Data-parallelism for plant disease detection

1st Tijana Atanasovska
Ss. Cyril and Methodius University in Skopje
Faculty of Computer Science and Engineering
Skopje, North Macedonia
atanasovskatijana25@yahoo.com

2nd Vladimir Zdraveski Ph.D.
Ss. Cyril and Methodius University in Skopje
Faculty of Computer Science and Engineering
Skopje, North Macedonia
vladimir.zdraveski@finki.ukim.mk

***Abstract*—The age of decentralization inspires ideas of implementation in the field of machine learning. The first step towards it is the model parallelization. Out of the three possible solutions, only one is suitable for every neural network architecture (model-agnostic). In this paper, we develop data-parallelism approach for plant disease detection. Inspired by state-of-art algorithms, we use convolutional neural network as it gives best performances when data is images. Our dataset is split in equal chunks and same neural network is used to test the learning on different number of workers. We propose solution architecture for different number of workers. The synchronization of weights is implemented with MPI, for between processes communication. Comparison between execution time and model performance metrics is discussed.**

***Index Terms*—data-parallelism, image classification, neural networks, decentralization, message passing interface**

I. Introduction

Over the past decade the information technology world was faced with two main development branches. Alex Krizhevsky has opened new era of possibilities for machine learning enthusiasts with his neural network AlexNet. On the other hand, the rise of Web3.0 pointed out the importance of decentralization. Both ideas naturally raises the question how can we combine them to result in better progress?

The amount of data created on a daily basis is tremendous. More data means wider learning space for our models, but also increasing training time. We must think how to use decentralization as a solution. On the other hand, the fear of artificial intelligence algorithms will be minimized as users will have part of them running in their devices. Distributed Machine Learning is new term in the age of AI, one of the solutions to speeding up learning for neural networks. The essence of Distributed ML is the use of multiple GPUs for training. But, the idea of utilizing a wide range of devices and different software kinds is getting more spread as Web3.0 rises in popularity. To fulfill our mission, we must think in dimensions of neural networks parallelization.

Neural networks are inspired by the human brain, mimicking the way that biological neurons signal to one another. They are built of node layers, connected one to another with associated weight. Each node is computing unit, representing linear regression model. The output, passed to activation layer, signals different information, which ends as classification or regression task. To expect significant results, we depend on many decisions - type and of layers, model architecture, weights, data to feed the neurons and lots of parameters. The process itself is challenge and the parallelization is even greater one.

There are three approaches to parallelize neural network - data parallelism, model parallelism and combination of them. Data parallelism is when we use the same model for every thread, but feed it with different parts of the data; model parallelism is when we use the same data for every thread, but split the model among threads. Each one has different benefits and drawbacks, explained in more details later in the paper, and we must consider the problem and available resources before making decision which one to use.

Our goal is to implement data-parallelism approach for plant disease classification model and compare performances on one CPU core, two CPU cores and a GPU. We will deeply explore the model architecture for data-parallelism. The communication between processes is implemented with Python library - mpi4py, allowing programmers to exploit multiple processor computing systems. After that, comparison is made between the results in execution time. As the most popular performance metric for models is accuracy, we will compare it on training and test data. At the end, exploration of benefits and drawbacks will be presented, and some ideas for future development.

Our dataset consists of plant disease images. The choice was made on purpose, to stress the importance of climate changes.Consequences as disrupted food availability, reduced access, and affected quality which leads to global economic and health problems. M.S.Hunjan has introduced us to these challenges, describing how the global disease scenario will be impacted by climate change[1]. Modern problems require intertwined sciences to search for solutions.

II. Related work

Comparison of many CNN-based models build for devices with different computational and storage resources are suggested by Ali Beikmohammadi et al [2]. Three stage algorithm can be separately used using the advantages of distributablity. The first model could be implemented on a user's mobile phone, whereas the second model is on a user's computer, and the third model is installed on a server. Each of these three models experienced a different view of data.

Modern techniques available, for plant disease detection, like processing, similarity identification and deep learning based classification techniques better in respect of time saving than the old methods used [3].

Convolutional neural networks have performed well in crop classification tasks [4], fruit counting, yield prediction [5], disease detection [4], [6], [7], and vision tasks in general. AlexNet and GoogLeNet architectures have shown state-of-the-art performance in these experiments [8], [9], [10]. Additionally, it has been shown that better results are acquired if networks are pre-trained [8].

Parallelization strategy that maximizes the overlap of inter-process communication with the computation is presented [11]. Stanford scientists have explored ways to parallelize/distribute deep learning in multi-core and distributed setting [12].
Zhiyi Huang makes comparison between the performance of two parallelization strategies for a backpropagation neural network on a cluster computer: exemplar parallel and node parallel strategies. The results are collated according to different sizes of neural network, different dataset sizes, and number of processors.[15] Another type - Bulk Synchronous Parallelism proves that exemplar parallelism outperforms techniques that partition the neural network across processors, especially when the number of exemplars is large, typical of applications such as data mining. [15]

III. Solution overview

A. *Parallelism in neural networks*

There are three approaches to parallelize neural network - data parallelism, model parallelism and combinational approach. Data parallelism is when we use the same model for every thread, but feed it with different parts of the data; model parallelism is when we use the same data for every thread, but split the model among threads. As long as the training objective decomposes into a sum over training examples, data parallelism is model-agnostic and applicable to any neural network architecture. In contrast, the maximum degree of model parallelism (distributing parameters and computation across different processors for the same training examples) depends on the model size and structure. The drawback of data parallelism is the number of copies of the model for every unit. The gradients synchronization must be observed as different mini-batches are used in each unit which implies the parameters need to be synchronized, i.e. averaged, after each pass. This leads to slower execution because of the communication needed. As Alex Krishevsky explains in his paper[14], model parallelism is efficient when the amount of computation per neuron activity is high (because the neuron activity is the unit being communicated), while data parallelism is efficient when the amount of computation per weight is high (because the weight is the unit being communicated). Solutions are suggested by Sunwoo Lee et al [12].
Another challenge is the batch size. In data parallelism, the model is updated after every pass which implies bigger batches will perform less updates. This approach decreases the communication between units, but big batches affect the rate of the convergence and might lead to longer training or divergence of the model. If the neural networks are set on different hardware devices, it should be considered scalable portion of data depending on the processing power and memory available on the device.

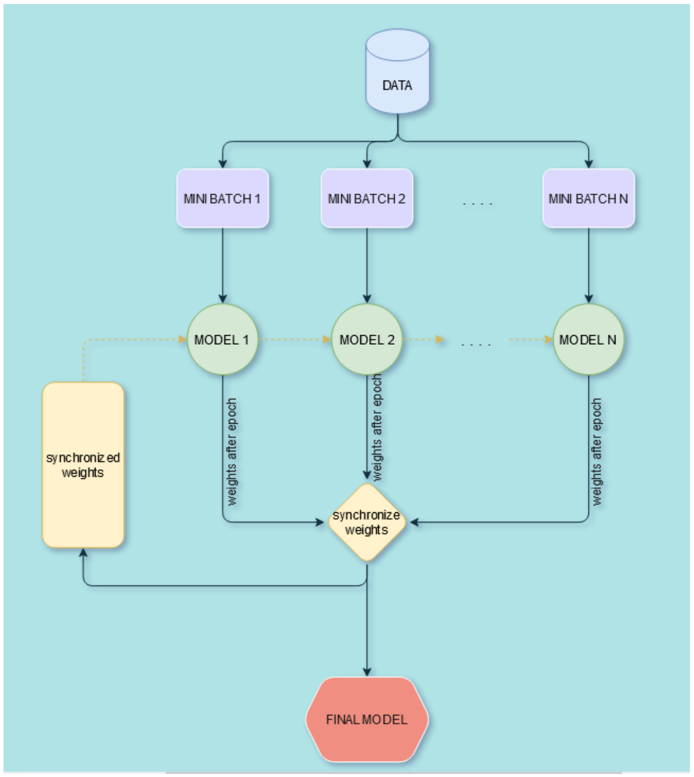

Fig. 1. Data-parallelism architecture

B. *Data-parallelism solution architecture*

The model used for this paper is based on deep convolutional neural networks. Another discussion that should be considered is about the types of layers that these networks use - convolutional and fully-connected layers. Convolutional layers cumulatively contain about 90-95% of the computation, about 5% of the parameters, and have large representations. Fully-connected layers contain about 5-10% of the computation, about 95% of the parameters, and have small representations. In particular, data parallelism appears attractive for convolutional layers, while model parallelism appears attractive for fully-connected layers. While there are proposals for combinations, we will use only data-parallelism, as we have many convolutional layers. On Figure 1 is illustrated solution for N workers.
First, the dataset is split on mini-batches. The preprocessing part consists of loading the data into memory, dimension reduction and conversion to suitable format for neural network input - in our case Numpy arrays. This part might be done by one unit, but this paper is dedicated on extensive use of parallelism. To increase robustness we use data augmentation, implemented with Image Data Generator from Keras. Next

step is model building in all of the units. The details are explained in section 4 - Model architecture. Every worker is training the model for one epoch on its own chunk of data. In data parallel, there is no synchronization in forward computing, because each unit has a fully copy of the model, including the deep net structure and parameters. But the parameter gradients computed in backpropagation must be synchronized afterwards.The process is repeated for as many epochs as we want to train the networks. The final model, used for predictions, is using the last synchronized weights.
The classification model is trained on PlantVillage dataset [14]. It consists plant images of 14 plants with 38 diseases, total of 20.639 images. The dimensions of the pictures are variable, but reduction is done in the preprocessing step. We reduce them to size of 128px, both width and height. The dataset is split into test and train sections by random selection to make the algorithm more robust. Proportions are 80%-20% and random state parameter is used so the division won't affect comparison results. RGB images are multi-dimensions matrices which are suitable subject for utilization of parallelism. This makes them more suitable for GPUs. Because this is problem of identification of diseases and not only detection, images are organized into folders by disease. The division implies that each worker will read similar data which will lead to bad performances of the models. As a consequence, shuffling and new splitting of the dataset is performed in one unit after every unit finishes with the preprocessing step. Communication between processes is enabled with Message Passing Interface (MPI) which is a standardized and portable message-passing standard designed to function on parallel computing architectures. We use mpi4py - version made for the programming language Python.

Type	Filters	Size
Conv2D	32	3x3
ReLU activation		
BatchNormalization		
MaxPooling2D		3x3
Dropout		0.25
Conv2D	64	3x3
ReLU activation		
BatchNormalization		
Conv2D	64	3x3
ReLU activation		
BatchNormalization		
MaxPooling2D		2x2
Dropout		0.25
Conv2D	128	3x3
ReLU activation		
BatchNormalization		
MaxPooling2D		2x2
Dropout		0.25
Flatten		
Dense		1024
ReLU activation		
BatchNormalization		
Dropout		0.25
Dense		15
Softmax		

TABLE I
NEURAL NETWORK LAYERS

For the purposes of this paper, data used for training was reduced to 3000 images. Number of training epochs is 25 and steps per epoch is 32. Batch size was set on 32 images.

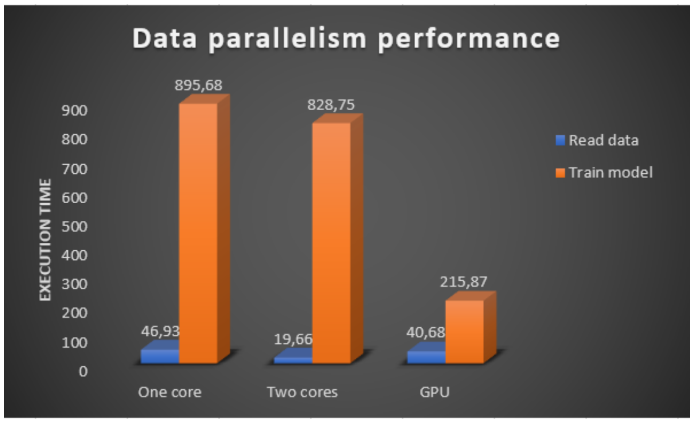

Fig. 2. Execution time comparison

IV. Model architecture

When dataset consists of images, the best performance models are built of convolutional layers. Named after the operation performed - convolution, which is performing dot product between the input image and a filter (set of weights designed to expose some features with size smaller than the input image) resulting in one scalar. The output is new array of different size (depends on the filter size). Next step is activation function and we are using ReLU. The result is given to normalization layer, which scales the output of the layer, explicitly by normalizing the activations of each input variable per mini-batch. To reduce the size, one of the layers used in every CNN is pooling layer. We are using Max pooling strategy. Dropout layer is a regularization method to reduce overfitting and improve generalization error in deep neural networks of all kinds.

Concrete architecture layers are shown in Table 1 with the information for number of filters and their size used in convolutional layers. Another parameters are dropout rate and number of units - dimensionality of the output space.
Implementation is done with Tensorflow and Keras. TensorFlow is an open-source library for building Machine learning models at large scale. Keras is a high-level neural networks API, written in Python and capable of running on top of Tensorflow.

After running forward step in the network, loss is computed and weights need to be adjusted. For that purpose, Adam optimizer is used. An optimization algorithm finds the value of the parameters(weights) that minimize the error when mapping inputs to outputs. Adam takes two parameters - initial learning rate (set on 1e-3) and decay which is a ratio between the learning rate and number of epochs. To compute the loss we use categorical cross-entropy, used to quantify deep learning model errors, typically in single-label, multi-class classification problems.

V. Results

Experiments were made on one core of CPU, two cores of CPU and GPU. System specifications on the hardware used: CPU - Intel Core i7-11370H, 11th gen. It has 4 cores (8 threads). Installed physical memory (RAM) - 16GB. GPU - NVIDIA GeForce RTX 3050Ti.
The first comparison of performances was with the images loading. One core: 46.93s. Two cores: 15.02s and 24.3s (average:19,66s). GPU: 40.68s. We can see how GPU has worse performance than two cores, which is caused because data has to be passed to them from RAM memory to GRAM in a ”costly” manner, every once in a while, so they can process it.
As Google Colab is very famous platform for neural network training, we tried the same process on their GPU. It finished after 1m 32s, which is understandable because it depends on network quality and Google Drive API, where the data is attached.
Next step is model loading and training the network. Parameters are same for every approach. On Figure 2, we can compare execution time. One CPU core finished the task for 895,68s which is only 66,87s more than the results achieved with two cores. Still, we must take into account the time needed for between processes communication, which is done quite often - after every epoch, so the models trained on different cores can synchronize their weights. GPU once again prove its superiority, being 3.83 times faster than two cores. One of the obstacles for CPU is the RAM needed to load two models, where every one of them has 13,259,023 parameters.
Accuracy comparison was made on train and test data. Figure 2 represents changes through epochs. Training done on one core has similar results with the GPU, but two-core approach converges slower. In the case of two cores, we compared accuracy between them after every epoch in the training process. One of them achieved better results throught the whole process. This leads us to idea of using the approach in preprocessing steps, to gain more information for different chunks of data.
Another comparable clue is the training loss - metric used to assess how a deep learning model fits the training set. Computationally, the training loss is calculated by taking the sum of errors for each example in the training set. The results are quite similar as with the accuracy performances.
Test data accuracy shows interesting behaviour. After 25 epochs of training, the three models trained with different approaches, achieved similar results. One core - 55%, two cores - 54,9% and GPU - 52,6%.

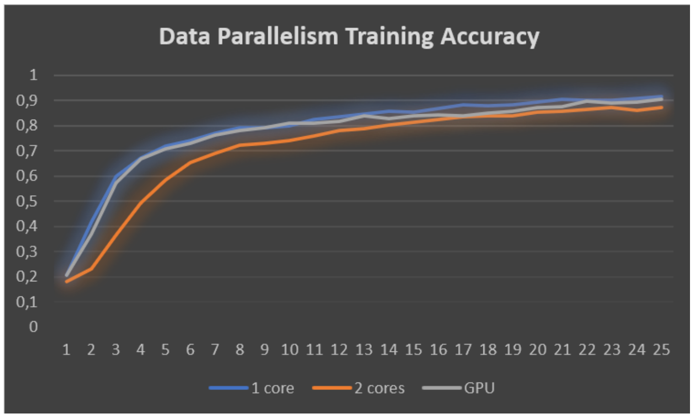

Fig. 3. Training accuracy

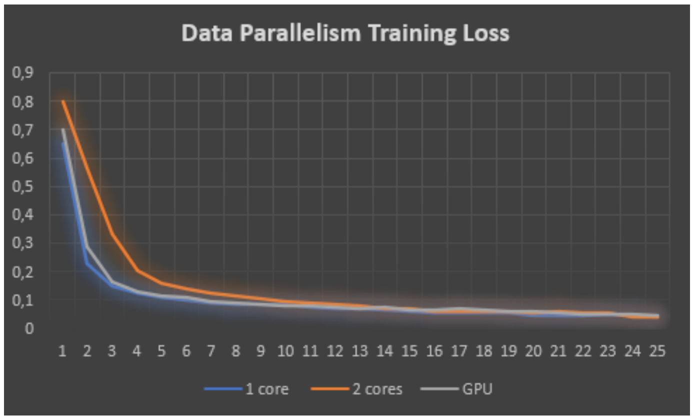

Fig. 4. Training loss

VI. Conclusion and future work

In this paper, we implemented data-parallelism for plant disease classification model. The fact that all of the networks achieved similar accuracy results motivates us to use the power of hardware components with different performances and expand the idea of model decentralization. GPU convincingly defeated its popularity in machine learning models, but our goal was to experiment parallelism with CPU to make use of many processing units.
The experiments implemented could be extended in future research. Data-parallelism on more GPUs or distributed nodes are some of the projects that might develop. As processes must communicate to exchange information, we would like to explore a few design strategies to maximize the degree of overlap between computation and communication. Also, another approach is to train separate models and synchronize their weights at the end, after all epochs. We can test how different architectures and change of parameters will act in our environment settings.
Many challenges will arise, but this paper is an inception of the experiments in the overlap of machine learning and decentralization.

References

[1] Mandeep Singh Hunjan and Jagjeet Singh Lore Chapte, “Climate change: Impact on plant pathogens, diseases, and their management”, August 2020
[2] Ali Beikmohammadi, Karim Faez, Ali Motallebi, “SWP-LeafNET: A novel multistage approach for plant leaf identification based on deep CNN”, September 2022

[3] Vijai Singh, Namita Sharma, Shikha Singh, “A review of imaging techniques for plant disease detection”, 2020

[4] Yao Chunjing, Zhang Yueyao, Zhang Yaxuan, Haibo Liu,“Application of convolutional neural network in classification of high resolution agricultural remote sensing images”, 2017

[5] Maryam Rahnemoonfar and Clay Sheppard, “Deep count: Fruit counting based on deep simulated learning ”, 2017

[6] Konstantinos P. Ferentinos, “Deep learning models for plant disease detection and diagnosis”, 2018

[7] Hyungtae Lee, Heesung Kwon, “Going deeper with contextual CNN for hyperspectral image classification”, 2017

[8] Sharada Mohanty, David Hughes, Marcel Salathe, “Using deep learning for image-based plant disease detection”, 2016

[9] Jayme Garcia Arnal Barbedo, “Plant disease identification from individual lesions and spots using deep learning”,2019

[10] Diego Inácio Patrício, Rafael Riederb, “Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review”, 2018

[11] Sunwoo Lee, Dipendra Jha, Ankit Agrawal, Alok Choudhary, and Wei-keng Liao, “Parallel deep convolutional neural network training by exploiting the overlapping of computation and communication”, 2017

[12] Vishakh Hegde, Sheema Usmani, “Parallel and distributed deep learning”, 2016

[13] Alex Krizhevsky, “One weird trick for parallelizing convolutional neural networks”, 2014

[14] PlantVillage Dataset https://github.com/spMohanty/PlantVillage-Dataset

[15] Zhiyi Huang - “Parallelization of a Backpropagation Neural Network on a Cluster Computer”, 2003

[16] R. Rogers and D. Skillicorn, “Strategies for parallelizing supervised and unsupervised learning in artificial neural networks using the bsp costmodel”, 1997.