\[ y = f(W_1 x + b_1) \]
\[ z = f(W_2 y + b_2) \]

The autoencoder can be learned in a number of ways, for example, stochastic gradient descent as in the backpropagation algorithm.

Stacked autoencoders are deep autoencoders with a number of hidden layers. One application of autoencoders is feature extraction for HSI classification in which the last hidden layer output represent the features to be classified by another method such as logistic regression or SVM [25].

In order to achieve spatial-spectral classification, the autoencoder can be used together with some other methods. In one approach called AE-SVM, the autoencoder was created by choosing $W_1 = W_2 = W$, and the hidden layer output was fed in to another classifier SVM classifier trained independently [25-26]. This was further generalized by using a 4-layer stacked autoencoder which was first trained. This was followed by retraining after attaching a logistic regression stage to the last stage of the stacked autoencoder.
In order to use both spatial and spectral information, principal component analysis (PCA) was first used to extract several, say, 3 features in the spectral direction. In the spatial direction, a window of size, say, 7x7, was chosen around the current pixel. This yields a volume element of size 7x7x3. This was flattened to a vector of length 63, which was fed in to a stacked autoencoder + classifier system.

The experiments were done with the Kennedy Space Center (KSC) data. The AE-SVM method was compared to pure SVM as well PCA followed by SVM (PCA-SVM) methods. In all cases, the AE-SVM method gave more accurate classification results. In turn, the deep AE-SVM method was better than purely AE-SVM method, and the improvement increased as the number of layers of the autoencoder increased.

8. WIDE AND DEEP NEURAL NETWORKS
In addition to the PSHNN related networks discussed in sections 3 thru 6, there have recently been several new directions in wide and deep neural network research. They are reviewed below.

8.1 Contextual CNN
These networks have been develop for classification of current hyperspectral images which are not very large [27]. Here deeper and wider mean using relatively large number of layers in the depth direction coupled with wide layer networks with large number of nodes. This is coupled with residual learning allowing increase in both depth and width of the overall network. Both spatial and spectral are included. The result is a fully convolutional network capable of processing hyperspectral images without any pooling layers. The network has been tested with the Indian Pines, University of Pavia and Salinas datasets. RELU nonlinear units and dropout techniques were utilized in inner layers. Softmax nonlinearity was used at the output layer. In the implementation, 9 layers of CNNs were utilized. For each training pixel, a 5x5 pixel neighborhood was used for convolutions. The learning techniques was stochastic gradient descent with a batch size equal to 10, and 100K iterations.
The proposed network was better than the previous networks by 2.58% for the Indian Pines dataset, 2.47% for the Salinas dataset, and 2.86% for the University of Pavia dataset. Residual learning helped improve the performance by optimizing training efficiency on a relatively small number of samples.

8.2 Wide Residual Networks
This is similar to Section 8.1 with deeper and wider networks together in a residual architecture [28]. Wider deep neural networks gave significantly improved results with 50 times less number of layers, and being more than 2 times faster than previous residual networks. These studies mostly used CIFAR datasets [29].

8.3 Wide and Deep Neural Networks in Parallel
In recent research by Google, wide and deep neural networks have been developed as two parallel branches, the wide network specializing in memorization while deep network specializing in generalization [30]. This is in analogy to biological intelligence in which
memorization and generalization occur together. During training, the prediction errors are backpropagated in both networks to learn the model parameters. The experiments showed that the wide model is able to memorize interactions with data with a large number of features, but not able to generalize these learned interactions on new data. The deep model generalizes well but is unable to learn exceptions within the data. Thus, the wide network is used to memorize all the specific rules while the deep network generalizes to similar items via embeddings. The Python program package is made available in TensorFlow implementation. The experimental studies show that the wide and deep model performs very well with large datasets in which each feature has many unique possible values.

9. CONCLUSIONS
Wide and deep neural networks have potential to achieve optimal architectures in various remote sensing classification applications, especially with hyperspectral remote sensing images. PSHNN’s provide a general framework to build wide neural networks with a number of stage neural networks (SNN’s). Each SNN can be a deep neural network among others. It is possible to reverse this process, for example, by building a deep neural network with layers in which each layer is a PSHNN. Various types of PSHNN’s have been investigated in classification of remote sensing images. They usually have higher accuracy of classification than other methods such as backpropagation neural networks and statistical classifiers.

Deep neural networks have been an area of intense investigation in recent years mainly because of their superior generalization performance and thereby high classification accuracies during testing. However, in order to build very deep networks, it is necessary to have very large datasets. This is often not practical. For example, hyperspectral remote sensing image datasets are large, but not very large. There are also other problems such as vanishing gradients. There are currently various research efforts to reduce these problems. For example, residual networks help reduce such problems.

Deep convolutional neural networks (CNN’s) and deep autoencoders (AEs) are the ones most often used with hyperspectral remote sensing image classification. They have achieved very high testing accuracy rates with hyperspectral remote sensing images. Residual network principles help achieve deep networks more effectively. In contextual CNN’s and wide residual networks, these principles have been effectively utilized to achieve very high accuracy in classification of images.

Partially due to moderate sizes of current hyperspectral image sets, design and implementation of wide and deep neural networks hold the potential to yield most effective solutions. For example, the number of layers of deep neural networks can be reduced by including growth in the wide direction. This is also expected to simplify the design of each layer of a deep neural network since errors are compensated in the wide direction. Parallel implementation of stages as in PSHNN's allow faster speed of processing during testing. As a whole, the overall designed network is expected to be more compact than a single deep neural network. PSHNN type of architecture also means new types of feature maps corresponding to subsets of input vectors corresponding to easy to classify and hard to classify vectors.

REFERENCES