remote sensing

MDPI

Article

Distributed Training and Inference of Deep Learning Models for Multi-Modal Land Cover Classification

Maria Aspri [1,2], Grigorios Tsagkatakis [2,*] and Panagiotis Tsakalides [1,2]

[1] Institute of Computer Science, Foundation for Research and Technology-Hellas (FORTH), GR70013 Heraklion, Greece; aspri@ics.forth.gr (M.A.); tsakalid@ics.forth.gr (P.T.)
[2] Computer Science Department, University of Crete, GR70013 Heraklion, Greece
* Correspondence: greg@ics.forth.gr; Tel.: +30-2811-392725

Received: 8 July 2020; Accepted: 17 August 2020; Published: 19 August 2020

Abstract: Deep Neural Networks (DNNs) have established themselves as a fundamental tool in numerous computational modeling applications, overcoming the challenge of defining use-case-specific feature extraction processing by incorporating this stage into unified end-to-end trainable models. Despite their capabilities in modeling, training large-scale DNN models is a very computation-intensive task that most single machines are often incapable of accomplishing. To address this issue, different parallelization schemes were proposed. Nevertheless, network overheads as well as optimal resource allocation pose as major challenges, since network communication is generally slower than intra-machine communication while some layers are more computationally expensive than others. In this work, we consider a novel multimodal DNN based on the Convolutional Neural Network architecture and explore several different ways to optimize its performance when training is executed on an Apache Spark Cluster. We evaluate the performance of different architectures via the metrics of network traffic and processing power, considering the case of land cover classification from remote sensing observations. Furthermore, we compare our architectures with an identical DNN architecture modeled after a data parallelization approach by using the metrics of classification accuracy and inference execution time. The experiments show that the way a model is parallelized has tremendous effect on resource allocation and hyperparameter tuning can reduce network overheads. Experimental results also demonstrate that proposed model parallelization schemes achieve more efficient resource use and more accurate predictions compared to data parallelization approaches.

Keywords: distributed deep learning; model parallelization; convolutional neural networks; multi-modal observation classification; land cover classification

1. Introduction

With the proliferation of high-resolution platforms continuously acquiring massive amounts of observations, extracting actionable knowledge from massive Earth Observation (EO) data is a crucial issue spanning different disciplines within the exploding research area of Big Data analytics. In contrast to typical Big Data scenarios in social media and web analytics, Big Earth Observation Data [1] correspond to high volumes of complex measurements streaming from scientific EO observation platforms, where each observation carries critical information. A representative example of such challenges is the Copernicus EO platform which is producing more than 10 TB of observations per day (velocity), leading to petascales of generated data (volume), from instruments ranging from radar (Sentinel 1) to spectroscopy (Sentinel 2) and radiometry (Sentinel 3) (variety), each one associated with different quality and observation characteristics (veracity). The driving force for addressing these challenges is the added value for long-term EO monitoring applications, such as climate change

monitoring, precision agriculture, and humanitarian aid among others, in a market expected to exceed 40B$ in the next decade. In the traditional paradigm, experts were tasked with investigating and characterizing observations, but in the era of Big EO Data such labor-intensive approaches cannot effectively scale-up. Hence, the development of large-scale distributed machine learning systems, capable of extracting actionable information from diverse data sources is extremely urgent.

Machine learning models, and more specifically Deep Neural Networks (DNNs) with multiple layers between the input and output layers, are becoming a pervasive tool in modern EO data analysis applications due to their efficiency in learning appropriate data representations [2,3]. When trained with large amounts of data, DNNs provide extremely accurate solutions and as a result, they have successfully been considered in a plethora of fields, including remote-sensing [4,5], speech recognition [6] and computer vision [7] among others. Graphical Processing Units (GPUs) are the ideal commodity hardware for training DNNs since they were developed to deal with parallel computations and have large memory bandwidth. Even though significant advances have been made in both GPU hardware and DNN architectures, the fact remains that as datasets increase in size and network architectures increase in complexity, DNNs require even more resources in both computational power and memory demand. As a result, both training and inference on large-scale models to competitive accuracy, essentially require a cluster of multiple machines connected by a network. Training under a distributed paradigm is also appropriate when dealing with Big Data, such as the ones associated with Earth Observation [8]. In such cases, and especially when considering scenarios involving multiple sources of observations, each stored at a different location, transferring of all required data into a single location can be very challenging, even prohibiting for very large volumes [9,10].

Subsequently, Distributed Deep Learning has received considerable attention, and many approaches and frameworks have been proposed and implemented [11], making various distributed and parallel computing schemes applicable to a variety of problems that have been previously addressed by sequential approaches. The performance of those approaches is however highly dependant not only on the connectivity structure and the computational needs of the model, but also on the cluster's characteristics. For example, DNN layers with local connectivity structures, such as the one in Convolutional Neural Networks (CNNs), have generally low communication requirements, and tend to be more compliant to distribution, than fully connected architectures such as Multi-Layer Perceptron. Moreover, processing times between different machines tend to differ, causing bottlenecks when transcending to different computation phases and leading to non-ideal speedups. Intercommunication is also a major factor to take into consideration, since different network technologies provide different performance [12].

Motivated by this, we explore the optimization of a distributed multi-modal CNN architecture and evaluate its performance in multi-class land cover classification. We consider CNN models since these models are proven to be highly effective on remote sensing observation classifications scenarios [13,14]. We explore scenarios of model parallelization as our distributed approach and consider different parallelization architectures in order to understand which one is the most appropriate for model distribution. Moreover, we compare our architecture with another popular distributed deep learning approach called data parallelization [15,16], and provide experimental results on each approach's ability to make fast and accurate predictions. As a result, our study makes the following contributions:

- A detailed analysis on the trade-offs between different parallelization architectures, focusing on the case of remote sensing data analysis.
- Formulation of a novel distributed processing model for the case of multi-modal remote sensing observation classification.
- A detailed experimental investigation of the merits of the proposed scheme compared to single-modality-based learning and non-deep learning methods.
- Comparison between model and data distributed DNN approaches for the problem of multi-modal data classification.

- Insights into model parallelization behavior, providing directions on better allocation of the resources of computer clusters.

2. State of the Art

2.1. Distributed Deep Learning and Model Parallelization

Several methods have been proposed for parallelizing and distributing a DNN model [17,18] as it is possible to partition both the forward evaluation and the backpropagation phases among parallel processors and across different machines. In general, two main classes of distributed processing have been considered, namely data parallelism, model parallelism and hybrid approached [19].

In model parallelization, each worker has a local copy of the entire network and is responsible for a different part of the processing pipeline. In model parallelism, different machines are responsible for the computations in different parts of the network. In this case, the weights are split equally among the machines, and all of them train on the same mini batch. The general idea behind hybrid parallelism is to combine different parallelization schemes in order to overcome the drawbacks of each other.

The DNN is not stored in a single device helps in memory conservation; however it also incurs additional communication after every layer. Considering this, it suffices to say that the architecture of a DNN plays an important role in the overall performance, since it creates layer inter-dependencies, which, generate a significant amount of communication overhead. To resolve this issue, the authors in [20] propose a method for partitioning a DNN in such a way that each machine will be responsible for twice the number of its neurons. The authors in [21] propose an alternative method, in which they use Cannon's matrix multiplication algorithm in order to reduce communication between fully connected layers, but it only produces better efficiency and speedups over simple partitioning on small-scale fully connected networks.

In contrast of model parallelism, in data parallelization every worker receives a complete copy of the model and its parameters. The dataset is distributed among the machines through different partitions, which train the copies locally. Each copy must use the same weights but trains on different batches, which means that in the weight update phase the results of the partitions have to be averaged to obtain the global gradient. The most straightforward strategy for data parallelism is to partition the work of the dataset samples among multiple computational resources, either cores or different devices. Therefore, datasets too large to be stored on a single machine use this method to achieve faster training [22].

Since data parallel approaches train a copy of the entire model on individual computer systems, they require some method of combining results and synchronizing the model parameters between machines. The prominent examples of parameter synchronization involve synchronous and asynchronous training, while coordination is facilitated by a machine named Parameter Server (PS) [23]. Synchronous methods [24] require all replicas to update their parameters at the same timestamp. By doing so, the batch size is effectively multiplied by the number of replicas. Many synchronous approaches for data parallelism were proposed in the literature. In ParallelSGD [25], SGD is run k times in parallel, dividing the dataset among k processors. After the convergence of every SGD instance, the resulting weights are aggregated to obtain a global averaged weight vector. MapReduce [16] has also been used in many distributed deep learning implementations [26,27], due to its easy to schedule parallel tasks. However, despite the overall successful first results, it generality hindered DNN-specific optimizations. More recent implementations however, make use of high-performance communication interfaces to implement parallelism features, such as reducing latency via Asynchronous methods [28]. Asynchronous executions can potentially provide higher throughput, since machines spend more time performing computations, instead of waiting for the parameter averaging step to be completed [28].

Regarding hybrid parallelism, the general idea behind it is to combine different parallelization schemes in order to overcome the drawbacks of each other. The authors in [29] perform a hybrid scheme

on AlexNet [3], in which they apply data parallelism to convolutional layers, and model parallelization to fully connected parts. The work presented in [28] proposed an asynchronous implementation of DNN training on CPUs, which uses an intermediate representation to implement fine-grained hybrid parallelism. Finally, in [30] distributed training is performed simultaneously on multiple model replicas, while each replica is trained on different data samples.

An earlier version of this work was presented in [31]. That work differs from this one in that we considered a different type of problem, namely spectroscopic redshift estimation from astrophysical observations, and we considered the scenario of data-parallelism for training a CNN model.

2.2. Deep Learning in Land Cover Classification

In the last few years the number of Earth observation satellites has significantly increased, providing continuous streams of high resolution imagery of the Earth's surface [32]. Analyzing the massive volumes of Earth observations can provide valuable insights into the state of the planet, including the ability to monitor and track changes in land cover and land use and among others. Regarding land cover classification, different approaches have been proposed, which mostly exploit user-defined features, such as the Scale Invariant Feature Transform (SIFT), extracted from remote sensing images [33,34]. However, it is difficult for user-defined features to capture the complexity of real data, especially when considering multiple source and modalities. As a result, such feature extraction processes are unable to achieve an optimal balance between discriminability between different classes and robustness to class variation.

With DNNs on the other hand, with its ability to automatically learn hierarchical intermediate representations [3,35], has been established as the next trend in the remote sensing classification [36]. As a result, a considerable amount of studies related to deep learning for remote sensing image analysis have been published in recent years, with many of them focusing on unsupervised generative graphical models, such as Deep Belief Networks (DBNs) or Sparse AutoEncoders (SAEs). The authors in [37] tested a DBN on urban land use, using polarimetric SAR (polSAR) data. Meanwhile, the authors in [38] applied a stacked SAE to PolSAR image classification and those in [39] applied adaptive boosting of restricted Boltzmann machines to PolSAR image classification. CNNs are also extremely popular in the literature for land cover classification approaches. The authors in [40] applied CNNs to PolSAR image classification. In [41], the authors replace pooling layers in their CNN with wavelet-constrained pooling layers. The Neural Network is then used in conjunction with superpixels and a Markov random field to produce the final segmentation map.

One important thing to note is that remote sensing data are often provided in different modalities, spectrally, spatially and temporally inhomogeneous. As a result, data fusion is becoming of significant importance in remote sensing classification. Data fusion is based on the exploitation of complementary information of data originating from different sources. Current works of remote sensing dealing with DNNs and data fusion mostly revolve around two strategies: early fusion and late fusion. Early fusion uses multiple modalities simultaneously in a Neural Network, usually by concatenating extracted features of each data source into a single data cube. The authors in [42] extract different types of spatial filters and concatenate them in a single tensor, which is then used to learn a supervised stack of AEs. Furthermore, the authors in [43] train a two-stream CNN that process PolSAR and hyperspectral data in parallel while fusing the resulting information at a later convolutional layer.

On the other hand, late fusion approaches train a separate classifier for each modality present in a dataset. Afterwards, the individual classification scores are fused into a final classification score. Moreover, these approaches are able to fuse predictions from different models, producing more robust and accurate results. For example, the authors in [44] fuse predictions obtained by a CNN with ones obtained by a random forest classifier, trained using user-defined features. However, the authors in [45] propose a different approach in late fusion, in which they fuse two decision maps obtained by pre-trained models by learning a fusion network based on residual learning.

For the case of remote sensing observations, fusion of observations from different modalities for land use and land cover classification was introduced very recently, due to the lack of appropriate datasets that could be used for training machine learning models. One such case is the dataset provided in the context of the data IEEE GRSS 2018 Data Fusion Contest which provided RGB, HSI and LiDAR measurements, as well as land cover/use annotation [46]. Unlike the proposed work, the authors in [46] do not consider distributed training of deep learning models. Furthermore, in the proposed CNN, the fusion amounts to majority voting of modality-specific classifier and not a unified end-to-end trainable network as it is in this work, while leads to lower classification accuracy.

Last, we note a very recent work which explores in great detail the benefits of distributed deep learning for remote sensing applications [8]. In that work the authors explore the potential of typical (data parallelism) distributed deep learning training on remote sensing scene classification. This framework is significantly different to this work where we explore the potential of optimizing the training processing in distributed processing environment for the case of multi-modal land cover classification.

3. Proposed Methodology

3.1. Overall CNN Architecture

In this work, we consider a CNN architecture that efficiently combines features from multiple modalities by employing a two-stream CNN architecture for assigning each pixel to a specific class. In order to capture spatial characteristics from neighboring pixel, instead of selecting a single pixel as the input, a spatial neighborhood is selected, centered on the query pixel, is used.

In general, a state-of-the-art CNN architecture is composed of a sequence of layers performing specific operations. Representative types of layers, which are also considered in this work are:

- Convolution layer:
- Activation layer. Activation layers introduce non-linear operations which enable the networks to act a universal function approximates. Representative examples are ReLU activation, which applies a non-negative thresholding operator, and Sigmoid, which constraints the output to the range $[0, 1]$ and is typically used in the last layer as the predictor.
- Batch normalization layer. This layer add a normalization step to the network, which make the inputs of each trainable layers comparable across features. By doing this, they ensure a high learning rate while keeping the network learning.
- Pooling layer. A pooling layer's objectives are firstly, to provide invariance to slightly different inputs and secondly, to reduce the dimensionality of feature maps. Max Pooling is most frequent choice, as it avoids cancellation of negative elements, and prevents blurring of activations and gradients throughout the network.
- Dropout layer. Dropout is a most popular techniques for limiting the effects of overfitting by temporarily reducing the total parameters of the network at each training epoch.
- Fully connected layer. A fully Connected layer is responsible for producing predictions, while taking into account the input features.

The network architecture considered in this work consists of a sequence of convolutional and pooling layers in order to extract features, and dropout and batch normalization layers to avoid overfitting. These are followed by two fully connected layers and a final softmax scoring layer. In order to construct a more memory efficient DNN, we adopted an early fusion approach, fusing the features before the first fully connected layer. Specifically, the proposed multi-modal CNN consists of three modules, one for processing HSI data, one for processing VHR-RGB data, and one for feature fusion and classification. The first two modules apply 3D convolutional layers that operate on their respective input patches to generate features. We decided on three dimensional convolution, since our hyperspectral data also contain spatial information that 2D models fail to accurately encode.

Due to the fact that CNNs require a considerable amount of data in order to ensure that sufficient generalization is achieved during training, the following preprocessing steps are applied in our dataset before going ahead with any learning or classification. Firstly, the four separate VHR-RGB tiles had to be merged, as well as resized along with the HSI image, to match the spatial size of the ground truth data. Then, image patches of size 25 × 25 pixels were cropped out by sliding through each training image. The generated patches contain the same pixels from both images, in order to get complementary information. Furthermore, for each class, we randomly chose 400 distinct patches from each image with a selected pixel as a starting point, resulting in overall 8000 (8 K) patches. The proposed processing pipeline is depicted in Figure 1 while the specific parameters of the network are shown in Figure 2.

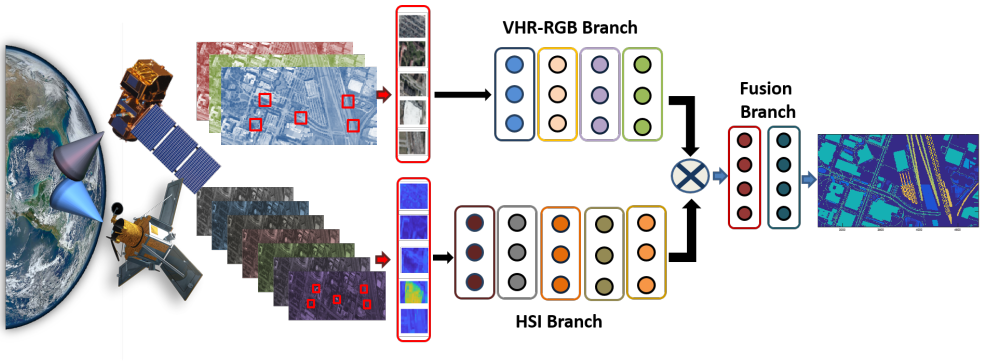

Figure 1. Processing pipeline for the proposed multimodal CNN. The overall network architecture is composed of three parts, one for processing the RGB (top), and one for the HSI (bottom), are are fused into a third part (right) in order to produce the classification results.

VHR branch
Conv3D: #filter =2 + ReLU
Conv3D: #filter =4 + ReLU
Max-Pooling
Batch Normalization
Conv3D: #filter =8 + ReLU
Conv3D: #filter =8 + ReLU
Max-Pooling
Batch Normalization
Conv3D: #filter =16 + ReLU
Conv3D: #filter =16 + ReLU
Max-Pooling
Batch Normalization
Conv3D: #filter =32 + ReLU
Conv3D: #filter =64 + ReLU
Max-Pooling
Batch Normalization
Conv3D: #filter =128 + ReLU
Conv3D: #filter =256 + ReLU
Conv3D: #filter =512 + ReLU

HSI branch
Conv3D: #filter =8 + ReLU
Conv3D: #filter =16+ ReLU
Max-Pooling
Batch Normalization
Conv3D: #filter =16+ ReLU
Activation – ReLU
Conv3D: #filter =32+ ReLU
Max-Pooling
Batch Normalization
Conv3D: #filter =32+ ReLU
Conv3D: #filter =64+ ReLU
Max-Pooling
Batch Normalization
Conv3D: #filter =128+ ReLU
Conv3D: #filter =256+ ReLU

Merge(VHR,HSI)
Flatten
Dropout(0.9)
Dense: #filter =1000 + ReLU
Dense: #filter =500 + ReLU
Dense: #filter =20 +
Activation – SoftMax

Figure 2. Parameters of the proposed multi-modal classification network. Conv3D corresponds to a 3D convolution, with the provided number of filters, followed by a ReLU non-linear activation. Dense indicates a fully connected layer with the provided number of filters, followed by a ReLU activation function.

3.2. Distributed CNN Architectures

Figure 3 presents a high level block diagram of the two different schemes, showcasing the access to the same data by all workers for the case of data parallelism and different data for the proposed model parallelism scheme. To quantify the performance gains associated with each methods, we explore the differences in terms of both processing time and accuracy.

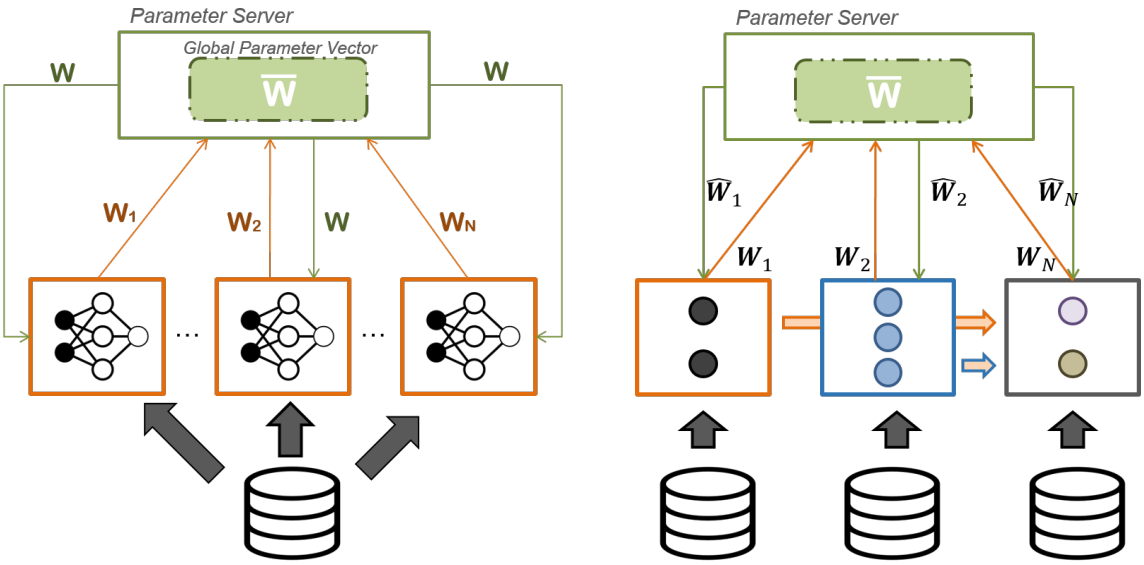

Figure 3. Data (left) and model (right) parallelization schemes. W_i indicate the network weights associated with each worker i while $\bar{W}$ indicates the process of averaging the weights collected by all workers at the parameter server. For tee case of data parallelism, once the averaging is executed, the same weights W are distributed to all workers. For the case of model parallelism, each worker obtains the portion of weights $\hat{W}_i$ corresponding to its local section of the network.

In order to fully exploit our cluster resources, we consider two different distributed deep learning architectures, based on the multimodal CNN described in Section 3.1. As a result our CNNs are consisted of three branches, one for each modality and a final one for fusion and classification. For our initial architecture, we adopted a baseline approach, in which we assign each branch to a different worker, as illustrated in Figure 4.

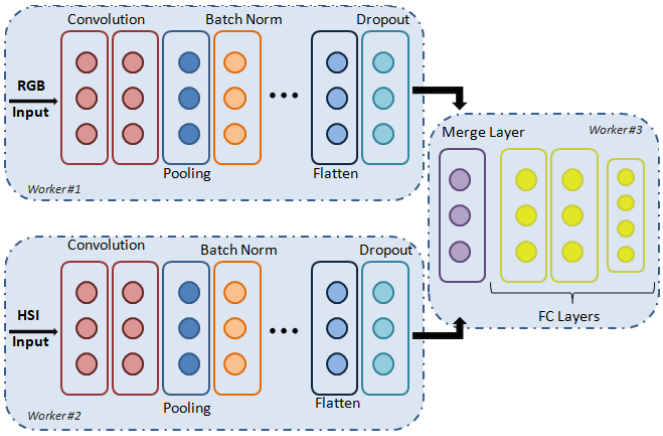

Figure 4. The **baseline** architecture of proposed distributed multimodal CNN consists of three main parts, each assigned to a different worker: The VHR-RGB part at the top branch, the HSI part at the bottom branch and merging part that leads to classification output is shown on the right.

For the second architecture, we took into consideration the traffic loads of our network. Thus, we modified our previous model distribution into the one depicted in Figure 5. In this approach we migrated the last convolutional-pooling block along with flatten and dense layers of both RGB and HSI branches. The rest of the layers remained on the same machines as before.

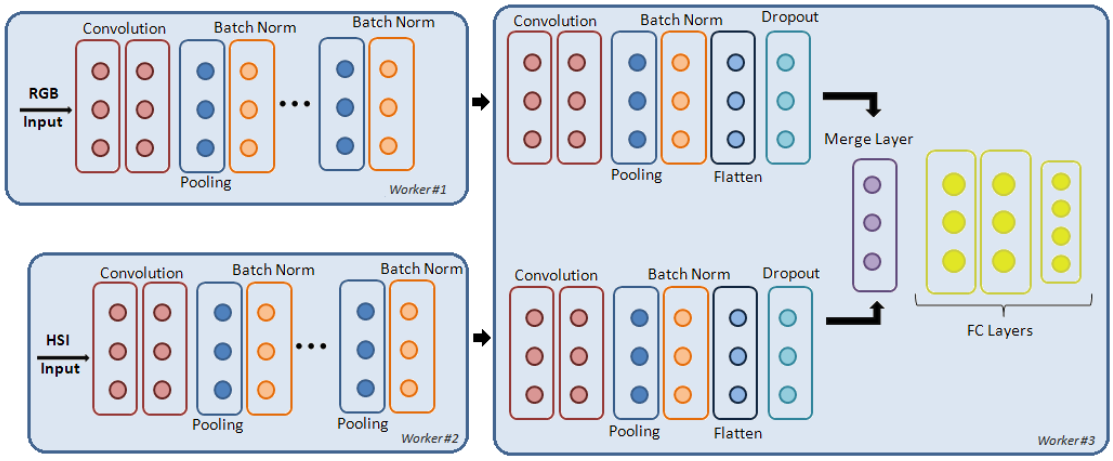

Figure 5. The **optimized** architecture. The architecture is similar to the "baseline" case, however, intermediate results from convolution layers are communicated amongs workers.

4. Distributed Processing Framework

4.1. System Overview

The model parallel architecture considered in this work is based on TensorFlowOnSpak (https://github.com/yahoo/TensorFlowOnSpark) [47], for managing TensorFlow-based distributed DNN workflows on Apache Spark clusters [15]. Apache Spark extends the MapReduce model [16] by the use of an elastic persistence model, which provides the flexibility to persist data records, either in memory, on disk, or both. Therefore, Spark favors the iterative processes met in machine learning and optimization algorithms. Briefly, Spark organizes the underlying infrastructure into a hierarchical cluster of computing elements, comprised of a Master and a set of N Workers. The Master is responsible for the configuration of the cluster, while the workers perform the learning tasks submitted to them through a driver program, along with an initial dataset.

TFoS exploits Spark's native mechanisms for automatic cluster configuration while bypassing its communication philosophy, thereby allowing for direct tensor communication between TF processes. This framework also implements the concept of Parameter Server (PS) [23] for enabling the smooth integration of TensorFlow code over Spark clusters, with minimum changes on the original TensorFlow code. In a data parallelization approach, PS is responsible for providing a global average of network parameters, while the rest of the nodes are responsible for training, whereas in a model parallelization approach PS is responsible for distributing parts of a model to the nodes as well as acting as a training coordinator between the nodes.

4.2. Cluster Setup

To benchmark our proposed distributed DNN architectures, the experiments were conducted on an Apache Spark Standalone cluster of 5 PCs, featuring TensorFlow Core version 1.2.1, Apache Spark version 2.1.1, Apache Hadoop version 2.6, and TensorFlowOnSpark version 1.0.0. The DNN architectures were implemented using Keras (F. Chollet, Keras. https://github.com/fchollet/keras, 2015.) version 2.0.4, a high level neural network API written in Python. The Master of the cluster is configured with an Intel Core i7-6700 3.40 GHz CPU, and has allocated 8 GB memory and 450 GB disk

space. Meanwhile, the workers are configured with Intel Core i5-3470 3.20 GHz CPUs, have allocated 2 GB memory and 450 GB disk space.

Figure 6 presents our cluster setup. It depicts Spark nodes connected via Ethernet. The Master configures the cluster through the driver, and Workers communicate through TFoS tensor communication. All nodes run Ubuntu Linux 16.04. For the purpose of identifying our system's bottlenecks, Ganglia is used to monitor CPU, memory, and Network traffic on every node [48].

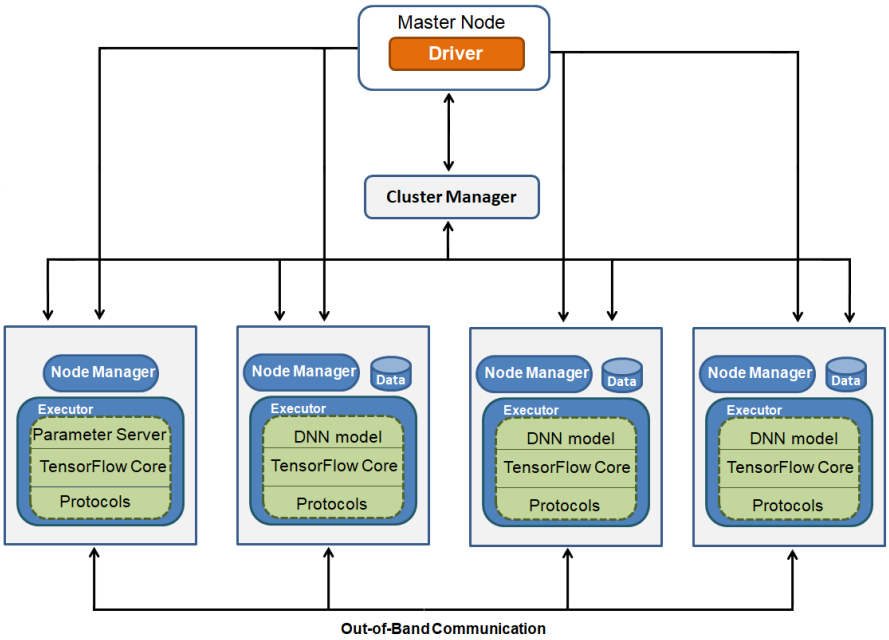

Figure 6. FoS Cluster Setup. The cluster consists of five machines, namely one master node responsible to managing the cluster, one node acting as the parameter server (lower left) and three worker nodes. Each worker node has access to a local version of the dataset, and a executer which implements the training/inference of the DNN. The Parameters does not access data, and is tasked with organising the training/inference process.

5. Experimental Analysis

5.1. Dataset

The dataset used for our experiments was collected by NCALM at the University of Houston on 16 February 2017, covering the University of Houston campus and its surrounding areas. The dataset contains images of multi-source optical remote sensing data, containing (a) hyperspectral (HSI) and (b) Very High Resolution RGB imagery (VHR-RGB).

Specifically, the HSI observations were acquired by a ITRES CASI 1500 sensor, covering a 380–1050 nm spectral range with 48 bands at a 1-m GSD. The observations were orthorectified and radiometrically calibrated to units of spectral radiance. The VHR-RGB observations were acquired by a DiMAC ULTRA-LIGHT+ camera with a 70-mm focal length and post-acquisition processed through white balance, calibration with respect to plane instruments, and orthorectified. The entire image is organized into four separate tiles, resulting in an image of size 12,020 $\times$ 47,680 pixel, when concatenated. A Ground truth image presented in Figure 7 was also provided as raster at a 0.5 m GSD, and it is corresponding to 20 urban land use and land cover classes, listed in Table 1. This dataset was developed in the context of the 2018 IEEE GRSS Data Fusion Contest "Advanced multi-sensor optical remote sensing for urban land use and land cover classification" [49].

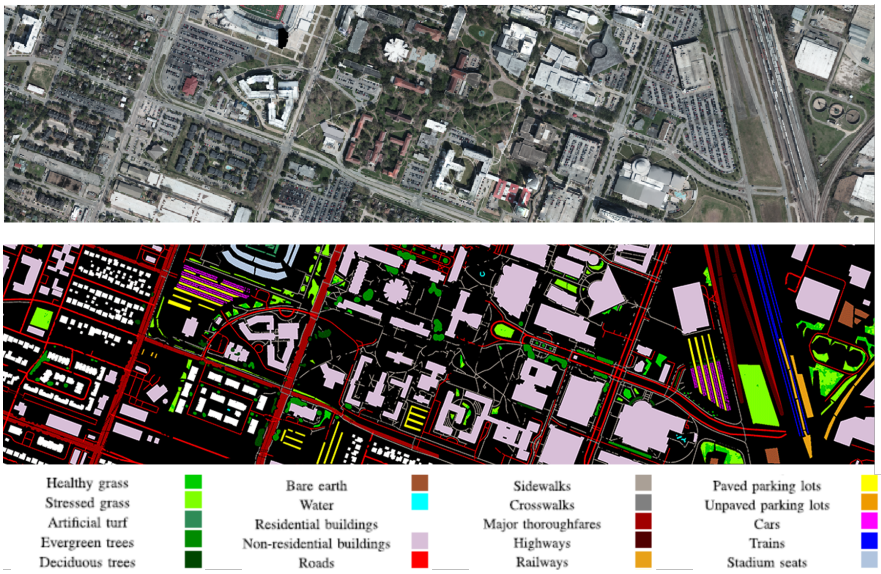

Figure 7. Exemplary RGB (top) and land cover annotation (bottom).

Table 1. The original set of class and the corresponding description.

Class ID	Corresponding Label
0	Healthy Grass
1	Stressed Grass
2	Artificial Turf
3	Evergreen Trees
4	Deciduous Trees
5	Bare Earth
6	Water
7	Residential Buildings
8	Non-Residential Buildings
9	Roads
10	Sidewalks
11	Crosswalks
12	Major thoroughfares
13	Highways
14	Railways
15	Paved Parking Lots
16	Unpaved Parking Lots
17	Cars
18	Trains
19	Stadium Seats

5.2. Result Evaluation

In this section, the results of our experiments are presented and discussed, in order to illustrate the performance of the model parallelization schemes introduced in this paper. More specifically, we first investigate the impact training under a distributed scenario in Section 5.2.2 and subsequently investigate different parameters including the batch size in Section 5.2.3. In Section 5.2.4, we consider the two different schemes for model-parallelism training of the proposed multi-modal CNN and report the use of resources associated with each approach. Last, in Section 5.3, we consider the case of model and data parallelism for the process of inference and compare these two approaches in terms of achieved processing time and resource use.

The proposed multi-modal CNN architecture was trained to minimize the categorical cross-entropy using the Adam optimization algorithm [50] with learning rate set to 10^{-3} and batch size equal to 40. For both training and validation all available bands were used, namely 48 for the

HSI and 3 for the VHR-RGB (see Figure 2). Furthermore, we also compare the proposed scheme with state-of-the-art classification methods for the single machine case. Specifically we consider the K-Nearest Neighbors, using 5 neighbor and Euclidean distance and Support Vector Machines (SVM) using the radial basis function as the kernel and the one-versus-all strategy for multi-class classification. We note that for the cases of the KNN and SVM, the two signals, namely the patch from the HSI and the one from the VHR-RGB are vectorized and concatenated before introduced to the classifiers.

5.2.1. Performance of the Proposed Multi-Modal CNN Architecture

We begin the experimental analysis by first exploring the learning capabilities of the proposed multi-modal CNN on land cover classification, on a single machine. Figure 8 present the accuracy and the loss achieved as a function of training epoch using 8000 examples, equally representing each class, as the training set. The results indicate that the proposed scheme is capable of achieving high accuracy, even from a limited number of training epochs, without demonstrating signs of overfitting.

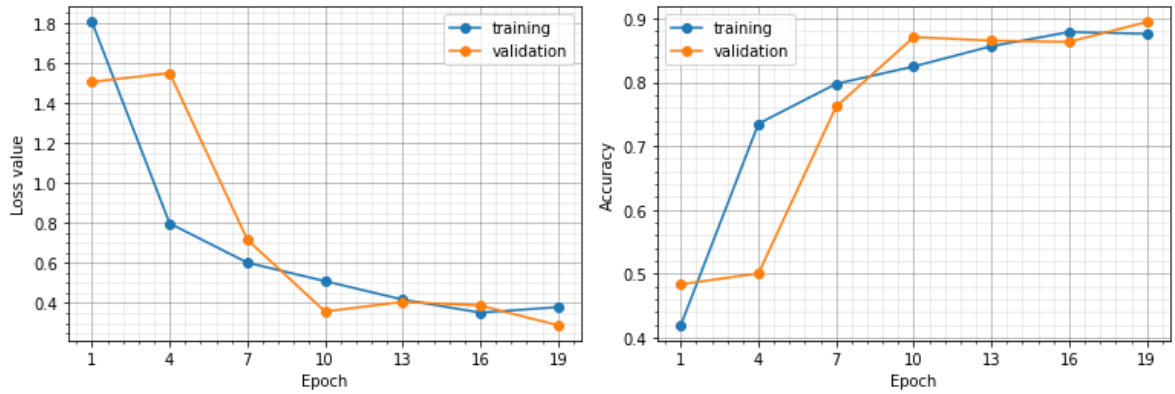

Figure 8. Training and validation set (left) loss function value, (right) overall accuracy.

In order to understand the impact of each modality on the final classification accuracy, Table 2 presents the accuracy achieved in both training and validation set for three scenarios, namely using RGB only, HSI only and using both sources of observations. We observe that in both training and validation cases, the best accuracy is achieved when both source of observations are simultaneously exploiting, justifying the motivation of this work for multi-modal data analytics. With regards to the individual data source, we observe that the RGB case exhibits very strong indications of overfitting, a phenomenon not observed for the case of the HSI.

Table 2. Accuracy for training and validation set using individual and fused data sources.

	Accuracy (%)	
	Training	**Validation**
RGB	73.3	43.1
HSI	75.5	77.6
RGB + HSI	88.4	83.6

Table 3 provides a comparison between the proposed methods and non-deep learning classifiers, specifically the SVM with the rbf kernel and the K-Nearest Neighbors. These results demonstrate that the proposed Multi-Modal (MM) CNN achieves the best performance, closely followed by the KNN, which however, is difficult to parallelize, which is the main objective of this work.

Table 3. Validation set accuracy for state-of-the-art and proposed methods.

	Accuracy (%)
SVM (rbf)	74.1
KNN (k = 5)	82.4
MM-CNN (proposed)	83.6

5.2.2. Effects of Distributed Environments on Training Time

In this section we compare the performance of the (baseline) distributed architecture with an identical multimodal CNN architecture deployed on a single machine setup. Figure 9 illustrates the performance of each architecture, when trained for the same amount of time.

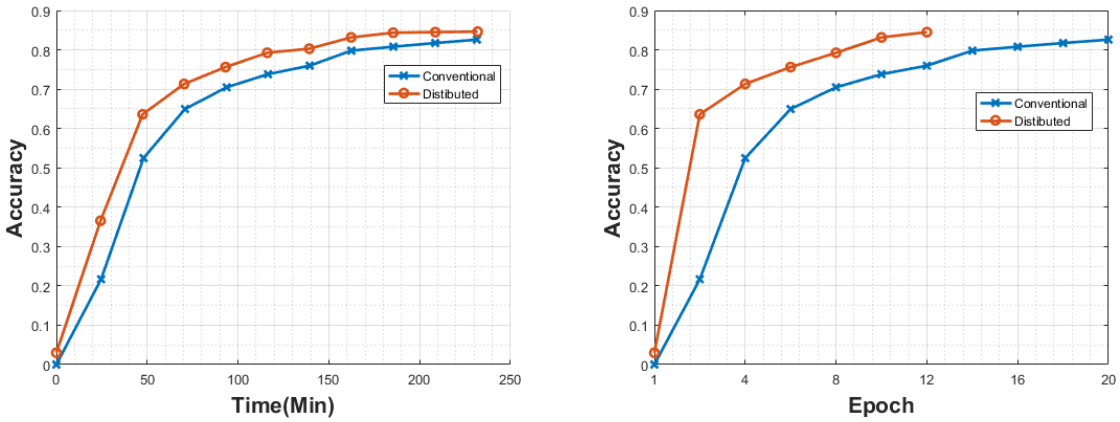

Figure 9. Training Accuracy with respect to elapsed training time (left) and training epoch (right).

The results demonstrate the advantage of the distributed architecture in terms of training accuracy, in contrast to the conventional single-processing-unit approach. Specifically, while both implementations achieve comparable accuracy, the distributed training architecture attains higher accuracy in less time as shown in Figure 9 right. Furthermore, for the same amount of time, the distributed processing scheme achieves comparable accuracy to the conventional scheme from a smaller number of epochs as shown in Figure 9 right. The reason while the distributed scheme completes a smaller number of epochs compared to the single-processing-unit is that in addition to processing, the distributed processing scheme must also accommodate communication transfer delays.As a result, a large part of the gains achieved by the distributed allocation of resources are lost due to communications overheads. We note that in both cases, the processing units are CPU which is not optimized for CNN training and therefore, significantly higher gains could potentially be achieved if GPUs were used.

5.2.3. Impact of Batch Size

The objective of this set of experiments was to examine if and how the hyperparameter of batch size affects model parallelization, motivated by the fact that since batch size defines the number of samples that will be propagated through the network, it will by extension affects the amount of data transfer over the network between different machines.

Figure 10 depicts both the incoming (left) and outgoing (right) traffic of the cluster for different batch sizes, during a time period of about 70 min, for the case of the 'baseline' architecture. These plots indicate that for a batch size of 10, network traffic can reach values as high as 120 MB/s compared to the batch sizes of 50 and 200, which get values more than 5 times lower. As a result, high batch sizes can greatly accelerate the training process.

It should however be noted that by using a very large batch can lead to a significant degradation in terms of the model capabilities, as measured by its ability to generalize. This occurs due to the fact that large-batch methods tend to converge to local minima [51]. In conclusion, larger batch sizes can be beneficial for model parallelism approaches since they greatly reduce network overheads, as long as they do not negatively impact the model's convergence.

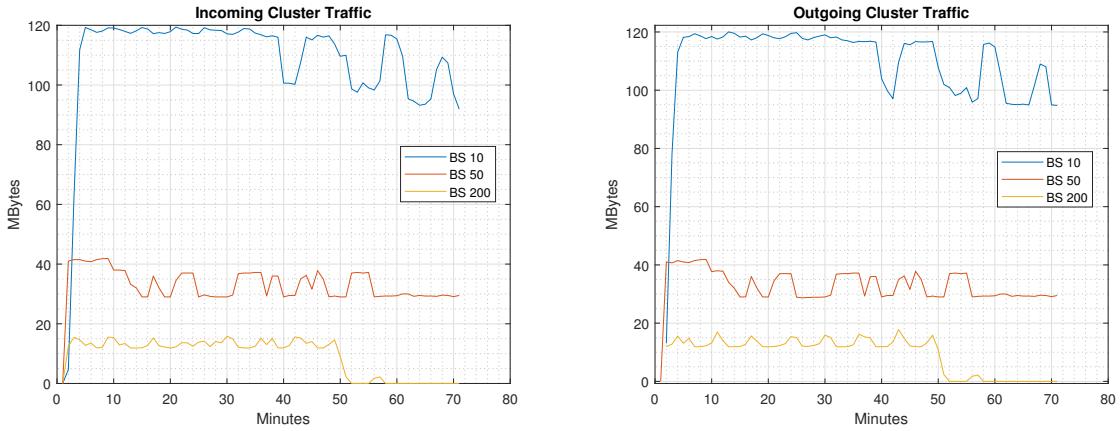

Figure 10. Incoming and Outgoing cluster traffic of the baseline architecture for different Batch Sizes.

5.2.4. Impact of Architectures on Resource Utilization

Even though batch size adjustments can significantly reduce communication overheads, further improvements are feasible by considering different allocations of the processing pipeline among workers, through different architectures. Figures 11 and 12, present the incoming and outgoing network traffic for the two architectural scenarios considered in the work, namely the 'baseline', and the 'optimized' respectively. In these plots, the horizontal axis corresponds to the individual worker, the diagonal axis to the sample batch and the vertical to the traffic measured in MBytes per second. From these plots, several observations can be made.

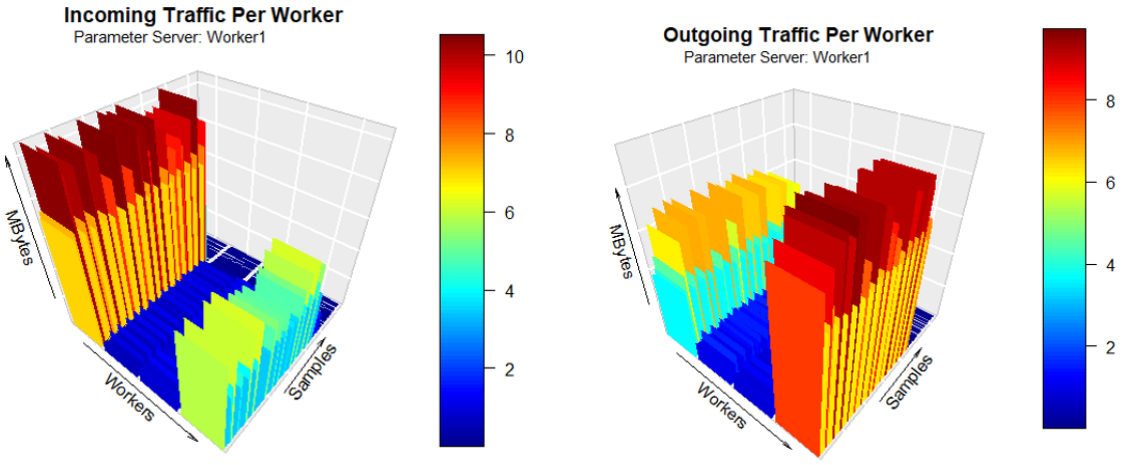

Figure 11. Communication traffic per worker for the 'baseline' architecture.

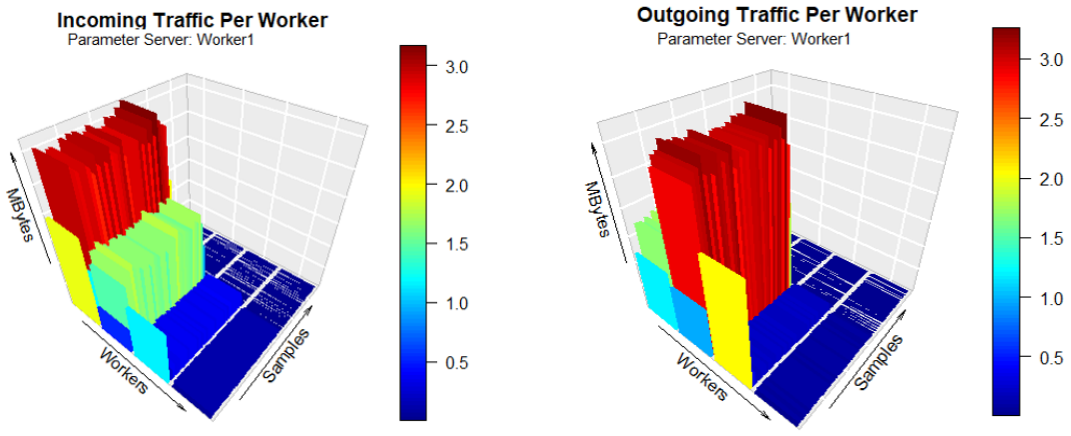

Figure 12. Communication traffic per worker for the 'optimized' architecture.

The first observation one can make is that for the case of the baseline architecture, bulk of the traffic is actually associated with two workers, worker 1 which acts as the PS, and worker 4 which is responsible for the merging and classification part of our multimodal CNN. The high incoming network traffic on the worker 4 is expected since it is responsible for the merging stage and thus need to collect data from both workers 2 and 3. It also expected that PS (worker 1) will have the highest incoming traffic, as it receives data form a branch consisted mostly of fully connected layers, which have much more parameters than the convolutional ones. The cluster's behavior also makes sense for outgoing traffic. In this case, worker 4 has the highest traffic compared to other workers of the cluster, since, as stated before, is responsible for the part of the model with the fully connected layers, and therefore has more parameters to send. The PS also exhibits high outgoing traffic since it communicates the parameters received from the fully connected layers to both workers 2 and 3.

Comparing the two architectures, we observe that the 'optimized' architecture is characterized by significantly less incoming and outgoing traffic compared to the 'baseline' architecture. More specifically, the outgoing traffic of the merging worker can be as low as 2.5 MB/s compared to the baseline one, where it reaches about 10 MB/s. As a result, traffic on the PS is significantly reduced, reaching about 2.5 MB/s for the incoming and 1.5 MB/s for the outgoing traffic. We also observe that for the case of the outgoing traffic, worker 2 exhibits the highest amount of traffic compared to the rest. In fact, the amount of outgoing traffic for the worker is approximately the same as the case of the 'baseline' architecture, but become significantly more pronounced due to the drop in traffic of the other workers. Overall, for both incoming and outgoing traffic, we can clearly see the significant benefits offered by the 'optimized' architecture compared to the 'baseline'.

In addition to network traffic, we also consider the use of available computational resources, measured in terms of CPU use. Figure 13 illustrates the performance of our cluster in terms of CPU usage, when executing the 'baseline' and 'optimized' approach respectively. From this figure, we can make the observation that both PS (worker 1) and the main classification machine (worker 4) do not use significant amounts of CPU, since the former just redistributes parameters to the rest of the cluster, and the latter is in charge of only a small number of layers. On the other hand, worker 3, which is responsible for the HSI branch, requires significantly more computational resources, reaching around 70% of CPU of the 'baseline' and 50% for the 'optimized' on average.

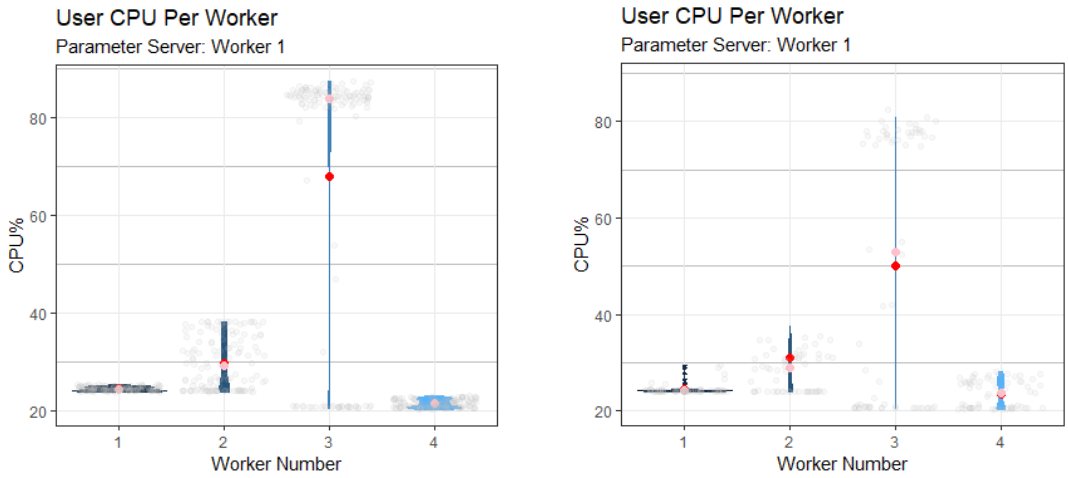

Figure 13. CPU Usage of the baseline (left) and Traffic Based (right) architectures. Red dots indicate mean value, while pink dots indicate median. The continuous lines encode the standard deviation around the mean.

5.3. Training Using Model and Data Parallelization Comparison

The experimental results reported in the previous sections demonstrated the impact that a carefully designed architecture can have on the use of resources. In this section, we study and compare the performance characteristics of the 'optimized' model parallelism architecture with the more traditional data parallelism approaches, during both training and inference stages. We compared the data and model parallelism scheme with respect to two critical system variables, namely network traffic and associated CPU load.

Regarding the network traffic, Figures 14 and 15 demonstrate that for the data parallelization scheme, a significant amount of traffic is associated with PS since its is primarily responsible for synchronizing the different workers, while all workers exhibit similar network traffic characteristics. On the other hand, for the proposed model parallelization scheme, the traffic associated with PS is significantly lower compared to the data parallelization scheme, at the same level as the other workers. Furthermore, worker #3, the one associated with the fusion, and thus the majority of the fully connected layers, has higher network traffic compared to the other workers, nevertheless, it is still less compared to the traffic for the data parallelization scheme.

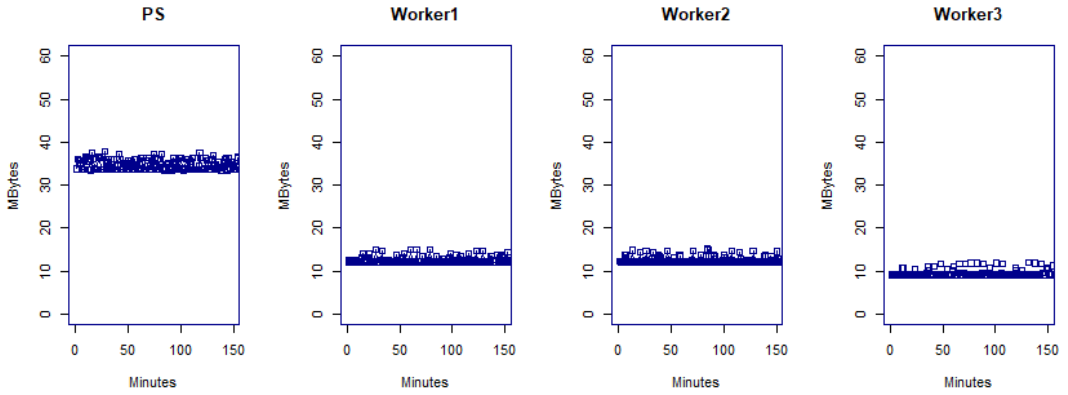

Figure 14. Incoming traffic per worker for the **data** parallelism scheme.

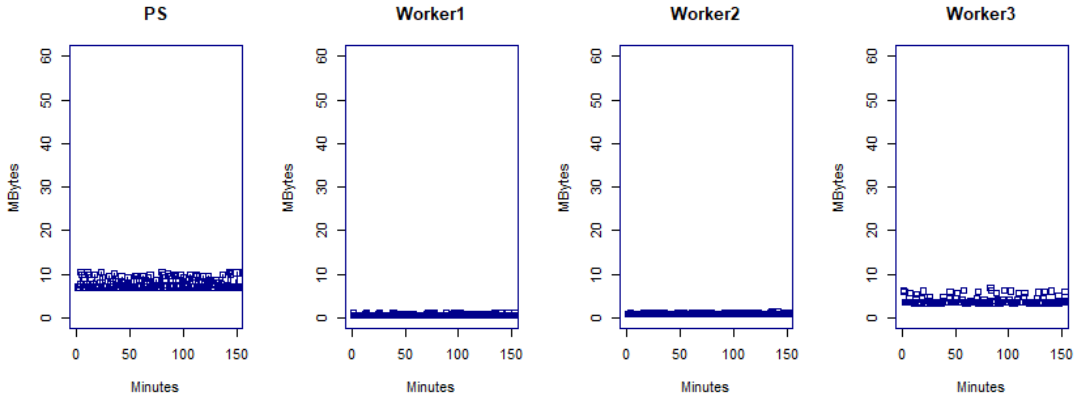

Figure 15. Incoming traffic per worker for the **model** parallelism scheme.

Regarding the CPU load for the data and model parallelism schemes shown in Figures 16 and 17, a first observation is that in both data and model parallelism scheme, the PS worker is associated with the lowest CPU load, although based on the previous discussion, it is associated with the bulk of the network traffic. As expected, for the case of data parallelism, all three workers exhibit the same CPU load, which is in the order to 90% percent. On the other hand, the majority of processing for the model parallelism scheme is associated with worker #3, the one responsible for the feature fusion and classification using fully connected layers.

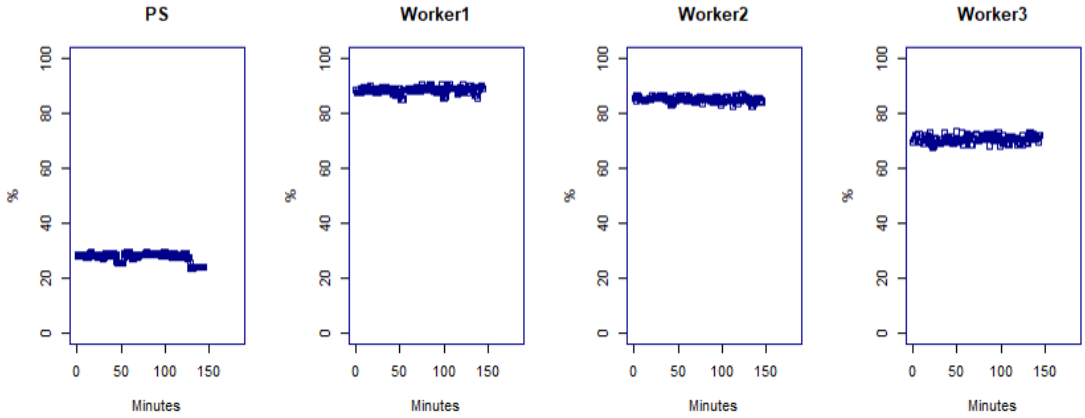

Figure 16. CPU load per worker for the **data** parallelism scheme.

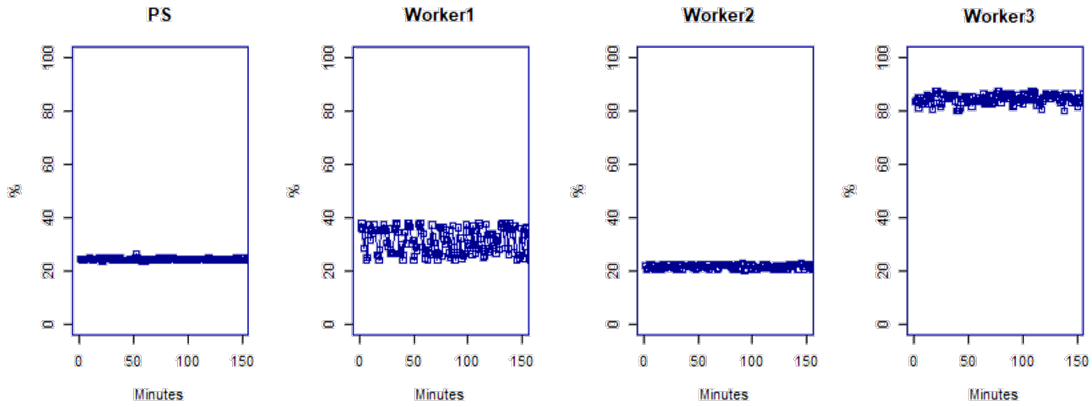

Figure 17. CPU load per worker for the **model** parallelism scheme.

5.4. Inference Using Model and Data Parallelization Comparison

In order to derive convincing conclusion, we consider the inference process of the trained networks using 1000 (1 K) and 14,000 (14 K) testing examples. For both parallelization approaches, we trained each model using an identical dataset of 8K patches and performed inference with the aforementioned datasets. Table 4 presents the results of our experiments with regards execution time, while Table 5 illustrates inference accuracy for each parallelization scheme. We note that for the data parallelization scheme, the entire testing dataset was split among the workers and executed in parallel.

Table 4. Evaluation time comparison between the model parallelization and the 'baseline' and 'optimized' data parallelization schemes.

# Examples	Data Parallelization	Model Parallelization	
		Baseline	Optimized
1 K	51.7 s	26.0 s	25.1s
14 K	787.2 s	500.1 s	364.5s

Table 5. Accuracy comparison between data parallelization and each one of our proposed model parallelization architectures.

# Examples	Data Parallelization	Model Parallelization	
		Baseline	Optimized
1 K	0.86	0.86	0.86
14 K	0.83	0.83	0.84

The results in Tables 4 and 5 demonstrate the clear superiority of the model parallelism scheme compared to the data parallelism scheme in terms of both time required for inference. In terms of the processing speed, i.e., the execution time, we observe that both model parallelization schemes are able to perform the inference at significantly less time compared to the data-splitting scheme employed in data parallelization. Between the two model parallelization schemes, there is a clear benefit in using the optimized scheme, especially when dealing with larger amounts of data, in which can the optimized scheme is 27% faster.

Furthermore, Table 5 suggests that even though inference accuracy between both parallelization approaches is similar for both datasets, model parallelization provides slightly better results. Moreover, we observe that the optimized scheme can achieve superior performance for the more 14 K examples case. These results clearly indicate that the optimized model parallelization scheme is the preferable approach in the case of inference, since it can make faster and more reliable predictions.

6. Conclusions

In this paper, we proposed a multimodal CNN for land cover classification and performed model parallelization architectures, in order to improve its performance on a distributed environment. Moreover, we compared our most efficient architectures with the popular data parallelization approach. We conducted experiments with patches extracted from RGB and HSI images, and our results provide the following contributions:

- Migrating branches of multimodal CNNs to different machines does not affect the model's learning capacity, and a distributed deep learning system can show similar behavior to traditional setups.
- The number of samples that will be propagated through the network greatly affects communication overheads between workers of a cluster. Model Parallelization approaches benefit from larger batch sizes, since they significantly reduce network traffic. However, a very big number of samples causes significant degradation in the quality of the model.

- Through this study, we gave insight on model parallelization's behavior, and as a result, provide directions for better resource allocation on commodity clusters. For instance, since the PS is the most memory consuming module of model parallelism, we can assign it to the machine with the most memory available. Or because the merging part of a multimodal CNN accumulates a lot of traffic, we can assign it to the machine with the most expensive networking cables.
- The way we parallelize our model has a tremendous impact on a cluster's resource allocation. Smart model split can reduce the load of heavy duty workers and put to use otherwise idle resources, while a naive split can cause severe bottlenecks through heavy network traffic and waiting learning tasks due to a lack of non-allocated resources.
- For land cover classification, model parallelization schemes tend to perform better than data parallelization since they provide significantly faster and slightly more accurate predictions.

As a last point, we should note that although distributed training is highly recommend for training a DNN, testing/inference may be more efficiently executed using more appropriate hardware platform. For instance, platforms such as Field Programmable Gate Arrays (FPGAs) appear offer much more favorable characteristics for inference compared to more typical GPU platforms [52].

Author Contributions: G.T. developed the methodology; M.A. did the experimental analysis and provided the original draft; G.T. and P.T. did the review and editing of the final document. All authors have read and agreed to the publisher version of the manuscript.

Funding: This research work work was partially funded by the Marie Skłodowska-Curie CALCHAS project (contract no. 842560) within the H2020 Framework Program of the European Commission and by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under HFRI Faculty grant no. 1725 (V4-ICARUS).

Acknowledgments: The authors would like to thank the National Center for Airborne Laser Mapping and the Hyperspectral Image Analysis Laboratory at the University of Houston for acquiring and providing the data used in this study, and the IEEE GRSS Image Analysis and Data Fusion Technical Committee.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
EO	Earth Observation
GPU	Graphical Processing Unit
GSD	Ground Sampling Distance
HSI	Hyperspectral Image
PS	Parameter Server
RGB	Red-Green-Blue
VHR	Very High Resolution

References

1. Guo, H.; Liu, Z.; Jiang, H.; Wang, C.; Liu, J.; Liang, D. Big Earth Data: A new challenge and opportunity for Digital Earth's development. *Int. J. Digit. Earth* **2017**, *10*, 1–12. [CrossRef]
2. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
3. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–8 December 2012; pp. 1097–1105.
4. Sharma, A.; Liu, X.; Yang, X.; Shi, D. A patch-based convolutional neural network for remote sensing image classification. *Neural Netw.* **2017**, *95*, 19–28. [CrossRef]
5. Tsagkatakis, G.; Aidini, A.; Fotiadou, K.; Giannopoulos, M.; Pentari, A.; Tsakalides, P. Survey of deep-learning approaches for remote sensing observation enhancement. *Sensors* **2019**, *19*, 3929. [CrossRef]

6. Graves, A.; Mohamed, A.R.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 6645–6649.
7. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2818–2826.
8. Sedona, R.; Cavallaro, G.; Jitsev, J.; Strube, A.; Riedel, M.; Benediktsson, J.A. Remote Sensing Big Data Classification with High Performance Distributed Deep Learning. *Remote Sens.* **2019**, *11*, 3056. [CrossRef]
9. Yao, X.; Li, G.; Xia, J.; Ben, J.; Cao, Q.; Zhao, L.; Ma, Y.; Zhang, L.; Zhu, D. Enabling the Big Earth Observation Data via Cloud Computing and DGGS: Opportunities and Challenges. *Remote Sens.* **2020**, *12*, 62. [CrossRef]
10. Gomes, V.C.; Queiroz, G.R.; Ferreira, K.R. An Overview of Platforms for Big Earth Observation Data Management and Analysis. *Remote Sens.* **2020**, *12*, 1253. [CrossRef]
11. Gaunt, A.L.; Johnson, M.A.; Riechert, M.; Tarlow, D.; Tomioka, R.; Vytiniotis, D.; Webster, S. AMPNet: Asynchronous model-parallel training for dynamic neural networks. *arXiv* **2017**, arXiv:1705.09786.
12. Chahal, K.S.; Grover, M.S.; Dey, K.; Shah, R.R. A hitchhiker's guide on distributed training of deep neural networks. *J. Parallel Distrib. Comput.* **2020**, *137*, 65–76. [CrossRef]
13. Yu, X.; Wu, X.; Luo, C.; Ren, P. Deep learning in remote sensing scene classification: A data augmentation enhanced convolutional neural network framework. *GISci. Remote Sens.* **2017**, *54*, 741–758. [CrossRef]
14. Kussul, N.; Lavreniuk, M.; Skakun, S.; Shelestov, A. Deep learning classification of land cover and crop types using remote sensing data. *IEEE Geosci. Remote Sens. Lett.* **2017**, *14*, 778–782. [CrossRef]
15. Zaharia, M.; Xin, R.S.; Wendell, P.; Das, T.; Armbrust, M.; Dave, A.; Meng, X.; Rosen, J.; Venkataraman, S.; Franklin, M.J.; et al. Apache spark: A unified engine for big data processing. *Commun. ACM* **2016**, *59*, 56–65. [CrossRef]
16. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
17. Zhang, K.; Chen, X.W. Large-scale deep belief nets with mapreduce. *IEEE Access* **2014**, *2*, 395–403. [CrossRef]
18. Oyama, Y.; Ben-Nun, T.; Hoefler, T.; Matsuoka, S. Accelerating deep learning frameworks with micro-batches. In Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER), Belfast, UK, 10–13 September 2018; pp. 402–412.
19. Bekkerman, R.; Bilenko, M.; Langford, J. *Scaling Up Machine Learning: Parallel and Distributed Approaches*; Cambridge University Press: Cambridge, UK, 2011.
20. Muller, U.; Gunzinger, A. Neural net simulation on parallel computers. In Proceedings of the 1994 IEEE International Conference on Neural Networks (ICNN'94), Orlando, FL, USA, 28 June–2 July 1994; Volume 6, pp. 3961–3966.
21. Ericson, L.; Mbuvha, R. On the performance of network parallel training in artificial neural networks. *arXiv* **2017**, arXiv:1701.05130.
22. Le, Q.V. Building high-level features using large scale unsupervised learning. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 8595–8598.
23. Li, M.; Andersen, D.G.; Park, J.W.; Smola, A.J.; Ahmed, A.; Josifovski, V.; Long, J.; Shekita, E.J.; Su, B.Y.Scaling distributed machine learning with the parameter server. In Proceedings of the 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14), Broomfield, CO, USA, 6–8 October 2014; pp. 583–598.
24. Das, D.; Avancha, S.; Mudigere, D.; Vaidynathan, K.; Sridharan, S.; Kalamkar, D.; Kaul, B.; Dubey, P. Distributed deep learning using synchronous stochastic gradient descent. *arXiv* **2016**, arXiv:1602.06709.
25. Zinkevich, M.; Weimer, M.; Li, L.; Smola, A.J. Parallelized stochastic gradient descent. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 6–9 December 2010; pp. 2595–2603.
26. Jiang, J.; Cui, B.; Zhang, C.; Yu, L. Heterogeneity-aware distributed parameter servers. In Proceedings of the 2017 ACM International Conference on Management of Data, Chicago, IL, USA, 14–19 May 2017; pp. 463–478.

27. Le, Q.V.; Ngiam, J.; Coates, A.; Lahiri, A.; Prochnow, B.; Ng, A.Y. On optimization methods for deep learning. In Proceedings of the 28th International Conference on International Conference on Machine Learning, Bellevue, WA, USA, 28 June–2 July 2011; pp. 265–272.
28. Zhang, S.; Zhang, C.; You, Z.; Zheng, R.; Xu, B. Asynchronous stochastic gradient descent for DNN training. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 6660–6663.
29. Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv* **2014**, arXiv:1404.5997.
30. Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Ranzato, M.; Senior, A.; Tucker, P.; Yang, K.; et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*; Google Inc.: Mountain View, CA, USA, 2012; pp. 1223–1231.
31. Aspri, M.; Tsagkatakis, G.; Panousopoulou, A.; Tsakalides, P. On Realizing Distributed Deep Neural Networks: An Astrophysics Case Study. In Proceedings of the 2019 27th European Signal Processing Conference (EUSIPCO), A Coruna, Spain, 2–6 September 2019; pp. 1–5.
32. Shen, H.; Meng, X.; Zhang, L. An integrated framework for the spatio–temporal–spectral fusion of remote sensing images. *IEEE Trans. Geosci. Remote Sens.* **2016**, *54*, 7135–7148. [CrossRef]
33. Belgiu, M.; Drăguţ, L. Random forest in remote sensing: A review of applications and future directions. *ISPRS J. Photogramm. Remote Sens.* **2016**, *114*, 24–31. [CrossRef]
34. Guo, Y.; Jia, X.; Paull, D. Effective sequential classifier training for SVM-based multitemporal remote sensing image classification. *IEEE Trans. Image Process.* **2018**, *27*, 3036–3048.
35. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
36. Stivaktakis, R.; Tsagkatakis, G.; Tsakalides, P. Deep Learning for Multilabel Land Cover Scene Categorization Using Data Augmentation. *IEEE Geosci. Remote Sens. Lett.* **2019**, *16*, 1031–1035. [CrossRef]
37. Yang, W.; Yin, X.; Xia, G.S. Learning high-level features for satellite image classification with limited labeled samples. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 4472–4482. [CrossRef]
38. Zhang, L.; Ma, W.; Zhang, D. Stacked sparse autoencoder in PolSAR data classification using local spatial information. *IEEE Geosci. Remote Sens. Lett.* **2016**, *13*, 1359–1363. [CrossRef]
39. Qin, F.; Guo, J.; Sun, W. Object-oriented ensemble classification for polarimetric SAR Imagery using restricted Boltzmann machines. *Remote Sens. Lett.* **2017**, *8*, 204–213. [CrossRef]
40. Zhou, Y.; Wang, H.; Xu, F.; Jin, Y.Q. Polarimetric SAR image classification using deep convolutional neural networks. *IEEE Geosci. Remote Sens. Lett.* **2016**, *13*, 1935–1939. [CrossRef]
41. Duan, Y.; Liu, F.; Jiao, L.; Zhao, P.; Zhang, L. SAR Image segmentation based on convolutional-wavelet neural network and markov random field. *Pattern Recognit.* **2017**, *64*, 255–267. [CrossRef]
42. Geng, J.; Wang, H.; Fan, J.; Ma, X. Deep supervised and contractive neural network for SAR image classification. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 2442–2459. [CrossRef]
43. Hu, J.; Mou, L.; Schmitt, A.; Zhu, X.X. FusioNet: A two-stream convolutional neural network for urban scene classification using PolSAR and hyperspectral data. In Proceedings of the Joint Urban Remote Sensing Event (JURSE), Dubai, UAE, 6–8 March 2017; pp. 1–4.
44. Hirschmuller, H. Stereo processing by semiglobal matching and mutual information. *IEEE Trans. Pattern Anal. Mach. Intell.* **2007**, *30*, 328–341. [CrossRef]
45. Audebert, N.; Le Saux, B.; Lefèvre, S. Semantic segmentation of earth observation data using multimodal and multi-scale deep networks. In Proceedings of the Asian Conference on Computer Vision, Taipei, Taiwan, 20–24 November 2016; Springer: Cham, Switzerland, 2016; pp. 180–196.
46. Sukhanov, S.; Budylskii, D.; Tankoyeu, I.; Heremans, R.; Debes, C. Fusion of LiDAR, hyperspectral and RGB data for urban land use and land cover classification. In Proceedings of the IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium, Valencia, Spain, 22–27 July 2018; pp. 3864–3867.
47. Lu, X.; Shi, H.; Biswas, R.; Javed, M.H.; Panda, D.K. DLoBD: A Comprehensive Study of Deep Learning over Big Data Stacks on HPC Clusters. *IEEE Trans. Multi-Scale Comput. Syst.* **2018**, *4*, 635–648. [CrossRef]
48. Massie, M.L.; Chun, B.N.; Culler, D.E. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput.* **2004**, *30*, 817–840. [CrossRef]

49. Xu, Y.; Du, B.; Zhang, L.; Cerra, D.; Pato, M.; Carmona, E.; Prasad, S.; Yokoya, N.; Hänsch, R.; Le Saux, B. Advanced multi-sensor optical remote sensing for urban land use and land cover classification: Outcome of the 2018 IEEE GRSS Data Fusion Contest. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2019**, *12*, 1709–1724. [CrossRef]
50. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015.
51. Keskar, N.S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; Tang, P.T.P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv* **2016**, arXiv:1609.04836.
52. Pitsis, G.; Tsagkatakis, G.; Kozanitis, C.; Kalomoiris, I.; Ioannou, A.; Dollas, A.; Katevenis, M.G.; Tsakalides, P. Efficient convolutional neural network weight compression for space data classification on multi-fpga platforms. In Proceedings of the ICASSP 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; pp. 3917–3921.