remote sensing

MDPI

Article

Convolutional Recurrent Neural Networks for Hyperspectral Data Classification

Hao Wu and Saurabh Prasad *

Hyperspectral Image Analysis Lab, Department of Electrical and Computer Engineering, University of Houston; 4800 Calhoun Rd., Houston, TX 77004, USA; hwu@uh.edu

* Correspondence: saurabh.prasad@ieee.org; Tel.: +1-713-743-4743

Academic Editors: Lênio Soares Galvão and Prasad S. Thenkabail

Received: 9 January 2017; Accepted: 14 March 2017; Published: 21 March 2017

Abstract: Deep neural networks, such as convolutional neural networks (CNN) and stacked autoencoders, have recently been successfully used to extract deep features for hyperspectral data classification. Recurrent neural networks (RNN) are another type of neural networks, which are widely used for sequence analysis because they are constructed to extract contextual information from sequences by modeling the dependencies between different time steps. In this paper, we study the ability of RNN for hyperspectral data classification by extracting the contextual information from the data. Specifically, hyperspectral data are treated as spectral sequences, and an RNN is used to model the dependencies between different spectral bands. In addition, we propose to use a convolutional recurrent neural network (CRNN) to learn more discriminative features for hyperspectral data classification. In CRNN, a few convolutional layers are first learned to extract middle-level and locally-invariant features from the input data, and the following recurrent layers are then employed to further extract spectrally-contextual information from the features generated by the convolutional layers. Experimental results on real hyperspectral datasets show that our method provides better classification performance compared to traditional methods and other state-of-the-art deep learning methods for hyperspectral data classification.

Keywords: deep learning; convolutional neural network; recurrent neural network; convolutional recurrent neural network; hyperspectral image

1. Introduction

In the last decade, with the advances of the computing power of computers and the availability of large-scale datasets, deep learning [1] techniques, such as deep belief networks (DBN), deep convolutional neural networks (CNN) and deep recurrent neural networks (RNN), have gained great success in a variety of machine learning tasks, such as computer vision, speech recognition, natural language processing, etc. For example, deep CNNs have been widely used for various computer vision tasks, such as large-scale detection and classification of object categories [2–6], deep CNN-based transfer learning [7,8] and speech recognition [9]. RNNs, another important branch of the deep neural networks family, were mainly designed for sequence modeling. The long short-term memory (LSTM) [10,11] network is a special type of RNN, which is able to capture very long-term dependencies embedded in sequence data. Both the regular RNN and the LSTM networks have been successfully used for time series data analysis, such as speech recognition [12–15], machine translation [16–18], etc.

CNN was first proposed in [19] for digit recognition where the network had two convolutional layers coupled with pooling layers. With a larger training dataset and more powerful computing hardware (GPUs), deep CNNs became popular since "AlexNet" [2], which had five convolutional layers, was used for ImageNet classification. Thus, deep CNN is usually used when the network has

more than three convolutional layers coupled with pooling layers, and each layer has a large number of nodes. Similarly, a deep RNN usually refers to a network with multiple recurrent layers.

Hyperspectral data, which capture spectral information of objects over a wide range of the electromagnetic spectrum, have played a key role in remote sensing data analysis [20]. With rich spectral information, hyperspectral image data have been used for target detection [21,22], land cover classification [23–25], space surveillance [26] and environmental science [27]. Modern advanced hyperspectral sensors are able to collect hyperspectral images of very high resolution, from which we can collect a large number of labeled pixels for training deep neural networks. Recently, deep learning techniques have been introduced into the remote sensing community especially for hyperspectral data classification and achieved state-of-the-art performance; for instance, a stacked autoencoder to extract deep features from hyperspectral data in [28]. One-dimensional CNNs were used to extract spectral features for hyperspectral data in [29]. Two-dimensional CNNs had been employed to extract features for hyperspectral data in [30,31]. Features extracted by a CNN were used for a sparse representation-based classifier in [32]. CNNs can also be used for scene classification of high-resolution remote sensing RGB images [33]. In [34], an RNN was used for land cover change detection on multi-spectral images.

However, the existing work, such as [28,31], that employed deep learning techniques for hyperspectral data classification, all treat the hyperspectral data samples as high dimensional input vectors to the network and try to extract deep features from them. In this paper, we try to analyze hyperspectral data from a sequential point of view, i.e., each hyperspectral sample (pixel) is seen as a data sequence, which can be modeled by recurrent neural networks. As the sequence model, RNN assumes that the current output of a sequence depends not only on the current input, but also the previous outputs. In other words, RNNs are suited for sequences where there are dependencies between different time steps. Since hyperspectral data are densely sampled from the entire spectrum, they are expected to have dependencies between different spectral bands. First, it is easy to observe that for any material, the adjacent spectral bands tend to have very similar values, which implies that adjacent spectral bands are highly dependent on each other. In addition, some materials also demonstrate long-term dependency between non-adjacent spectral bands.

In this paper, we propose that RNNs are a natural choice for hyperspectral data classification. Although CNNs are also able to capture some dependencies in the input sequence data, they only model local dependencies, since the convolutional filters often have a short length. Thus, recurrent neural networks are better for modeling sequence dependency. In addition, we also propose to use a convolutional recurrent neural network (CRNN) [35,36], which is composed of a few convolutional layers followed by a few recurrent layers. The motivation to use convolutional layers (and pooling layers) before recurrent layers is that convolutional layers can first extract middle-level, locally invariant features from the input sequence. Pooling layers make the sequence shorter by subsampling, which will accelerate the backpropagation through the recurrent layer, since the computational complexity of recurrent layers grows linearly with respect to the length of the input sequence. Thus, recurrent layers can then extract the spectral dependencies more efficiently from the middle-level features provided by convolutional layers. Another benefit of RNN (and hence, CRNN) is that, unlike CNN, it does not require the input sequence to have a fixed length. Although each pixel in a hyperspectral image has the same length of spectrum in our problem, the method can still be extended to handle problems where input sequences may have variable lengths.

There are mainly three contribution of this work: First, RNNs are used to extract contextual information for hyperspectral data. Second, we propose to use CRNN for hyperspectral image classification, which combines the advantages of both CNNs and RNNs. Finally, a spatial constraint based on decision fusion was applied to the output of the neural networks to further improve the classification performance.

The remainder of this paper is organized as follows: Section 2 talks about different deep neural networks for hyperspectral data classification: we provide a basic overview for CNNs and how

they have been used for hyperspectral data classification, introduce RNNs for hyperspectral data classification and describe how to combine CNN with RNN to get the proposed method CRNN for hyperspectral data classification. A spatial constraint based on decision fusion is also described in this section for spectral-spatial classification. The details of the experiments conducted for validating the proposed methods are given in Section 3, which includes descriptions of the datasets, the experimental setup and results. Discussions on the results are presented in Section 4. Section 5 summarizes the key ideas in this work and provides concluding remarks.

2. Materials and Methods

In this section, we first review the current most related works for hyperspectral image classification based on deep CNNs in Section 2.1. Then, we introduce the proposed methods for hyperspectral data classification, which include RNNs in Section 2.2 and CRNNs/CLSTMs in Section 2.3. Finally, Section 2.4 describes a spatial constraint based on decision fusion for spectral-spatial hyperspectral image classification.

2.1. CNN

As in [29], 1D CNNs have been successfully used for hyperspectral image pixel-level classification. Moreover, [30,31] made use of 2D CNNs for classification by taking a neighborhood window of size $w \times w$ (where w usually takes values about 20) around each labeled pixel and treat the whole window as a training sample, which is basically image-level classification instead of pixel-level classification. Additionally, this 2D CNN framework will need a large number of labeled pixels to generate enough 2D neighborhood regions to train a deep CNN. However, we usually do not have enough labeled pixels on a remotely-sensed hyperspectral image to generate, so many neighborhood regions for training a deep network. Another concern is that having a fixed size of neighborhood cannot guarantee that each neighborhood region contains only one class of object. Since different classes tend to have different spatial sizes, the optimal neighborhood size should also be class specific. Thus, 1D CNNs are more practical for remotely-sensed hyperspectral image classification, and we will only focus on them in the following discussion.

A graphical illustration of the 1D CNN we used in this work is shown in Figure 1 where a hyperspectral vector is fed to the input layer and then propagated through several successive convolutional and pooling layers for feature extraction. Each convolutional layer has multiple 1D convolutional filters (kernels). The size of the kernel is a hyperparameter and data dependent. The pooling layers are used for subsampling to reduce the dimensionality of the network, which can help reduce computation and control overfitting.

Let us denote an input hyperspectral vector as $\boldsymbol{x} = (x_1, x_2, ..., x_T) \in \mathbb{R}^T$ where T is the length of the input vector. In the first convolutional layer, a set of d filters $\{\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, ..., \boldsymbol{\phi}_d\}$ of receptive field size (i.e., length of convolutional filters), r, are applied to the input vector via convolution operation ($*$) to get the feature map:

$$\boldsymbol{F} = (\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_T) = g(\boldsymbol{x} * \{\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, ..., \boldsymbol{\phi}_d\}) \quad (1)$$

where $\boldsymbol{f}_t \in \mathbb{R}^d$ and g is a nonlinear activation function, such as tanh or a rectified linear unit (ReLU). ReLU is defined as $g(x) = max(0, x)$, which has become the most used activation function in CNNs, and we also choose to use it in this work.

The max pooling layer subsamples every depth slice of the input by the *max* operation. The most common form is a pooling layer with filters of a length of two applied with a stride of two. After applying the pooling layer, the depth (dimension) remains unchanged, but the length is reduced by half.

Finally, the extracted high-level features will be flattened to be a fixed-dimensional vector, which is then fully connected to the output layer for classification, where we have a softmax activation function to compute the predictive probabilities for all of the categories. This is done by:

$$p(y = k|\boldsymbol{x}) = \frac{exp(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)}{\sum_{k'=1}^{K} exp(\boldsymbol{w}_{k'}^\top \boldsymbol{x} + b_{k'})} \tag{2}$$

where $\boldsymbol{w}_k$'s and b_k's are the weight and bias vectors, and there are K categories.

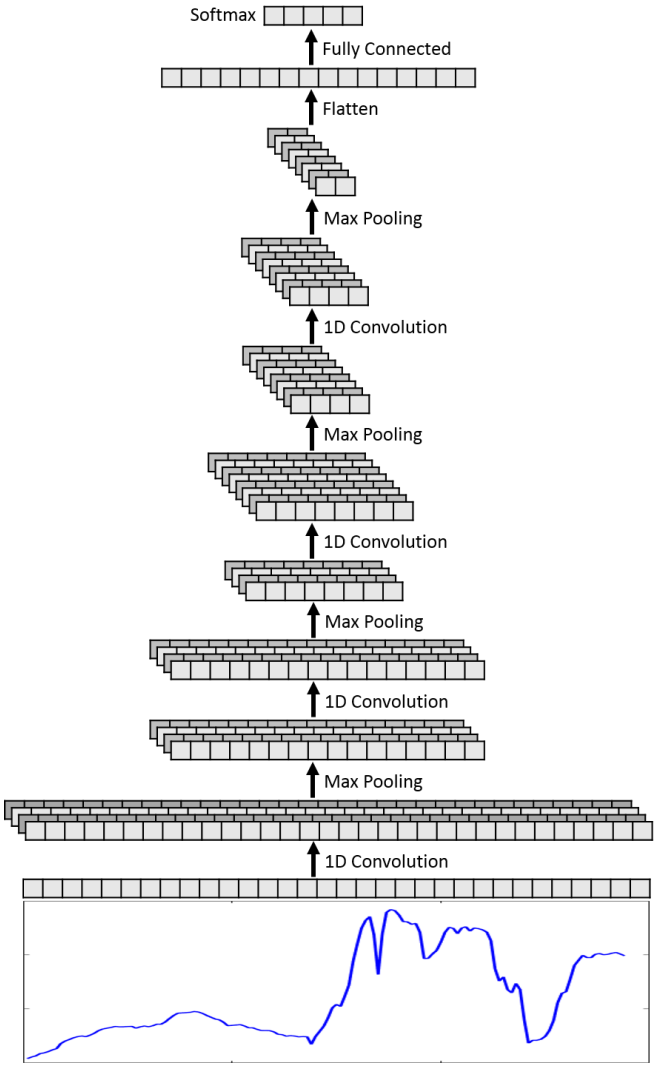

Figure 1. Architecture of the CNN for hyperspectral data classification.

Training a neural network is to find the best parameters (weights of the network) to minimize the loss function, which in a classification task measures the compatibility between a prediction (e.g., the class scores in classification) and the ground truth label. The loss takes the form of an average over the losses for every training example: $L = \frac{1}{N}\sum_{i=1}^{N} L_i$, where N is the number of samples and L_i is the loss for sample i. For the output layer with softmax activation, the cross-entropy loss (also known as negative log likelihood) is most widely used:

$$L_i = -log(p(y_i|\boldsymbol{x}_i)) \tag{3}$$

The network is trained with stochastic gradient descent (SGD), and gradients are calculated by the back-propagation algorithm. A mini-batch strategy is utilized in our implementation to reduce loss fluctuation, so the gradients are calculated with respect to mini-batches. The algorithm will be running iteratively until the loss converges, when the change of training and validation loss is below some threshold.

2.2. RNN

Recurrent neural networks, which consists of successive recurrent layers, are sequential models, which map a sequence to another sequence. RNNs have a strong capability of capturing contextual information within a sequence. The contextual cues are stable and useful for classifying hyperspectral data. What is more, RNN is able to operate on sequences of arbitrary lengths, although this advantage is not utilized in this work. However, it is worth noting that our work can be extended to handle the problem where the input sequences have variable lengths.

The structure of a basic RNN with one recurrent layer is illustrated in Figure 2 where we have a sequence of vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T\}$ as input; $\{\boldsymbol{h}_1, \boldsymbol{h}_2, ..., \boldsymbol{h}_T\}$ is a sequence of hidden sates, and $\{\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_T\}$ is a sequence of outputs.

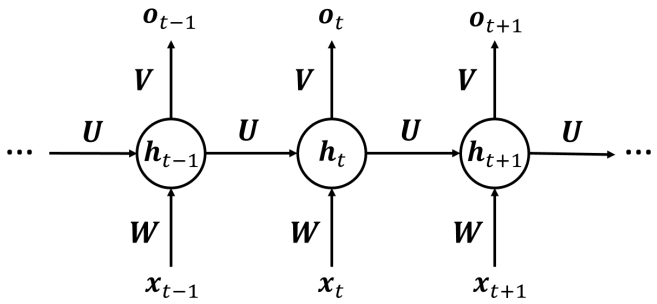

Figure 2. Structure of a basic recurrent layer.

A recurrent layer has a recursive function f, which takes as input one input vector $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$ and returns the new hidden state as:

$$\boldsymbol{h}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) = tanh(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1}) \tag{4}$$

and the outputs are calculates as:

$$\boldsymbol{o}_t = softmax(\boldsymbol{V}\boldsymbol{h}_t) \tag{5}$$

where $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are the weight matrices, which are shared across all steps, and activation function *tanh* is the hyperbolic tangent function. This recursive function however is known to suffer from the problem of the vanishing gradient [37] for long input sequences, such as the speech signal or text document, which makes it difficult to learn for long-term dependencies. To overcome this problem, the long short-term memory (LSTM) RNN [10,11] was introduced, which uses a more complicated function that learns to control the flow of information, allowing the recurrent layer to capture long-term dependencies more easily. The structure of a basic LSTM unit is illustrated in Figure 3.

The LSTM unit consists of four sub-units: input gate, output gate, forget gate and new memory, which are computed by:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{U}^{(i)}\boldsymbol{h}_{t-1}) \tag{6}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{U}^{(o)}\boldsymbol{h}_{t-1}) \tag{7}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{U}^{(f)}\boldsymbol{h}_{t-1}) \tag{8}$$

$$\tilde{\boldsymbol{c}}_t = tanh(\boldsymbol{W}^{(c)}\boldsymbol{x}_t + \boldsymbol{U}^{(c)}\boldsymbol{h}_{t-1}) \tag{9}$$

where activation functions σ and $tanh$ are logistic sigmoid and hyperbolic tangent functions, respectively. Based on these, the LSTM unit then computes the memory cell and output as:

$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \tag{10}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot tanh(\boldsymbol{c}_t) \tag{11}$$

where the point-wise multiplication of two vectors is denoted with $\odot$.

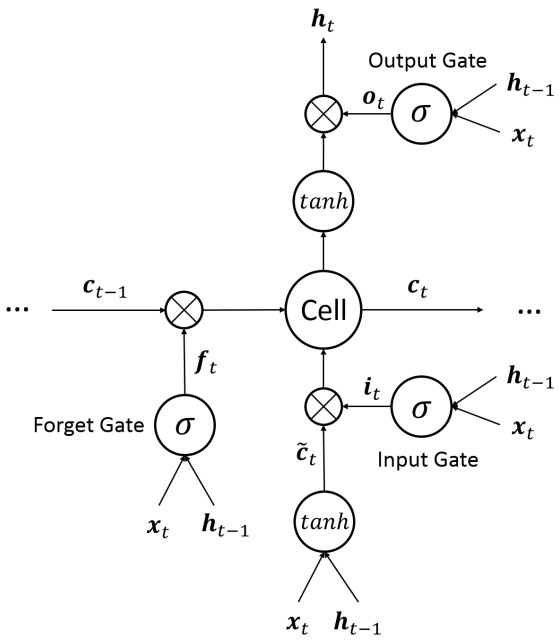

Figure 3. Structure of a basic recurrent layer.

A graphical illustration of the RNN framework (regular RNN or LSTM RNN) we used in this work is shown in Figure 4 where the input hyperspectral data $\boldsymbol{x} = (x_1, x_2, ..., x_T)$ is viewed as a sequence of 1D vectors, which is propagated through several recurrent layers to extract deep features. The hidden states of the last recurrent layer are a sequence of high-level features. As indicated in Figure 4, we only take the last hidden state of the last recurrent layer as the input to the classification layer, since the last hidden state should already contain all of the useful contextual information in previous time steps. Like in CNN, the softmax activation function is applied at the output layer, and the loss function is formulated as the cross-entropy. Training is done by mini-batch gradient descent. However, gradients of the loss function are calculated by the back-propagation through time (BPTT) algorithm [38].

2.3. CRNN

The CRNN is a hybrid of convolutional and recurrent neural networks [35,36]. It is composed of several convolutional (and pooling) layers followed by a few recurrent layers, as indicated in the graphical illustration of CRNN used in this work in Figure 5. CRNN has the advantages of both convolutional and recurrent networks. First, the convolutional layers are able to efficiently extract middle-level, abstract and locally invariant features from the input sequence. The pooling layers help reduce computation and control overfitting. Second, the recurrent layers extract contextual information from the feature sequence generated by the previous convolutional layers. Contextual information captures the dependencies between different bands in the hyperspectral sequence, which is more stable and useful for classification. Third, recurrent layers can handle variable-length input sequences, though we are not making using of this benefit in this work.

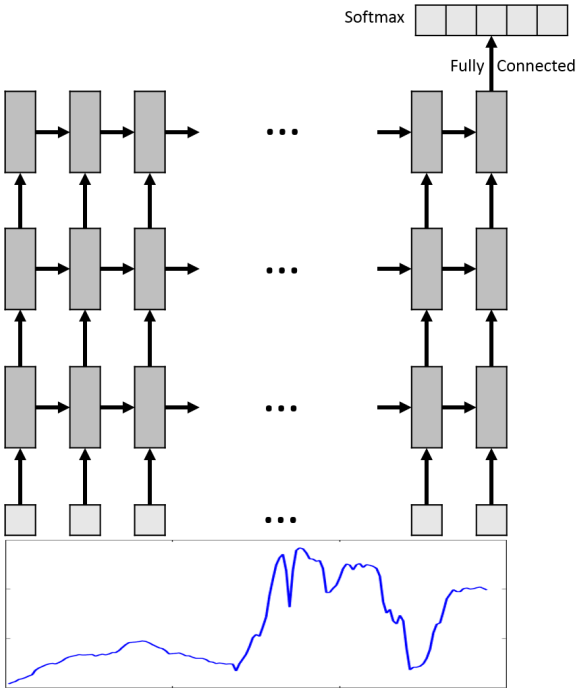

Figure 4. Architecture of the recurrent neural network (RNN) for hyperspectral data classification.

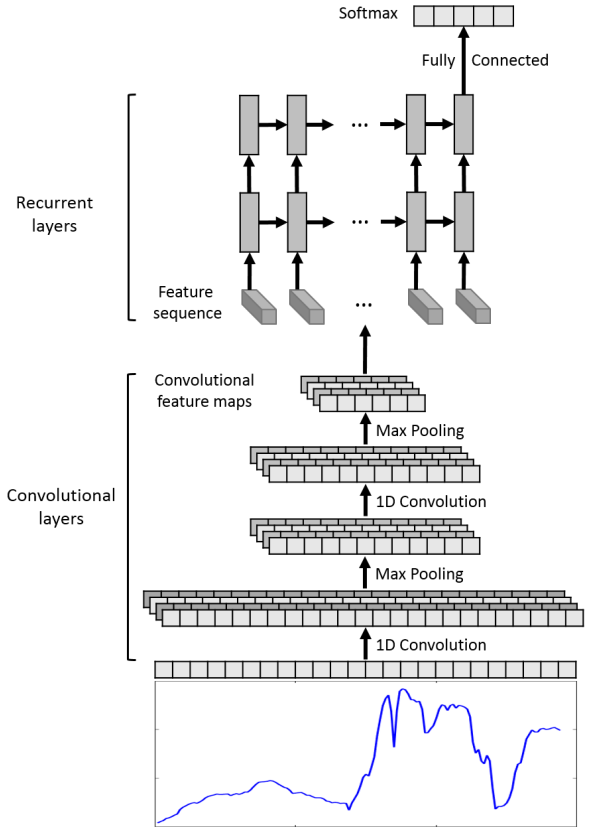

Figure 5. Architecture of the convolutional recurrent neural network (CRNN) for hyperspectral data classification.

For the recurrent layers, we can either use the regular recurrent function or more complicated ones, like LSTM, which can capture very long dependencies more efficiently. However, since the lengths of input hyperspectral sequences are not very long (usually below 200) and the pooling layers additionally reduce the length greatly (usually below 50), we will not have long-term dependency and vanishing gradient problems in most cases. Thus, regular RNN should work as well as LSTM networks, which has been verified in our experiments.

Finally, as in RNN, the last hidden state of the last recurrent layer will be fully connected to the classification layer where a softmax activation function is applied. For training, as in CNN and RNN, the loss function is chosen as cross-entropy, and mini-batch gradient descent is used to find the best parameters of the network. The gradients in the convolutional layers are calculated by the back-propagation algorithm, and gradients in the recurrent layers are calculated by the back-propagation through time (BPTT) algorithm [38].

2.4. Spatial Constraint by Decision Fusion

The neural networks described in Sections 2.1–2.3 all extract deep spectral features for each hyperspectral pixel independently. In order to further improve the classification performance, we integrate the spatial constraint by linear opinion pools (LOP) [39,40] based on the fact that, in pixel-based image classification, spatially neighboring pixels tend to have similar categories. The spatial constraint encourages piecewise smooth segmentation of images by smoothing out noisy predictions due to noisy data or outliers.

Based on the predictive probabilities computed by the output layer (softmax) of the neural networks, LOP-based decision fusion calculates the posterior probability for each pixel $\boldsymbol{x}_i$ as a weighted sum of the posterior probabilities of the spatial neighbors of that pixel:

$$p(y_i|\boldsymbol{x}_i) = \sum_{j \in \mathcal{N}_i} \omega_j p(y_j|\boldsymbol{x}_j) \tag{12}$$

where $\mathcal{N}_i$ are the spatial neighbors of pixel $\boldsymbol{x}_i$ and ω_j's are weights for each neighbor that satisfy $\sum_{j \in \mathcal{N}_i} \omega_j = 1$. For simplicity, we use uniform weights for neighbors in our implementation.

A flowchart illustrating the proposed classification system is shown in Figure 6.

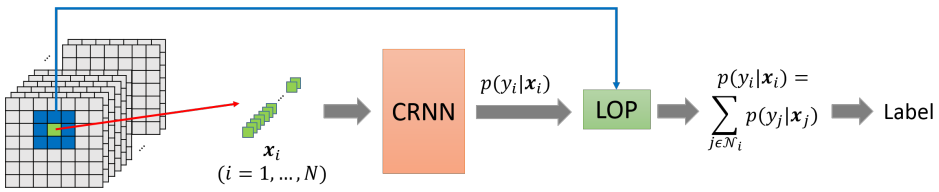

Figure 6. Flowchart for the proposed method.

3. Experimental Setup and Results

This section is devoted to illustrate the capabilities of the presented deep neural networks (CNN, RNN and CRNN) for two hyperspectral image datasets in remote sensing. A traditional method based on support vector machines (SVM) with the radial basis function (RBF) kernel is also implemented for comparison.

3.1. Datasets

Two modern high resolution hyperspectral images are used in our study. One covers an urban area over the University of Houston at Houston, Texas, and the other covers a mixed vegetation site at the Indian Pines area in Indiana.

The first dataset used in this work, the University of Houston (UH) hyperspectral image, was acquired by the NSF-funded National Center for Airborne Laser Mapping (NCALM) over

the University of Houston campus and the neighboring urban area using the ITRES-CASI (Compact Airborne Spectrographic Imager) 1500 hyperspectral imager in 2012. The hyperspectral image contains 15 labeled land cover classes and consists of 144 spectral bands over the 364 nm–1046 nm wavelength range. It has a spatial dimension of 1905 × 349 with a spatial resolution of 2.5 m. Figure 7 shows the true color image of the UH dataset with the ground truth for all 15 classes, and Figure 8 shows the corresponding mean spectral signatures (radiance) for each class. The spectral signatures tell that different classes have different shapes and local structures. Thus, when we treat the hyperspectral pixels as sequences, RNNs can be used to extract discriminative contextual information from them, which is very useful for classification. This dataset is available online (http://hyperspectral.ee.uh.edu).

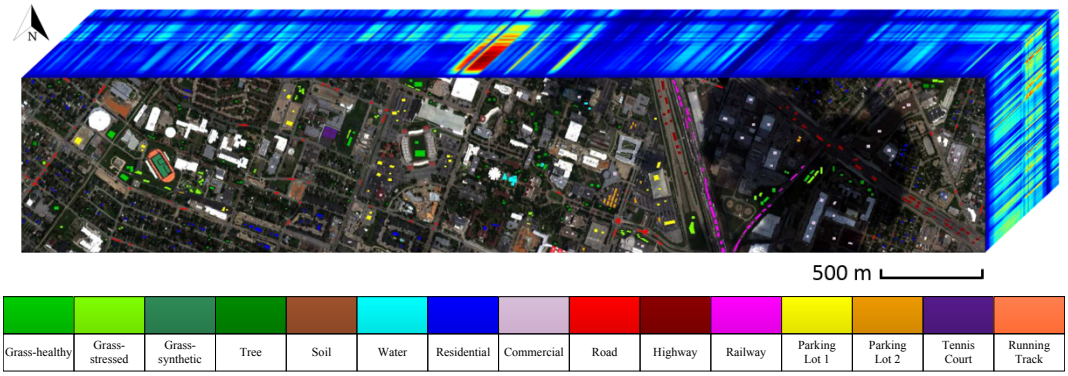

Figure 7. The color image of the University of Houston (UH) dataset with the ground truth for 15 classes in different colors.

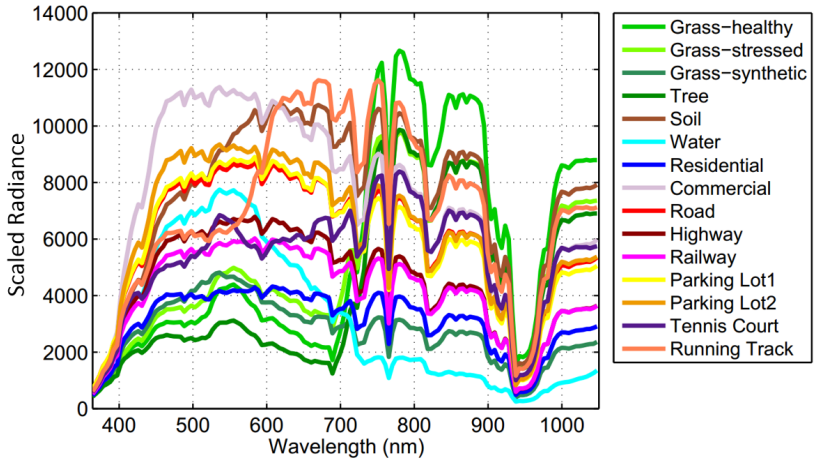

Figure 8. Mean spectral signatures of 15 classes from the UH dataset where different colors correspond to different classes in Figure 7.

The second dataset, the Indian Pines hyperspectral image, was acquired using the ProSpecTIRinstrument in May 2010 over an agriculture area in Indiana, USA. The image has a spatial dimension of 1342 × 1287 with a spatial resolution of 2 m. It consists of 180 spectral bands over the 400 nm–2500 nm wavelength range. There are 19 labeled classes contained in this dataset, and 16 of them belong to different vegetation types. Figure 9 shows the true color image with the corresponding ground truth for this dataset, and Figure 10 presents the mean spectral signatures (reflectance) of all 19 classes.

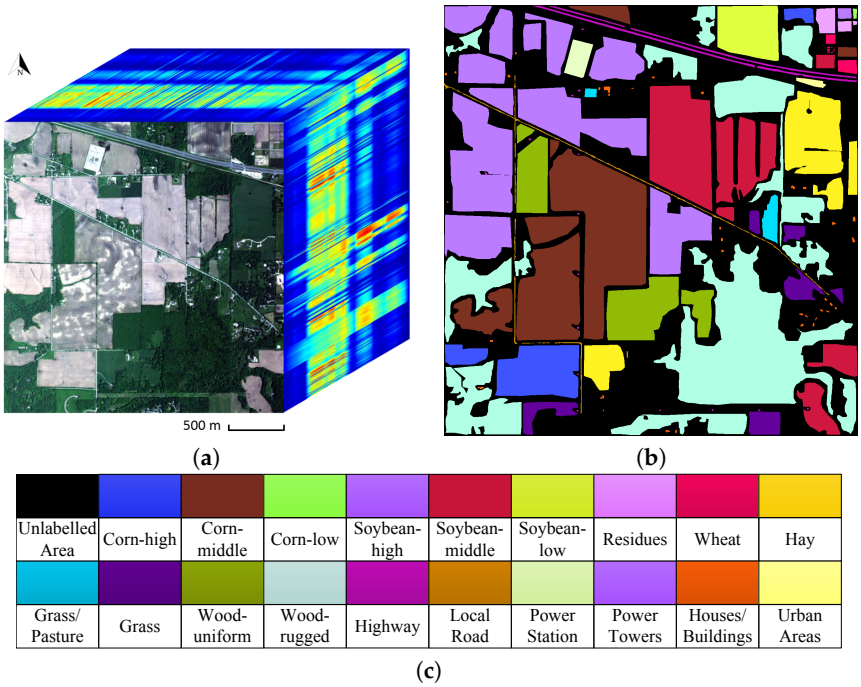

Figure 9. The Indian Pines dataset: (**a**) true color image and (**b**) ground-truth, with class names in (**c**).

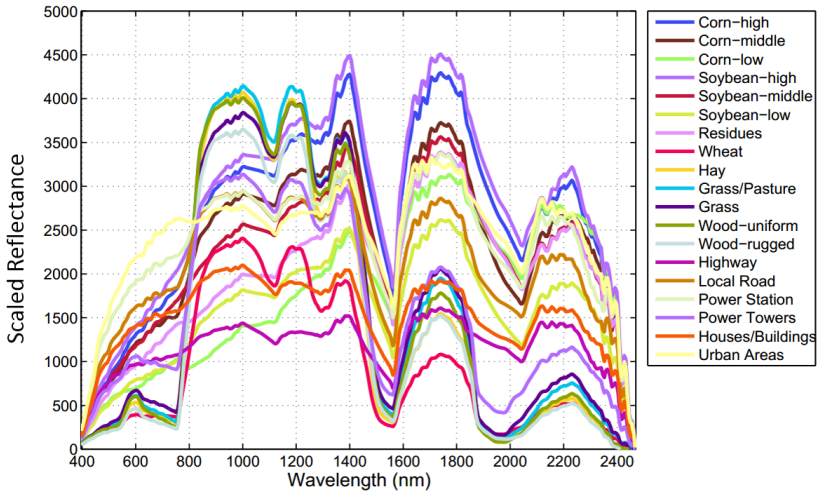

Figure 10. Mean spectral signatures of the 19 classes in Indian Pines dataset.

3.2. Experimental Setup

For both datasets, we randomly split the labeled samples into training and test sets. During training, 10% of the training samples are used as a validation set to learn the hyperparameters of the neural networks (i.e., layer size, number of layers, learning rate and mini-batch size) using a grid search strategy.

In our experiments, the CNN has four convolutional (with pooling layers); RNN and LSTM have three recurrent layers; CRNN and CLSTM have two convolutional layers (with pooling layers) and two recurrent layers. We also implemented the 2D CNN in the same way as [30,31] to extract spatial features from hyperspectral images. In particular, PCA was first employed on the whole image to reduce the dimensionality down to three, and then, we take a neighborhood region of size 11×11 around each labeled pixel to form 2D images, which will be fed to the input of the 2D CNN, which has three convolutional layers.

The configurations of all networks used in our experiments are summarized in Tables 1 and 2, where convolutional layers are denoted as "conv⟨receptive field size⟩-⟨number of filters⟩" and recurrent layers are denoted as "recur-⟨feature dimension⟩".

Table 1. Summary of network configurations for the University of Houston dataset. conv, convolutional; LSTM, long short-term memory.

CNN	2D CNN	RNN/LSTM	CRNN/CLSTM
Input-144			
conv6-32	conv3 × 3-32	recur-128	conv6-32
maxpool	conv3 × 3-32	recur-256	maxpool
conv6-32	maxpool	recur-512	conv6-32
maxpool	conv3 × 3-64		maxpool
conv3-64	maxpool		recur-256
maxpool			recur-512
conv3-64			
maxpool			
fully connected-15			

Table 2. Summary of network configurations for the Indian Pines dataset.

CNN	2D CNN	RNN/LSTM	CRNN/CLSTM
Input-180			
conv10-32	conv3 × 3-64	recur-128	conv10-32
maxpool	conv3 × 3-64	recur-256	maxpool
conv10-32	maxpool	recur-512	conv10-32
maxpool	conv3 × 3-96		maxpool
conv5-64	maxpool		recur-256
maxpool			recur-512
conv5-64			
maxpool			
fully connected-19			

We implemented the neural networks using the TensorFlow [41] and Keras [42] framework. Experiments are carried out on a workstation with a 3.0-GHz Intel(R) Core i7-5960X CPU and an NVIDIA(R) GeForce Titan X GPU. The training process starts with the weights of all networks randomly initialized, and the initial learning rate is set to 10^{-4}. For mini-batch stochastic gradient descent, we use a batch size of 128 in all experiments. During training, the learning rate will be decreased by half every 500 epochs until loss reaches convergence.

3.3. Results

Since the datasets are highly unbalanced, so we run our experiments with the same number of training samples for each class. For the University of Houston dataset, which has 15 classes, we run experiments with 50 samples per class (750 in total), 100 samples per class (1500 in total) and 200 samples per class (3000 in total). For the Indian Pines dataset, which has 19 classes, we run experiments with 100 samples per class (1900 in total), 200 samples per class (3800 in total) and 300 samples per class (5700 in total). The classification performances for different models (SVM, CNN, 2D CNN, RNN, LSTM, CRNN, CLSTM) and the corresponding methods with the LOP spatial constraint on the University of Houston dataset are presented in Table 3, and results on the Indian Pines dataset are presented in Table 4. The classification results on both datasets show that our proposed method, CRNN, achieved the best performance in all scenarios.

Table 3. Classification results obtained by different approaches with different numbers of training samples on the University of Houston dataset. LOP, linear opinion pools.

Training Set Size	750	1500	3000
RBF-SVM	89.42 (±1.91)	92.86 (±1.13)	95.43 (±0.77)
CNN	90.91 (±0.69)	93.42 (±0.71)	96.25 (±0.46)
2D CNN	88.85 (±1.52)	94.26 (±0.61)	97.14 (±0.62)
RNN	78.67 (±1.93)	82.84 (±1.72)	92.16 (±0.66)
LSTM	86.55 (±0.89)	91.87 (±0.63)	94.05 (±0.54)
CRNN	93.42 (±0.46)	95.33 (±0.41)	97.64 (±0.30)
CLSTM	90.52 (±0.77)	94.53 (±0.47)	97.55 (±0.44)
CNN-LOP	93.36 (±0.81)	95.02 (±1.03)	97.87 (±0.50)
RNN-LOP	88.11 (±1.39)	93.52 (±1.43)	96.40 (±0.88)
LSTM-LOP	91.86 (±1.50)	94.67 (±0.62)	97.11 (±0.44)
CRNN-LOP	95.17 (±0.11)	97.08 (±0.36)	98.61 (±0.37)
CLSTM-LOP	93.22 (±1.16)	96.28 (±0.82)	98.21 (±0.45)

Table 4. Classification results obtained by different approaches with different numbers of training samples on the Indian Pines dataset.

Training Set Size	1900	3800	5700
RBF-SVM	92.82 (±1.07)	94.36 (±0.93)	95.13 (±0.64)
CNN	93.11 (±0.95)	94.53 (±0.39)	95.84 (±0.31)
2D CNN	88.78 (±0.85)	92.04 (±0.58)	92.82 (±0.65)
RNN	84.83 (±1.62)	89.74 (±0.98)	91.86 (±0.77)
LSTM	85.04 (±1.15)	89.83 (±0.74)	92.15 (±0.51)
CRNN	94.43 (±1.01)	96.24 (±0.60)	96.83 (±0.47)
CLSTM	92.72 (±1.08)	95.13 (±0.57)	96.16 (±0.55)
CNN-LOP	94.99 (±0.85)	96.78 (±0.76)	97.26 (±0.46)
RNN-LOP	91.39 (±1.11)	94.44 (±0.53)	95.27 (±0.52)
LSTM-LOP	91.62 (±0.36)	94.92 (±0.38)	95.32 (±0.41)
CRNN-LOP	96.61 (±0.75)	96.98 (±0.29)	98.08 (±0.44)
CLSTM-LOP	95.17 (±0.68)	96.52 (±0.81)	97.01 (±0.44)

We also show the training and validation loss and training and validation accuracy when training the neural networks on the University of Houston dataset and Indian Pines dataset in Figures 11 and 12, respectively. For every neural network, the training losses all converged to a level close to zero, and training accuracy converged to almost 100%. At the same time, the validation losses all converged to a low level, and the validation accuracies all reached a high number. It is also worth noting that CRNN and CLSTM converge faster than CNN and RNN. The corresponding number of parameters and training time (in minutes) for each network are summarized in Table 5. We can see that RNN and LSTM need much more training time than the others because the computational complexity for RNN/LSTM grows linearly with respect to the length of the input sequence, and most of the computations need to be done sequentially.

Table 5. Number of parameters and training time for the University of Houston dataset.

Network	# Parameters	Training Epochs	Run Time (min)
CNN	33,615	5000	15
2D CNN	37,295	500	3.6
RNN	516,623	5000	158
LSTM	2,043,407	2000	330
CRNN	481,807	500	4.7
CLSTM	1,884,943	500	14

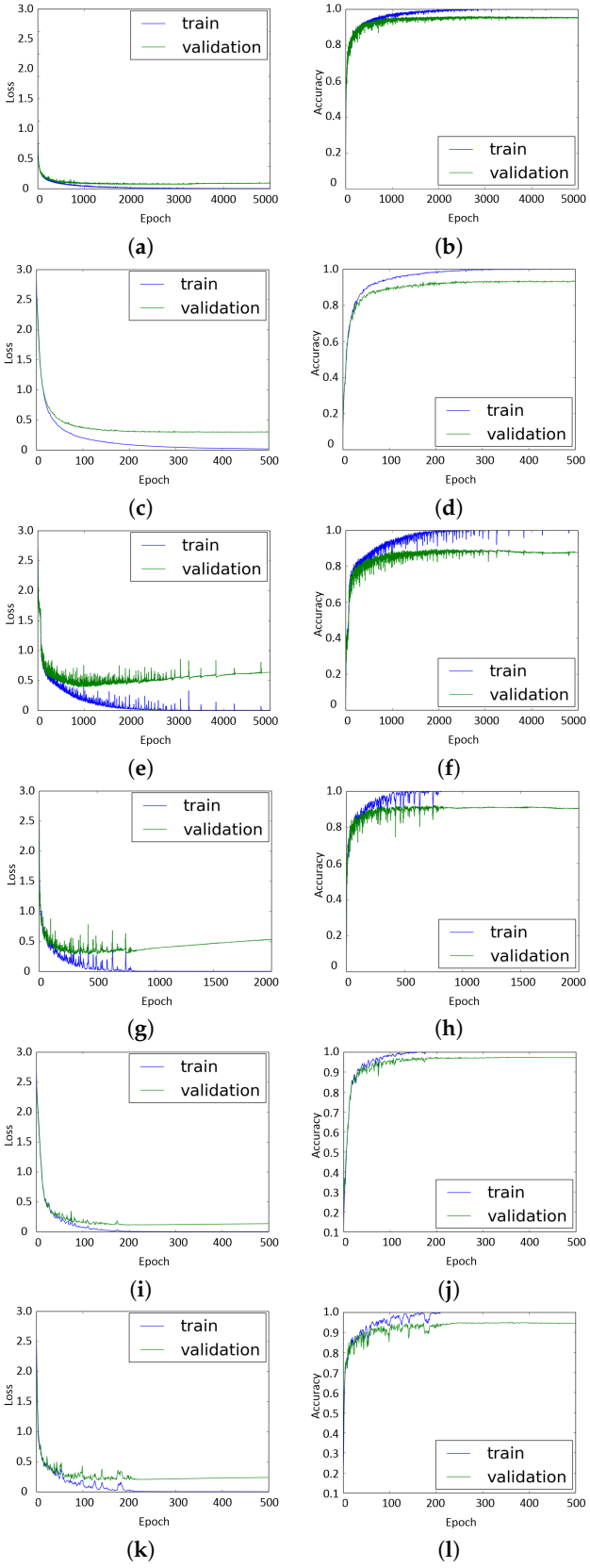

Figure 11. Loss and accuracy for the training and validation set when training different models on the University of Houston dataset: (**a**,**b**) for CNN; (**c**,**d**) for 2D CNN; (**e**,**f**) for RNN; (**g**,**h**) for LSTM; (**i**,**j**) for CRNN; (**k**,**l**) for CLSTM.

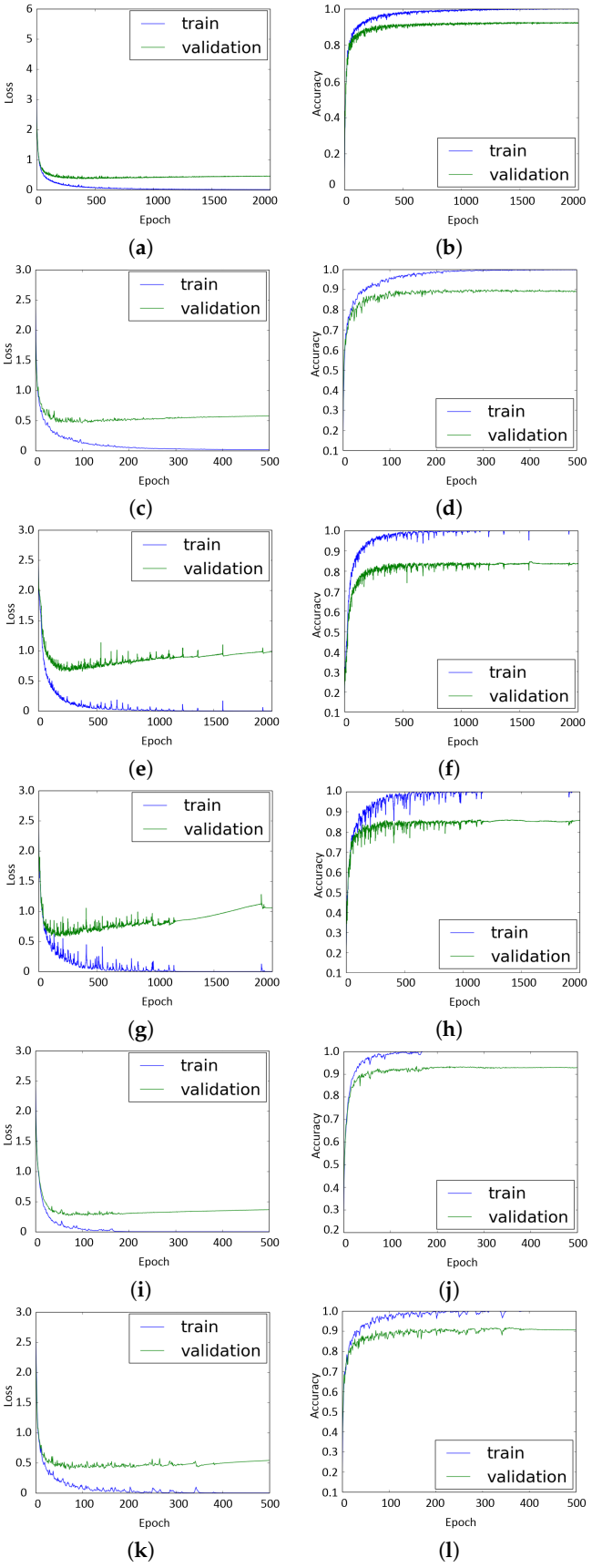

Figure 12. Loss and accuracy for the training and validation set when training different models on the Indian Pines dataset: (**a**,**b**) for CNN; (**c**,**d**) for 2D CNN; (**e**,**f**) for RNN; (**g**,**h**) for LSTM; (**i**,**j**) for CRNN; (**k**,**l**) for CLSTM.

Finally, we show the classification maps for different models used in this work in Figures 13 and 14. It is easy to see that the classification map of 2D spatial CNN is smoother than the other 1D models, but many important details (like edges between different classes) got smoothed out, as well, which resulted in incorrect prediction around the edges. When adding LOP as the spatial constraint, the maps become smoother, while the important edges still were preserved.

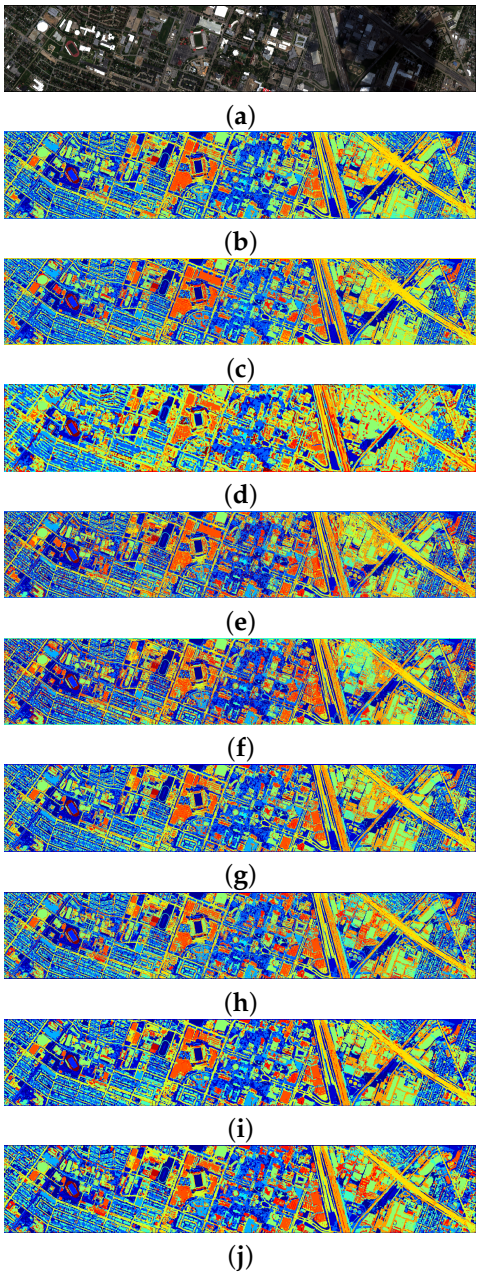

Figure 13. University of Houston dataset: (**a**) RGB image; (**b**–**f**) classification maps for different models: (**b**) SVM; (**c**) CNN; (**d**) 2D CNN; (**e**) RNN; (**f**) LSTM; (**g**) CRNN and (**h**) CLSTM; (**i**) CRNN-LOP and (**j**) CLSTM-LOP.

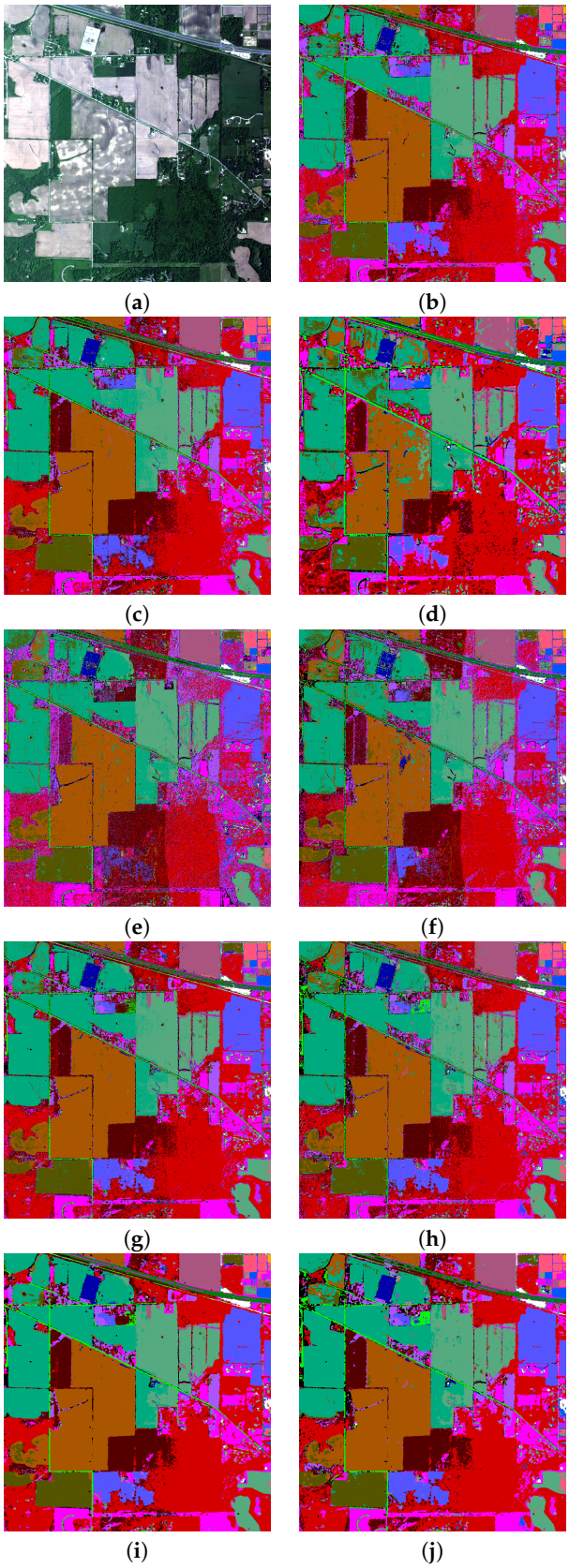

Figure 14. Indian Pines dataset: (**a**) RGB image. (**b**–**f**) Classification maps for different models: (**b**) SVM; (**c**) CNN; (**d**) 2D CNN; (**e**) RNN; (**f**) LSTM; (**g**) CRNN and (**h**) CLSTM; (**i**) CRNN-LOP and (**j**) CLSTM-LOP.

4. Discussion

The classification results in Tables 3 and 4 show that our proposed method, CRNN, achieved the best performance in all scenarios. In particular, a few interesting and practically relevant observations can be made, which readers will find helpful when applying deep learning techniques to extract deep features from hyperspectral images:

- CNN and CRNN/CLSTM achieved better classification results than the traditional method RBF-SVM in all scenarios, while the performances of RNN/LSTM are still worse than RBF-SVM.
- The 2D spatial CNN performs better than SVM on the University of Houston dataset, but worse on the Indians Pines dataset. The reason is that the University of Houston dataset contains urban objects that have much more spatial features than the vegetation categories in the Indian Pines dataset.
- As expected, the performances of CRNN/CLSTM are better than CNN because CRNN/CLSTM have the advantages of both convolutional networks and recurrent networks.
- The fact that the performance of CRNN/CLSTM are significantly better than RNN/LSTM tells us that the middle-level features extracted by the convolutional layers in CRNN/CLSTM help the following recurrent layers to better capture the contextual information.
- LSTM network has better performance than the regular RNN, especially for the University of Houston dataset, because LSTM networks are capable of capturing the long-term dependencies in the input sequence and, thus, avoid the gradient vanishing problem.
- CLSTM performs no better than CRNN in all cases, meaning that CRNN does not have the long-term dependency and gradient vanishing problem because the length of the sequence is already much reduced by the two pooling layers before the recurrent layers. The reason why CLSTM is even worse than CRNN when the training set is small is that the CLSTM has much more parameters than CRNN, so it tends to overfit the training data and performs worse on test data.
- The LOP-based spatial constraint further improved the performances of all of the 1D models.

To better understand the classification power of each neural network, we take the high dimensional features extracted by all models trained on the University of Houston dataset with 1500 training samples and use the t-SNE [43] algorithm to reduce the dimensionality to two. The results are visualized in Figure 15, where different colors stands for 15 classes in the University of Houston dataset. Similarly, the feature visualizations for the Indian Pines dataset are depicted in Figure 16. Features extracted by all models are more discriminative than the original hyperspectral data. Features extracted by the CRNN for different classes are better separated than the other models, which means that features extracted by CRNN are the most discriminative. The reason is that the first few convolutional (and pooling) layers in CRNN extract middle-level features where the spectral variation has been decreased, which makes it easier for the following recurrent layers to learn the contextual information. This can be verified in Figure 17, which shows an example of features extracted by the convolutional layers. Compared to the input signal, the extracted features are smoother since convolutional layers are able to remove local variations. Furthermore, the convolutional layer also successfully captures some important information, such as the locations of peaks and slopes, which are quite discriminative between different classes. Thus, recurrent layers can capture the contextual information more effectively from the features extracted by convolutional layers.

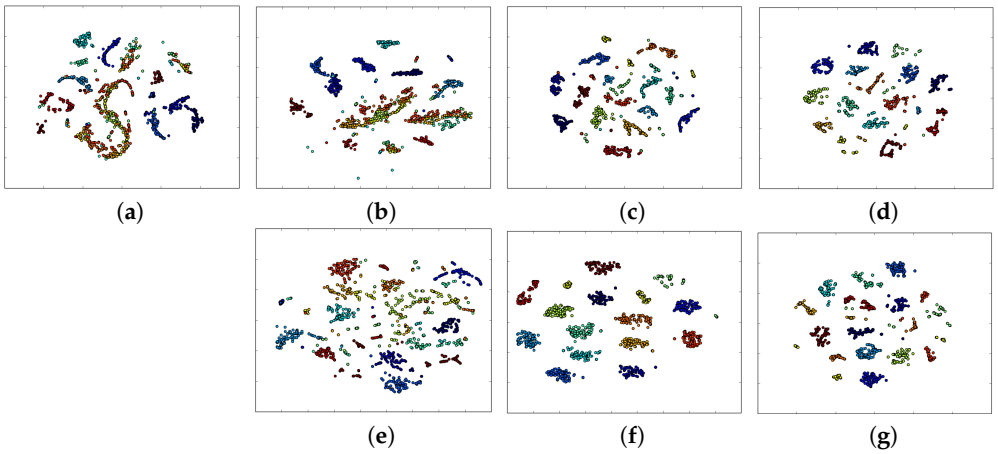

Figure 15. University of Houston dataset: two-dimensional embedding extracted by t-SNE for (**a**) input data, and features extracted by: (**b**) CNN; (**c**) RNN; (**d**) LSTM; (**e**) 2D CNN; (**f**) CRNN; (**g**) CLSTM.

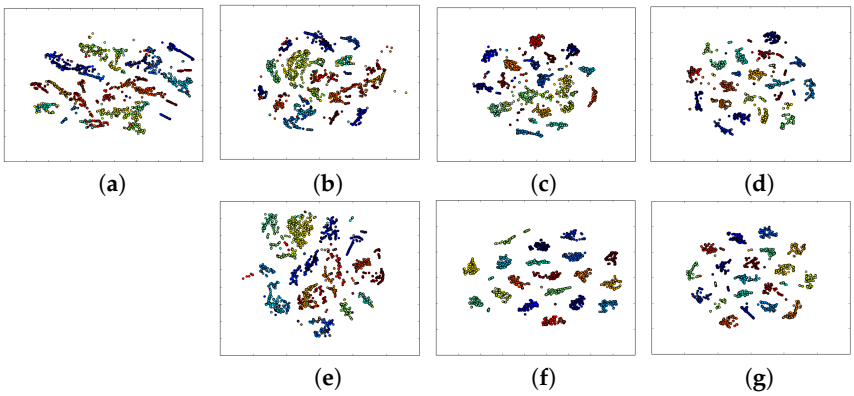

Figure 16. Indian Pines dataset: two dimensional embedding extracted by t-SNE for (**a**) input data, and features extracted by: (**b**) CNN; (**c**) RNN; (**d**) LSTM; (**e**) 2D CNN; (**f**) CRNN; (**g**) CLSTM.

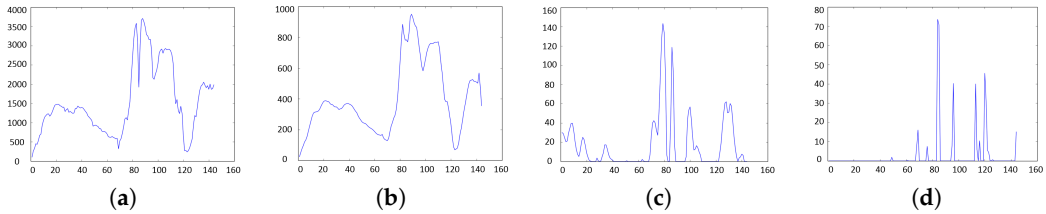

Figure 17. An input hyperspectral pixel in (**a**) and features extracted by the convolutional layer in (**b**–**d**): (**b**) is a smoothed curve of (**a**); (**c**) tries to extract local peaks in (**a**); and (**d**) captures slope information in (**a**).

5. Conclusions

In this paper, we proposed to extract the contextual information in hyperspectral data by modeling the dependencies between different spectral bands using an RNN. In particular, we employed a CRNN model, which consists of several convolutional layers and a few recurrent layers, for hyperspectral data classification. The first convolutional layers are utilized to extract middle-level and locally invariant features, which are then fed to a few recurrent layers to additionally extract the contextual information between different spectral bands. By combining convolutional and recurrent layers, our CRNN model is able to extract more discriminative feature representations for classification, and it outperformed other state-of-the-art methods on real hyperspectral image datasets.

In the future, we would like to explore semi-supervised deep learning techniques for hyperspectral image classification because it is usually very expensive to get labeled data for remote sensing applications.

Acknowledgments: Our project funding covers the cost to publish in open access journals.

Author Contributions: H.W. and S.P. designed the study and planned the experiments. H.W. ran all experiments and organized the results. H.W. wrote the draft, and S.P. reviewed and revised the paper.

Conflicts of Interest: The authors declare no conflict of interest. The funding sponsors had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; nor in the decision to publish the results.

References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444.
2. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*; Neural Information Processing Systems Foundation, Inc.: South Lake Tahoe, NV, USA, 2012; pp. 1097–1105.
3. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587.
4. Girshick, R. Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 13–16 December 2015; pp. 1440–1448.
5. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
6. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 27–30 June 2016.
7. Nanni, L.; Ghidoni, S. How could a subcellular image, or a painting by Van Gogh, be similar to a great white shark or to a pizza? *Pattern Recognit. Lett.* **2017**, *85*, 1–7.
8. Barat, C.; Ducottet, C. String representations and distances in deep Convolutional Neural Networks for image classification. *Pattern Recognit.* **2016**, *54*, 104–115.
9. Sainath, T.N.; Mohamed, A.R.; Kingsbury, B.; Ramabhadran, B. Deep convolutional neural networks for LVCSR. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 8614–8618.
10. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780.
11. Gers, F.A.; Schmidhuber, J.; Cummins, F. Learning to forget: Continual prediction with LSTM. *Neural Comput.* **2000**, *12*, 2451–2471.
12. Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **2005**, *18*, 602–610.
13. Graves, A.; Mohamed, A.R.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 6645–6649.
14. Graves, A.; Jaitly, N. Towards end-to-end speech recognition with recurrent neural networks. In Proceedings of the International Conference on Machine Learning (ICML), Beijing, China, 21–26 June 2014; Volume 14, pp. 1764–1772.
15. Sak, H.; Senior, A.; Rao, K.; Beaufays, F. Fast and accurate recurrent neural network acoustic models for speech recognition. *arXiv* **2015**, arXiv:1507.06947.
16. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*; Neural Information Processing Systems Foundation, Inc.: South Lake Tahoe, NV, USA, 2014; pp. 3104–3112.
17. Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
18. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.

19. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324.
20. Landgrebe, D. Hyperspectral image data analysis. *IEEE Signal Process. Mag.* **2002**, *19*, 17–28.
21. Shaw, G.; Manolakis, D. Signal processing for hyperspectral image exploitation. *IEEE Signal Process. Mag.* **2002**, *19*, 12–16.
22. Manolakis, D.; Siracusa, C.; Shaw, G. Hyperspectral subpixel target detection using the linear mixing model. *IEEE Trans. Geosci. Remote Sens.* **2001**, *39*, 1392–1409.
23. Huang, C.; Davis, L.; Townshend, J. An assessment of support vector machines for land cover classification. *Int. J. Remote Sens.* **2002**, *23*, 725–749.
24. Li, J.; Marpu, P.R.; Plaza, A.; Bioucas-Dias, J.M.; Benediktsson, J.A. Generalized composite kernel framework for hyperspectral image classification. *IEEE Trans. Geosci. Remote Sens.* **2013**, *51*, 4816–4829.
25. Wu, H.; Prasad, S. Dirichlet process based active learning and discovery of unknown classes for hyperspectral image classification. *IEEE Trans. Geosci. Remote Sens.* **2016**, *54*, 4882–4895.
26. Yuen, P.W.; Richardson, M. An introduction to hyperspectral imaging and its application for security, surveillance and target acquisition. *Imaging Sci. J.* **2010**, *58*, 241–253.
27. Malthus, T.J.; Mumby, P.J. Remote sensing of the coastal zone: An overview and priorities for future research. *Int. J. Remote Sens.* **2003**, *24*, 2805–2815.
28. Chen, Y.; Lin, Z.; Zhao, X.; Wang, G.; Gu, Y. Deep learning-based classification of hyperspectral data. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2014**, *7*, 2094–2107.
29. Hu, W.; Huang, Y.; Wei, L.; Zhang, F.; Li, H. Deep convolutional neural networks for hyperspectral image classification. *J. Sens.* **2015**, *2015*.
30. Yue, J.; Zhao, W.; Mao, S.; Liu, H. Spectral—Spatial classification of hyperspectral images using deep convolutional neural networks. *Remote Sens. Lett.* **2015**, *6*, 468–477.
31. Zhao, W.; Du, S. Spectral–Spatial feature extraction for hyperspectral image classification: A dimension reduction and deep learning approach. *IEEE Trans. Geosci. Remote Sens.* **2016**, *54*, 4544–4554.
32. Liang, H.; Li, Q. Hyperspectral imagery classification using sparse representations of convolutional neural network features. *Remote Sens.* **2016**, *8*, 99.
33. Hu, F.; Xia, G.S.; Hu, J.; Zhang, L. Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery. *Remote Sens.* **2015**, *7*, 14680.
34. Lyu, H.; Lu, H.; Mou, L. Learning a transferable change rule from a recurrent neural network for land cover change detection. *Remote Sens.* **2016**, *8*, 506.
35. Zuo, Z.; Shuai, B.; Wang, G.; Liu, X.; Wang, X.; Wang, B.; Chen, Y. Convolutional recurrent neural networks: Learning spatial dependencies for image representation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Boston, MA, USA, 7–12 June 2015; pp. 18–26.
36. Xiao, Y.; Cho, K. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv* **2016**, arXiv:1602.00367.
37. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166.
38. Werbos, P.J. Backpropagation through time: What it does and how to do it. *Proc. IEEE* **1990**, *78*, 1550–1560.
39. Benediktsson, J.A.; Sveinsson, J.R. Multisource remote sensing data classification based on consensus and pruning. *IEEE Trans. Geosci. Remote Sens.* **2003**, *41*, 932–936.
40. Wu, H.; Prasad, S. Infinite Gaussian mixture models for robust decision fusion of hyperspectral imagery and full waveform LiDAR data. In Proceedings of the 2013 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Austin, TX, USA, 3–5 December 2013; pp. 1025–1028.
41. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: tensorflow.org (accessed on 9 January 2017).
42. Chollet, F. Keras. Available online: https://github.com/fchollet/keras (accessed on 9 January 2017).
43. Maaten, L.V.D.; Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.