MS5. HPC System architecture design

Version 0.4

Documentation Information

Contract Number	2018-EU-IA-0104
Project Website	www.saintgeorgeonabike.eu
Contratual Deadline	31/08/2020
Nature	Report
Author	Maria-Cristina Marinescu (BSC)
Contributors	Artem Reshetnikov (BSC)
Reviewer	Antoine Isaac (EF), José Eduardo Cejudo Grano de Oro (EF), Ariadna Lobo (BSC)
Keywords	HPC, object detection, distributed implementation, batch size, OpenMPI, GPU

Co-financed by yhe Connecting Europe Facility of the European Union

Change Log

Version	Author	Description Change
V0.1	Maria-Cristina Marinescu (BSC)	First iteration
V0.2	Artem Reshetnikov (BSC)	Reviewed
V0.3	Antoine Isaac (EF), José Eduardo Cejudo Grano de Oro (EF)	Reviewed
V0.4	Ariadna Lobo (BSC)	Reviewed

Table of Contents

1. Introduction

This document accompanies the MS6: *System and module-level architecture development* document to explain the computational requirements for the implementation of techniques that are explained in MS6. Just like MS6, this is a first iteration. Due to future implementation improvements, the increase of the training datasets, as well as to the probable implementation of new techniques, we can only estimate at this point what the final requirements are going to be in terms of the HPC architecture.

2. HPC requirements for the current system architecture and implementation

MS6 explains in detail the five main lines of work that we have been following during the first year of the project. These are as follows:

Improving object detection based on time context

Object detection based on Iconographic classes

Improving object detection based on language model

Caption generation based on attention mechanism

Caption classifier

The task that requires significant HPC resources, and which is part of all but the last line of work, is the object detector. The classifier depends only on the size of the dataset, and predicts a simple feature, that is, if a statement is likely or not to be a caption. The rest of the tasks are either relying on the MS COCO pre-trained model or on our own object detector, and caption generation does not require many resources. Given the nature of the domain we address - cultural heritage - we will definitely need to use our own object detection model, which is able to predict symbolic or iconographic classes, imaginary beings, symbolic actions, concepts that do not exist in everyday life as captured in pictures. At this point in the project, this is the only task that requires HPC resources, and a massive amount of them. At the same time, it is a key component which cannot be circumvented if we want to generate rich and precise metadata.

The implementation of the object detection model runs on the MinoTauro cluster at BSC-CNS. MinoTauro is a heterogeneous cluster in which the main computational power is provided by NVIDIA GPUS, which significantly increases the training speed of deep learning models. MinoTauro contains 39 servers, each of them containing:

- 2 Intel Xeon E5-2630 v3 (Haswell) 8-core processors, (each core at 2.4 GHz, and with 20 MB L3 cache)
- 2 K80 NVIDIA GPU Cards
- 128 GB of Main memory
- 120 GB SSD

The distributed implementation of the training algorithm for the object detection model uses HOROVOD - a distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet, and MasKRCNN - a framework for instance segmentation and object detection based on the FastRCNN architecture. Underneath, the implementation is based on

- NVIDIA CUDA: CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs).

- OpenMPI: Open MPI is a framework for the Message Passing Interface implementation, which allows distributing computation in supercomputers.

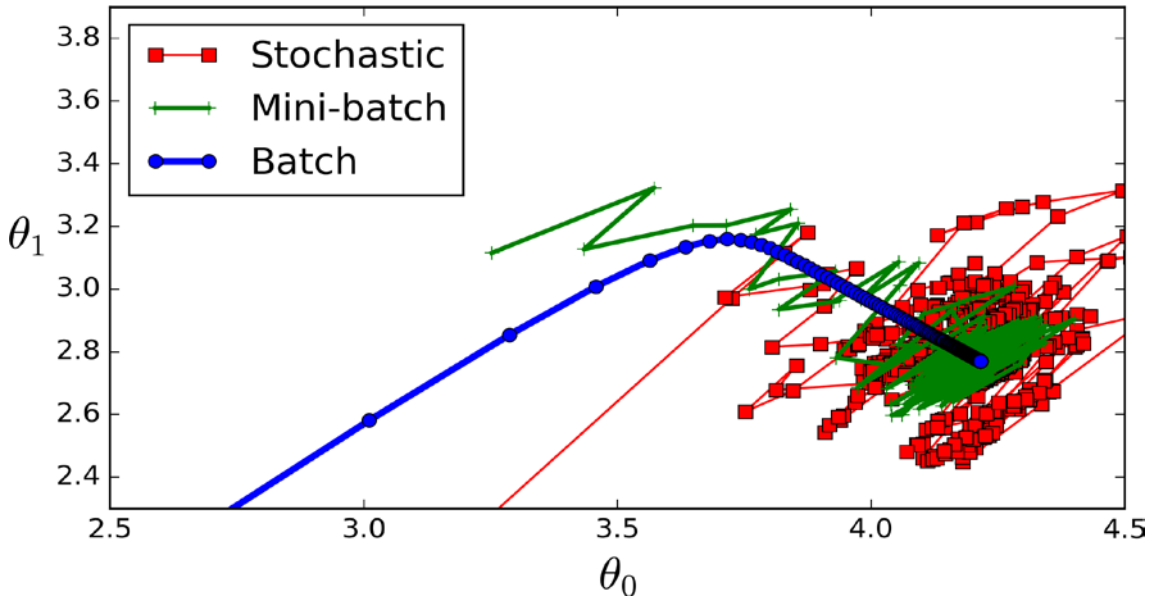
The key metric when scaling (from the computational viewpoint) the training task for Neural Networks in general is batch size. The batch size itself depends on the memory requirements and defines the number of samples that will be propagated through the network in a step (and there may be many steps per iteration).

As an example, let's say you have 7050 training samples and you want to set up a `batch_size` equal to 1000. The algorithm takes the first 1000 samples (from 1st to 1000th) from the training dataset and trains the network. Next, it takes the second 1000 samples (from 1001st to 2000th) and trains the network again. We can keep doing this until we have propagated all samples through the network. The last set of samples is not complete, in our case containing only 50 samples. The simplest solution is just to get the final samples and train the network, as a last step.

The advantages of using a batch size that is less than the number of all samples are the following:

- It requires less memory. Since you train the network using fewer samples, the overall training procedure requires less memory. That's especially important if you are not able to fit the whole dataset in your machine's memory.
- Typically networks train faster with mini-batches. That's because we update the weights after each propagation. In our example we've propagated 5 batches (4 of them had 1000 samples and 1 had 50 samples) and after each of them we've updated our network's parameters. If we used all samples during propagation, we would make only 1 update for the network's parameters.

There are also disadvantages of using a batch size less than the number of all samples. The smaller the batch is, the less accurate the estimate of the gradient will be. In the figure below (https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network), you can see that the direction of the mini-batch gradient (green color) fluctuates much more in comparison to the direction of the full batch gradient (blue color). In the future we will run our own experiments to confirm this for our case.



Considering this, scaling from a dataset with 4K images (with their bounding box information) to (a minimum of) 10K can be done without any additional resources and just maintaining the number

for batch_size. However, in that case the accuracy of the generated model will decrease and may not be enough if we want to use the model in production.

We have conducted experiments on training the object detection model based on the Mask RCNN architecture, using the 4k dataset mentioned above. The table below illustrates several approximations in terms of the number for batch_size, the computing resources, estimated training time, and range for the dataset size, which we consider to be sufficient to be able to achieve enough accuracy. At this point, we have to take into account limitations imposed when executing on the cluster; specifically, training cannot take longer than 48 hours, which means that we need to increase the resources in case of achieving this time limit. The model was trained with 50 epochs.

Batch size	**Resources**	**Training time**	**Size of dataset**
1000	1 node, 2 GPU (GPU parallelism)	13-16 hours	1000-4000
2000	1 node, 2 GPU (GPU parallelism)	48 hours	1000-4000
2000	5 nodes, 10 GPU (node parallelism)	12 hours	1000-4000
4000	5 nodes, 10 GPU (node parallelism)	Approximately 26-30 hours	4000-8000
6000	5 nodes, 10 GPU (node parallelism)	Approximately 38-45 hours	8000-10000
8000	Approximately 8 nodes, 16 GPU (node parallelism)	Approximately 38-45 hours	8000-10000

3. Discussion: Future HPC requirements

The current system implementation includes 5 techniques that all address use case 3.1 General service for enriching collections. The rest of the use cases are as follows:

Ingestion of results into Europeana (use case 3.2)

Search API (use case 3.3)

Crowdsourcing campaign for validation of enrichments (use case 3.4)

Uploading in data sharing platforms (use case 3.5)

None of these use cases impose any further computational power requirements. It is important to repeat here that the enrichment service offered by Saint George on a Bike is based on offline processing; it does not offer any real time scenario. The search use case does not trigger any enrichment processing, but rather finds relevant information based on the metadata for the images in the existing dataset. Crowdsourcing implies running the enrichment tool for the selected collections, but the enrichment process finishes before the end user can access and validate the metadata.

In terms of the global system architecture, documents MS3 and MS6 describe the options we foresee, without having reached a final decision. This is due to many factors, including the quality

of the enrichments and the facility with which the Europeana portal and search API can be adapted to exploit annotations that represent our project's enrichments (currently the Search APIs only exploits transcriptions, which do not include the enrichments our project would create). In any case, the decision of how to offer these services does not affect the HPC resources that the project will need.

Lastly, it is possible that implementations of new techniques may require additional computational resources, particularly if we do find it viable to extract semantic graphs as metadata. We think it is unlikely that these requirements will be of a totally different magnitude, given that actions - relationships between objects - will likely be deduced based on semantic information rather than intensive computer vision algorithms.