IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 14 February 2023, accepted 7 March 2023, date of publication 9 March 2023, date of current version 15 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3255164

RESEARCH ARTICLE

Hyperspectral Image Classification: An Analysis Employing CNN, LSTM, Transformer, and Attention Mechanism

FELIPE VIEL[1,2], (Student Member, IEEE), RENATO COTRIM MACIEL[3], LAIO ORIEL SEMAN[1,2], CESAR ALBENES ZEFERINO[1], (Member, IEEE), EDUARDO AUGUSTO BEZERRA[2], (Member, IEEE), AND VALDERI REIS QUIETINHO LEITHARDT[4,5], (Senior Member, IEEE)

[1]Laboratory of Embedded and Distributed Systems, University of Vale do Itajaí, Itajaí 88302-901, Brazil
[2]Space Technology Research Laboratory (SpaceLab), Federal University of Santa Catarina, Florianopolis 88040-370, Brazil
[3]Department of Production and Systems Engineering, Federal University of Santa Catarina, Florianopolis 88040-370, Brazil
[4]COPELABS, Lusófona University of Humanities and Technologies, 1749-024 Lisbon, Portugal
[5]VALORIZA, Research Center for Endogenous Resources Valorization, Instituto Politécnico de Portalegre, 7300-555 Portalegre, Portugal

Corresponding author: Felipe Viel (viel@univali.br)

This work was supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnologico (CNPq), Agência Espacial Brasileira (AEB), under Grant 436.982/2018-8, Grant 313.513/2021-0, Grant 140368/2021-3, and Grant 308361/2022-9; in part by the National Funds through Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) by the Project "VALORIZA—Research Center for Endogenous Resource Valorization" under Grant UIDB/05064/2020; and in part by Fundação para a Ciência e a Tecnologia under Project UIDB/04111/2020.

ABSTRACT Hyperspectral images contain tens to hundreds of bands, implying a high spectral resolution. This high spectral resolution allows for obtaining a precise signature of structures and compounds that make up the captured scene. Among the types of processing that may be applied to Hyperspectral Images, classification using machine learning models stands out. The classification process is one of the most relevant steps for this type of image. It can extract information using spatial and spectral information and spatial-spectral fusion. Artificial Neural Network models have been gaining prominence among existing classification techniques. They can be applied to data with one, two, or three dimensions. Given the above, this work evaluates Convolutional Neural Network models with one, two, and three dimensions to identify the impact of classifying Hyperspectral Images with different types of convolution. We also expand the comparison to Recurrent Neural Network models, Attention Mechanism, and the Transformer architecture.. Furthermore, a novelty pre-processing method is proposed for the classification process to avoid generating data leaks between training, validation, and testing data. The results demonstrated that using 1 Dimension Convolutional Neural Network (1D-CNN), Long Short-Term Memory (LSTM), and Transformer architectures reduces memory consumption and sample processing time and maintain a satisfactory classification performance up to 99% accuracy on larger datasets. In addition, the Transfomer architecture can approach the 2D-CNN and 3D-CNN architectures in accuracy using only spectral information. The results also show that using two or three dimensions convolution layers improves accuracy at the cost of greater memory consumption and processing time per sample. Furthermore, the pre-processing methodology guarantees the disassociation of training and testing data.

INDEX TERMS Hyperspectral imaging, CNN, LSTM, transformer, remote sensing.

I. INTRODUCTION

Hyperspectral imaging is one of the several methods for capturing data from a scene in remote sensing systems. A Hyperspectral Image (HSI) is a 3D data cube composed of 2D images that represent a scene at different electromagnetic wavelengths (bands) [1], [2], [3], [4], [5]. In HSIs, electromagnetic wave spectra captured by sensors range from tens to hundreds, depending on the sensor used. Furthermore, the spectrum ranges from visible (400 nm) to infrared (2500 nm)

The associate editor coordinating the review of this manuscript and approving it for publication was Qiangqiang Yuan.

wavelengths, with a nominal spectral resolution of 10 nm [1], [4]. Thus, an HSI is an image with a high spectral resolution that provides more information about a single point than other types of image used in remote sensing [6], [7], [8], [9].

The large amount of data in HSIs produces practical implications, such as identifying compounds and materials in the scene. This data volume also has implications against computational restrictions in sensing platforms, such as limited memory and processing. The relationship between processing large volumes of data and space applications, especially with onboard processing, has been the subject of several investigations in recent years [10], [11], [12], [13]. This issue mainly affects the use of more complex onboard computers (OBC) in spacecraft, which have been the subject of study and constant evolution in various parts of their architecture [10].

One of the main processes applied to HSIs is classification, which mainly benefits from the high spectral resolution of these images. This high resolution creates a unique spectral signature at each point (pixel) of the scene, which allows the extraction of which compounds are in the scene [14]. Among the models used to classify HSI, the Convolutional Neural Network (CNN) models stand out. The CNN models process the data in various ways and can be one-dimensional, two-dimensional, or three-dimensional. Using one or more dimensions directly impacts processing and computational resource requirements, such as the memory used by the CNN and the model inference time [15], [16], [17].

CNN is a type of artificial neural network that is particularly well-suited for analyzing and processing data with a grid-like structure, such as images. A CNN consists of a series of hidden layers, each one performing a convolution operation on the input data. The convolution operation involves applying a set of weights (also called filters) to a small region of the input data and producing a transformed output. The weights are learned through training and are used to extract features from the input data. The transformed output is then passed through a nonlinear activation function and is used as input to the next hidden layer [15], [18].

One of the key benefits of CNNs is their ability to learn hierarchical features from the input data. Multiple convolutional layers achieve this learning, and each layer can learn more complex features based on the features learned by the previous layer. Additionally, CNNs can process data with a large spatial size (such as an image) much more efficiently than fully connected neural networks, which require many weights to process the data. This property makes CNNs particularly well-suited for tasks such as image classification and object detection [18].

One Neural Network architecture that compares to 1D-CNN is the Recurrent Neural Network (RNN). RNNs are particularly useful for processing sequential data because they allow the model to incorporate information from previous time steps. This approach contrasts traditional feedforward neural networks, which process input data independently and do not consider any information from previous time steps. One of the key advantages of RNNs is that they can process data of variable length, unlike feedforward neural networks, which require fixed-length input. This characteristic makes RNNs useful for tasks such as natural language processing, as text length can vary significantly. RNNs can be unrolled in time, meaning that the same network architecture is applied to each time step in the sequence. This property allows the network to learn long-term dependencies between time steps, which is important for language translation or speech recognition tasks. There are several variants of RNNs, and one of the most important is Long Short-Term Memory (LSTM) [17], [18], [19].

Transformer is an architecture that has been gaining popularity for 1-dimensional data applications and replacing RNN [20]. Generally, natural language processing (NLP) processing applies the Transformer architecture, including machine translation, language modeling, and text summarization. However, Transformer architecture can also be employed for time series processing. Using the Transformer design, image processing operations can also be accomplished. The processing of visual features utilizing a 2D self-attention mechanism, as opposed to the 1D self-attention mechanism used in NLP tasks, is a standard method. This method allows the network to dynamically weigh the importance of different visual regions when making predictions. The Transformer architecture is a versatile method for processing grid-like and sequential data. It has shown promising performance in various image processing applications, such as image classification, segmentation, and generative modeling [20], [21], [22].

Works in the literature report the classification of HSIs with CNN. Among the existing literature, some solutions address comparisons between 1D-CNN and 2D-CNN approaches applied to HSI with their own architectures [15], [23], [24], [25]. Other works expand the comparison or integration of CNNs, including 1D, 2D, and 3D architectures for images originated from remote sensings, such as [26], [27], [28], [29], and [30]. Works such as [14], [31], [32], [33], [34], [35], [36], and [37] compare distinct architectures focusing mainly on spatial information from remote sensing images, including HSIs. The authors of these works propose approaches or CNN architectures that fuse spatial information with spectral information. Another point worth highlighting and relevant to CNN is how to generate and split the training and testing dataset. The works of [38], [39], [40], [41], [42], and [43] show different techniques to split the dataset or demonstrate techniques for data augmentation to present a better balance and dissociation between training and testing data.

Other works explore and combine RNN architectures for HSI classification, including LSTM and Gated Recurrent Unit (GRU). The authors of [44] propose an architecture combining a Dense Connected Convolutional Network with

a bidirectional RNN with an attention mechanism network. In [45], a bidirectional-convolutional long short-term memory architecture is proposed that explores spatial-spectral features for HSI classification. The work of [46] presents two RNN architectures using LSTM and GRU for HSI. Also, it proposes a new activation function called Parametric Rectified Tanh (PRetanh) focused on RNNs. Finally, in [47], a combination of convolutional layers with GRU-type RNN layers is also presented to combine spatial-spectral features present in HSI for more robust classification.

The Transformer architecture is already being applied to the classification of HSIs with an option for the CNN and RNN architectures. The works [48], [49], [50], [51] classify HSIs using a spectral-spatial approach, using steps to extract spatial information and using the spectral signature, and applying the concept of Vision Transformer (ViT). There are authors, as [52], [53], and [54], that unite features of CNN and LSTM architecture with Transformer for HSI classification. Some works like [55] and [56] also present an extensive comparison of the transformer architecture with other types of architectures, comparing approaches using spectral and spatial information.

In this context, the main contribution of the present work relies on:

- Evaluating three types of CNN (1D, 2D, and 3D), considering performance, processing time, and memory consumption metrics. This evaluation identifies which models concomitantly balance accuracy, resource consumption (required memory and number of parameters), and processing time.
- An extensive comparison among 1D-CNN, LSTM, and Transformer architectures for classifying HSIs using only spectral information. The comparison is also expanded, adding Attention Mechanism Layer in 1D-CNN and LSTM architectures.
- A proposed pre-processing methodology that performs a total dissociation of training and testing data, avoiding poor generalization and metrics overestimation to 2D and 3D architectures.
- Demonstration that 1D architectures are faster, require less memory, and can achieve 99% accuracy. We also demonstrate a correlation between the size of the training dataset and the accuracy that allows the 1D architectures to be equivalent to the 2D architecture and the 3D architectures of reference in the literature, even using less information.

The remainder of this work is structured as follows. Section II describes the materials and methods used in the work development, providing details about the implementation and verification steps, with the main point being the methodology used to split the training and testing data in the pre-processing stage. Section III presents the CNN architectures, Section IV presents the LSTM architectures, and Section V presents the Transformer architecture used and explored in work. Section VI discusses the results obtained and analyzed. Concluding, Section VII presents the final remarks.

II. MATERIALS AND METHODS

A. DATASET

The datasets used are literature benchmarks available in [57]. Those datasets represent aerial views captured by three distinct sensors, NASA/JPL AVIRIS (Airborne Visible/Infrared Imaging Spectrometer), Hyperion on NASA Earth Observation 1 (EO-1), and the ROSIS (Reflective Optics System Imaging Spectrometer) sensor. The HSIs used as datasets and benchmarks were Indian Pines, the University of Pavia, the Center of Pavia, Kennedy Space Center, Botswana, and Salinas Valley. The AVIRIS sensor captured Indian Pines, Salinas Valley, and Kennedy Space Center images. The ROSIS sensor captured the University of Pavia and the Center of Pavia scenes. In contrast, the Hyperion sensor on NASA EO-1 satellite captured the Botswana scene [57].

Indian Pines (IP), Salinas Valley (SA), and Kennedy Space Center (KC) scenes have 224 spectral bands covering the electromagnetic spectrum from 400 to 2500 nm. However, 24 bands are removed because they are in water absorption regions for IP and SA. For the HSI KC, 176 bands were utilized for the analysis after water absorption, and low Signal-to-Noise Rate (SNR) bands were eliminated. The IP scene has a spatial dimension of 145 $\times$ 145 pixels and comprises two-thirds agriculture and one-third forest or other perennial natural vegetation. This scene has some of the corn and soybean crops in the early stages of growth, two main dual-lane highways, a railway line, some low-density housing, other built structures, and more minor roads. This HSI has 16 previously mapped classes in an 145 $\times$ 145$\times$224 hypercube. The SA HSI represents the region of the same name in California, with a spatial resolution of 512 $\times$ 217 pixels and the hypercube being 512$\times$217$\times$224. This HSI includes vegetation, bare soil, and vineyards, with 16 mapped classes. The KC scene 512 $\times$ 614 pixels comprises different spectral fingerprints of vegetation species in 13 classes representing the various land cover types that occur in this ecosystem [57].

The Botswana (BT) scene has a spatial resolution of 1476$\times$ 256 pixels and 242 bands that covers the 400-2500 nm region of the spectrum in 10 nm windows, representing the region Okavango Delta in Botswana. After preprocessing to reduce the impacts of subpar detectors, inter-detector miscalibration, and sporadic abnormalities, only 145 bands remain. Their observations from 14 recognized classes represent the various forms of land cover in the distal seasonal wetlands, occasional swamps, and dry forests. Finally, the University of Pavia (UP) and Center of Pavia (CP) scenes also represent the region of Pavia in Italy. CP has a spatial resolution of 1096 $\times$ 1096 pixels and 102 bands, with the hypercube being 1096 $\times$ 1096$\times$102. UP has a spatial resolution of 610$\times$610 pixels and 103 spectral bands, with the hypercube being 610$\times$610$\times$103. These HSIs are previously mapped and divided into nine classes ranging from asphalt to bare soil [57].

B. DEVELOPMENT PLATFORM AND LANGUAGE

The development of the machine learning models was performed using Python version 3.8, with the NumPy, Keras, TensorFlow, Spectral, and Time libraries. Keras is a high-level user-friendly Application Programming Interface (API) running on top of TensorFlow, enabling the development of machine learning models with great ease. In contrast, TensorFlow is a robust library focused on DNN training, inference, and serving. NumPy library is used to manipulate arrays and create tensors for CNN models. Spectral is employed to manipulate the HSIs used as the dataset. Finally, the Time library enables the capture of execution time from the prediction.

We used the Google Collaboratory Pro cloud service platform for training, inference, and evaluation. Three configurations were used: CPU only, CPU+NVIDIA Tesla T4 GPU, and CPU+NVIDIA A100 GPU. The CPU-only infrastructure offers three Intel Xeons running at 2.2 GHz, 25 GB of main memory, and 225 GB of disk. The CPU+NVIDIA Tesla T4 GPU infrastructure offers three Intel Xeons at 2.2 GHz, 25 GB of main memory, 166 GB of disk, and an NVIDIA Tesla T4 GPU with 15 GB of memory. Finally, The CPU+NVIDIA A100 GPU infrastructure offers a computational infrastructure with an Intel Xeon CPU with 2.2 GHz of operating frequency and 11 processors, 83.78 GB of main memory, 166 GB of disk storage, and an NVIDIA A100-SXM4 offboard graphics card with 40 GB of memory. This setup enables the execution of experiments with high performance, mainly due to the use of a dedicated graphics card.

C. BASELINE, DATA PREPROCESSING, AND EVALUATION METRICS

The CNN architectures used [14] as a baseline. Also, a distinct HSI dataset-splitting methodology is proposed. The purpose is to enable splits that do not contaminate training data in testing data and testing data in training data. This splitting methodology aims to improve the classification process, avoiding over-optimistic model metrics and resulting in increased confidence in the precision results displayed in [14]. In [14], the authors use a data split methodology for training and testing that, when creating the spatial-spectral cubes, causes information leakage (data leakage) between training, validation, and testing sets. This contamination causes a high degree of distrust regarding the macro precision of the prediction in the testing set on production in real-world applications.

This proposed data division process splits the HSI into even and odd rows, with the even rows for training and the odd rows for testing. After this row division, it generates a column division. After this splitting, the even rows and columns are allocated for training and the odd rows and columns are allocated for testing. Figure 1 illustrates the proposed data division methodology. It is important to note that 1D-CNN makes a pixel classification, and this methodology is applied to maintain a comparison between the three architectures.

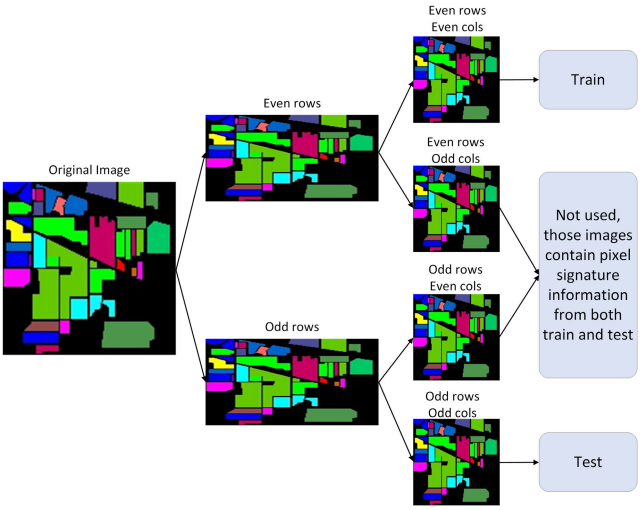


FIGURE 1. Proposed dataset-splitting approach to avoid contamination between training and testing datasets.

However, the *train_test_split* function from the scikit-learn library was also used.

In addition, to achieve efficient training and classification regarding memory requirements, a transformation using the PCA (Principal Component Analysis) algorithm is applied to reduce the number of bands of the HSIs used. The basic idea behind PCA is to compress the original data into a lower-dimensional set of variables, called principal components (PCs), that are orthogonal (i.e., uncorrelated) and are ranked by their importance in explaining the variance in the data. The first principal component (PC1) is the direction in the data that captures the greatest amount of variance, the second principal component (PC2) is the direction that captures the second greatest amount of variance, and so on. The PCA transformation is applied to the training and testing sets [58]. Exclusively for the 1D-CNN with the Indian Pines image, class balancing was applied for the architecture based in [14]. The class balancing aimed to allow equal training for the IP HSI due to this HSI having classes with almost 2,000 samples and classes with less than ten samples during training. In addition, a validation set was applied to the architectures to observe a prediction result without compromising the testing set.

The metrics used to extract results were the Overall Accuracy (OA), Average Accuracy (AA), Cohen Kappa Score (kappa), prediction time, and the amount of memory used by the model. The Python libraries Time (prediction time), Keras (model size in bytes), and scikit-learn (model accuracy) were used in the results extraction. The OA will measure the number of correctly classified samples out of the total samples. Kappa is a statistical measurement metric that provides mutual information about the strong agreement between the ground truth map and the classification map. AA represents the mean of the classwise classification accuracy. The equations that define each metric are:

$$OA = \frac{tp + tn}{tp + tn + fp + fn} \tag{1}$$

$$AA = \frac{\sum_{i=1}^{n_{\text{classes}}} AccuaracyClass_i}{n_{\text{classes}}} \quad (2)$$

$$Kappa = \frac{p_o - p_e}{1 - p_e} \quad (3)$$

where tp are true positives, tn are true negatives, fp are false positives, fn are false negatives, $n_classes$ indicates the number of classes present in the HSI, $AccuracyClass_i$ indicates the individual accuracy of the class. In Kappa, p_o is the observed relative agreement between raters (identical to precision) and p_e is the hypothetical probability of chance agreement, using the observed data to calculate the probability that each observer randomly sees each category.

III. CNN ARCHITECTURE

The basic idea behind convolution is to take a small matrix of numbers (called a "kernel" or "filter") and slide it over the input data, performing an element-wise multiplication between the entries in the kernel and the input data and summing the results. This process is repeated for each position of the kernel, resulting in a new feature map matrix. The process of sliding the kernel over the input data and performing the element-wise multiplication and summing is called a convolutional operation. Applying a kernel to an input matrix is called a convolutional layer in a CNN. A CNN typically consists of multiple convolutional layers. Each layer is followed by a nonlinear activation function (such as a ReLU function) to introduce nonlinearity into the model. The output of the convolutional layers is then often passed through one or more fully-connected layers (also called "dense layers") before a final output layer that produces the predictions [18].

For the implementation of the 1D-, 2D-, and 3D-CNN architectures, we used the model proposed in [14] as a reference. However, the CNNs underwent adaptations to fit the proposed architecture implementation of this work. For the 1D-CNN, we used 1D convolutional layers and dense layers. Following, we used 2D and 1D convolutional layers, as well as dense layers, to implement the 2D-CNN. Finally, we applied the same architecture proposed in [14] to build the 3D-CNN.

In addition, all CNNs have the Flatten layers to vectorize the data and Dropout layers to prevent overfitting by randomly setting a fraction of the activations in the network to zero during training.

All architectures were subjected to the Adaptive Moment Estimation (Adam) optimizer with a learning rate ranging from 0.001 to 0.0001. The first moment, which is an exponentially weighted moving average of the gradients, and the second moment, which is an exponentially weighted moving average of the squares of the gradients, are the two moving averages that the Adam optimizer employs to calculate the adaptive learning rates [59], [60]. The following is how the adaptive learning rates are calculated using these moving averages:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (4a)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (4b)$$

TABLE 1. 1D-CNN for the IP and KC scenes.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
Conv1d_1	(1,22,16)	160
Conv1d_2	(1,16,32)	3,616
Conv1d_3	(1,12,64)	103,304
Conv1d_4	(1,11,128)	16,512
Flatten	(1024)	0
Dense_6	(256)	360,704
Dropout	(256)	0
Dense_8	(128)	32,896
Dropout	(128)	0
Dense_10	(16)	2,064

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (4c)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (4d)$$

$$w_t = w_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (4e)$$

where m_t is the first moment at time t, v_t is the second moment at time t, g_t is the gradient at time t, w_t is the weight at time t, α is the learning rate, β_1 and β_2 are the exponential decay rates for the first and second moments, respectively, and epsilon is a small constant used to prevent division by zero.

Below, the architectures for HSI classification are described in more detail.

A. 1D-CNN

1D-CNN is intended to explore only the spectral characteristics of the HSIs, performing a pixel-wise classification. In its development, we used an architecture that adapted better to the two image types after applying PCA. This approach is due to the number of PCs used for the IP HSI being different from the number used for the SA and UP HSIs. We used 30 components for the IP and KC scenes and 15 for the SA, UP, CP, and BT scenes. In both cases, the PCs were generated by the transformation with the PCA algorithm. The number of components affects the size of the kernels used in the convolutional layers, but the number of layers is the same. For example, the kernels used in the Conv 1D layers (1, 2, 3, and 4) for the IP HSI were 9, 7, 5, and 2, respectively, while for the SA, UP, CP, KC, and BT HSIs were 7, 5, 3, and 3, respectively.

Figure 2 illustrates the architecture of the developed 1D CNN. Tables 1 and 2 describe the characteristics of the layers of the models created, respectively, for the IP scene and the SA, PU, CP, KC, and BT scenes.

In addition, two methodologies for dividing training and testing data were used, as mentioned in Section II-C.

Subsequently, the proposed architecture for 1D-CNN was expanded to use the Attention layer in conjunction with MaxPooling layers. Additionally, the 1D-CNN-R and 1D-CNN-RA architectures explore the characteristic of using few resources, in this case, parameters, to observe whether accuracy can be satisfactory using fewer storage resources. The R in 1D-CNN-R stands for Reduced, and

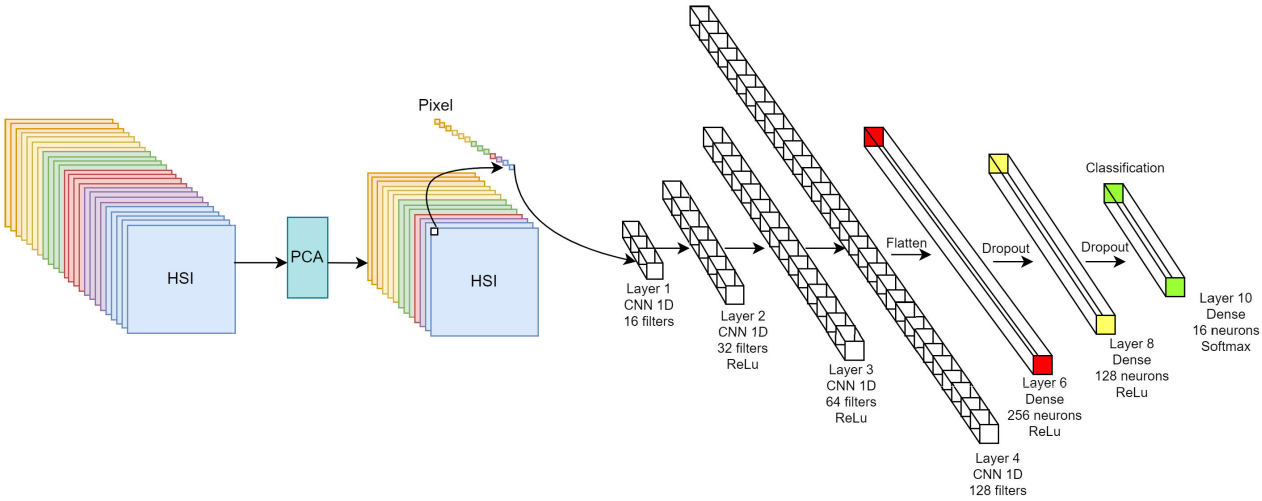


FIGURE 2. 1D-CNN architecture (based in [14]). We named this architecture 1D-CNN-O (-O stands for original architecture).

TABLE 2. 1D-CNN for the SA, PU, CP, and BT scenes. The values of the last layer are 2064 (16 classes) to SA scene, 1161 (9 classes) to UP and CP scenes, 1677 (13 classes) to KC scene, and 1806 (14 classes) to BT scene. The values in Dense 6 layer are 524,544 to KC scene and 33,024 for other scenes.

Layer	Output Shape	Number of Parameters
Input	(1,15,1)	0
Conv1d_1	(1,9,16)	128
Conv1d_2	(1,5,32)	2,592
Conv1d_3	(1,3,64)	6,208
Conv1d_4	(1,1,128)	24,704
Flatten	(128)	0
Dense_6	(256)	33,024/524,544
Dropout	(256)	0
Dense_8	(128)	32,896
Dropout	(128)	0
Dense_10	(16)/(9)/(13)/(14)	2,064/1,161/1,677/1,806

RA in 1D-CNN-RA stands for Reduced with Attention. Figure 3 illustrates the 1D-CNN-RA, with the difference to 1D-CNN-R being only the absence of the Attention layer. Tables 3 and 4 show the characteristics of the created architectures.

It is worth mentioning that the reduced architectures decrease the number of components after the PCA to 30. Adding the layers is intended to observe the positive and negative impacts primarily generated by adding the Attention layer. For example, in a neural network, the Attention mechanism allows the model to selectively focus on certain parts of the input data when processing it rather than using a fixed weighting or considering all parts of the input equally. This characteristic is advantageous when dealing with long sequences of data, such as natural language sentences, where certain words or phrases may be more relevant to the task at hand than others [21], [22]. The kernels used in the Conv 1D 1, 2, 3, and 5 layers for the IP HSI were 15, 1, 3, and 2. The choice of hyperparameters kernels size and number of filters of the convolutional layers, number of units of the Attention layer, number of neurons of the Dense layers, and the learning rate performed were found via Hyperband of the keras_tunner library.

TABLE 3. Reduced 1D-CNN without Attention layer.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
Conv1d_1	(16,120)	1,920
Conv1d_2	(16,104)	12,584
Conv1d_3	(14,24)	7,512
MaxPolling1d_4	(7,24)	0
Conv1d_5	(6,16)	784
MaxPolling1d_6	(3,16)	0
Attention_7	(32)	1,280
Flatten	(48)	0
Dense_8	(96)	3,168
Dropout	(96)	0
Dense_9	(96)	9,312
Dropout	(96)	0
Dense_10	(16)	1,552

TABLE 4. Reduced 1D-CNN with Attention layer.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
Conv1d_1	(16,120)	1,920
Conv1d_2	(16,104)	12,584
Conv1d_3	(14,24)	7,512
MaxPolling1d_4	(7,24)	0
Conv1d_5	(6,16)	784
MaxPolling1d_6	(3,16)	0
Flatten	(48)	0
Dense_7	(96)	4,704
Dropout	(96)	0
Dense_8	(96)	9,312
Dropout	(96)	0
Dense_9	(16)	1,552

The attention mechanism works by introducing a set of ‘‘attention weights’’ that indicate the importance of each part of the input to the model’s output. These weights are learned during training and can be thought of as a set of coefficients applied to the input data before passing through the rest of the network. There are several different ways in which attention mechanisms can be implemented in a neural network. One common approach is to use a separate ‘‘attention layer’’ that takes the input data and calculates the attention weights based on some measure of relevance or importance. These weights

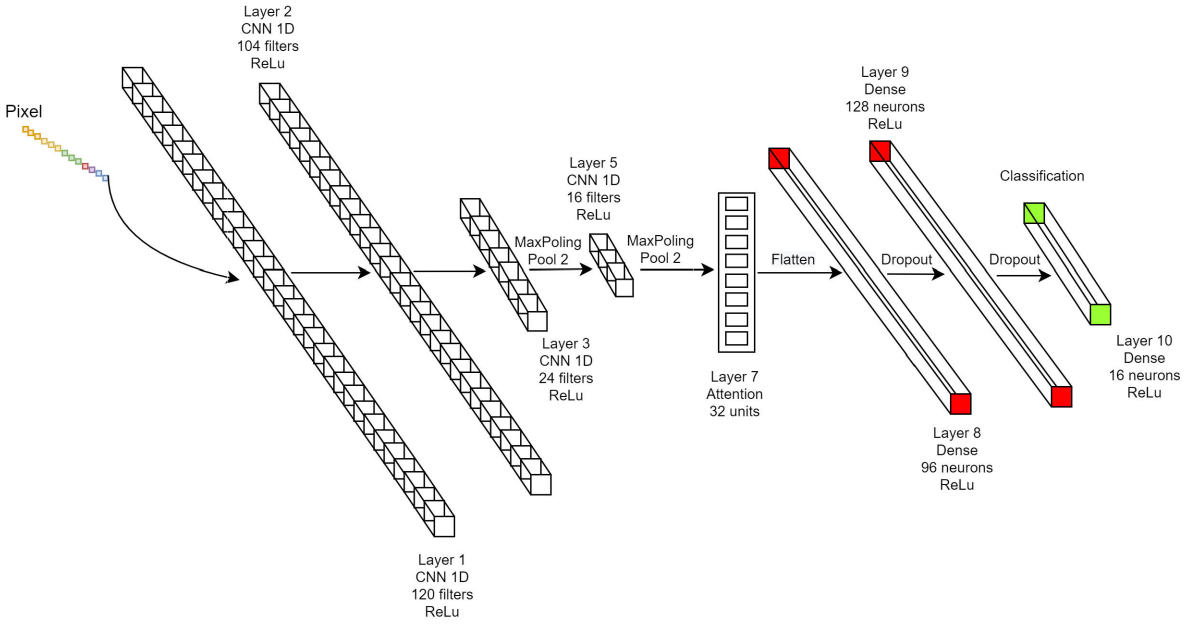


FIGURE 3. 1D-CNN-RA architecture proposed to explore satisfactory performance with a reduced number of parameters.

are then applied to the input data before it is passed through the rest of the network [21], [22].

This process involves calculating the dot product between the query vector and each key vector in the input data, as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (5)$$

where, Q is the query vector, K is the set of key vectors, and V is the set of value vectors. The dot product between the query vector and each key vector is normalized by the square root of the key vector dimension, d_k. The resulting dot products are then passed through a softmax function to produce a set of weights, which are used to weight the value vectors. The weighted sum of the value vectors is then returned as the output of the attention mechanism.

B. 2D-CNN

The 2D-CNN developed in this work relies on the architecture of the model proposed in [14]. However, unlike that work, the model uses three 2D convolutional layers and one 1D layer, followed by three dense layers. This architecture means that the model is more than just 2D. This architecture allows decreasing the number of parameters while still exploring the spatial characteristics of the HSI. A purely 2D model resulted in a considerable number of parameters (75 times more) for the same number of convolutional layers. The dense layers have the same number of units as the proposal in [14], which is the same architecture used in the 3D approach. In addition, we used a 19 × 19 spatial filtering window, which is smaller than the one applied in the reference work (i.e., 25 × 25).

Tables 5 and 6 provide a more detailed description of the architectures for the IP scene and the SA and UP scenes, which also differ in the number of components generated by the PCA algorithm.

TABLE 5. 2D-CNN for the IP scene.

Layer	Output Shape	Number of Parameters
Input	(19,19,30,1)	0
Conv2d_1	(19,13,24,8)	400
Conv2d_2	(19,9,20,16)	3216
Conv2d_3	(19,7,18,32)	4640
Reshape	(19,7,576)	0
Conv1d_4	(19,5,64)	110656
Flatten	(6080)	0
Dense_5	(256)	1556736
Dropout_6	(256)	0
Dense_7	(128)	32896
Dropout_8	(128)	0
Dense_9	(16)	2064

TABLE 6. 2D-CNN for the SA, PU, CP, and BT scenes. The values of the last layer is 2064 (16 classes) to SA scene, 1161 (9 classes) to UP and CP scenes, 1677 (13 classes) to KC scene, and 1806 (14 classes) to BT scene.

Layer	Output Shape	Number of Parameters
Input	(19,19,15,1)	0
Conv2d_1	(19,13,9,8)	400
Conv2d_2	(19,9,5,16)	3216
Conv2d_3	(19,7,3,32)	4640
Reshape	(19,7,96)	0
Conv1d_4	(19,5,64)	18496
Flatten	(6080)	0
Dense_5	(256)	1556736
Dropout_6	(256)	0
Dense_7	(128)	32896
Dropout_8	(128)	0
Dense_9	(16)/(9)/(13)/(14)	2,064/1,161/1,677/1,806

Unlike the methodology used in [14], the division of training and testing data was done using the methodology proposed in this work (described in Section II-C).

Figure 4 illustrates the developed 2D CNN architecture designed to work primarily with spatial characteristics.

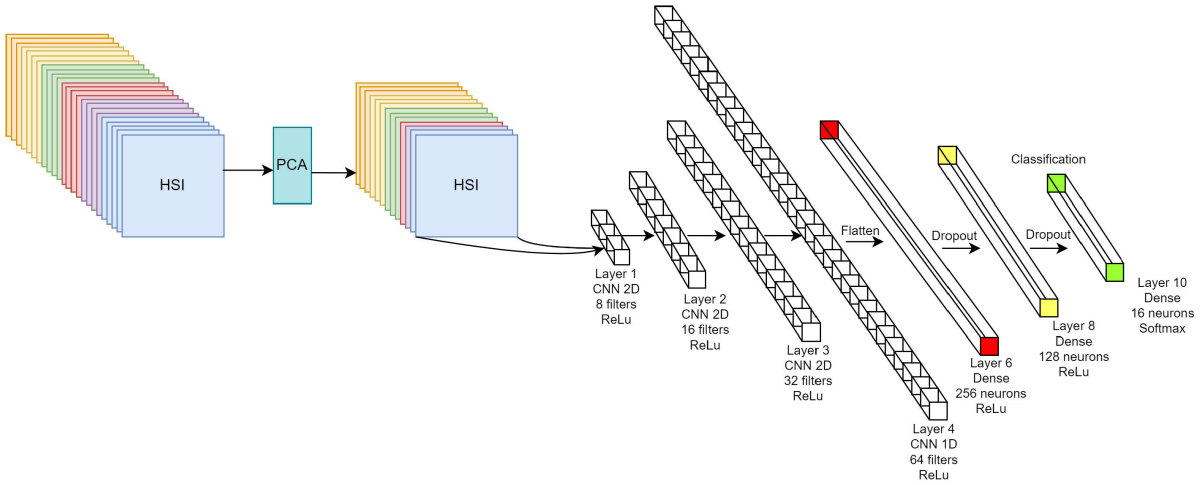


FIGURE 4. Proposed 2D-CNN architecture (based in [14]).

TABLE 7. 3D-CNN for the IP scene.

Layer	Output Shape	Number of Parameters
Input	(19,19,30,1)	0
Conv3d_1	(17,17,24,18)	512
Conv3d_2	(15,15,20,16)	5776
Conv3d_3	(13,13,18,32)	13856
Reshape_3	(13,13,576)	0
Conv2d_4	(11,11,64)	331840
Flatten	(7744)	0
Dense_6	(256)	1982720
Dropout	(256)	0
Dense_8	(128)	32896
Dropout	(128)	0
Dense_10	(16)	2064

TABLE 8. 3D-CNN for the SA, PU, CP, and BT scenes. The values of the last layer is 2064 (16 classes) to SA scene, 1161 (9 classes) to UP and CP scenes, 1677 (13 classes) to KC scene, and 1806 (14 classes) to BT scene.

Layer	Output Shape	Number of Parameters
Input	(19,19,15,1)	0
Conv3d_1	(17,17,24,18)	512
Conv3d_2	(15,15,20,16)	5776
Conv3d_3	(13,13,18,32)	13856
Reshape_3	(13,13,576)	0
Conv2d_4	(11,11,64)	55360
Flatten	(7744)	0
Dense_6	(256)	1982720
Dropout	(256)	0
Dense_8	(128)	32896
Dropout	(128)	0
Dense_10	(16)/(9)/(13)/(14)	2,064/1,161/1,677/1,806

C. 3D-CNN

The 3D-CNN used in this work was proposed and developed in [14], which was used as a guide for the other two architectures described in this work. However, unlike the reference architecture, the training and testing data were split using the methodology proposed by the authors (described in Section II). Again, this methodology prevents contamination of the training and testing datasets. As discussed above, we used a spatial filtering window smaller than the one applied in the reference work. Figure 5 ilustrated the architecture proposed by [14].

Tables 7 and 8 provide a more detailed description of the architectures for the IP scene for the SA and UP scenes, respectively, which also differ in the number of components generated by the PCA algorithm. Unlike 1D-CNN, 30 components are used for the IP scene.

IV. RNN ARCHITECTURE

RNNs are a type of neural network especially well-suited for processing data sequences, such as text, audio, or video. They are called recurrent because they have connections between neurons that allow information to be passed from one-time step to the next. It means that RNNs can store and retrieve information from previous steps and use it to make decisions in the present. RNNs are a powerful tool for processing data sequences and making inferences and predictions based on previous information. They are well-suited for applications involving time series data such as natural language processing or sensor sampling at different times [18], [19].

RNNs are composed of layers of neurons, each connected to the previous and subsequent layers through weights. During training, the weights are adjusted according to the errors generated by the network when processing the input data. The more accurate the network's prediction, the smaller the errors will be; therefore, the more accurate the weights will be [18], [19]. Several types of RNN architectures exist, such as LSTM and GRU [18], [19].

A. LSTM

LSTM is an RNN architecture proposed to handle long-term dependency problems in data sequences. These RNNs are called LSTM because they have an internal structure called an LSTM cell that allows the network to store information for extended periods [19], [45], [46].

The LSTM cell comprises three inputs, output, and forget gates that control the input, output, and deletion of information from the cell. This feature enables the LSTM network to control what is stored in memory and what is discarded, allowing it to capture broader context

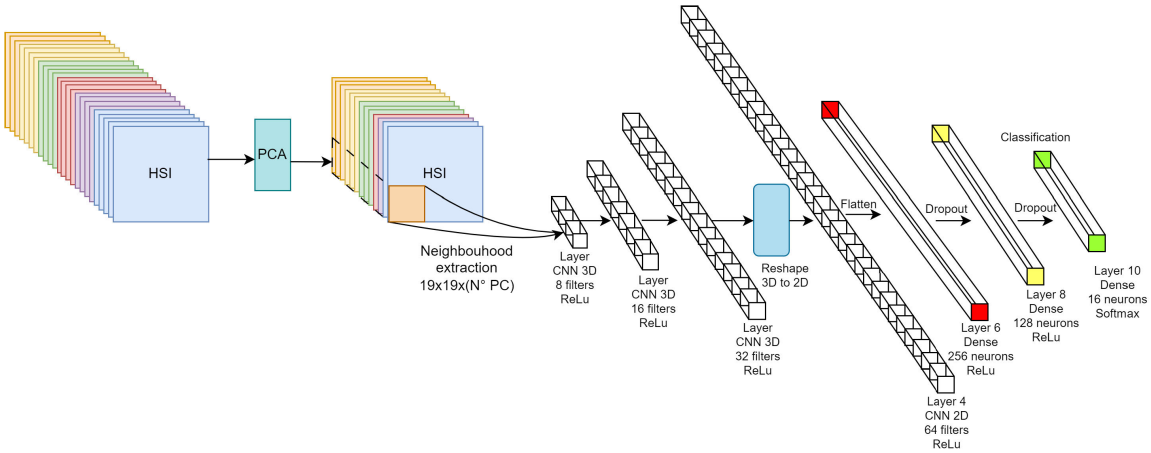


FIGURE 5. HybridSN proposed by [14].

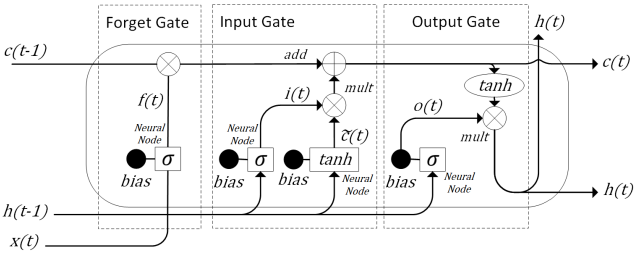


FIGURE 6. LSTM cell with Forget Gate.

dependencies in sequences of data [19], [45], [46]. The LSTM network can be mathematically represented as follows:

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \tag{6a}$$
$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \tag{6b}$$
$$\tilde{c}_t = \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \tag{6c}$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{6d}$$
$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \tag{6e}$$
$$h_t = o_t \cdot \tanh(c_t) \tag{6f}$$
$$y_t = W_{yh}h_t + b_y \tag{6g}$$

where x_t is the input at time step t, h_t is the hidden state at time step t, y_t is the output at time step t, i_t, f_t, and o_t are the input, forget, and output gates, respectively, $\tilde{c}_t$ is the output of the hyperbolic tangent activation function, and c_t is the memory cell. The weights W and biases b are learnable parameters of the model. Figure 6 illustrates the LSTM cell (and equations) with a forget gate [19].

LSTM networks can capture complex syntactic and semantic relationships in a text. Due to this, LSTMs are used in time series prediction, for example. In summary, LSTMs are a form of RNN that is particularly well-suited for processing sequences of data and capturing broader context dependencies [19], [45], [46].

We create two architectures of LSTM to apply HSI classification at the pixel level, as made in 1D-CNN. Figure 7 illustrates the architecture of the proposed LSTM with Attention Layer (LSTM-A).

TABLE 9. LSTM without Attention layer.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
LSTM_1	(30,40)	6720
Dropout	(30,40)	0
LSTM_2	(30,90)	47160
Dropout	(30,90)	0
LSTM_3	(120)	101280
Dropout	(120)	0
Dense_4	(24)	2904
Dropout	(24)	0
Dense_5	(240)	6000
Dropout	(240)	0
Dense_6	(16)	3856

TABLE 10. LSTM with Attention layer.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
LSTM_1	(30,40)	6720
Dropout	(30,40)	0
LSTM_2	(30,90)	47160
Dropout	(30,90)	0
LSTM_3	(30,120)	101280
Dropout	(30,120)	0
Attention	(56)	27840
Dense_4	(24)	1368
Dropout	(24)	0
Dense_5	(240)	6000
Dropout	(240)	0
Dense_6	(16)	3856

Tables 9 and 10 provide a more detailed description of the LSTM architectures. As made in the 1D-CNN architecture proposed with and without the Attention layer, the proposed LSTMs use 30 components in PCA for three HSI images used in this work. The choice of the hyperparameters kernels size, number of filters of the convolutional layers, number of units of the Attention layer, number of neurons of the Dense layers, number of units in the LSTM, and the learning rate performed were found via Hyperband from the keras_tunner library.

V. TRANSFORMER

Transformer is an ANN architecture presented by [20]. It is primarily used for Natural Language Processing (NLP) tasks such as machine translation, language modeling, and

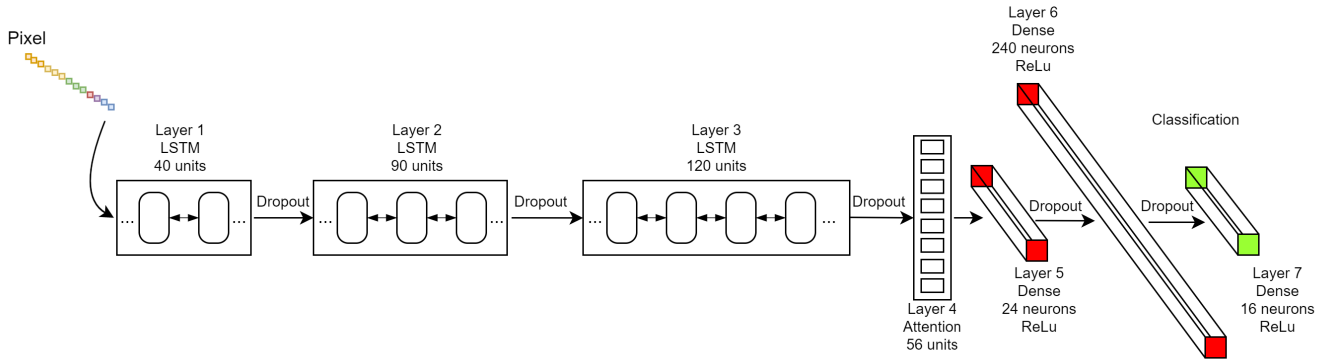


FIGURE 7. LSTM architecture with Attention layer.

text summarization, but it can also be used for time series processing. One of the main features of Transformer is the use of a Self-Attention Mechanism, or just Attention Mechanism [20], [21], [22].

Transformer also uses Multi-Head Attention, allowing the model to simultaneously attend to different parts of the input sequence of the [20] input sequence. This can be expressed as:

$$MH(Q, K, V) = Concat(head_1, .., head_h)W^O \quad (7)$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad (8)$$

where the function Attention(Q, K, V) is applied several times (in this case, h times) with different linear projections for Q, K and V for each Attention head and W^O is a [20] projection matrix. Finally, Transformer also makes use of feedforward networks with residual connections and layer normalization [20], which can be expressed as:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (9)$$

$$\text{LayerNorm}(x) = \frac{\gamma_i(x - \mu_i)}{\sqrt{\sigma_i^2 + \epsilon}} + \beta_i \quad (10)$$

where FFN(x) is the feedforward network and W_1, W_2, b_1, b_2 are trainable weights. The terms W_1 and W_2 are the weight matrix, while b_1 and b_2 are the bias. The LayerNorm(x) is the layer normalization function, μ and σ are the mean and the standard deviation of input x, respectively, and parameters γ and β are learnable scale and shift parameters, respectively [20], [61].

The Transformer architecture used is based time series example of [62]. The architecture input was modified to support pixel-level HSI classification as done in 1D-CNN. For the model, the pixel with 30 PCs is used for the IP, SA, UP, CP, KC, and BT HSIs.

The architecture is composed of four blocks named Transformer Layers. Internally, each block is divided into Multi-head Attention block and 1D-CNN block. Multi-head Attention blocks comprise four *heads* of 128 units and a normalization layer. The 1D-CNN block comprises two convolutional layers of 4 filters with kernel size 1, in addition to the normalization layer. Between the blocks, there is the

TABLE 11. Transformer layers. Layers 2 to 9 are repeated 4 times.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
LayerNormalization	(30,1)	2
Multi-head Attention	(30,1)	7,169
Dropout	(30,90)	0
Lambda	(30,120)	0
LayerNormalization	(30,120)	2
Conv1D	(30,4)	8
Dropout	(30,4)	0
Conv1D	(30,1)	5
Lambda	(30,1)	0
–	–	–
GlobalAveragePooling	(30)	0
Dense	(128)	3,968
Dense	(16)	2,064

TOFpLambda layer, also called Lambda layer, which serves as an adapter for sequential data between different layers and is inserted by the TensorFlow library.

VI. EXPERIMENTAL RESULTS

As a result, the performance metrics proposed architectures are related to prediction time, precision, and size (in bytes) were used.

A. CLASSIFICATION PERFORMANCE

Table 12 illustrates the results obtained in prediction (classification) with the testing data set. This table presents the result for the six HSIs used with the proposed division methodology and the size of the data sets used for training and testing.

The process of dividing the HSIs used in the tests was also evaluated for division using the proposed methodology and the *train_test_split* function from the scikit-learn library. Table 13 presents the results obtained.

Table 14 presents the accuracy difference only of the architectures that process the spectral information, comparing the 1D-CNN based on [14], the proposed CNN without and with Attention and LSTM architecture with and without the Attention layer. It is observed that the proposed CNN and Attention architectures present adequate results with fewer parameters. However, they do not affect the classification performance with the three metrics, showing a general result worse than their use. However, the architectures demand a

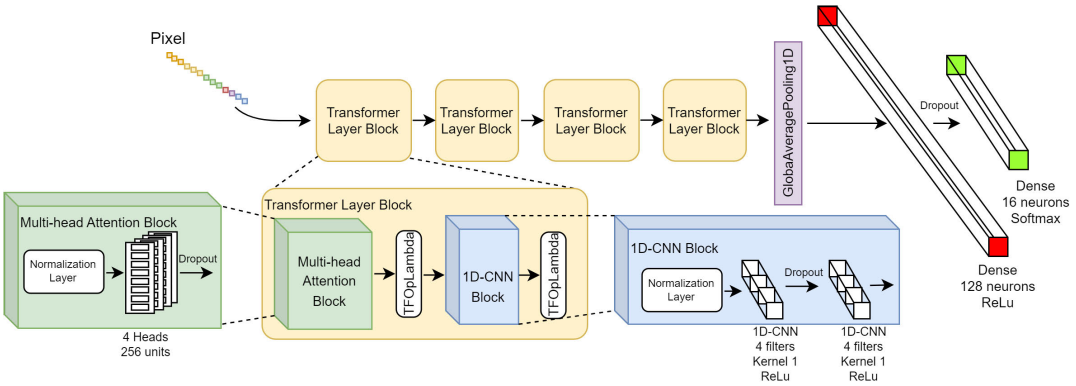


FIGURE 8. Transformer architecture implemented based in [62].

TABLE 12. Classification performance of the three CNN architectures using the new dataset-splitting methodology.

HSI	CNN Arch.	OA (%)	AA (%)	Kappa (×100)	Training Dataset	Testing Dataset
IP	1D	74.23	66.69	70.72	2.560	2.569
	2D	97.00	93.05	96.58	2.560	2.569
	3D	97.58	98.45	97.25	2.560	2.569
SA	1D	94.21	97.45	93.55	13.580	13.490
	2D	99.92	99.97	99.91	13.580	13.490
	3D	100.00	100.00	100.00	13.580	13.490
UP	1D	93.20	90.92	90.98	10.647	10.729
	2D	99.35	99.30	99.14	10.647	10.729
	3D	99.92	99.91	99.90	10.647	10.729
CP	1D	99.08	97.14	98.69	37,109	36,959
	2D	99.94	99.81	99.92	37109	36959
	3D	99.71	98.70	99.59	37109	36959
KC	1D	92.64	90.10	91.81	1,308	1,305
	2D	91.03	87.19	90.01	1308	1305
	3D	88.73	82.29	87.42	1308	1305
BT	1D	93.91	92.97	93.51	817	802
	2D	99.87	99.88	99.86	817	802
	3D	99.25	99.12	99.18	817	802

TABLE 13. Accuracy of the two dataset-splitting methodologies for 1D-CNN.

HSI	Divison Method	OA (%)	AA (%)	Kappa (×100)	Train. Size	Test. Size
IP	Proposed	75.36	70.45	72.01	2.560	2.569
	scikit-learn	83.17	84.96	80.78	8.199	2.050
SA	Proposed	94.21	97.45	93.55	13.580	13.490
	scikit-learn	96.24	98.19	95.81	43.303	10.826
UP	Proposed	93.20	90.92	90.98	10.647	10.729
	scikit-learn	95.17	93.61	93.59	34.220	8.556
CP	Proposed	99.08	97.14	98.69	37,109	36,959
	scikit-learn	99.16	97.38	98.82	118,521	29,631
KC	Proposed	92.64	90.10	91.81	1,308	1,305
	scikit-learn	96.06	94.10	95.62	4,168	1,043
BT	Proposed	93.91	92.97	93.51	817	802
	scikit-learn	96.92	97.26	96.66	2,598	650

TABLE 14. Performance of the 1D architectures.

HSI	Architecture	OA (%)	AA (%)	Kappa (×100)
IP	1D-CNN-O	83.17	84.96	80.78
	1D-CNN-R	77.34	73.97	74.03
	LSTM	75.70	73.08	72.28
	1D-CNN-RA	73.49	71.36	69.80
	LSTM-A	75.48	76.73	72.22
	Transformer	82.56	79.13	80.00
SA	1D-CNN-O	96.24	98.19	95.81
	1D-CNN-R	95.04	97.47	94.47
	LSTM	94.76	97.59	94.17
	1D-CNN-RA	94.39	96.95	93.75
	LSTM-A	94.77	97.48	94.16
	Transformer	96.23	98.22	95.80
UP	1D-CNN-O	95.17	93.61	93.59
	1D-CNN-R	92.16	89.94	89.63
	LSTM	93.99	92.39	92.02
	1D-CNN-RA	89.28	86.34	85.75
	LSTM-A	94.43	91.81	92.59
	Transformer	95.10	93.05	93.50
CP	1D-CNN-O	99.16	97.38	98.82
	1D-CNN-R	98.92	96.85	98.47
	LSTM	99.13	9747	9877
	1D-CNN-RA	98.41	94.95	97.75
	LSTM-A	99.04	97.15	98.65
	Transformer	99.20	97.34	98.87
KC	1D-CNN-O	96.06	94.10	95.62
	1D-CNN-R	94.11	91.43	93.45
	LSTM	91.24	87.11	90.24
	1D-CNN-RA	94.56	91.83	93.94
	LSTM-A	91.56	88.82	90.61
	Transformer	92.45	88.32	91.59
BT	1D-CNN-O	96.92	97.26	96.66
	1D-CNN-R	91.89	91.64	91.21
	LSTM	95.28	95.92	94.89
	1D-CNN-RA	75.38	75.46	73.32
	LSTM-A	95.79	96.43	95.44
	Transformer	97.12	97.23	96.88

number and variety of training samples, observed in the SA and UP datasets.

B. MEMORY REQUIREMENTS

Model storage is a feature that must be considered for computational systems with limited memory available. This requirement is essential for any application to run on these platforms. Therefore, the amount of memory (space) used by the model in bytes (without the dataset) was also measured. For more accurate model comparison, the number of parameters used by each architecture of each dataset was also obtained.

Table 15 presents the memory required by the three CNN architectures and by the three datasets based in [14]. There is a difference in batch size between the architectures, with 1D-CNN using 32 and 2D-CNN and 3D-CNN using 256. This difference was used based on the size of the training samples.

TABLE 15. Number of parameters and memory required by the three CNN architectures.

HSI	CNN Architecture	Size (bytes)	Number of Parameters
IP	1D	1,095,952	426,256
	2D	287,543,824	1,710,608
	3D	360,030,336	2,369,664
SA	1D	298,224	100,616
	2D	78,786,064	1,618,448
	3D	97,266,816	2,093,184
UP	1D	296,425	100,713
	2D	78,777,993	1,617,545
	3D	97,258,745	2,092,281
CP	1D	296,425	100,713
	2D	78,777,993	1,617,545
	3D	97,258,745	2,092,281
KC	1D	297,453	101,229
	2D	78,782,605	1,618,061
	3D	97,263,357	2,092,797
BT	1D	297,710	101,358
	2D	78,783,758	1,617,545
	3D	97,264,510	2,092,926

Comparison of required memory and number of parameters has also been expanded to the 1D architectures, as shown in Table 16. It is observed that the memory demands mainly for the 1D-CNN-R and -RA architecture, in joint analysis with the performance results in classification, show that this architecture offers a good trade-off. This feature is essential for use in the final application with the restriction of computational resources such as embedded systems. By default, we use the split method via scikit-learn since it has more samples, and 1D architectures do not have the problem of separating training and testing samples as in 2D and 3D architectures.

TABLE 16. Number of parameters and consumed memory of the 1D classification architectures.

HSI	CNN Architecture	Size (bytes)	Number of Parameters
IP	1D-CNN-O	1,095,952	426,256
	1D-CNN-R	637,408	38,368
	LSTM	1,266,672	167,920
	1D-CNN-RA	639,200	38,112
	LSTM-A	2,191,024	194,224
	Transformer	307,672	34,776
SA	1D-CNN-O	298,224	101,616
	1D-CNN-R	637,408	38,368
	LSTM	1,266,672	167,920
	1D-CNN-RA	639,200	38,112
	LSTM-A	2,191,024	194,224
	Transformer	307,672	34,776
UP	1D-CNN-O	296,425	100,713
	1D-CNN-R	635,833	37,689
	LSTM	1,264,089	166,233
	1D-CNN-RA	637,625	37,433
	LSTM-A	2,188,441	192,537
	Transformer	305,873	33,873
CP	1D-CNN-O	296,425	100,713
	1D-CNN-R	635,833	37,689
	LSTM	1,264,089	166,233
	1D-CNN-RA	637,625	37,433
	LSTM-A	2,188,441	192,537
	Transformer	547,793	37,713
KC	1D-CNN-O	297,453	101,229
	1D-CNN-R	636,733	38,077
	LSTM	1,265,565	167,197
	1D-CNN-RA	638,525	37,821
	LSTM-A	2,189,917	193,501
	Transformer	548,821	38,229
BT	1D-CNN-O	297,710	101,358
	1D-CNN-R	636,958	38,174
	LSTM	1,265,934	167,438
	1D-CNN-RA	638,750	37,918
	LSTM-A	2,190,286	193,742
	Transformer	549,078	38,358

C. TIME FOR PREDICTION

The prediction time of CNNs with different architectures was measured at the testing dataset and sample levels to observe the impact of classifying HSIs using 1D, 2D, and 3D data formats. This approach allows us to see which architecture fits best when processing time is a constraint (CPU or SoC with slow or low-power cores). Table 17 presents the results obtained with the prediction of the different architectures using the NVIDIA A100 GPU, NVIDIA Tesla T4 GPU, and CPU-only infrastructures. The number of samples used for testing and training is shown in Tab. 12 for 1D-, 2D-, and 3D-CNNs and in Tab. 13 for 1D architectures. For 1D architectures, the size in Tab. 13 is used for scikit-learn.

This comparison of time results on different GPUs demonstrates that the use of communication resources and the computing system makes the computing time vary. As can be seen, using an A100 GPU offers a communication latency that makes a 1D or 3D sample irrelevant to what is available in terms of bandwidth between CPU and GPU. However, the sample size is already relevant when we use Tesla T4 CPU-only and CPU+GPU mode. Furthermore, if we calculate the total sample time divided by the total time to process the dataset by the number of samples, we can see that the acceleration is much greater when processing files with the CPU and more than 2× with the T4 Tesla GPU. Therefore, even with the A100 GPU, we will obtain a result similar to the Tesla T4 GPU if we calculate the processing time per sample based on the processing time of the dataset by the number of samples. The calculated processing time came to mitigate this effect and evaluate how much the inference costs (in the processor that will make it – CPU or GPU). We made an approximation by dividing the processor time of the entire dataset by the number of samples in the dataset. This operation minimizes the impact of latency generated by the above factors and allows for a closer look than just the processed inference.

Table 18 expands the processing time results to sort the complete and sample-only dataset. The presented results compare the different classification architectures at the pixel level, considering only the spectral information. The low time required is a consequence of the low amount of parameters and memory and the lack of data structuring as in the 2D-CNN and 3D-CNN architectures.

D. DISCUSSION

When we compare only the 1D architectures, we see that the processing performance is uniform. The variation is

TABLE 17. Processing time for the three CNN architectures running on different infrastructures and using the new dataset-splitting methodology.

HSI	CNN Arch.	Test dataset size	Time all dataset A100 (s)	Time 1 sample capt. A100 (ms)	Time 1 sample calc. A100 (ms)	Time all dataset T4 (s)	Time 1 sample T4 capt. (ms)	Time 1 sample T4 calc. (ms)	Time all dataset CPU (s)	Time 1 sample CPU capt. (ms)	Time 1 sample CPU calc (ms)
IP	1D	2.569	0.62	87.48	0.24	0.3	193.1	0.12	0.6	55.41	0.24
	2D	2.569	0.77	87.57	0.30	0.7	238.3	0.29	9.3	55.26	3.63
	3D	2.569	0.54	84.78	0.21	0.7	242.6	0.30	32.3	61.29	12.58
SA	1D	13.490	1.66	81.59	0.12	1.1	191.0	0.08	1.2	60.95	0.09
	2D	13.490	2.21	92.15	0.16	1.7	229.5	0.12	15.3	46.13	1.13
	3D	13.490	1.85	87.71	0.14	1.6	204.3	0.12	46.8	49.98	3.47
UP	1D	10.729	1.33	65.83	0.12	0.9	211.4	0.08	0.9	54.43	0.09
	2D	10.729	1.75	68.86	0.16	1.3	173.6	0.12	12.3	46.56	1.15
	3D	10.729	1.47	69.83	0.14	1.3	130.0	0.12	36.8	50.04	3.43
CP	1D	36,959	3.51	78.86	0.09	2.9	87.95	0.08	3.8	56.72	0.10
	2D	36,959	5.56	63.89	0.34	6.8	75.27	0.20	44.3	54.41	1.43
	3D	36,959	4.35	51.66	0.11	5.6	63.95	0.15	128.3	63.03	3.47
KC	1D	1,305	0.43	63.47	0.33	0.3	47.85	0.25	0.4	55.97	0.28
	2D	1,305	0.40	59.08	0.35	0.4	60.05	0.22	1.7	52.62	1.46
	3D	1,305	0.24	74.47	0.18	0.2	57.66	0.17	4.6	63.49	3.52
BT	1D	802	0.37	59.95	0.46	0.3	46.41	0.43	0.3	57.03	0.43
	2D	802	0.37	60.79	0.47	0.3	57.67	0.44	1.2	57.14	1.57
	3D	802	0.15	55.81	0.18	0.2	54.39	0.24	2.8	71.63	3.54

TABLE 18. Processing time for the 1D architectures running on the CPU+NVIDIA A100 GPU infrastructure.

HSI	1D Architecture	Time test dataset (ms) captured	Time 1 test sample (ms) captured	Time 1 test sample (ms) calculated
IP	1D-CNN-O	423.36	51.69	0.13
	1D-CNN-R	444.42	66.91	0.14
	LSTM	1,742.48	56.35	0.56
	1D-CNN-RA	555.94	50.84	0.18
	LSTM-A	1,630.76	54.56	0.53
	Transformer	897.95	65.62	0.29
SA	1D-CNN-O	1,132.53	51.66	0.06
	1D-CNN-R	1,472.39	49.31	0.09
	LSTM	3,642.60	51.56	0.22
	1D-CNN-RA	2,060.54	52.62	0.12
	LSTM-A	4309.86	53.10	0.26
	Transformer	2,705.60	56.11	0.16
UP	1D-CNN-O	945.23	50.09	0.07
	1D-CNN-R	1,207.85	49.86	0.09
	LSTM	3,077.71	51.26	0.23
	1D-CNN-RA	1,657.27	52.59	0.12
	LSTM-A	3,659.06	54.99	0.28
	Transformer	2,231.02	54.80	0.17
CP	1D-CNN-O	3029.52	67.30	0.10
	1D-CNN-R	3948.82	73.13	0.08
	LSTM	9153.98	62.29	0.20
	1D-CNN-RA	5421.70	65.52	0.12
	LSTM-A	11946.48	81.16	0.26
	Transformer	7937.89	82.39	0.17
KC	1D-CNN-O	415,41	60.81	0.39
	1D-CNN-R	369.36	61.13	0.23
	LSTM	1633.70	65.37	1.04
	1D-CNN-RA	453.93	59.66	0.29
	LSTM-A	1768.09	68.27	1.13
	Transformer	706.98	68.59	0.45
BT	1D-CNN-O	337.75	57.35	0.51
	1D-CNN-R	275.59	58.89	0.28
	LSTM	2564.81	68.81	2.63
	1D-CNN-RA	341.33	60.23	0.52
	LSTM-A	1674.84	71.07	1.71
	Transformer	621.34	65.54	0.63

presented when the prediction of the entire dataset is made due to the variation of samples to be processed. What is worth mentioning is that regardless of the 1D architecture presented, CNN with or without the Attention layer, LSTM with or without the Attention layer, and Transformer, the processing is around from 40 to 90 ms per sample. This result can be motivated by the processing infrastructure that Google Colaboratory offers. However, it is worth mentioning that the memory consumption was more discrepant, which may impact computing platforms differently from the one used in this work. When considering the processing time, using only CPU and CPU+NVIDIA Tesla T4 GPU, we observe that 1D-CNNs enable an acceleration of almost a hundred times compared to 2D- and 3D-CNNs in the case of only CPU and more than two times in the case CPU+NVIDIA Tesla T4 GPU. This acceleration is only visible when we minimize the impact of data communication offered by the computing infrastructure. We divided the dataset processing time by the number of samples to minimize the impact.

In an analysis taking into account pixel sampling, the 1D-CNN architecture has an advantage. It can perform the classification without storing multiple samples and window size (19×19) and only perform the classification after sampling by the camera. A point that is also worth mentioning is the lack of control offered by the SaaS architecture of Google Colaboratory, where there is no certainty of how the computational system is organized, making the time obtained an estimate.

In an analysis taking into account pixel sampling, the 1D-CNN architecture has an advantage, as it can perform the classification without storing multiple samples and window size (19×19) and only then perform the classification after sampling by the camera.

Regarding the accuracy metric of the three architectures, we observed that the 1D-CNN architecture performs as well as the 2D- and 3D-CNNs when considering a training set with a good balance of samples and many training samples. This analysis comes when we observe the results of the IP scene, regardless of the division methodology, which presents poor results due to the significant class imbalance (number of samples). Comparing the results of

the SA, UP, CP, KC and BT scenes, we observe that the 1D-CNNs do not cope well with this imbalance compared to 2D- and 3D- CNNs. Therefore, HSI with a significant class imbalance, due to the region covered by the sensing system having this characteristic, should be classified using 2D- or 3D-CNNs. This statement is due to the ability of the 2D- and 3D-CNNs to relate much information, mitigating the imbalance. Therefore, if accuracy is the most critical requirement for the application, 2D- or 3D-CNNs should be considered. These two architectures also stand out compared to the 1D architecture, even with the balance in the dataset. Another impacting factor is that the new dataset division methodology generates a smaller number of samples for training, directly impacting the generalization ability of the network.

Nevertheless, the results in Tab. 12 and 14 show a correlation directly proportional to the number of samples for training with accuracy above 99%. The results in these tables show that using spectral-spatial information in the case of 3D architectures has the same impact as using larger datasets with good balance in the case of the CP scene. This result allows us to conclude that architectures such as Transformer, which has better average performance among 1D architectures, can be used instead of 3D architectures as long as it is trained with a dataset with a high number of samples. In addition, the 1D architecture will provide less processing time and memory demand.

When we focus on comparing 1D architectures, we observe that adding the Attention layer had no effect on the CNN performance but (little) impacted the LSTM architecture. As already mentioned, the IP dataset highly affects the reduced and unbalanced samples. However, the 1D architectures with the SA dataset have expressive results, which are also verified with the UP dataset. The classification performance in the three evaluated metrics shows that the 1D architecture can reach an accuracy close to, or even greater than, 95%, presenting a significantly lower number of parameters, processing time, and required storage memory. In addition, HSI, because they present enough scene information in the pixel, can only be classified with 1D architectures when the application does not demand special knowledge, such as structures, present in the scene of interest.

VII. CONCLUSION

In this work, an evaluation of CNN using different numbers of dimensions for convolution in HSI classification was performed. The results lead to and support the conclusion that the amount of dimensions directly impacts the most relevant metric for the target HSI application. For applications with strict resource restrictions, such as memory and processing, 1D-CNNs are ideal when there is an adequate class balance. On the other hand, for applications that demand higher accuracy, 1D-CNNs can be used when they have a large dataset with an excellent class balance. Even so, prioritizing precision, 2D- and 3D-CNNs are more indicated for being more accurate even without class balancing. With these results, designers can base themselves on implementing CNN for embedded platforms such as satellites, which have computing resource constraints and can use 1D-CNN. On the hand, classification systems in the food industry usually do not have computational limitations and value the accuracy of the analysis/classification [63].

From the presented results, using 1D-CNNs and LSTM for HSI classification has several benefits compared to using 2D- or 3D-CNNs. 1D-CNNs and LSTMs demand lower processing time and memory consumption and require fewer parameters, making them a good choice for systems with memory constraints. In acceleration, 1D-CNNs and LSTMs performed significantly faster than 2D- and 3D-CNNs, particularly when considering a single sample (pixel) classification. However, LSTM and 1D-CNN may not perform as well as 2D- and 3D-CNNs in cases with a large class imbalance in the data, as these architectures can better relate more information and mitigate the imbalance.

In future work, we intend to evolve the study on the Transformer architecture for per-pixel classification of HSIs. In addition, the studies will evolve towards implementing the hardware accelerator format of the models, particularly the Transformer architecture.

REFERENCES

[1] N. Yokoya, T. Yairi, and A. Iwasaki, "Coupled nonnegative matrix factorization unmixing for hyperspectral and multispectral data fusion," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 2, pp. 528–537, Feb. 2012.

[2] M. Simoes, J. Bioucas-Dias, L. B. Almeida, and J. Chanussot, "A convex formulation for hyperspectral image superresolution via subspace-based regularization," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 6, pp. 3373–3388, Jun. 2015.

[3] H. Zhang, L. Zhang, and H. Shen, "A super-resolution reconstruction algorithm for hyperspectral images," *Signal Proc.*, vol. 92, no. 9, pp. 2082–2096, 2012.

[4] A. Plaza, J. A. Benediktsson, J. W. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, A. Gualtieri, and M. Marconcini, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, pp. 110–122, Sep. 2009.

[5] Y. Jin, Y. Dong, Y. Zhang, and X. Hu, "SSMD: Dimensionality reduction and classification of hyperspectral images based on spatial–spectral manifold distance metric learning," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5538916.

[6] M. Moeller, T. Wittman, and A. L. Bertozzi, "A variational approach to hyperspectral image fusion," in *Proc. SPIE*, vol. 7334, Apr. 2009, Art. no. 73341E.

[7] N. Akhtar, F. Shafait, and A. Mian, "Bayesian sparse representation for hyperspectral image super resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3631–3640.

[8] Z. Chen, H. Pu, B. Wang, and G.-M. Jiang, "Fusion of hyperspectral and multispectral images: A novel framework based on generalization of pan-sharpening methods," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 8, pp. 1418–1422, Aug. 2014.

[9] J. Qu, Y. Li, and W. Dong, "Hyperspectral pansharpening with guided filter," *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 11, pp. 2152–2156, Nov. 2017.

[10] R. L. Davidson and C. P. Bridges, "Error resilient GPU accelerated image processing for space applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 1990–2003, Sep. 2018.

[11] L. A. Martins, G. A. M. Sborz, F. Viel, and C. A. Zeferino, "An SVM-based hardware accelerator for onboard classification of hyperspectral images," in *Proc. 32nd Symp. Integr. Circuits Syst. Design*, Aug. 2019, pp. 1–6.

[12] A. Perez, A. Rodriguez, A. Otero, D. G. Arjona, A. Jimenez-Peralo, M. A. Verdugo, and E. De La Torre, "Run-time reconfigurable MPSoC-based on-board processor for vision-based space navigation," *IEEE Access*, vol. 8, pp. 59891–59905, 2020.

[13] F. Viel, W. D. Parreira, A. A. Susin, and C. A. Zeferino, "A hardware accelerator for onboard spatial resolution enhancement of hyperspectral images," *IEEE Geosci. Remote Sens. Lett.*, vol. 18, no. 10, pp. 1796–1800, Oct. 2021.

[14] S. K. Roy, G. Krishna, S. R. Dubey, and B. B. Chaudhuri, "HybridSN: Exploring 3-D–2-D CNN feature hierarchy for hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 17, no. 2, pp. 277–281, Feb. 2020.

[15] T.-H. Hsieh and J.-F. Kiang, "Comparison of CNN algorithms on hyperspectral image classification in agricultural lands," *Sensors*, vol. 20, no. 6, p. 1734, Mar. 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/6/1734

[16] A. A. Abdelhamid, E.-S.-M. El-Kenawy, B. Alotaibi, G. M. Amer, M. Y. Abdelkader, A. Ibrahim, and M. M. Eid, "Robust speech emotion recognition using CNN+LSTM based on stochastic fractal search optimization algorithm," *IEEE Access*, vol. 10, pp. 49265–49284, 2022.

[17] N. Lopac, F. Hrzic, I. P. Vuksanovic, and J. Lerga, "Detection of non-stationary GW signals in high noise from Cohen's class of time–frequency representations using deep learning," *IEEE Access*, vol. 10, pp. 2408–2428, 2022.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[19] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural Comput.*, vol. 31, no. 7, pp. 1235–1270, 2019.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[22] M. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015, *arXiv:1508.04025*.

[23] M. E. Paoletti, J. M. Haut, J. Plaza, and A. Plaza, "Deep learning classifiers for hyperspectral imaging: A review," *ISPRS J. Photogramm. Remote Sens.*, vol. 158, pp. 279–317, Dec. 2019.

[24] L. Huang and Y. Chen, "Dual-path Siamese CNN for hyperspectral image classification with limited training samples," *IEEE Geosci. Remote Sens. Lett.*, vol. 18, no. 3, pp. 518–522, Mar. 2021.

[25] Q. Liu, Y. Dong, Y. Zhang, and H. Luo, "A fast dynamic graph convolutional network and CNN parallel network for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5530215.

[26] Y. Wang, Z. Fang, and H. Hong, "Comparison of convolutional neural networks for landslide susceptibility mapping in Yanshan county, China," *Sci. Total Environ.*, vol. 666, pp. 975–993, May 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0048969719307612

[27] T. Hakala, I. Polonen, E. Honkavaara, R. Nasi, T. Hakala, and A. Lindfors, *Using Aerial Platforms in Predicting Water Quality Parameters From Hyperspectral Imaging Data With Deep Neural Networks*. Cham, Switzerland: Springer, 2020, pp. 213–238, doi: 10.1007/978-3-030-37752-6_13.

[28] C. Chen, J.-J. Zhang, C.-H. Zheng, Q. Yan, and L.-N. Xun, "Classification of hyperspectral data using a multi-channel convolutional neural network," in *Intelligent Computing Methodologies*, D.-S. Huang, M. M. Gromiha, K. Han, and A. Hussain, Eds. Cham, Switzerland: Springer, 2018, pp. 81–92.

[29] Y. Li, H. Zhang, and Q. Shen, "Spectral–spatial classification of hyperspectral imagery with 3D convolutional neural network," *Remote Sens.*, vol. 9, no. 1, p. 67, Jan. 2017. [Online]. Available: https://www.mdpi.com/2072-4292/9/1/67

[30] P. Dou and C. Zeng, "Hyperspectral image classification using feature relations map learning," *Remote Sens.*, vol. 12, no. 18, p. 2956, Sep. 2020. [Online]. Available: https://www.mdpi.com/2072-4292/12/18/2956

[31] W. L. Hakim, F. Rezaie, A. S. Nur, M. Panahi, K. Khosravi, C.-W. Lee, and S. Lee, "Convolutional neural network (CNN) with metaheuristic optimization algorithms for landslide susceptibility mapping in Icheon, South Korea," *J. Environ. Manag.*, vol. 305, Mar. 2022, Art. no. 114367. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301479721024294

[32] Y. Luo, J. Zou, C. Yao, X. Zhao, T. Li, and G. Bai, "HSI-CNN: A novel convolution neural network for hyperspectral image," in *Proc. Int. Conf. Audio, Lang. Image Process. (ICALIP)*, Jul. 2018, pp. 464–469.

[33] B. Becker, M. Vaccari, M. Prescott, and T. Grobler, "CNN architecture comparison for radio galaxy classification," *Monthly Notices Roy. Astronomical Soc.*, vol. 503, no. 2, pp. 1828–1846, Feb. 2021, doi: 10.1093/mnras/stab325.

[34] S. K. Roy, J. M. Haut, M. E. Paoletti, S. R. Dubey, and A. Plaza, "Generative adversarial minority oversampling for spectral–spatial hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5500615.

[35] M. E. Paoletti, J. M. Haut, N. S. Pereira, J. Plaza, and A. Plaza, "Ghostnet for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 59, no. 12, pp. 10378–10393, Dec. 2021.

[36] M. Liang, H. Wang, X. Yu, Z. Meng, J. Yi, and L. Jiao, "Lightweight multilevel feature fusion network for hyperspectral image classification," *Remote Sens.*, vol. 14, no. 1, p. 79, Dec. 2021. [Online]. Available: https://www.mdpi.com/2072-4292/14/1/79

[37] M. He, B. Li, and H. Chen, "Multi-scale 3D deep convolutional neural network for hyperspectral image classification," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 3904–3908.

[38] J. A. Benediktsson, X. C. Garcia, B. Waske, J. Chanussot, J. R. Sveinsson, and M. Fauvel, "Ensemble methods for classification of hyperspectral data," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2008, pp. 1–62.

[39] P. Ramzi, F. Samadzadegan, and P. Reinartz, "Classification of hyperspectral data using an AdaBoostSVM technique applied on band clusters," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2066–2079, Jun. 2014.

[40] W. Li, C. Chen, M. Zhang, H. Li, and Q. Du, "Data augmentation for hyperspectral image classification with deep CNN," *IEEE Geosci. Remote Sens. Lett.*, vol. 16, no. 4, pp. 593–597, Apr. 2019.

[41] J. Nalepa, M. Myller, and M. Kawulok, "Training- and test-time data augmentation for hyperspectral image segmentation," *IEEE Geosci. Remote Sens. Lett.*, vol. 17, no. 2, pp. 292–296, Feb. 2020.

[42] S. Rashwan and N. Dobigeon, "A split-and-merge approach for hyperspectral band selection," *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 8, pp. 1378–1382, Aug. 2017.

[43] Y. Li, Z. Du, S. Wu, Y. Wang, Z. Wang, X. Zhao, and F. Zhang, "Progressive split-merge super resolution for hyperspectral imagery with group attention and gradient guidance," *ISPRS J. Photogramm. Remote Sens.*, vol. 182, pp. 14–36, Dec. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271621002616

[44] L. Liang, S. Zhang, and J. Li, "Multiscale DenseNet meets with bi-RNN for hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 15, pp. 5401–5415, 2022.

[45] Q. Liu, F. Zhou, R. Hang, and X. Yuan, "Bidirectional-convolutional LSTM based spectral–spatial feature learning for hyperspectral image classification," *Remote Sens.*, vol. 9, no. 12, p. 1330, Dec. 2017. [Online]. Available: https://www.mdpi.com/2072-4292/9/12/1330

[46] L. Mou, P. Ghamisi, and X. X. Zhu, "Deep recurrent neural networks for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 7, pp. 3639–3655, Jul. 2016.

[47] R. Hang, Q. Liu, D. Hong, and P. Ghamisi, "Cascaded recurrent neural networks for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 8, pp. 5384–5394, Aug. 2019.

[48] X. He, Y. Chen, and Z. Lin, "Spatial-spectral transformer for hyperspectral image classification," *Remote Sens.*, vol. 13, no. 3, p. 498, Jan. 2021.

[49] L. Sun, G. Zhao, Y. Zheng, and Z. Wu, "Spectral–spatial feature tokenization transformer for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5522214.

[50] Z. Zhong, Y. Li, L. Ma, J. Li, and W.-S. Zheng, "Spectral–spatial transformer network for hyperspectral image classification: A factorized architecture search framework," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, pp. 1–15, 2021.

[51] D. Hong, Z. Han, J. Yao, L. Gao, B. Zhang, A. Plaza, and J. Chanussot, "SpectralFormer: Rethinking hyperspectral image classification with transformers," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5518615.

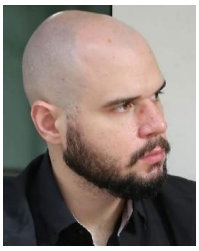
[52] L. Yang, Y. Yang, J. Yang, N. Zhao, L. Wu, L. Wang, and T. Wang, "FusionNet: A convolution–transformer fusion network for hyperspectral image classification," *Remote Sens.*, vol. 14, no. 16, p. 4066, Aug. 2022.

[53] Z. Zhang, T. Li, X. Tang, X. Hu, and Y. Peng, "CAEVT: Convolutional autoencoder meets lightweight vision transformer for hyperspectral image classification," *Sensors*, vol. 22, no. 10, p. 3902, May 2022.

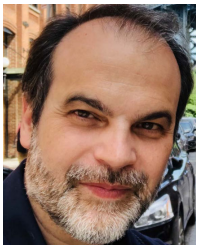
[54] Q. Xu, C. Yang, J. Tang, and B. Luo, "Grouped bidirectional LSTM network and multistage fusion convolutional transformer for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5541314.
[55] W. Wang, L. Liu, T. Zhang, J. Shen, J. Wang, and J. Li, "Hyper-ES2T: Efficient spatial–spectral transformer for the classification of hyperspectral remote sensing images," *Int. J. Appl. Earth Observ. Geoinf.*, vol. 113, Sep. 2022, Art. no. 103005.
[56] J. Yang, B. Du, and L. Zhang, "From center to surrounding: An interactive learning framework for hyperspectral image classification," *ISPRS J. Photogramm. Remote Sens.*, vol. 197, pp. 145–166, Mar. 2023.
[57] B. A. M. Grana and M. A. Veganzons. (Mar. 2021) *Hyperspectral Remote Sensing Scenes*. [Online]. Available: http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes
[58] M. D. Farrell and R. M. Mersereau, "On the impact of PCA dimension reduction for hyperspectral detection of difficult targets," *IEEE Geosci. Remote Sens. Lett.*, vol. 2, no. 2, pp. 192–195, Apr. 2005.
[59] S. Bock, J. Goppold, and M. Weiß, "An improvement of the convergence proof of the ADAM-optimizer," 2018, *arXiv:1804.10587*.
[60] S. Bock and M. Weis, "A proof of local convergence for the Adam optimizer," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.
[61] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu, "On layer normalization in the transformer architecture," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 10524–10533.
[62] T. Ntakouris. (Jun. 2021). *Timeseries Classification With a Transformer Model*. [Online]. Available: https://keras.io/examples/timeseries/timeseries_transformer_classification/
[63] N. Lambert. (2023). *CS 188: Artificial Intelligence—Reinforcement Learning II [PowerPoint Slides]*. [Online]. Available: https://inst.eecs.berkeley.edu/~cs188/sp23/assets/lectures/cs188-sp23-lec14.pdf

FELIPE VIEL (Student Member, IEEE) received the master's degree in applied computing from the University of Vale do Itajaí, Brazil, in 2019. He is currently pursuing the Ph.D. degree in electrical engineering with the Federal University of Santa Catarina. He is also a Computer Engineer. He has been an Assistant Professor with the Polytechnic School, University of Vale do Itajaí (UNIVALI), Brazil, since 2019, and a Researcher with the Laboratory of Embedded and Distributed Systems, Univali. His research interests include digital systems design, embedded systems, reconfigurable systems (FPGAs), hardware accelerators, machine learning, digital image processing, and space applications.

RENATO COTRIM MACIEL received the bachelor's degree (Hons.) in electrical engineering with a minor in industrial engineering from the Universidade Federal de Santa Catarina. During his studies, he earned a place on the Dean's List for the 2014 Fall Semester with the University of Nebraska–Lincoln and the 2015 Spring Semester with North Dakota State University, Fargo. He is the Lead Machine Learning Engineer with significant experience in the field of big data and machine learning. He is involved in open-source software projects, where he contributes to the development of libraries and tools for machine learning. His research interests include machine learning, big data, natural language processing, computer vision, time series analysis, and forecasting. He focuses on designing and implementing scalable and available machine learning models and data pipelines on cloud-based architectures.



LAIO ORIEL SEMAN received the graduate and master's degrees from the Regional University of Blumenau (FURB), in 2013 and 2015, respectively, and the Ph.D. degree from the Federal University of Santa Catarina (UFSC), in 2017, all in electrical engineering. He is currently a permanent Professor with the Masters in Applied Computing, University of Vale do Itajaí (UNIVALI), and a Collaborating Professor with PosAutomação (Postgraduate Program in Automation and Systems), UFSC. He has experience in electrical engineering with an emphasis on computational systems, working mainly on modeling, control and optimization strategies (linear, non-linear, and mixed integer programming), and applied artificial intelligence. His research interests include intelligent transport systems, cyber-physical systems, aerospace systems (CubeSats), and oil and gas production.



CESAR ALBENES ZEFERINO (Member, IEEE) received the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 2003, with a sandwich internship at the Sorbonne University, Paris, France. He has been a Full Professor with the Polytechnic School, University of Vale do Itajaí (UNIVALI), Brazil, since 2002, teaching undergraduate and graduate courses in the fields of computer architecture, embedded systems, and digital systems. He is currently the Director of the Polytechnic School and the Head of the Laboratory of Embedded and Distributed Systems (LEDS). He was granted the Researcher Productivity Scholarship from the National Council for Scientific and Technological Development (CNPq), Brazil. His interests include digital systems design, embedded systems design, networks-on-chip, hardware accelerators, and the Internet of Things.

EDUARDO AUGUSTO BEZERRA (Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Sussex, England, where he developed his research work at Space Science Centre from 1998 to 2002. He is currently a Researcher and a Lecturer with the Electrical and Electronic Engineering Department, Federal University of Santa Catarina, Brazil. He is the Head of the Space Technology Research Laboratory (SpaceLab) and leads space missions, such as the GOLDS-UFSC, Catarina Constellation, and FloripaSat, which is the first satellite of the family was launched in 2019. His research interests include embedded systems for space applications, computer architecture, reconfigurable systems (FPGAs), software and hardware testing, and fault tolerance.

VALDERI REIS QUIETINHO LEITHARDT (Senior Member, IEEE) received the Ph.D. degree in computer science from INF-UFRGS, Brazil, in 2015. He is currently an Adjunct Professor with the Polytechnic Institute of Portalegre, where he is a Researcher integrated with the VALORIZA Research Group, School of Technology and Management (ESTG). He is also a Collaborating Researcher with the following research groups: COPELABS, Universidade Lusófona de Lisboa, Portugal; and the Expert Systems and Applications Laboratory, University of Salamanca, Spain. His research interests include distributed systems with a focus on data privacy, communication, and programming protocols, involving scenarios and applications for the Internet of Things, smart cities, big data, cloud computing, and blockchain.