A Parallelization Approach for Hybrid-AI-based Models: an Application Study for Semantic Segmentation of Medical Images

Marco Scarfone[1], Pierangela Bruno[1,*] and Francesco Calimeri[1]

[1]*Department of Mathematics and Computer Science, University of Calabria, Rende, Italy*

Abstract

In the last decades, Artificial Intelligence (AI) approaches have been fruitfully employed in many tasks; for instance, Deep Learning (DL)-based methods have shown great ability in extracting meaningful features from images, providing valuable support to computer-aided diagnosis and medicine. Including prior knowledge in DL-based approaches could help in making their decisions more powerful, understandable, and explainable. However, even if this combination has raised a lot of interest in the scientific community, still remains an open problem due to several difficulties, for example, in modeling complex domains, handling missing specifications, and identifying the most suitable architecture able to properly combine the two AI worlds.

In this work, we rely on an existing framework defined for combining deductive and inductive approaches; in particular, explicit knowledge is encoded using Answer Set Programming (ASP), included in the training, and used to improve the quality of the images via a post-processing phase.

We propose a parallelization of this approach that drastically reduces the execution time. The proposed approach has been tested using different neural networks for semantic segmentation tasks over Laryngeal Endoscopic Images.

Keywords

Parallel Computing, Answer Set Programming, Deep Learning, Semantic Segmentation, Inductive-deductive coupling

1. Introduction

Semantic image segmentation refers to the task of segmenting an image into regions corresponding to meaningful objects and then assigning them an object category label [1]. In medical contexts, semantic segmentation of images can be extremely useful to support clinicians in providing proper diagnosis, identifying pathological conditions, and highlighting image regions related to a specific disease. In this context, Deep Learning (DL)-based approaches represent a huge breakthrough, showing a great deal of potential in extracting meaningful information from different types of images (e.g., computed tomography (CT), magnetic resonance imaging (MRI), endoscopic imaging). These approaches show to be particularly suitable for semantic

---

*Corresponding author.

bruno@mat.unical.it (P. Bruno); calimeri@mat.unical.it (F. Calimeri)
0000-0002-0832-0151 (P. Bruno); 0000-0002-0866-0834 (F. Calimeri)

segmentation and, in general, for supporting automated diagnosis, surgical scene understanding, and computer-assisted interventions.

Furthermore, including methods explicitly conceived for modeling prior knowledge in the DL-based process can improve the quality of the results, paving the way for better interpretability and explainability of neural networks. Indeed, such approaches have been widely studied and used in different areas of AI, such as planning, probabilistic reasoning, bioinformatics, etc. (see, e.g., [2, 3, 4]). Also, very recently, ways of combining deductive and inductive approaches have raised a lot of interest in the scientific community.

However, AI-based approaches, and, in particular, DL-based, require huge computational time and even better processors to perform specific operations, especially in computer vision in which images and videos of high quality are analyzed.

For this reason, many scientists decided to rely on data parallelism (DP), by partitioning (and distributing) the workload among the cluster processes across the batch size [5]. In this way, neural networks can be trained in parallel and, at each batch size, all processes collaborate to modify local weights that, globally, concur to the ”best” global parameters, so defining the DL-based model [6].

Among different existing methods, we made use of Message Passing Interface (MPI) [7] which is a standardized communication protocol designed to function on parallel computing architectures and distributed applications. MPI can be very useful to speed up the learning phase which could be very slow according, for example, to the number of images received as input or to the complex network structure [5].

We propose a parallelization approach to perform semantic segmentation of Laryngeal endoscopic images [8]. We make use of a hybrid-AI-based model proposed in [9] and [10]. In this work, the authors combined different neural network architectures (i.e., DeepLab-v3, SegNet, U-Net) and the potential coming from the declarative nature of Answer Set Programming (ASP) to improve the overall performance via (*i*) an ad-hoc loss function and (*ii*) a proper post-processing phase. Our approach exploits parallel computing to drastically reduce the execution time of the baseline work [9], keeping a comparable performance.

The remainder of the paper is structured as follows. In Section 2 we provide a detailed description of our approach, which has been assessed via a careful experimental activity, which is in turn discussed in Section 3; we analyze and discuss results in Section 4, eventually drawing our conclusions in Section 5.

2. Methods

This approach relies on the seminal work that appeared in [9] which is used as the basis of our parallelization. The authors in [9] proposed a framework to combine DL and ASP-based models in performing semantic segmentation. Specifically, they used ASP to:

- drive the network’s learning and penalize the misclassification during the training phase: the ASP-based model is used to quantify a penalty value by comparing the network’s prediction to medical knowledge and ground truth segmentation. In this way, the approach

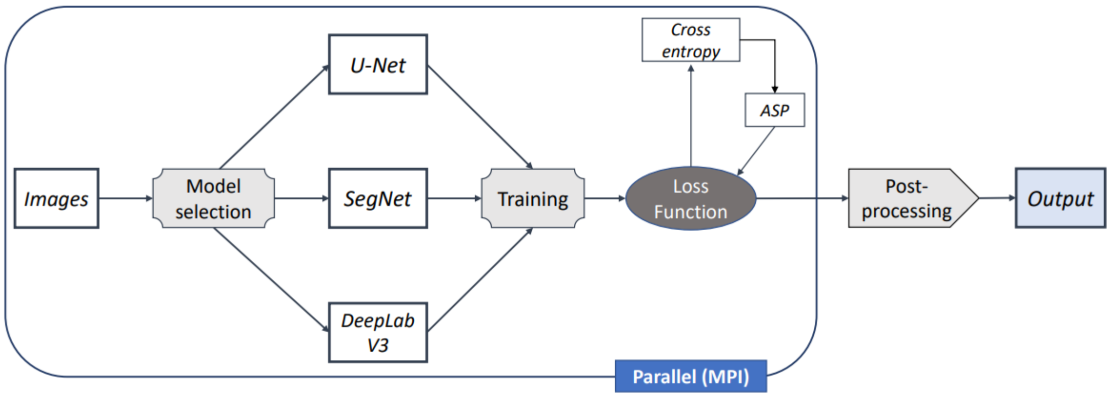


Figure 1: Workflow of the proposed framework. Images are used to train three different neural networks. The training phase is supported by an ASP-based model through loss function. The whole process is computed in parallel. The predicted output is refined by rule-based post-processing

is able to express "how wrong" the classification is; this value takes part in defining the loss function [9].

- improve the quality of the results via ASP-based post-processing. The approach is able to remove noise (i.e., small "islands" of misclassified pixels) and wrong predicted classes (i.e., classes which do not respect medical requirements). Specifically, after converting the network's prediction into logical rules, the ASP-based model identifies pixels that need to be removed and, eventually, re-assigns misclassified pixels/elements to the more frequent class in the neighborhood.

In this paper, we rely on data decomposition to parallelize the above-described approach and, in particular, the training process. The workflow of the approach is shown in Fig. 1.

2.1. Parallelization

MPI offers two kinds of communication functions: point-to-point and collective. In particular, collective communications execute an exchange of data with all interested nodes; these communications can be: (*i*) one-to-many where data are sent from the root process to all others; (*ii*) many-to-one where the root node receives data from all communicator process; (*iii*) many-to-many where no root node exists and all processes send and receive from all the other ones.

In order to allow the parallelization of the approach, the input was split into various processes through the scatter function to resolve the group of images. To better explain how it works, we provide the following example. Given three processes (P_0, P_1, P_2), a tensor-shaped $300x224x224x3$, and 300 images with dimensions $224x224$ at 3 channels, the scatter method divides the 300 images into groups of 100 and send them to each process. The resulting processes P_0, P_1, P_2 have a tensor $100x224x224x3$ each. This is allowed thanks to the Single Program Multiple Data (SPMD) paradigm by which each process performs the same program [11]. Then, using

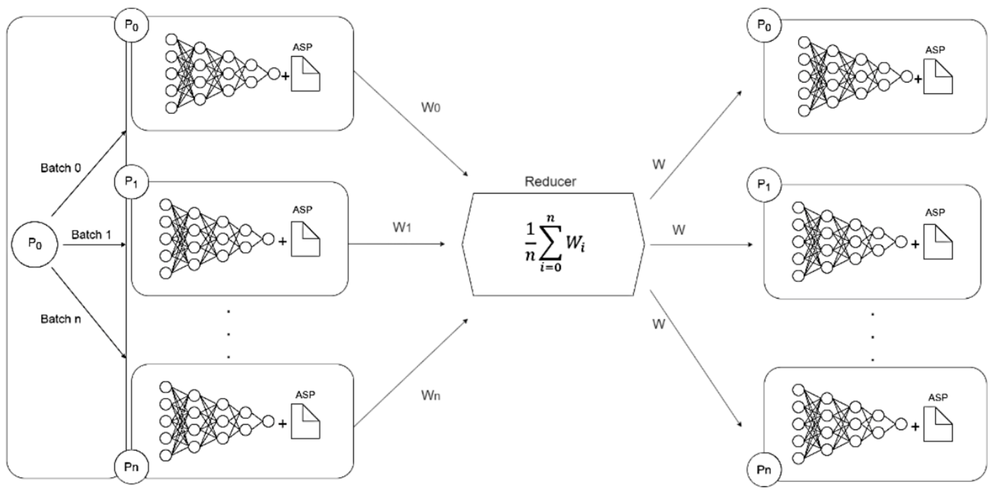


Figure 2: Knowledge exchange between neural networks on each processor for each batch size.

this paradigm, if we had three processes, we would be able to execute three neural networks contemporarily and each of them would take as input three different batches of images.

After the data decomposition and during the learning phase, each neural network is able to communicate and adjourn the learned information to the other networks.

The communication occurs through the function *Allreduce* which is a many-to-many communication and operates similarly to the reduce standard method. Specifically, it executes mathematical operations on weights calculated from each network and adjourns the status of the other ones. More in detail, when a process completes its own batch or a specific epoch, it waits until all the other processes finish and, only after, an average of all weights is computed.

The average is computed via a personalized operation in which a reducer sums all tensors with the same key and, after, it averages them and sends the result to each process. In this way, it is possible to use different neural networks in parallel. Each network operates on different data (then, the quantity of images is reduced) and exchanges the information just obtained.

Particularly, at each iteration, many models are obtained with equal information; these models correspond to the number of processes started. Afterward, in the next batch, each neural network is ready to independently update its weights based on its own batch of images, and, then, the information is exchanged with the other models. An example of this process is shown in Fig. 2.

2.1.1. ASP

Our approach also provides the possibility to parallelize the ASP model which can be executed by each process at the same time. Then, ASP works on a sub-group of images in each process, generating several loss functions for each batch of images and for each process. These ASP-based loss functions are then added to the loss function obtained by the neural network to define the final loss, according to [9]. To handle parallel computing with ASP, the outputs produced by neural networks are separately and simultaneously stored. Therefore, each process can access its own ASP-based output according to the specific batch of images received. This makes the entire computing process much faster and more efficient.

Similarly, our approach could allow us to parallelize ASP-based post-processing such that each process can simultaneously access the rule-based model and the knowledge base describing the specific batch of images. In this way, each process is able to accurately identify the wrong classes or noise and re-assign the right locations in the image.

3. Experimental Activities

For the experimental analysis, we used the same dataset proposed in [9]: the Laryngeal Endoscopic Images dataset [8]. It consists of 536 manually segmented in vivo color images of the larynx captured during two different resection surgeries. In particular, the images are categorized in 7 classes: *void, vocal folds, other tissue, glottal space, pathology, surgical tool, and intubation*, corresponding to index 0, 1, 2, 3, 4, 5, 6, respectively.

In order to ensure a proper comparison w.r.t. the results obtained in [9], we kept the same configuration. The dataset was split into training (80%) and testing (20%) sets, each network was implemented in Pytorch using the SGD optimizer and cross-entropy (CE) as a loss function. In order to complete the experiments, due to time constraints, we reduced the number of epochs from 1000 to 300; future experiments are planned, with increased limits.

Also, to assess the effectiveness of our approach we used Intersection-over-Union (*IoU*) evaluation metric, which is as $IoU = \frac{TP}{TP+FP+FN}$, where *TP* is the number of true positive, *FP* false positive and *FN* false negative pixels, respectively.

We point out that, at present, the results refer to parallel training without the inclusion of ASP in the learning phase. Furthermore, even if we designed the parallel workflow, currently the presented post-processing results are obtained without parallelization. Full experiments are carried out at the time of writing, and the updated results will be released in the near future.

3.1. Parallelization metrics

Speed-up is an important factor to consider when evaluating a parallel approach. It is computed as follows:

$$speed - up = \frac{T_s}{T_p}$$

Where T_s is the execution time of the sequential algorithm, while T_p is the parallel time. Two kinds of speed-up are existing, absolute and relative: relative indicates the speed-up where T_s is the time of the algorithm with one process; absolute indicates the speed-up where T_s is the time of the best sequential algorithm. For example, if a sequential algorithm needs 10 minutes of calculation time and a correspondent parallel algorithm needs 2 minutes, we can say that the speed increases by 5 times.

Since speed-up measures how fast a parallel algorithm goes, the ideal speed-up is equal to the number of processes in use. In this case, we talk about linear speed-up. It could happen that the speed-up stops or drastically reduces the growth when the number of processes increases.

This behavior can be explained by Amdahl's law [12] which evaluates the maximum value that the speed-up can reach on a determined algorithm as follows:

$$speed - up(n) = \frac{1}{(1-f) + \left(\frac{f}{n}\right)}$$

where f indicates the portion of parallel code, $1 - f$ indicates the sequential part remaining, and n is the number of processes. In other words, the speed-up exclusively depends on the number of sequential portions, independently of the number of processes utilized. In addition, maintaining constant the number of processes, the parallel part $f < 1$, whatever big it is, has as an upper bound the number of processes; then, linear speed-up occurs when the $speed - up(n) = n$.

Another reason why acceleration does not grow linearly is due to overhead. The overhead is the overload of work due to different factors:

- *Time to start activity*
- *Synchronization between processes*
- *Communication of data*
- *Libraries and operating, system overload, etc.*
- *Time to conclude the activity.*

The overhead can be calculated via the following formula:

$$Overhead(n) = n \cdot T_p - T_s$$

At last, another parameter to be considered is efficiency. The efficiency is a value between 0 and 1 and it is computed as:

$$E_n = \frac{S_n}{n}$$

Where n is the number of processes and S_n is the speed-up with n processes. This relation indicates the fraction of time in which each element is really utilized. In particular:

- If $E_n = 1$, then we have a linear speed-up (very difficult)
- If $E_n < \frac{1}{n}$, then the algorithm has a slackening

The efficiency metric is useful to quantify the scaling down. Specifically, if the efficiency remains constant to the variation of the number of processes, we obtain a linear scaling down. Actually, by increasing the number of processes p and fixing the problem dimension W the efficiency decreases (see Fig. 3 (a)), on the contrary, fixing the number of processes and increasing the problem dimension, the efficiency increases (see Fig. 3 (b)). The object is therefore to maintain, as previously described, a constant efficiency. Consequently, we need to increase the number of processes, increasing the problem dimension, bringing us to the iso-efficiency concept [13].

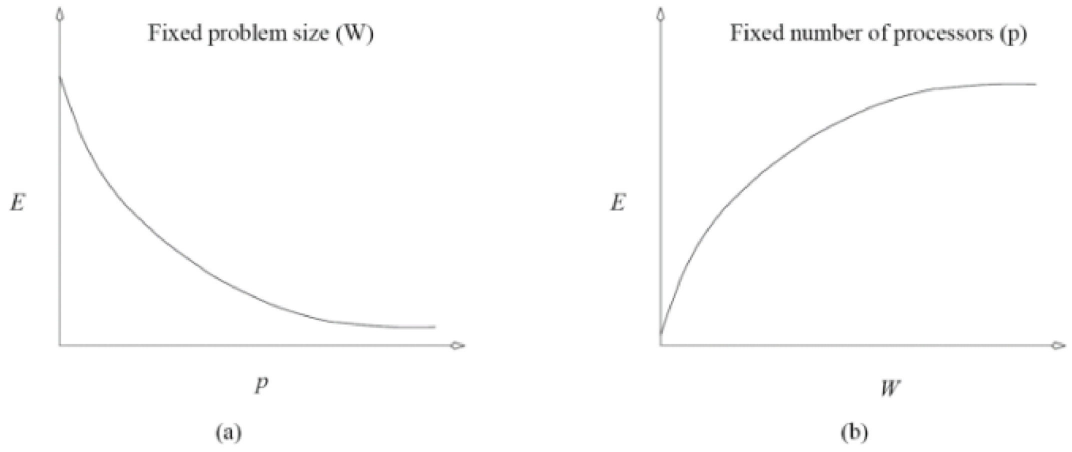
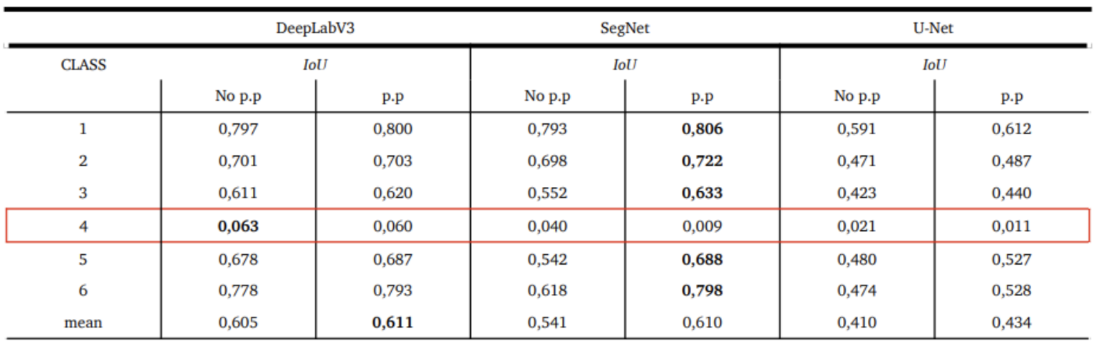


Figure 3: Efficiency trend by fixing the problem size (a) and the number of processes (b).



	DeepLabV3		SegNet		U-Net	
CLASS	*IoU*		*IoU*		*IoU*	
	No p.p	p.p	No p.p	p.p	No p.p	p.p
1	0,797	0,800	0,793	**0,806**	0,591	0,612
2	0,701	0,703	0,698	**0,722**	0,471	0,487
3	0,611	0,620	0,552	**0,633**	0,423	0,440
4	**0,063**	0,060	0,040	0,009	0,021	0,011
5	0,678	0,687	0,542	**0,688**	0,480	0,527
6	0,778	0,793	0,618	**0,798**	0,474	0,528
mean	0,605	**0,611**	0,541	0,610	0,410	0,434

Figure 4: Per-class and mean IoU for the 3 tested neural networks that are trained in parallel. The first column reports the results obtained without post-processing (no p.p.) and the second refers to the results after ASP-based post-processing (p.p.). The most significant results are highlighted.

4. Results and Discussion

Table 4 shows the results achieved using parallel training. We can see that the IoU of neural networks follows the same trend as the sequential approach used in [9]. However, the class *pathology* (i.e., 4), which is considered the most difficult since the lower occurrence in the dataset [8], achieved a very low IoU value. This is most likely caused by the decrease of epochs that negatively affects the performance of the network in recognizing this class and, consequently, in the overall averaged IoU value.

In general, SegNet and Deeplabv3 show better performance than U-Net and the post-processing is able to slightly but systematically improve the image quality in almost all classes.

A visual example of the results is shown in Fig. 5. These results, which are graphically compared with raw images and ground truth (GT) segmentation, show the capability of our approach in assigning the right class to each pixel and removing misclassification errors via

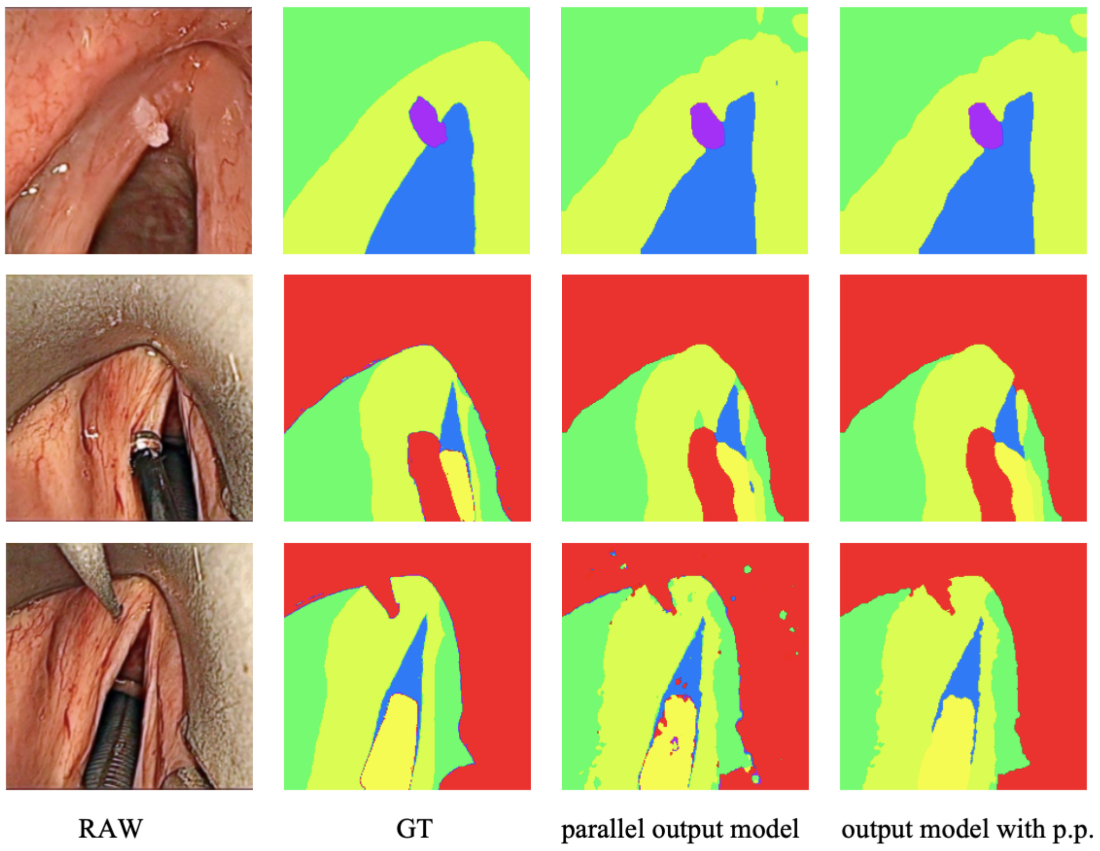


Figure 5: Example results obtained using 3 different patients. From left to right: raw image, ground truth (GT) segmentation, semantic segmentation obtained using parallel model, and results obtained using post-processing.

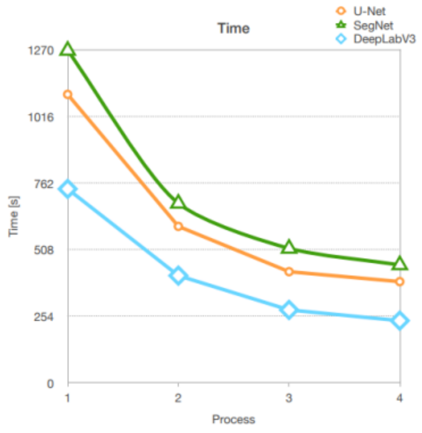
ASP-based post-processing.

4.1. Parallelization performance

Figure 6 shows the execution time for each epoch required to train each network according to the number of processes used. We can notice that SegNet results in the heaviest network, taking about 21 minutes to conclude an epoch. However, thanks to parallel processing the execution time is reduced up to ~ 7 minutes using 4 processes.

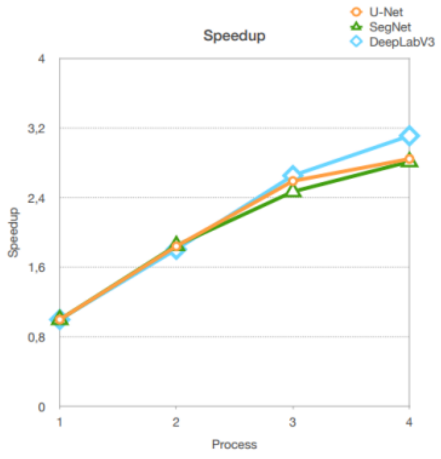
Figure 7 shows the relative speed-up achieved using the three neural networks. We can notice that speed-up for two processes reaches a value close to two, similarly using three processes, meaning that the network's learning is going two or almost three times faster. Instead, when using four processes (on the same machine), there are no huge improvements; this could be explained via Amdahl's law described in Sec. 3.1.

We also computed the efficiency trend on the different models achieved using the neural networks, as shown in Fig. 8. SegNet network shows an efficiency value of 92% using two



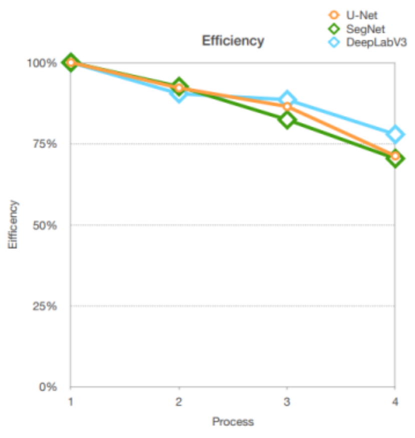
Process	DeepLabV3	SegNet	U-Net
1	12min 21s	21min 10s	18min 22s
2	6min 50s	11min 26s	9min 58s
3	4min 39s	8min 34s	7min 5s
4	3min 58s	7min 31s	6min 27s

Figure 6: Average execution time trend per epoch with different numbers of processes.



Process	DeepLabV3	SegNet	U-Net
1	1	1	1
2	1,807	1,851	1,843
3	2,656	2,471	2,593
4	3,113	2,816	2,848

Figure 7: Trend of the average speed-up per epoch with the different number of processes.



Process	DeepLabV3	SegNet	U-Net
1	1,00000	1,00000	1,00000
2	0,904	0,926	0,921
3	0,885	0,824	0,864
4	0,778	0,704	0,712

Figure 8: Average efficiency trend per epoch with a different number of processes.

processes but, when the number of processes increases, at the same problem dimension, the efficiency starts to decrease. However, the performance of the approach still remains comparable, showing good results.

5. Conclusion

In this work, we proposed a parallel AI-based approach to perform semantic segmentation of medical images. We used an existing framework as a baseline for our approach. This framework combines Neural Networks and ASP to define a novel loss function and a post-processing phase. We performed a thorough experimental analysis; our proposal reduced the execution time and achieved comparable results w.r.t. baseline approach.

Actually, the reported results achieved via the parallel approach refer to an experimental activity performed without the inclusion of the ASP-based model in the training phase. As future work is concerned, we aim to refine experimental analysis including ASP-based knowledge in the parallel training phase and investigate misclassification errors and improve the generalization capability of the model, as well as the overall performance.

References

[1] Y. Mo, Y. Wu, X. Yang, F. Liu, Y. Liao, Review the state-of-the-art technologies of semantic segmentation based on deep learning, Neurocomputing 493 (2022) 626–646.

[2] C. Dodaro, G. Galatà, A. Grioni, M. Maratea, M. Mochi, I. Porro, An asp-based solution to the chemotherapy treatment scheduling problem, Theory and Practice of Logic Programming 21 (2021) 835–851.

[3] C. Dodaro, D. Ilardi, L. Oneto, F. Ricca, Deep learning for the generation of heuristics in answer set programming: A case study of graph coloring, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2022, pp. 145–158.

[4] E. Di Rosa, E. Giunchiglia, M. Maratea, A new approach for solving satisfiability problems with qualitative preferences, in: ECAI 2008, IOS Press, 2008, pp. 510–514.

[5] A. Castelló, E. S. Quintana-Ortí, J. Duato, Accelerating distributed deep neural network training with pipelined mpi allreduce, Cluster Computing 24 (2021) 3797–3813.

[6] T. Ben-Nun, T. Hoefler, Demystifying parallel and distributed deep learning: An in-depth concurrency analysis, ACM Computing Surveys (CSUR) 52 (2019) 1–43.

[7] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, Mpi: The complete reference the mit press, Cambridge, Massachusetts (1996).

[8] M.-H. Laves, J. Bicker, L. A. Kahrs, T. Ortmaier, A dataset of laryngeal endoscopic images with comparative study on convolution neural network-based semantic segmentation, International journal of computer assisted radiology and surgery 14 (2019) 483–492.

[9] P. Bruno, F. Calimeri, C. Marte, M. Manna, Combining deep learning and asp-based models for the semantic segmentation of medical images, in: International Joint Conference on Rules and Reasoning, Springer, 2021, pp. 95–110.

[10] P. Bruno, F. Calimeri, C. Marte, Dedudeep: An extensible framework for combining deep learning and asp-based models, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2022, pp. 505–510.

[11] P. Czarnul, Parallel programming for modern high performance computing systems, CRC Press, 2018.

[12] G. M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of the April 18-20, 1967, spring joint computer conference, 1967, pp. 483–485.

[13] S. Kumar, Introduction to Parallel Programming, Cambridge University Press, 2022.