The Journal of Supercomputing (2020) 76:8866–8882
https://doi.org/10.1007/s11227-020-03187-0

Scalable recurrent neural network for hyperspectral image classification

Mercedes E. Paoletti[1] · Juan M. Haut[1] · Javier Plaza[1] · Antonio Plaza[1]

Published online: 4 February 2020

Abstract
Hyperspectral imaging (HSI) collects hundreds of images over large spatial observation areas on the Earth's surface, recording scenes at different wavelength channels and providing a vast amount of information. Recurrent neural networks (RNNs) have been widely used for the classification of HSI datasets, understood as a single sequence of pixel vectors with high dimensionality. However, the RNN model scales poorly when dealing with HSI scenes with large dimensionality. In order to mitigate this problem, this paper presents a new RNN classifier based on simple recurrent units that performs HSI classification in a highly scalable and efficient way. Our experimental results (conducted on four real HSI datasets) reveal very good performance, not only in terms of classification accuracy (in line with existing methods), but also in terms of computational performance when dealing with large datasets.

Keywords Hyperspectral image · Recurrent neural networks · CUDA

1 Introduction

The significant advances in computing technology achieved in the last decade, coupled with the newest developments in imaging spectroscopy [18], have allowed the development of new Earth observation (EO) missions with powerful airborne and satellite hyperspectral imaging (HSI) sensors, which can capture high-quality

Source codes: https://github.com/mhaut/scalable_RNN_HSI.

✉ Juan M. Haut
juanmariohaut@unex.es

Mercedes E. Paoletti
mpaoletti@aunex.es

Javier Plaza
jplaza@unex.es; aplaza@unex.es

[1] Department of Technology of Computers and Communications, University of Extremadura, Escuela Politecnica, Avda. de la Universidad s/n, Cáceres, Spain

images composed by hundreds of measurements (at different wavelength channels) over extensive spatial areas, acquiring information in hundreds of continuous and narrow bands, ranging from the visible to the near-infrared (NIR) and shortwave-infrared (SWIR) [8] parts of the electromagnetic spectrum.

As a result, current spectrometers are able to produce very large HSI data cubes, where each pixel contains the spectral signature of the observed materials. These spectral signatures collect the physical–chemical behavior of materials in the presence of solar light, being unique for each kind of terrestrial object, and allowing to describe and identify each element of the scene, not only at an object level, but also at pixel (and even sub-pixel) level of detail [19], providing abundant information for the characterization of the surface of the Earth. Such information can be used in a wide range of human activities, such as hydrology [12], forestry [20], geology [22] and mineralogy [5]), as well as precision agriculture [7], urban planning [27], and prevention and management of disasters.

In this context, the analysis and processing of HSI datasets plays an important role in remote sensing [14], demanding the development of effective and efficient techniques for the analysis of these data. In particular, this paper focuses on methods that identify the content of an HSI scene by associating each pixel in the scene with a corresponding land-cover class. These methods can be described as the mapping function $f(\cdot, \theta)$, where $f : \mathbf{X} \rightarrow \mathbf{Y}$ is able to relate each pixel of the HSI scene $\mathbf{x}_i \in \mathbf{X}$ with a land-cover category $\mathbf{y}_i$ by adjusting its parameters θ, obtaining at the end a classification map $\mathbf{Y}$ with pairs of the form: $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n_{\text{pixels}}}$.

Besides, it must be noted that HSI data present several challenges related to its volume and complexity. While a detailed spectral signature allows for the unique identification of each material in the scene, its large spectral dimension and content can further complicate the classification process, imposing significant storage and processing restrictions. As a result, classification methods must be efficient and scalable in the use of computational and storage elements [25]. Moreover, the use of a large number of spectral features increases the complexity of classification methods, hampering their performance and leading to lower classification accuracies with more features (*peaking paradox*). In the field of HSI, this issue corresponds with the *curse of dimensionality* phenomenon that, coupled with high intraclass variability and interclass similarity, makes the classification problem an extremely ill-posed one.

Among available classification methods, deep neural networks (DNNs) [3] have attracted the attention of the HSI research community due to several characteristics that facilitate the classification of these kinds of data, including the following aspects:

- They do not need prior information about statistical properties of the HSI data to extract and process the spectral, spatial and spectral–spatial information contained in the scenes.
- Their working mode is based on the optimization of a loss function, for instance the mean square error (MSE) between the networks' outputs and the desired outputs, through the adjustment of the networks' parameters θ. To achieve this, they employ a forward-backward iterative mechanism (based on gradient descent

optimizers) to find the optimal θ, being able to work as universal approximators [6].
– They offer great flexibility in terms of learning models, i.e., unsupervised, supervised and semi-supervised.
– As stacks of layers composed by neurons, they provide a great variety of architectures, from shallow to deep and very deep ones, employing fully (or locally) connected neurons, and implementing one or more paths.

Moreover, advances in computing technology have allowed the implementation of deepest and more complex neural models, which have led to a revolution in deep learning (DL) techniques [15]. Focusing on HSI classification, these DNNs are able to extract representative features, learning simple representations at the first layers and extending them to more complex abstractions at the final layers (hierarchical learning), discovering nonlinear relationships in the input HSI data and yielding high performance in HSI classification [24, 32].

In particular, the recurrent neural network (RNN) [29] is an interesting classifier that presents an internal structure similar to a directed graph, implementing loops in the connections of the layers that force each node activation of the current step to depend on the activations of the preceding ones. In this sense, the RNN is a powerful model for learning sequences of data, storing an internal state that provides a memory to relate the current input data sample with the previous ones. At the end, these states allow to process the contextual information of the data, extracting temporal features. The RNN has been previously employed to perform HSI data classification, considering each spectral pixel as the input sequential data of the model, as Mou et al. propose in [21]. Other approaches even combine spectral information with spatial information. For instance, Zhou et al. [34] concatenate the spatial information of a neighborhood window (extracted with PCA [30]) to the spectral information of the pixel. Zhang et al. [33] consider several principal components (for which they extract Gabor textures and differential morphological profiles) which are combined and stacked to conform local spatial sequential (LSS) features that will be sent as input to the RNN model.

Nonetheless, all the aforementioned spectral and spectral–spatial methodologies must face an important restriction. In particular, RNNs have been shown to suffer from overfitting when the length of their input sequences is very large, which happens often when dealing with HSIs, since spectral bands are handled as sequences of length n_3 in which each sequence feature is a band, so overfitting becomes more evident as n_3 increases. To mitigate this problem, it is usual to include more information to the model, e.g., grouping the spectral bands to enlarge the size of the features and shorten the length of the sequence. However, this practice hampers the scalability and performance of the network. Also, as the sequence and/or feature length increases, the runtimes needed by the model become longer.

In order to improve the performance of RNN models (in terms of both runtimes and scalability), some interesting parallel versions and toolkits have been developed [1, 17, 23]. For instance, [28] presents a general RNN with a parallel implementation for graphics processing units (GPUs) based on NVIDIA CUDA. Other works focus on the skip of the hidden states [2] in order to speed up the data processing.

However, to the best of our knowledge, very few efforts have been focused on accelerating the processing of HSIs with RNNs. However, accelerating RNNs for HSI data processing can give an adequate solution to the problems of scalability, runtime and overfitting when dealing with the high spectral dimensionality of these kinds of data.

This paper investigates (for the first time in the available literature) the scalable implementation of a novel variant of the RNN model, called simple recurrent unit (SRU) [16], for HSI data classification. Comparing the SRU with traditional recurrent units, its architecture allows faster learning in terms of training speed, reducing the number of trainable parameters while maintaining a reliable performance (in terms of accuracy). Thus, the main goal of this paper is to reduce the internal complexity of the RNN model (i.e., the relationships between the current outputs and the previous ones), thus facilitating the parallelization of the computations performed by the recurrent units in order to enhance the performance of traditional RNN models for HSI data classification.

2 Recurrent neural units: overview

An HSI data scene $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ can be represented as a matrix of $n_1 \times n_2$ pixels, where each pixel $\mathbf{x}_i$ is composed by n_3 spectral bands. On this wise, the classification pursues to associate each pixel with a corresponding land-cover class (label), obtaining a classification map $\mathbf{Y} \in \mathbb{R}^{n_1 \times n_2 \times n_{\text{classes}}} \equiv \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n_1 \cdot n_2}$.

RNN models for HSI data interpretation process the spectral signature contained in each $\mathbf{x}_i$ as a time sequence, where the spectral bands are considered as time steps (see Fig. 1). This allows the pixels to be processed in band-by-band fashion [21]. Alternatively, the bands can be arranged into groups [31]. Moreover, the RNN model provides three different units: (i) vanilla recurrent unit, (ii) long short-term memory (LSTM) [11], and (iii) gated recurrent unit (GRU) [4]. The vanilla unit is the oldest and simplest model, as defined by Eq. (1):

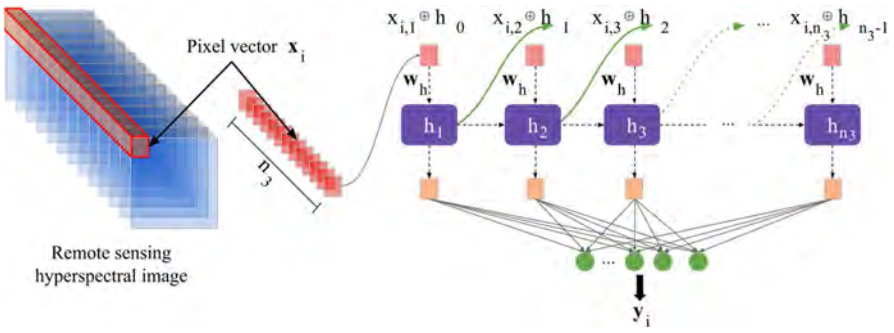


Fig. 1 Traditional band-by-band RNN model for HSI data classification. The spectral signature contained in each pixel is processed as a temporal sequence, where different spectral bands correspond to different time steps

$$\mathbf{h}_t = \begin{cases} 0 & \text{if } t = 0 \\ \mathcal{H}\left(\mathbf{W}_h \cdot \mathbf{x}_t + \mathbf{U}_h \cdot \mathbf{h}_{t-1} + b_h\right) & \text{if } t \neq 0 \end{cases} \tag{1}$$

Considering $\mathbf{x}_t \in \mathbb{R}^n$ as the input sequential sample, the vanilla model computes the output $\mathbf{y}_t$ as a hidden state at time t, i.e., $\mathbf{y}_t = \mathbf{h}_t$, by combining the current input with the unit's data weights $\mathbf{W}_h$ and bias b_h and the previous state $\mathbf{h}_{t-1}$, weighted by the connection weights $\mathbf{U}_h$, being $\mathcal{H}(\cdot)$ a nonlinear activation function such as sigmoid or hyperbolic tangent (tanh). As a result, the obtained hidden state works as the memory of the model, and it is applied to the subsequent sample $\mathbf{x}_{t+1}$.

Due to its simplicity, the vanilla RNN tends to quickly degrade the interpretation of high-dimensional data, reaching poor classification results. The LSTM deals with the data degradation by reinterpreting the original RNN as a cell composed by two main states, the hidden $\mathbf{h}_t$ and cell $\mathbf{c}_t$ states, which are controlled by three gates known as input $\mathbf{i}_t$, output $\mathbf{o}_t$ and forget $\mathbf{f}_t$ gates, in order to manage the information flow that goes through the unit. This mechanism allows the LSTM to learn useful information along time, disregarding the irrelevant one.

$$\begin{aligned} \mathbf{i}_t &= \mathcal{H}\left(\mathbf{W}_i \cdot \mathbf{x}_t + \mathbf{U}_i \cdot \mathbf{h}_{t-1} + b_i\right) \\ \mathbf{f}_t &= \mathcal{H}\left(\mathbf{W}_f \cdot \mathbf{x}_t + \mathbf{U}_f \cdot \mathbf{h}_{t-1} + b_f\right) \\ \mathbf{o}_t &= \mathcal{H}\left(\mathbf{W}_o \cdot \mathbf{x}_t + \mathbf{U}_o \cdot \mathbf{h}_{t-1} + b_o\right) \\ \mathbf{c}_t &= \begin{cases} 0 & \text{if } t = 0 \\ \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathcal{H}\left(\mathbf{W}_c \cdot \mathbf{x}_t + \mathbf{U}_c \cdot \mathbf{h}_{t-1} + b_c\right) & \text{if } t \neq 0 \end{cases} \\ \mathbf{h}_t &= \begin{cases} 0 & \text{if } t = 0 \\ \mathbf{o}_t \circ \mathcal{H}\left(\mathbf{c}_t\right) & \text{if } t \neq 0 \end{cases} \end{aligned} \tag{2}$$

As we can observe in Eq. (2), $\mathbf{h}_t$ works as the traditional output of the unit, while $\mathbf{c}_t$ includes (or removes) information into (from) the cell, depending on the gates' values. Thus, the input gate $\mathbf{i}_t$ determines whether a new input sample is allowed to reach inside the cell or not; the forget gate $\mathbf{f}_t$ deletes the irrelevant information; and the output gate $\mathbf{o}_t$ weights the unit's output signal at time t.

However, this control-gate mechanism introduces significant complexity to LSTM. With this in mind, the GRU model tries to make a compromise between the simplicity of the vanilla unit and the high performance of the LSTM. In fact, the GRU can be considered as a simplified LSTM, with the output gate removed (this involves fewer parameters) and the input and forget gates evolved into update ($\mathbf{z}_t$) and reset ($\mathbf{r}_t$) gates, as Eq. (3) shows:

$$\begin{aligned} \mathbf{z}_t &= \mathcal{H}\left(\mathbf{W}_z \cdot \mathbf{x}_t + \mathbf{U}_z \cdot \mathbf{h}_{t-1} + b_z\right) \\ \mathbf{r}_t &= \mathcal{H}\left(\mathbf{W}_r \cdot \mathbf{x}_t + \mathbf{U}_r \cdot \mathbf{h}_{t-1} + b_r\right) \\ \mathbf{h}'_t &= \tanh\left(\mathbf{W}_h \cdot \mathbf{x}_t + \mathbf{r}_t \circ \mathbf{U}_h \cdot \mathbf{h}_{t-1} + b_h\right) \\ \mathbf{h}_t &= \begin{cases} 0 & \text{if } t = 0 \\ \mathbf{z}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \circ \mathbf{h}'_t & \text{if } t \neq 0 \end{cases} \end{aligned} \tag{3}$$

Although these models are able to reach acceptable performance in HSI data classification, they present an important computational restriction due to the high dependence on previous steps. In fact, although algebraic operations can be optimized and parallelized in hardware, such dependencies hamper the speedup as the amount of data (and the dimensionality of the feature space) grow, making the RNN model scale poorly in this context. In addition, the RNN suffers from overfitting and vanishing gradient [10] problems. When processing HSI data, these models can see their performance severely compromised when the spectral dimension of the images is too high. In order to overcome these limitations, in the next section we introduce a new RNN-based architecture that employs a SRU as the main block of the model.

3 Proposed method

3.1 SRU methodology

The SRU simplifies the internal architecture of the recurrent cell. With a design in-between that of the LSTM and GRU models, it exhibits two main states: the hidden state $\mathbf{h}_t$ and the cell state $\mathbf{c}_t$, controlled by the forget $\mathbf{f}_t$ and reset $\mathbf{r}_t$ gates, as Eq. (4) indicates:

$$\begin{aligned} \mathbf{f}_t &= \mathcal{H}\left(\mathbf{W}_f \cdot \mathbf{x}_t + \mathbf{u}_f \odot \mathbf{c}_{t-1} + b_f\right) \\ \mathbf{r}_t &= \mathcal{H}\left(\mathbf{W}_r \cdot \mathbf{x}_t + \mathbf{u}_r \odot \mathbf{c}_{t-1} + b_r\right) \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \left(1 - \mathbf{f}_t\right) \odot \left(\mathbf{W}_c \cdot \mathbf{x}_t\right) \\ \mathbf{h}_t &= \mathbf{r}_t \odot \mathbf{c}_t + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \cdot \alpha \end{aligned} \tag{4}$$

Internally, Eq. (4) can be divided in two main components: (i) the *light recurrence*, which reads the input $\mathbf{x}_t$ at time t and computes the forget gate $\mathbf{f}_t$ and the cell state $\mathbf{c}_t$ values (capturing the relevant sequential information), and (ii) the *highway network*, which obtains the unit's output by using the reset gate $\mathbf{r}_t$ and the hidden state $\mathbf{h}_t$.

Focusing on the light recurrence concept, we observe that gates and cell states are not obtained by relying on previous hidden states (i.e., $\mathbf{h}_{t-1}$), but on the previous cell state (i.e., $\mathbf{c}_{t-1}$). Furthermore, the SRU units do not integrate the previous states through a matrix multiplication ($\cdot$), which is a quite complex mathematical operation to parallelize at the hardware level. This is because each dimension of the resulting output depends on all the entries of $\mathbf{c}_{t-1}$, imposing a "waiting time" (in fact, a delay) until the state $\mathbf{c}_{t-1}$ is fully computed. In turn, the SRU performs a point-wise multiplication ($\odot$), allowing the independence of each output's dimension to obtain part of the output without having the previous state $\mathbf{c}_{t-1}$ fully calculated. This point-wise multiplication is also employed by the highway network [26], which adaptively combines the current input $\mathbf{x}_t$ with the cell state $\mathbf{c}_t$ through the reset gate $\mathbf{r}_t$. Moreover, it reduces the vanishing gradient problem by implementing a skip connection $(1 - \mathbf{r}_t) \odot \mathbf{x}_t \cdot \alpha$, controlled by the *scaling correction* constant α, which helps to propagate the gradient signal.

3.2 Proposed CUDA-based SRU for HSI classification

The proposed SRU has been implemented over a GPU using CUDA, aiming at increasing the model's performance by parallelizing the operations given in Eq. (4). In this context, it must be noted that the HSI data have been reshaped into an $n \times n_3$ matrix (with $n = n_1 \times n_2$) and divided into training $\mathcal{D}_{\text{train}}$ and test $\mathcal{D}_{test}$ subsets to adjust the parameters and validate the model, being $\mathcal{D}_{\text{train}}$ organized as batches $\mathbf{X} \in \mathbb{R}^{n_3 \times n_b}$, having each one n_b sequences (i.e., pixels) of n_3 features (i.e., spectral bands).

Besides, the network has been implemented as a *many-to-one* model with one cell that computes across input sequences to perform a single prediction from each one, defining the hidden space $\mathbb{R}^{n_h}$. From Eq. 4, each weight matrix $\mathbf{W}_*$ dotted by $\mathbf{X}$ is extracted, fusing these computations (by using cuBLAS) into a single matrix multiplication that obtains the auxiliary matrix $\mathbf{M} \in \mathbb{R}^{n_b \times k \cdot n_h}$ of Fig. 2. It is worth mentioning that, as a highway network's point-wise operation is performed, if the input and output data sizes are not the same, the $\mathbf{M}$ will introduce a fourth matrix that will serve as the highway connection's weights, setting $k = 4$ and performing $(1 - \mathbf{r}_t) \odot (\mathbf{W} \cdot \mathbf{x}_t)$ in Eq. (4). With this in mind, a CUDA kernel composed by $n_b \cdot n_h$ threads, arranged in the form of one-dimensional blocks, has been implemented. Algorithm 1 shows a pseudocode of the aforementioned kernel, where all the data structures (i.e., matrices and vectors) have been stored considering the C-style (i.e., row-major order) memory storage scheme. Moreover, Fig. 3 gives a graphical example of how the CUDA threads access the positions of matrix $\mathbf{M}$. Finally, our model

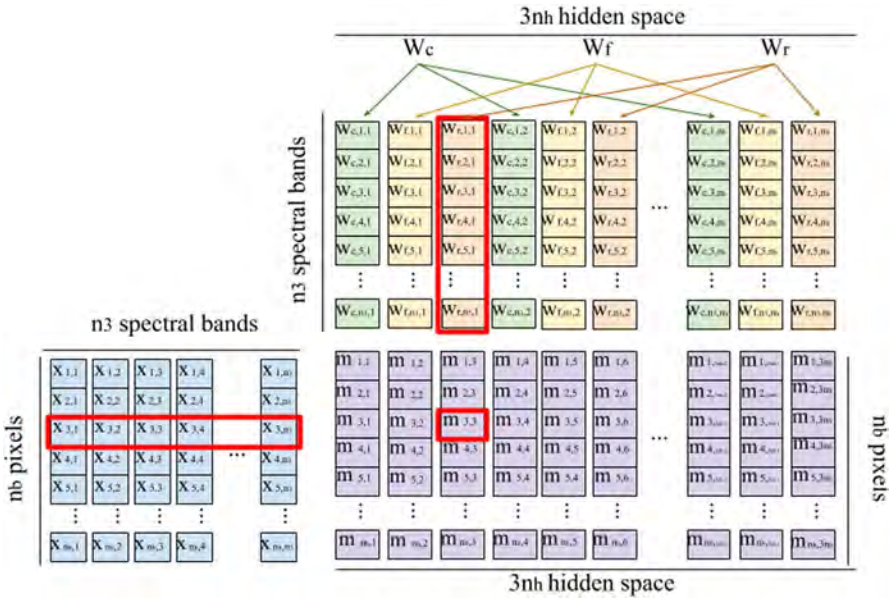


Fig. 2 General matrix-to-matrix multiplication between the current HSI data batch and the model's weights, optimized through cuBLAS library

Fig. 3 Diagram illustrating the threads' access to auxiliary matrix **M**, where black, red green and purple arrows represent those threads with identifier 0, 1, 2 and $n_b \cdot n_h - 1$, respectively

considers the following hyperparameters: $n_b = 100$, $n_h = 144$, $n_3 =$ spectral bands and $k = 4$, setting the maximum number of threads per block to 512.

Algorithm 1 Pseudocode of SRU model to perform HSI data classification

procedure HSI-SRU($\mathbf{X} \in \mathbb{R}^{n_b \times n_3}$: HSI data batch, $\mathbf{M} \in \mathbb{R}^{n_b \times k \cdot n_h}$: auxiliary matrix, $\mathbf{c}_{t-1} \in \mathbb{R}^{n_3 \times n_b \cdot n_h}$: previous cell state, $\mathbf{u} \in \mathbb{R}^{2n_h}$: weight vectors $\mathbf{u}_f$ and $\mathbf{u}_r$, $\mathbf{b} \in \mathbb{R}^{2n_h}$: biases vectors $\mathbf{b}_f$ and $\mathbf{b}_r$, n_b: batch size, n_h: hidden space size, n_3: spectral dimensions, k: factor)
  $i = blockIdx.x * blockDim.x + threadIdx.x$ ▷ current thread id
  if $i < (n_b \cdot n_h)$ **then** ▷ check that the maximum allowed dimension is not exceeded
    $N = n_b \cdot n_h \cdot k$ ▷ Elements of auxiliary matrix **M**
    $m = i \cdot k$ ▷ Index of auxiliary matrix **M**
    for $l = 0, \ldots, n_3$ **do** ▷ Go along the sequences
      $\mathbf{f}_t = \sigma\left(\mathbf{M}[m+1] + \mathbf{u}[i\%n_h] \cdot \mathbf{c}_{t-1}[i] + \mathbf{b}[(i\%n_h)]\right)$
      $\mathbf{r}_t = \sigma\left(\mathbf{M}[m+2] + \mathbf{u}[(i\%n_h) + n_h] \cdot \mathbf{c}_{t-1}[i] + \mathbf{b}[(i\%n_h)]\right)$
      $\mathbf{c}_t = \mathbf{f}_t \cdot \mathbf{c}_{t-1}[i] + (1 - \mathbf{f}_t) \cdot \mathbf{M}[m]$
      $\mathbf{h}_t = \mathbf{r}_t \cdot \mathbf{c}_t + (1 - \mathbf{r}_t) \cdot \mathbf{M}[m+3]$
      $m = m + N$ ▷ Updating index of **M**
      $\mathbf{c}_{t-1} = \mathbf{c}_t$ ▷ Updating previous state
      $\mathbf{c} = \text{Concatenate}(\mathbf{c}_t)$
      $\mathbf{h} = \text{Concatenate}(\mathbf{h}_t)$
    end for
  end if
  return h and **c**
end procedure

4 Experimental results

4.1 Hyperspectral datasets

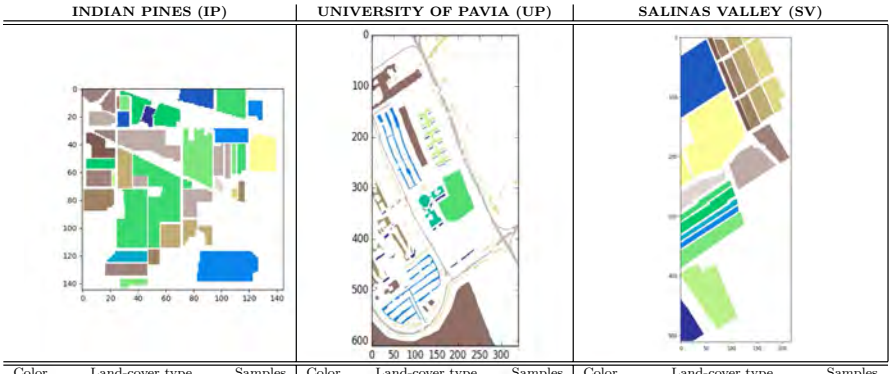
In order to test the proposed SRU model for HSI classification purposes, we perform an exhaustive comparison between all the available recurrent architectures: vanilla RNN, LSTM and GRU (implemented with CUDA and CuDNN) and the proposed method, with the aim of quantifying the computational improvements and the advantages that can be obtained in terms of performance and classification accuracy. To this end, four widely used images in the field of HSI data classification have been considered: Indian Pines (IP), Big Indian Pines (BIG) and Salinas Valley (SV), collected by AVIRIS, and the University of Pavia (UP) scene,

collected by ROSIS. Figure 4 shows a summary of these datasets, including the number of available labeled data in each case.

The IP, BIP and SV scenes were gathered by AVIRIS sensor [9] in 1992 over agricultural areas and comprise $145 \times 145 \times 200$, $2678 \times 614 \times 220$ and $512 \times 217 \times 204$ samples, respectively, organized in 16 and 58 different land-cover classes. The UP scene was captured by ROSIS [13] over an urban area, comprising $610 \times 340 \times 113$ samples labeled with 9 different classes.

4.2 Experimental environment

Several implementations of the considered RNN-based models for HSI classification have been developed and tested on a hardware environment with a i9-9940X processor, located over an Gigabyte X299 Aorus, 128GB of DDR4 RAM, and an NVIDIA Titan RTX GPU with 24GB GDDR6. In order to provide an efficient implementation, the proposed model (together with the different RNN architectures) has been parallelized on the GPU using CUDA 10.0.130 and cuDNN 7.6.0 language over the Pytorch framework, with Ubuntu 18.04.3 x64 as operating system.



INDIAN PINES (IP)

Color	Land-cover type	Samples
	Background	10776
	Alfalfa	46
	Corn-notill	1428
	Corn-min	830
	Corn	237
	Grass/Pasture	483
	Grass/Trees	730
	Grass/pasture-mowed	28
	Hay-windrowed	478
	Oats	20
	Soybeans-notill	972
	Soybeans-min	2455
	Soybean-clean	593
	Wheat	205
	Woods	1265
	Bldg-Grass-Tree-Drives	386
	Stone-steel towers	93
	Total samples	21025

UNIVERSITY OF PAVIA (UP)

Color	Land-cover type	Samples
	Background	164624
	Asphalt	6631
	Meadows	18649
	Gravel	2099
	Trees	3064
	Painted metal sheets	1345
	Bare Soil	5029
	Bitumen	1330
	Self-Blocking Bricks	3682
	Shadows	947
	Total samples	207400

SALINAS VALLEY (SV)

Color	Land-cover type	Samples
	Background	56975
	Brocoli-green-weeds-1	2009
	Brocoli-green-weeds-2	3726
	Fallow	1976
	Fallow-rough-plow	1394
	Fallow-smooth	2678
	Stubble	3959
	Celery	3579
	Grapes-untrained	11271
	Soil-vinyard-develop	6203
	Corn-senesced-green-weeds	3278
	Lettuce-romaine-4wk	1068
	Lettuce-romaine-5wk	1927
	Lettuce-romaine-6wk	916
	Lettuce-romaine-7wk	1070
	Vinyard-untrained	7268
	Vinyard-vertical-trellis	1807
	Total samples	111104

Fig. 4 Number of available labeled samples in the Indian Pines (IP), University of Pavia (UP), and Salinas Valley (SV) HSI datasets

4.3 Experimental settings

Two main experiments have been conducted to perform an exhaustive analysis of the performance and scalability of the proposed method, as compared with other RNN architectures:

- Our first experiment compares the RNN models with a fixed percentage of training samples. In particular, 15% of randomly selected samples from the IP and BIP scenes and 10% from the UP and SV scenes have been considered. The RNN models have been implemented following the architecture proposed in [21]. The main goal is to study the classification performance of the methods, using standard metrics such as the overall (OA) and average (AA) accuracy, and the kappa coefficient (K). Also run times and number of parameters have been measured. Moreover, two different implementations of vanilla RNN, LSTM and GRU have been considered: (i) direct CUDA implementations and (ii) CuDNN library-based implementations.
- Our second experiment evaluates the scalability and speedup of the proposed model with different sizes of the input features. For this purpose, we have reduced the dimensionality of the scenes (while retaining most of the information in the scenes) using PCA. In particular, for the AVIRIS datasets, in addition to the original 200, 204 and 220 spectral bands, the RNN models have been tested with 50, 100 and 150 principal components while, for the UP scene, 10, 40 and 80 principal components have been, respectively, considered, in addition to the original 103 spectral bands.

4.4 Experimental discussion

4.4.1 Experiment 1: analysis of classification results

The results obtained in our first experiment are reported in Tables 1, 2 and 3. If we focus on the classification accuracy, we can observe that, in general, the CUDA implementations and their CuDNN counterparts reach similar OA, AA and K values, where vanilla RNN usually reaches the lowest accuracies. Comparing these results with those obtained by the SRU-based model, we can conclude that the SRU achieves intermediate results between the vanilla and LSTM/GRU, except for BIP and SV scenes, where it reaches slightly worse results than the vanilla model (the reason for this is that a strong overfitting was observed during the training stage, while the complexity of the BIP's classes is greater than in other datasets). This indicates that the simplicity of the SRU (compared with LSTM and GRU) can negatively affect the overfitting of the network (as it happens also with the vanilla RNN), requiring regularization and standardization mechanisms (such as dropout) in order to reduce this effect. The resulting classification maps can be seen in Fig. 5. As the considered methods are pixel based, they all exhibit some "salt & pepper" noise in the classification results being all very similar.

Table 1 Classification results obtained for the IP dataset, using 15% randomly selected samples to perform the training stage

Class	Vanilla RNN		LSTM		GRU		Proposed
	CUDA	CuDNN	CUDA	CuDNN	CUDA	CuDNN	
1	28.72 ± 7.5	22.05 ± 4.76	38.97 ± 13.51	43.59 ± 9.59	**55.9** ± 6.96	38.97 ± 14.27	35.9 ± 9.32
2	73.82 ± 2.15	70.7 ± 2.24	**78.19** ± 3.33	75.4 ± 3.28	76.57 ± 1.28	75.7 ± 3.34	75.8 ± 2.75
3	55.63 ± 3.86	54.52 ± 4.39	62.21 ± 4.25	**66.18** ± 2.5	64.62 ± 3.59	61.76 ± 2.75	65.11 ± 4.43
4	41.79 ± 7.41	43.28 ± 5.55	55.92 ± 6.51	**56.52** ± 9.64	55.42 ± 4.72	55.32 ± 7.18	54.23 ± 12.67
5	86.1 ± 2.7	84.98 ± 2.08	89.46 ± 3.36	89.46 ± 1.37	86.15 ± 4.23	**90.88** ± 2.83	86.49 ± 2.34
6	95.0 ± 3.03	94.26 ± 1.4	95.48 ± 1.17	96.42 ± 1.71	**97.0** ± 1.68	97.0 ± 0.8	96.97 ± 1.04
7	40.0 ± 12.72	33.91 ± 10.79	61.74 ± 16.36	63.48 ± 2.13	61.74 ± 6.96	66.09 ± 13.01	**68.7** ± 6.39
8	97.98 ± 1.06	97.83 ± 1.7	99.41 ± 0.51	99.46 ± 0.33	**99.8** ± 0.24	99.61 ± 0.25	98.87 ± 1.32
9	10.59 ± 6.86	15.29 ± 6.0	36.47 ± 9.41	25.88 ± 8.8	24.71 ± 14.6	**43.53** ± 17.69	29.41 ± 11.16
10	67.58 ± 2.72	70.0 ± 7.05	73.73 ± 2.76	**75.86** ± 3.33	73.34 ± 2.49	73.34 ± 3.21	74.55 ± 1.25
11	77.16 ± 1.04	76.66 ± 2.75	**82.17** ± 2.22	79.69 ± 1.28	81.11 ± 1.3	80.65 ± 2.74	79.37 ± 1.48
12	65.6 ± 3.94	67.5 ± 8.25	**81.35** ± 3.54	80.99 ± 0.93	75.4 ± 1.31	80.44 ± 5.54	75.28 ± 4.35
13	96.44 ± 3.28	97.24 ± 1.23	98.51 ± 1.07	98.74 ± 0.43	98.51 ± 0.46	98.28 ± 0.89	**99.2** ± 0.46
14	93.6 ± 1.52	94.55 ± 1.3	95.33 ± 2.57	94.98 ± 2.4	95.0 ± 0.98	**95.52** ± 1.38	94.85 ± 1.38
15	63.84 ± 4.24	59.21 ± 4.47	70.73 ± 6.24	70.85 ± 5.26	71.59 ± 3.77	**72.8** ± 2.82	65.18 ± 4.22
16	83.29 ± 4.41	85.06 ± 2.93	90.13 ± 4.03	90.13 ± 5.63	**91.65** ± 4.5	88.1 ± 5.47	88.61 ± 2.12
OA	76.74 ± 0.62	76.3 ± 0.73	**81.9** ± 0.59	81.47 ± 0.55	81.25 ± 0.2	81.33 ± 0.52	80.48 ± 0.47
AA	67.32 ± 1.05	66.69 ± 1.5	75.61 ± 2.33	75.48 ± 1.63	75.53 ± 1.04	**76.12** ± 2.41	74.28 ± 1.62
K(x100)	73.37 ± 0.74	72.86 ± 0.9	**79.29** ± 0.69	78.84 ± 0.63	78.55 ± 0.25	78.66 ± 0.6	77.68 ± 0.52
Parameters	234704		247568		243280		**230928**
Runtime(s)	177.5 ± 0.7	48.42 ± 0.77	170.26 ± 1.45	53.58 ± 1.37	175.9 ± 0.99	49.69 ± 1.54	**29.9** ± 0.23
Speedup	1.0	3.67	1.04	3.31	1.01	3.57	**5.94**

The best metric values are in bold

Table 2 Classification results obtained for the UP dataset, using 10% randomly selected samples to perform the training stage

Class	Vanilla RNN		LSTM		GRU		Proposed
	CUDA	CuDNN	CUDA	CuDNN	CUDA	CuDNN	
1	92.69 ± 1.79	91.47 ± 0.97	93.35 ± 1.09	**94.29** ± 0.93	94.04 ± 0.6	94.1 ± 1.12	93.67 ± 0.95
2	97.67 ± 0.44	97.22 ± 0.5	**97.99** ± 0.28	97.55 ± 0.43	97.92 ± 0.36	97.43 ± 0.54	97.56 ± 0.33
3	72.08 ± 4.46	72.9 ± 2.86	**78.73** ± 3.49	76.35 ± 1.71	77.81 ± 1.39	76.0 ± 3.18	76.83 ± 0.96
4	92.76 ± 1.5	93.46 ± 1.09	93.46 ± 0.64	94.86 ± 0.44	**94.89** ± 1.14	94.53 ± 1.51	93.95 ± 1.23
5	99.69 ± 0.06	99.6 ± 0.1	99.65 ± 0.19	99.8 ± 0.13	99.8 ± 0.15	**99.87** ± 0.11	99.62 ± 0.17
6	89.16 ± 0.85	**90.99** ± 1.39	89.32 ± 0.84	90.22 ± 1.78	89.21 ± 1.93	90.1 ± 1.86	89.38 ± 1.45
7	81.64 ± 7.1	**86.6** ± 5.41	86.52 ± 1.82	83.59 ± 5.03	85.63 ± 4.44	86.55 ± 2.58	83.76 ± 1.27
8	85.33 ± 3.13	85.46 ± 1.52	88.29 ± 1.41	88.03 ± 1.9	87.67 ± 2.26	**88.4** ± 2.19	87.2 ± 1.89
9	99.79 ± 0.11	**99.93** ± 0.06	99.69 ± 0.22	99.67 ± 0.38	99.6 ± 0.06	99.81 ± 0.16	99.86 ± 0.14
OA	92.84 ± 0.32	92.93 ± 0.09	93.88 ± 0.2	93.81 ± 0.06	**93.92** ± 0.06	93.81 ± 0.12	93.51 ± 0.1
AA	90.09 ± 0.98	90.85 ± 0.64	**91.89** ± 0.32	91.59 ± 0.45	91.84 ± 0.46	91.87 ± 0.5	91.31 ± 0.23
K(x100)	90.48 ± 0.41	90.61 ± 0.12	91.86 ± 0.26	91.78 ± 0.08	**91.92** ± 0.09	91.77 ± 0.17	91.38 ± 0.13
Parameters	76809		89673		85385		**73033**
Runtime(s)	351.9 ± 1.33	109.47 ± 0.91	303.16 ± 2.02	118.22 ± 0.61	295.75 ± 1.19	110.6 ± 1.39	**57.73** ± 0.21
Speedup	1.0	3.21	1.16	2.98	1.19	3.18	6.1

The best metric values are in bold

Table 3 Classification results obtained for the SV dataset, using 10% randomly selected samples to perform the training stage

Class	Vanilla RNN		LSTM		GRU		Proposed
	CUDA	CuDNN	CUDA	CuDNN	CUDA	CuDNN	
1	99.63 ± 0.09	99.69 ± 0.42	99.83 ± 0.09	99.63 ± 0.45	99.91 ± 0.09	**99.97** ± 0.07	99.7 ± 0.23
2	99.97 ± 0.03	99.88 ± 0.18	99.83 ± 0.15	99.98 ± 0.03	99.97 ± 0.03	**99.98** ± 0.01	99.96 ± 0.03
3	97.46 ± 1.49	98.19 ± 0.45	**99.73** ± 0.14	99.46 ± 0.33	98.66 ± 1.21	99.56 ± 0.37	98.82 ± 0.58
4	98.69 ± 0.62	99.17 ± 0.28	**99.44** ± 0.24	99.3 ± 0.24	98.87 ± 0.35	99.23 ± 0.26	99.31 ± 0.44
5	98.38 ± 0.53	98.24 ± 0.57	99.05 ± 0.41	99.15 ± 0.49	99.04 ± 0.53	**99.27** ± 0.34	98.74 ± 0.48
6	99.87 ± 0.1	99.81 ± 0.12	**99.9** ± 0.07	99.86 ± 0.09	99.88 ± 0.08	99.89 ± 0.1	99.89 ± 0.09
7	99.78 ± 0.14	99.69 ± 0.17	99.75 ± 0.1	**99.84** ± 0.2	99.79 ± 0.1	99.7 ± 0.11	99.76 ± 0.11
8	88.36 ± 1.04	89.42 ± 2.27	88.7 ± 1.98	**89.8** ± 0.97	88.9 ± 1.76	89.34 ± 0.6	88.33 ± 1.6
9	99.81 ± 0.14	99.8 ± 0.11	99.72 ± 0.2	99.83 ± 0.11	99.91 ± 0.1	**99.98** ± 0.01	99.86 ± 0.08
10	96.01 ± 0.43	96.17 ± 0.92	97.57 ± 0.32	97.57 ± 0.51	97.77 ± 0.49	**97.9** ± 0.23	96.61 ± 0.77
11	98.09 ± 0.75	96.75 ± 0.38	97.77 ± 1.84	98.06 ± 1.13	**99.33** ± 0.25	97.86 ± 1.08	98.38 ± 0.47
12	99.53 ± 0.31	99.08 ± 0.91	**99.88** ± 0.1	99.7 ± 0.47	99.64 ± 0.21	99.86 ± 0.11	99.71 ± 0.2
13	98.71 ± 0.76	98.54 ± 0.43	**98.88** ± 0.75	98.42 ± 0.92	98.86 ± 0.69	98.81 ± 0.49	98.76 ± 0.44
14	96.74 ± 1.14	97.09 ± 1.58	97.11 ± 1.77	97.63 ± 1.06	97.45 ± 0.71	**98.09** ± 0.89	96.91 ± 1.14
15	73.31 ± 1.12	69.99 ± 5.36	76.94 ± 3.22	75.72 ± 0.62	**77.62** ± 2.16	77.32 ± 0.99	69.94 ± 2.63
16	99.25 ± 0.32	**99.32** ± 0.28	99.07 ± 0.35	99.09 ± 0.38	99.07 ± 0.57	99.19 ± 0.61	99.03 ± 0.3
OA	93.32 ± 0.21	93.08 ± 0.31	94.1 ± 0.22	94.18 ± 0.18	94.26 ± 0.25	**94.37** ± 0.06	93.0 ± 0.09
AA	96.47 ± 0.11	96.3 ± 0.28	97.07 ± 0.24	97.07 ± 0.05	97.17 ± 0.12	**97.25** ± 0.12	96.48 ± 0.18
K(x100)	92.55 ± 0.23	92.29 ± 0.35	93.43 ± 0.24	93.51 ± 0.2	93.6 ± 0.28	**93.73** ± 0.07	92.19 ± 0.1
Parameters	239312		252176		247888		**235536**
Runtime(s)	564.77 ± 1.9	199.83 ± 2.09	552.8 ± 1.88	216.65 ± 1.54	567.29 ± 1.93	198.25 ± 1.51	**74.13** ± 0.31
Speedup	1.0	2.84	1.03	2.62	1.0	2.86	7.65

The best metric values are in bold

Focusing on the number of parameters and run times, we observe that both the CUDA and CuDNN versions require the same number of parameters. However, their run times are significantly different, being CuDNN faster. With this in mind, our model exhibits the lowest number parameters, resulting in less computational requirements, lower memory consumption and fast performance. In fact, the SRU is able to outperform all the optimized models in CuDNN. In order to analyze the speedup achieved by these methods, we first note that the slowest one is the most basic model (i.e., the vanilla RNN directly implemented in CUDA). Further, the CUDA versions of LSTM and GRU achieve low speedups, while the CuDNN counterparts achieve an approximate speedup of x3 for the IP, BIP and UP dataset, and x2 for the SV scene. However, the speedups achieved by the proposed method are close to x5, x6 and x7 for the IP, UP and SV scenes, respectively, and more than x10 for the BIP. This shows the improved computational performance (and scalability with size) of the proposed SRU implementation.

4.4.2 Experiment 2: speedup and scalability

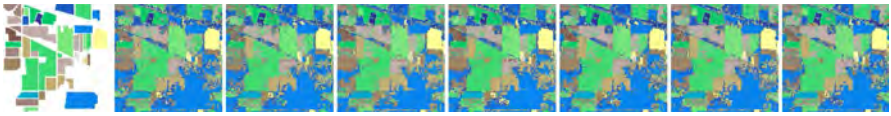
The results of our second experiment are reported on the last rows of Tables 1, 2 and 3. Once more, the CUDA implementation of vanilla RNN was the slowest model, and we used it as a baseline to evaluate the other implementations. If we focus on the CUDA LSTM and GRU, their speedup never exceeds x1.20 and even decreases when more input features are used (e.g., for the IP and SV scenes). In turn, the CuDNN-based models exhibit speedup values of x2 and x3, being the CuDNN-based vanilla model the one with the highest speedup (closely followed by the GRU network). However, these implementations do not scale well when additional input features are considered, even reducing their speedup in some cases (e.g., for the SV scene). In this sense, the proposed method not only reaches the highest speedup values, but the associated speedups always increase with the number of input features (for instance in BIP) (Table 4).

Figure 6 shows a graphical representation of the runtimes measured (in s) for the different tested methods, as a function of the number of input features (spectral bands). As we can observe, the CUDA versions of vanilla, LSTM and GRU are clearly the slowest methods for the considered HSI datasets. These models are all negatively affected by

Table 4 Classification results obtained for the BIP dataset, using 15% randomly selected samples. Due to space limitations, the accuracy of each class is not shown

Class	Vanilla RNN		LSTM		GRU		Proposed
	CUDA	CuDNN	CUDA	CuDNN	CUDA	CuDNN	
OA	57.9 ± 0.49	58.12 ± 0.33	60.71 ± 1.39	60.28 ± 0.51	**61.6** ± 0.33	61.38 ± 0.64	56.73 ± 0.22
AA	46.4 ± 0.58	46.09 ± 0.43	49.16 ± 1.14	48.31 ± 1.1	**50.83** ± 0.68	49.47 ± 0.93	43.42 ± 0.55
K(x100)	54.34 ± 0.53	54.57 ± 0.34	57.42 ± 1.51	56.92 ± 0.54	**58.42** ± 0.4	58.1 ± 0.69	53.04 ± 0.21
Parameters	849146		862010		857722		**845370**
Runtime(s)	5327.74 ± 107.52	1759.85 ± 107.58	5076.23 ± 77.47	1934.94 ± 68.97	5180.01 ± 52.87	1745.2 ± 89.31	**508.19** ± 56.77
Speedup	1.0	3.03	1.05	2.75	1.03	3.05	**10.48**

The best metric values are in bold



(a) Ground-truth **(b)** V-RNN CUDA **(c)** V-RNN CuDNN **(d)** LSTM CUDA **(e)** LSTM CuDNN **(f)** GRU CUDA **(g)** GRU CuDNN **(h)** Proposed

Fig. 5 Classification maps of IP scene, using 15% randomly selected samples to perform the training stage

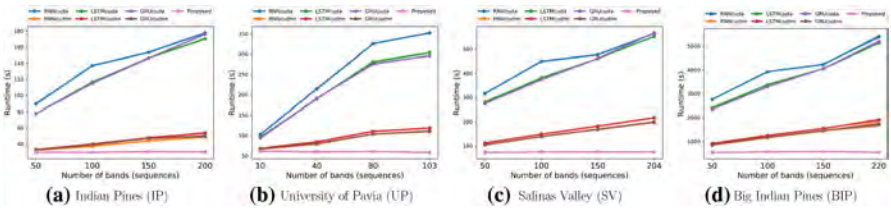


Fig. 6 Graphical representation of the runtimes (in s) measured for the different tested methods, as a function of the number of input features (spectral bands)

the increase in the number of input features (i.e., the runtimes increase significantly with the number of input features). However, the CuDNN counterparts exhibit better computational performance and scalability than CUDA versions when the number of input features increases. Finally, the proposed SRU-based model clearly exhibits the lowest runtimes (which do not increase with the number of features), thanks to the optimal parallelization of its point-wise operations. As a result, our method is not affected by the inclusion of additional input features, scaling significantly better than the other tested methods.

5 Conclusions and future work

In this paper, a new RNN model for HSI data classification is presented and discussed. The proposed model is based on the SRU architecture, which reduces the internal complexity of other previously developed recurrent approaches (i.e., LSTM and GRU) by decoupling the computational relationship between the current and previous states in the network architecture. This is achieved by resorting to easily parallelizable, point-wise operations. Our experiments, conducted using four benchmark HSI datasets, reveal that our method is able to achieve good classification results using much fewer parameters than the traditional models (vanilla, LSTM and GRU), therefore consuming less memory. Moreover, our parallelization strategy significantly reduces the measured runtimes of the proposed method, which obtains higher speedups as the number of pixels (and the dimensionality of input features) in the HSI increase.

As future work, we will study the inclusion of standardization and regularization methods to reduce the overfitting of the proposed model, with the goal of further improving the reliability and precision of the obtained classification results.

Acknowledgements This study was supported by Spanish Ministry (FPU14/02012-FPU15/02090, ESP2016-79503-C2-2-P), FEDER and Junta de Extremadura (GR18060), and the European Union (734541-EXPOSURE).

References

1. Appleyard J, Kocisky T, Blunsom P (2016) Optimizing performance of recurrent neural networks on gpus. arXiv:1604.01946
2. Campos V, Jou B, Giró-i Nieto X, Torres J, Chang SF (2017) Skip rnn: learning to skip state updates in recurrent neural networks. arXiv:1708.06834
3. Chen Y, Lin Z, Zhao X, Wang G, Gu Y (2014) Deep learning-based classification of hyperspectral data. IEEE J Select Top Appl Earth Obs Remote Sens 7(6):2094–2107
4. Cho K, Van Merriënboer B, Bahdanau D, Bengio Y (2014) On the properties of neural machine translation: encoder-decoder approaches. arXiv:1409.1259
5. Cudahy T, Hewson R, Huntington J, Quigley M, Barry P (2001) The performance of the satellite-borne hyperion hyperspectral VNIR-SWIR imaging system for mineral mapping at Mount Fitton, South Australia. In: IGARSS 2001. Scanning the present and resolving the future. Proceedings. IEEE 2001 international geoscience and remote sensing symposium (Cat. No. 01CH37217). IEEE, vol 1, pp 314–316
6. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2(4):303–314
7. Gevaert CM, Suomalainen J, Tang J, Kooistra L (2015) Generation of spectral-temporal response surfaces by combining multispectral satellite and hyperspectral UAV imagery for precision agriculture applications. IEEE J Select Top Appl Earth Obs Remote Sens 8(6):3140–3146
8. Goetz AF, Vane G, Solomon JE, Rock BN (1985) Imaging spectrometry for earth remote sensing. Science 228(4704):1147–1153
9. Green RO, Eastwood ML, Sarture CM, Chrien TG, Aronsson M, Chippendale BJ, Faust JA, Pavri BE, Chovit CJ, Solis M, Olah MR, Williams O (1998) Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS). Remote Sens Environ 65(3):227–248
10. Hochreiter S (1998) Recurrent neural net learning and vanishing gradient. Int J Uncertain Fuzziness Knowl-Based Syst 6(2):107–116
11. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
12. Kumar DN, Reshmidevi T (2013) Remote sensing applications in water resources. J Indian Inst Sci 93(2):163–188
13. Kunkel B, Blechinger F, Lutz R, Doerffer R, van der Piepen H, Schroder M (1988) ROSIS (Reflective Optics System Imaging Spectrometer)—A candidate instrument for polar platform missions. In: Seeley J, Bowyer S (eds) Proceedings of SPIE 0868 optoelectronic technologies for remote sensing from space, p 8 . https://doi.org/10.1117/12.943611
14. Landgrebe D (2002) Hyperspectral image data analysis. IEEE Signal Process Mag 19(1):17–28
15. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436
16. Lei T, Zhang Y, Wang SI, Dai H, Artzi Y (2018) Simple recurrent units for highly parallelizable recurrence. In: Proceedings of the 2018 conference on empirical methods in natural language processing, Brussels, Belgium, October 31–November 4, 2018, pp 4470–4481. https://doi.org/10.18653/v1/d18-1477
17. Li B, Zhou E, Huang B, Duan J, Wang Y, Xu N, Zhang J, Yang H (2014) Large scale recurrent neural network on gpu. In: 2014 International Joint Conference on Neural Networks (IJCNN). IEEE, pp 4062–4069
18. Li Z, Chen J, Baltsavias E (2008) Advances in photogrammetry, remote sensing and spatial information sciences: 2008 ISPRS congress book, vol 7. CRC Press, Boca Raton
19. Manolakis D, Shaw G (2002) Detection algorithms for hyperspectral imaging applications. IEEE Signal Process Mag 19(1):29–43
20. Meerdink SK, Roberts DA, Roth KL, King JY, Gader PD, Koltunov A (2019) Classifying California plant species temporally using airborne hyperspectral imagery. Remote Sens Environ 232:111308

21. Mou L, Ghamisi P, Zhu XX (2017) Deep recurrent neural networks for hyperspectral image classification. IEEE Trans Geosci Remote Sens 55(7):3639–3655
22. Murphy RJ, Monteiro ST, Schneider S (2012) Evaluating classification techniques for mapping vertical geology using field-based hyperspectral sensors. IEEE Trans Geosci Remote Sens 50(8):3066–3080
23. Nurvitadhi E, Sim J, Sheffield D, Mishra A, Krishnan S, Marr D (2016) Accelerating recurrent neural networks in analytics servers: comparison of FPGA, CPU, GPU, and ASIC. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL). IEEE, pp 1–4
24. Paoletti ME, Haut JM, Plaza J, Plaza A (2019) Deep learning classifiers for hyperspectral imaging: a review. ISPRS J Photogramm Remote Sens 158:279–317. https://doi.org/10.1016/j.isprsjprs.2019.09.006
25. Plaza A, Du Q, Chang YL, King RL (2011) High performance computing for hyperspectral remote sensing. IEEE J Select Top Appl Earth Obs Remote Sens 4(3):528–544
26. Srivastava RK, Greff K, Schmidhuber J (2015) Training very deep networks. In: Advances in neural information processing systems, pp 2377–2385
27. Weber C, Aguejdad R, Briottet X, Avala J, Fabre S, Demuynck J, Zenou E, Deville Y, Karoui MS, Benhalouche FZ et al (2018) Hyperspectral imagery for environmental urban planning. In: IGARSS 2018-2018 IEEE international geoscience and remote sensing symposium. IEEE, pp 1628–1631
28. Weninger F, Bergmann J, Schuller B (2015) Introducing currennt: the Munich open-source CUDA recurrent neural network toolkit. J Mach Learn Res 16(1):547–551
29. Williams RJ, Zipser D (1989) A learning algorithm for continually running fully recurrent neural networks. Neural Comput 1(2):270–280
30. Wold S, Esbensen K, Geladi P (1987) Principal component analysis. Chemom Intell Lab Syst 2(1–3):37–52
31. Xu Y, Zhang L, Du B, Zhang F (2018) Spectral–spatial unified networks for hyperspectral image classification. IEEE Trans Geosci Remote Sens 56(10):5893–5909
32. Yang X, Ye Y, Li X, Lau RY, Zhang X, Huang X (2018) Hyperspectral image classification with deep learning models. IEEE Trans Geosci Remote Sens 56(9):5408–5423
33. Zhang X, Sun Y, Jiang K, Li C, Jiao L, Zhou H (2018) Spatial sequential recurrent neural network for hyperspectral image classification. IEEE J Select Top Appl Earth Obs Remote Sens 11(11):4141–4155
34. Zhou F, Hang R, Liu Q, Yuan X (2019) Hyperspectral image classification using spectral-spatial LSTMs. Neurocomputing 328:39–47