An Optimized and Accelerated Object Instance Segmentation Model for Low-Power Edge Devices

Diego Bellani[a], Valerio Venanzi[b], Shadi Andishmand, Luigi Cinque[c] and Marco Raoul Marini[d]
Department of Computer Science, Sapienza University of Rome, Via Salaria 113, 00198, Rome, Italy

Keywords: Deep Learning Efficiency, Edge Computing, Embed Devices, Object Detection, Pruning.

Abstract: Deep learning, for sustainable applications or in cases of energy scarcity, requires using available, cost-effective, and energy-efficient accelerators together with efficient models. We explore using the Yolact model, for instance, segmentation, running on a low power consumption device (e.g., Intel Neural Computing Stick 2 (NCS2)), to detect and segment-specific objects. We have changed the Feature Pyramid Network (FPN) and pruning techniques to make the model usable for this application. The final model achieves a noticeable result in Frames Per Second (FPS) on the edge device while achieving a consistent mean Average Precision (mAP).

1 INTRODUCTION

Deep Learning (DL) has recently gained widespread recognition as a powerful technology, driving advancements in both research and industrial applications across a range of fields, e.g., medical area (Avola et al., 2022), cultural heritage (Gonizzi Barsanti et al., 2024), and even entertainment (Avola et al., 2023).

Deep Convolutional Neural Networks (CNNs) have made remarkable progress in tasks such as image analysis, object detection, and instance segmentation (Li et al., 2022; Khayya et al., 2024). However, deploying these models on resource-constrained edge devices poses significant challenges due to high computational, memory, and energy demands. For instance, models like ResNet-100 (He et al., 2015) and Mask R-CNN (He et al., 2017a) require millions of parameters and billions of floating-point operations (FLOPs), making them difficult to run on edge devices. Although offloading computation to cloud servers is an option, practical issues like connectivity, latency, and security concerns limit this approach.

The demand for efficient on-device inference has led to model optimization and compression innovations to reduce computational costs while maintaining acceptable accuracy. Companies have developed a range of domain-specific accelerators for inference on edge, e.g., Intel's Neural Compute Stick 2 (NCS2), Google's Edge Tensor Processing Unit (TPU), and NVIDIA's Jetson Nano.

1.1 Research Objectives

The goals of this research are to:

- Develop an instance segmentation model to identify specific objects in images accurately.
- Implement the model on an edge device.
- Optimize the model to reduce inference time on the device.
- Evaluate the model's performance for the Intel NCS2 platform.

We particularly care about the NCS2 because the financier of this project (Rome Technopole) was already invested in this hardware, and a solution targeting any other accelerator would not have made economic sense.

2 STATE OF THE ART

The task of Instance Segmentation is an evolution of Object Detection and Semantic Segmentation. While object detection focuses on identifying different objects in an image, semantic segmentation aims to predict the class of each pixel and create masks for each class. Both tasks are tackled simultaneously in Instance Segmentation, providing pixel-level accuracy for locating object instances. Each pixel is classified into one of the predefined classes, akin to Semantic Segmentation, but with the added capability of performing Object Detection. Consequently, each result-

[a] https://orcid.org/0009-0008-8312-9112
[b] https://orcid.org/0009-0006-2847-4998
[c] https://orcid.org/0000-0001-9149-2175
[d] https://orcid.org/0000-0002-2540-2570

Bellani, D., Venanzi, V., Andishmand, S., Cinque, L. and Marini, M. R.
An Optimized and Accelerated Object Instance Segmentation Model for Low-Power Edge Devices.
DOI: 10.5220/0013200700003905
In *Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2025)*, pages 485-495
ISBN: 978-989-758-730-6; ISSN: 2184-4313

ing mask corresponds to a specific class instance, allowing for identifying different instances of the same class within an image.

Similar to previous tasks, object detection, and semantic segmentation, various frameworks based on Convolutional Neural Networks (CNNs) have been proposed to address this challenge over the years. (Hariharan et al., 2014) were pioneers in addressing instance segmentation, referring to it as Simultaneous Detection and Segmentation. They extended the R-CNN architecture to obtain masks for every instance, not just bounding boxes. Their framework began with generating different proposals using Multiscale Combinatorial Grouping, resulting in 2000 region proposals per image. Subsequently, a CNN extracted features from each region proposal, and a Support Vector Machine was trained on these features to obtain class probabilities. Finally, a year later, the authors refined their model (Hariharan et al., 2014). They observed that using only the output of the last layer as a feature representation led to a loss of crucial spatial information. To address this limitation, they introduced the concept of hypercolumns. Instead of relying solely on the output of the last layer, each pixel was defined by a vector comprising activations from all CNN units above that pixel.

(Pinheiro et al., 2015) introduced an innovative approach with their algorithm DeepMask, which takes an image patch as input and produces two outputs. One output is a class-agnostic mask, while the other is a score indicating the probability of the patch containing a completely centered object. The model, based on a ConvNet, optimizes both outputs simultaneously through a cost function. DeepMask significantly reduced the number of generated proposals while enhancing performance compared to prior models.

While Fully Convolutional Networks (FCNs) have proven successful for Semantic Segmentation, they lack instance-level information. (Dai et al., 2016) proposed InstanceFCN, an extension of FCN capable of segmenting different instances of the same class. Unlike traditional FCNs, InstanceFCN's output is a score map the same size as the input image. Each pixel in the output serves as a classifier for relative positions of instances, incorporating information such as "left side" or "bottom". By leveraging local coherence, InstanceFCN reduces computational costs and the number of parameters compared to DeepMask.

However, InstanceFCN had drawbacks addressed in (Li et al., 2016). InstanceFCN lacked consideration for semantic categories, performed segmentation and detection in separate stages, and lacked an end-to-end model. The fixed-size sliding windows and the time-consuming process of finding instances at different scales were additional challenges. (Li et al., 2016) introduced Fully Convolutional Instance Segmentation (FCIS), the first end-to-end solution for instance segmentation. FCIS shares features and score maps between segmentation and detection tasks, leading to a reduction in parameters. Additionally, it utilizes bounding box proposals instead of sliding windows, improving efficiency.

Despite the enhancements introduced by (Li et al., 2016), the FCIS model exhibited deficiencies, particularly when dealing with overlapping instances. It demonstrated issues and inaccuracies in scenarios where two or more instances overlapped, resulting in false edges, even in cases of uniform background texture.

An alternative research approach addressed the problem differently, first performing semantic segmentation and subsequently attempting to delineate instances for each class. This is exemplified in works such as (Arnab and Torr, 2017; Bai and Urtasun, 2016; Kirillov et al., 2016; Liu et al., 2017).

(Kirillov et al., 2016) propose a model called InstanceCut. They tackled semantic segmentation using a standard CNN to achieve an instance-agnostic solution. Simultaneously, a separate CNN was employed to extract the edges of different instances. The outputs of these two CNNs were then combined. Notably, this framework trained two distinct models—one for semantic segmentation and another for instance-edge detection—avoiding the need to handle global features of instances from various classes. This modular design facilitates direct implementation of advancements in semantic segmentation or instance-edge detection. However, a limitation of this method is its inability to consider objects as part of the same instance if they are not connected.

In (Bai and Urtasun, 2016), the authors leveraged the classical grouping technique known as the watershed transform. Viewing a grayscale image as a topographic surface, the transform floods the surface from different points in its minima. A segmentation of the image's different components is obtained by preventing water from different sources from meeting. (Bai and Urtasun, 2016) proposed using a deep convolutional neural network to learn the energy of the transform, enabling segmentations containing only one instance. This allows the segmentation of different instances by cutting at distinct energy levels. Similar to (Kirillov et al., 2016), a primary limitation of this framework is its inability to handle occlusions.

In (Liu et al., 2017), Sequential Grouping Networks (SGN) were introduced, for instance, segmentation. SGN employs a concatenation of neural net-

works, each addressing sub-tasks with varying semantic complexity within the instance segmentation process. The task is divided into more manageable subtasks, each handled by a distinct neural network. The initial network focuses on grouping pixels along pixel rows and columns in the image by predicting object breakpoints and forming line segments. These line segments become the input for the second network, which is responsible for grouping them into connected elements. Another neural network integrates these components in the final step, generating masks for object instances. Despite its effectiveness, this framework exhibits limitations in segmenting small instances and occasionally grouping multiple instances together when they overlap.

As for more recent methodologies, instance segmentation approaches can be categorized into two primary groups: algorithms based on detection and those employing a single-shot strategy. The former tends to achieve higher mean Average Precision (mAP), reflecting the overlap area between predicted masks and ground truth masks. In contrast, the latter is particularly well-suited for real-time implementation scenarios, offering superior speed compared to the former approach.

Numerous frameworks have been developed within the category of algorithms based on detection. The approach based on detection is commonly known as the proposal-based method. In this method, the initial step involves detecting various objects within the image using a dedicated detection network. Subsequently, a segmentation head is applied to each detected bounding box to acquire instance segmentation like Mask R-CNN (He et al., 2017a).

Mask R-CNN (He et al., 2017a) was proposed as an extension of Faster R-CNN. The authors introduced an additional branch to the Faster R-CNN architecture. While the original branch was designed for classification and bounding box regression, producing bounding boxes and class labels, the new branch focused on predicting segmentation masks for each region of interest. This resulted in the mask of the instance within each bounding box as the output. Mask R-CNN demonstrated superior performance to previous state-of-the-art models, for instance, segmentation, particularly when utilizing the COCO dataset (He et al., 2017a).

Numerous frameworks have been developed within the category of single-shot strategy. Single-shot instance segmentation, or in other words, the proposal-free method, is a real-time approach, usually much faster than a detection-based approach. At the same time, the single-shot approach is less accurate. As representative examples of the single-shot architecture, we can mention YOLACT (You Only Look At CoefficienTs) (Bolya et al., 2019; He et al., 2017a), PolarMask (Xie et al., 2019) or SSAP (Single-Shot Instance Segmentation With Affinity Pyramid) (Gao et al., 2019). SSAP runs a pixel category classification network (semantic segmentation) and performs instance segmentation on a semantic segmentation map.

This work has been included in the research for industrial purposes in the Rome Technopole ecosystem. The content target of the research is currently confidential for the industrial secrets involved; thus, we decided to test the effectiveness of the proposed methodology on a different subject. Pepper segmentation is a task that receives attention in research for its applications in the real world. In (Cong et al., 2023), the authors perform Instance Segmentation on Greenhouse Sweet Peppers using Improved Mask RCNN. They achieve good results, proving the method robust even on complex scenes. They achieve 5 FPS performance on a system equipped with an RTX 2080; as the authors claim, this kind of performance satisfies the requirements of dynamic monitoring of sweet peppers' growth status, albeit still not meeting good enough real-time performance. In (Gómez-Zamanillo et al., 2024), Instance Segmentation for pepper phenotyping is based on DL. They tested the program on images of three pepper varieties. They improved on previous state-of-the-art programs, achieving 97% mean average precision for two of the three varieties and 52% on the third one. The algorithm has proved robust but still computationally expensive, requiring about 0.5s on GPU.

2.1 Mask R-CNNs

As previously mentioned, (He et al., 2017a) conceived Mask R-CNN as an extension of the preexisting Faster R-CNN. The Faster R-CNN architecture comprises two primary components. Initially, it features a Region Proposal Network (RPN) responsible for suggesting candidate bounding boxes that indicate potential locations of different objects within the input. Subsequently, these proposed bounding boxes serve as input for the second stage, a Fast R-CNN. In this stage, the model utilizes features from each bounding box to perform classification and bounding box regression tasks.

In the context of these primary modules, the raw image is not directly utilized as input. To enhance efficiency and reduce processing time, Fast R-CNN introduced a concept involving an initial convolutional neural network to generate the feature map for the entire image. This approach is time-saving compared

to individually computing the feature map for each proposed bounding box. The initial module, referred to as the backbone, is succeeded by two subsequent modules: the Region Proposal Network (RPN) and the network head.

2.2 YOLACT

In the context of Instance Segmentation, networks that proved to yield good results are FCIS (Li et al., 2016), Mask R-CNN (He et al., 2017a), RetinaMask (Fu et al., 2019), PANet (Liu et al., 2018), etc. Although performing relatively well, these frameworks cannot be used in real-time due to the computational complexity involved in creating such systems. The sheer number of parameters makes it impossible for these networks to perform on machines with lower computational capabilities. Therefore, the task requires a different architecture that can perform real-time computations.

YOLACT (Bolya et al., 2019), an advanced single-stage object detector, introduces a mask component, utilizing a fully convolutional network (FCN) to propose masks for the entire image. Simultaneously, a prediction head generates coefficients for each mask instance, facilitating effective instance localization. This unique approach distinguishes YOLACT from Mask R-CNN, notably omitting the Region Proposal Network (RPN) layer, resulting in significant speed improvements. However, this speed enhancement comes at the expense of reduced prediction quality, particularly for smaller objects.

The YOLACT model operates as a single-stage instance segmentation network, dividing the task into prototype mask generation and mask coefficient prediction. Its five components include a backbone network (Backbone) incorporating ResNet and feature pyramid (FPN), a mask template generation branch (Protonet), a prediction module, an aggregation branch, and a clipping module. Utilizing ResNet and FPN, the backbone network obtains feature images P5, P4, and P3 and performs convolution on feature image P5 to acquire feature images P6 and P7. The instance segmentation process involves two parallel subtasks. One subtask utilizes feature image P3 in Protonet to generate various mask templates (Prototype mask generation), each sensitive to different instances. The other subtask adds a mask coefficients prediction branch to the target detection branch, generating coefficients representing instance masks (Mask Coefficients). These coefficients are linearly combined with the mask template, and, following bounding box predictions, the image is cropped to achieve instance segmentation (Bolya et al., 2019). YOLACT's architecture is shown in Figure 1.

3 PROPOSED METHOD

As already mentioned, this work focuses on performing Instance Segmentation on Low-Power Edge Devices. Many works (e.g. (Bolya et al., 2019; He et al., 2017a; Wang et al., 2020a) are often meant to run on (possibly quite powerful) desktop GPUs.

While exploring the landscape of advanced models designed for semantic segmentation (Wang et al., 2017) and instance segmentation (Liu et al., 2020; Yang et al., 2019) on edge devices, we found that these models were mainly created for powerful GPU-based edge devices like Jetson Xavier. It is often the case that inference of the same model on the CPU is suboptimal.

Various strategies have been employed to accelerate instance segmentation models, such as Pruning, but many of them only decrease the accuracy. In this context, we now establish a baseline model and propose a solution to reduce the computational intensity of the instance segmentation task without compromising accuracy and inference speed in instance segmentation on NCS2.

Our study started with the Yolact model as our base for single-stage instance segmentation. We then made various structural changes to make it work with minimal computational requirements on the NCS2 edge device. Our main goal is to speed up and optimize the instance segmentation model to make it practical for real-world situations using such a device.

3.1 Architecture's Backbone: ResNet-50

Yolact is a real-time instance segmentation algorithm that divides the instance segmentation task into two parallel branches fed by the backbone detector.

Generally, the backbone refers to the CNN network, which takes the image as input and extracts the feature map upon which the rest of the network is based.

Several CNNs are available, such as AlexNet, VGGNet, and ResNet (Sultana et al., 2020), which are mainly used as a backbone in instance segmentation models. However, many complicated ConvNets deliver higher accuracy than the simple ones; the drawbacks become significant, especially when dealing with an edge device with a simple VPU processor and small RAM.

We opted for ResNet-50 as the backbone for the baseline model due to its more logical choice in terms

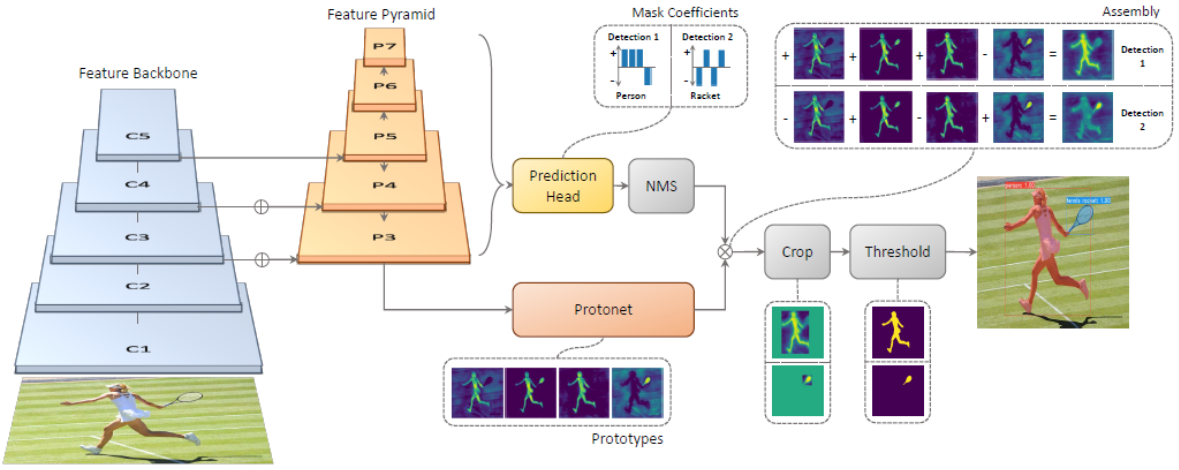

Figure 1: YOLACT's architecture.

of computational complexity with fewer parameters. This consideration is particularly crucial for edge devices, and our research aims to enhance this aspect. YOLACT has 30M parameters, while ResNet-50 has 23M parameters.

3.2 Acceleration with Pruning

Pruning (He et al., 2017b) Convolutional Neural Networks (CNNs) have emerged as a highly effective approach for compressing networks (Ding et al., 2021). Recent trends have witnessed networks growing deeper, resulting in a surge in parameters and convolution operations. However, this rise in high-capacity networks with substantial inference costs presents challenges, particularly in applications involving embedded sensors or mobile devices with limited computational and power resources (Li et al., 2016).

Impressive compression rates have been achieved on AlexNet and VGGNet by pruning weights with small magnitudes and retraining without significantly impacting overall accuracy (Han et al., 2015). Also, this technique seems particularly effective on complex models, where the depth and the number of layers are considerable (Fontana et al., 2024). However, it's important to note that pruning parameters might not always reduce computation time significantly, primarily because most of the removed parameters belong to fully connected layers, which inherently have low computation costs.

Unstructured pruning methods remove individual parameters, resulting in a sparse neural network (Vahidian et al., 2021). However, this approach faces limitations as most frameworks and hardware struggle to accelerate computations involving sparse matrices. For instance, the NCS2 cannot exploit zeros in parameter tensors to reduce the actual cost of the network. This pruning technique is commonly known as Weight Pruning.

Table 1: Size of ResNet-50 after pruning.

Pruning ratio	Param. (M)
40%	10.9
50%	9.2
60%	8.08
70%	7.22
80%	6.64

On the other hand, filter pruning is a structured approach that doesn't introduce sparsity, thereby eliminating the necessity for sparse libraries or specialized hardware (Shao et al., 2021). Pruning filters directly impact acceleration by reducing the number of matrix multiplications. One-shot pruning is performed initially, followed by a retraining strategy that saves retraining time by pruning filters across multiple layers, a crucial aspect for very deep networks. In our scenario, we opted for filter pruning, aiming for an effective device-friendly method and considering the interconnectivity between layers and filters. Unimportant filters are identified and pruned by utilizing ℓ_2-norm as a regularizer. A stage-wise pruning approach allows for a uniform pruning rate across all layers within a given stage. We pruned ResNet50 with 40%, 50%, 60%, and 70%, the results are shown in Table 1.

4 EXPERIMENTAL SETUP

This section provides the experimental setup involved, highlighting the hardware details and the dataset.

4.1 Hardware

Traditionally, as the requirements of deep learning architecture grow, one would need more computational resources and specialized hardware such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs). However, this work aims to involve a low computational end device, the NCS2.

However, the program has also been tested on an Intel i7-9750H. The rationale for the CPU choice is as follows. It is a high-end consumer CPU that came out around the same period of the NCS2 (i.e., Q2'19 for the CPU[1] and Q4'18 for the NCS2[2]). It is also a mobile CPU (i.e., mounted on laptop computers); recall that our goal is to run Instance Segmentation on a low-end piece of hardware, meant to be used in situations where energy consumption is a crucial factor, where powerful (and power-hungry) machines cannot be used. Thus, comparing the NCS2 with a CPU meant to run on a laptop makes sense.

It is also worth noting the difference in power consumption. Intel NCS2 absorbs around 1W (Libutti et al., 2020), while the i7-9750H has a rated TDP of 45W.

4.2 Dataset

Figure 2: Dataset sample.

As previously mentioned, the target dataset is covered by industrial confidentiality. Thus, we used a different dataset as a baseline. Starting from Sweet Pepper dataset on Kaggle [3], we selected 450 images and resized them to 550×550px using bilinear interpolation. The selection has been performed with a balance of the data among all the classes, simulating the sample availability of the target industrial context.

The dataset has been split into 70% for training and 30% for tests. Figure 2 is an example of what images from the dataset look like.

5 RESULTS

5.1 Models Accuracy

Following, we will discuss the accuracy and robustness of the tested models (i.e., YOLACT paired with ResNet-50 and its variations).

As a metric, we will use mean Average Precision (mAP).

5.1.1 Pruned ResNet-50

We first focus on the results obtained by pairing YOLACT with Pruned ResNet-50.

Figure 3 compares the performance of YOLACT paired with vanilla ResNet-50 with various degrees of pruning of ResNet-50.

Base ResNet-50 showed robust results, with a mAP of 92.11%. Pruning the backbone did impact the prediction accuracy. The lowest accuracy model is ResNet-50 pruned at 80%, with an mAP of 83.34%, and the highest accuracy is ResNet-50 pruned at 60%, with an mAP of 88.91%.

These tests show that pruning needs to be balanced; in our tests, we lost almost 10% accuracy on the most pruned backbone. On the other hand, some pruned models didn't lose as much, thus making pruning a viable option if having a smaller model is crucial and losing some accuracy is acceptable.

Interestingly, the mAP doesn't decrease monotonically as the model gets pruned. This behavior, although strange, has already been observed in the literature. A survey on DNN pruning (Cheng et al., 2024) shows the same pattern. Figure 8 of that paper shows pruning on a ResNet-152; accuracy follows an overall increasing trend up to around 60%, and drops down as the pruning percentage increases (the same pattern of Figure 3). In our case, although the trend is the same, the magnitude at which the accuracy changes is greater; this could be explained by the size of the dataset used in this work, which began so small that pruning might have had a greater impact on performance.

[1] https://ark.intel.com/content/www/us/en/ark/products/191045/intel-core-i7-9750h-processor-12m-cache-up-to-4-50-ghz.html

[2] https://www.intel.com/content/www/us/en/products/sku/140109/intel-neural-compute-stick-2/specifications.html

[3] https://www.kaggle.com/datasets/lemontyc/sweet-pepper

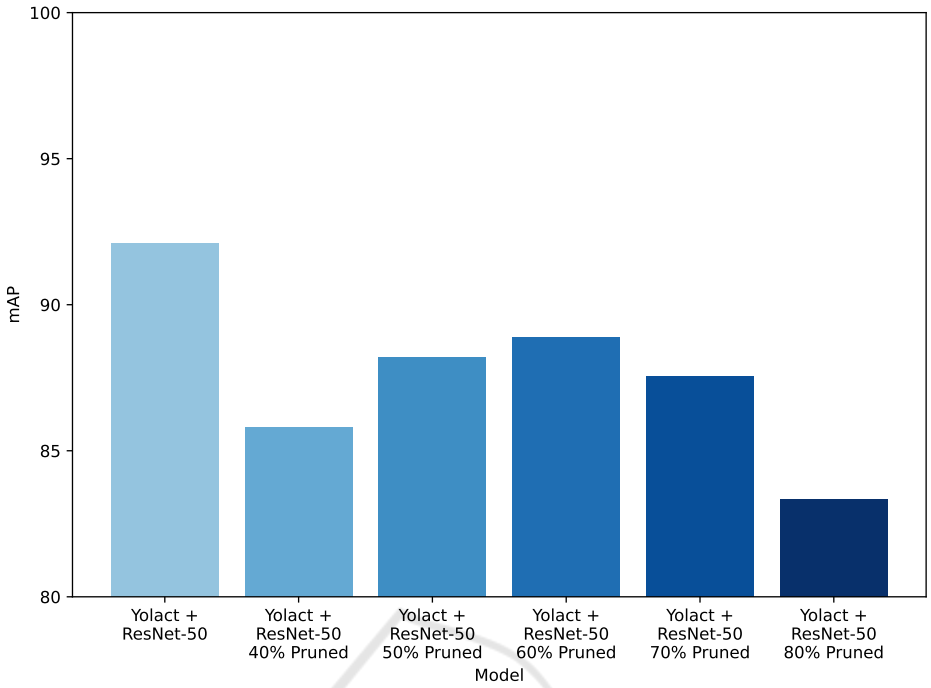

Figure 3: Yolact ResNet-50 pruned ratio trade-off.

5.1.2 ResNet-50 with Various Optimization Techniques

We now focus on many optimizations that can be performed on the backbone to make it lighter.

All the results are summarized in Figure 4.

Reparameterization involving the transformation of ResNet to a simplified VGG-like RMNet architecture revealed promise in improving mask mAP, especially with higher pruning ratios. This methodology showcased its potential for optimizing mask predictions while simplifying the model's architectural complexities.

Integrating CPAC-convolutional layers (Wang et al., 2020b) into the ResNet50 architecture yielded substantial reductions in model size while concurrently maintaining or augmenting mask mAP. Specifically, incorporating CPAC in ResNet50 resulted in a noteworthy increase in mask mAP alongside a considerable reduction in model size, indicating its effectiveness in bolstering mask accuracy.

Depth-wise separable convolutions (DWS) showcased variable impacts on mask accuracy when applied to distinct sections of the YOLACT model. Although effective in parameter reduction, careful consideration is warranted when implementing DWS to ensure it positively influences mask mAP.

It is worth noting that the best optimization method has been CPAC-convolution, which yielded 92% mAP, really close to the 92.11% of base ResNet-50 with no optimization applied.

5.2 Inference Evaluation on Hardware

Following, we show the inference time on both the CPU and the NCS2. The CPU is used as a benchmark for the NCS2.

All the results in this section are summarized in Table 2 and Table 3.

5.2.1 CPU

Figure 5 shows the CPU's mAP and FPS using Yolact paired with ResNet-50 and its optimized variations. These results have been used as a benchmark to compare the results of the NCS2.

CPAC convolution yielded good results, achieving around the same precision of base ResNet-50, but improving the FPS by 78.1%. Pruning at 60% achieved the best inference time, with an increase in FPS of 138.7%, but with a considerable drop in accuracy, going from 92.11% to 83.11%.

5.2.2 NCS2 Device

Following the results of the NCS2. On the NCS2, we have conducted two kinds of tests. In the first kind, we tested Yolact paired with only ResNet-50 with no optimizations, with images ranging from 200×200px to 550×550px. Results are shown in Figure 6.

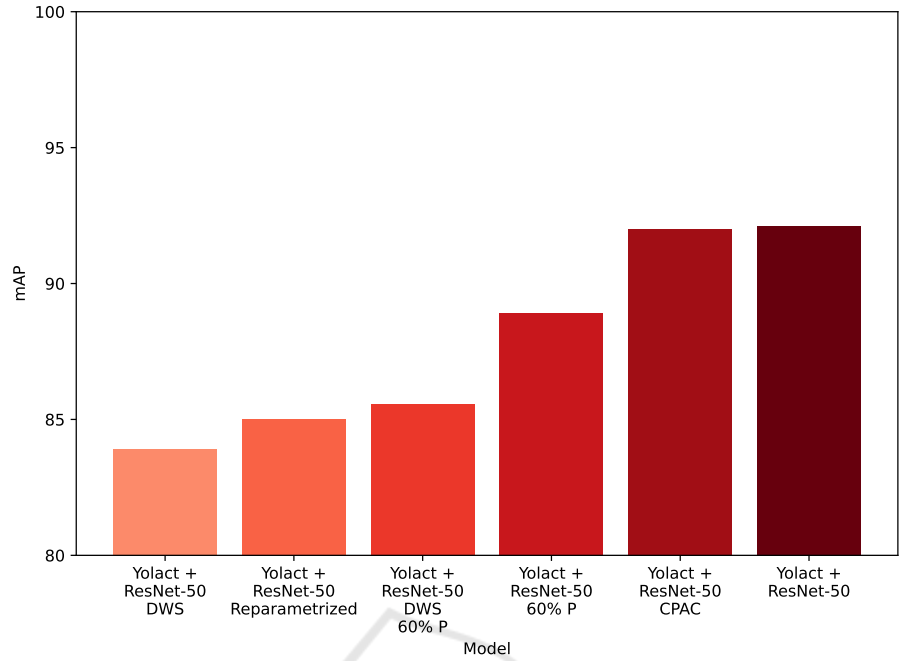

Figure 4: Yolact with ResNet-50 acceleration and optimization methods.

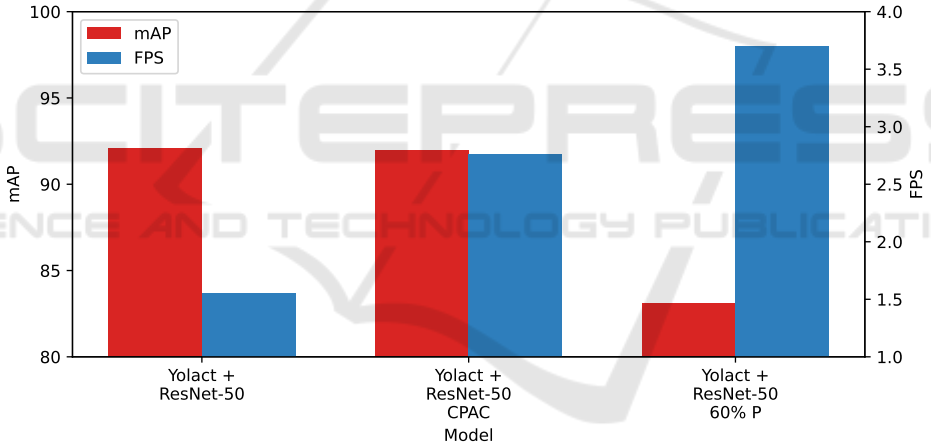

Figure 5: CPU Benchmark results in mAP and FPS.

Decreasing the image size, as expected, improved the inference time at the cost of reducing the precision. Starting from the baseline images at 550 × 550px, with 92.11% mAP and 0.13 FPS, and ending with 200 × 200px images, which yielded 81.11% mAP and 0.4FPS, and increased of 207.7% in FPS and -11.9% on the mAP.

Following, in Figure 7, the mAP and inference time of the NCS2 paired with Yolact and ResNet-50 pruned with increasing ratios, from 40% to 80%.

Increasing the pruning ratio did improve the FPS, but always by a small amount, going from a minimum of 0.254 FPS (pruning at 40%) to a maximum of 0.277 FPS (pruning at 80%). This small increase came at the cost of precision, which dropped precision from 85.81% (pruning at 40%) to 83.34% (pruning at 80%). It is also interesting to see that accuracy does not monotonically decrease: it first goes up, from 40% pruning to 60% pruning, and then decreases from 60% to 80% (see explanation in Section 5.1.1).

These tests on the NCS2 show inference times that, although unsuitable for real-time performance, can be used in practice depending on the application if power efficiency is a concern.

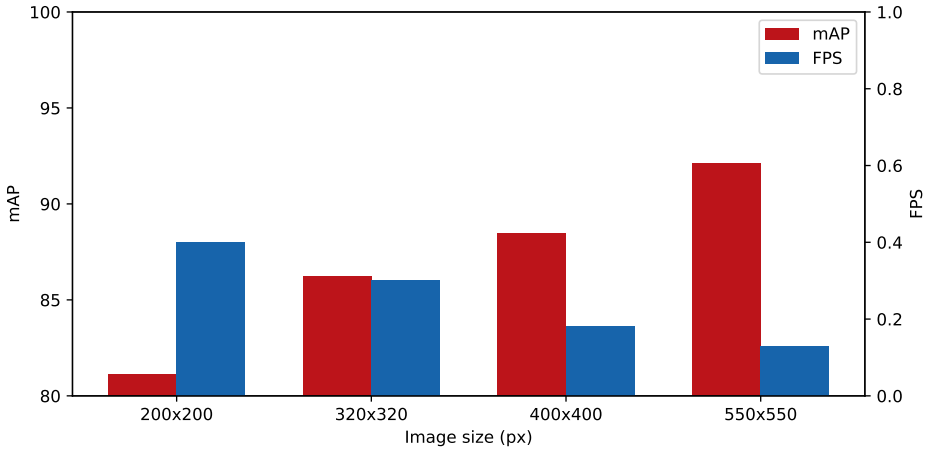

Figure 6: Image size mAP and FPS trade-off.

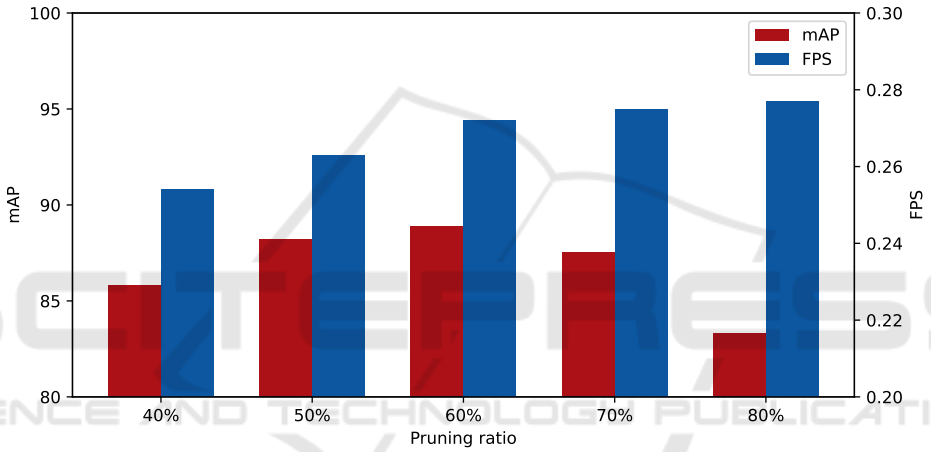

Figure 7: NCS2: image size mAP and FPS trade-off with Pruning.

5.2.3 CPU and NCS2 Comparison

Here, we compare the results of the NCS2 with the benchmark obtained from the CPU. Results shown in Table 2. Figure 6 is not summarized in the table because it represents different kinds of tests, but it is shown in Table 3 for completeness.

Even though the NCS2, in all tests, was performed with less than 1 FPS, in real-case scenarios, it might be as suitable as the performance obtained from the CPU, which maxed at 3.7FPS.

Table 2: Synopsis table for CPU and NCS2.

Hardware	Optimization	mAP (%)	FPS
CPU	None	92.11	1.55
CPU	CPAC	92	2.76
CPU	60% Pruning	83.11	3.70
NCS2	None	92.11	0.13
NCS2	40% Pruning	85.81	0.254
NCS2	50% Pruning	88.22	0.263
NCS2	60% Pruning	88.91	0.272
NCS2	70% Pruning	87.56	0.275
NCS2	80% Pruning	83.34	0.277

6 CONCLUSIONS

In conclusion, an extensive assessment of diverse YOLACT models featuring distinct backbones, optimization, and acceleration techniques were conducted to enhance the inference performance from the baseline. Subsequently, a sequence of hardware configu-

Table 3: Synopsis table for NCS2 with image resizing.

Hardware	Size (px)	mAP (%)	FPS
NCS2	200 x 200	81.11	0.4
NCS2	320 x 320	86.24	0.3
NCS2	400 x 400	88.5	0.18
NCS2	550 x 550	92.11	0.13

rations was carried out to enable inference on the Intel NCS2. To the best of our knowledge, this is the first implementation of this kind of model on this type of device.

Although we reached an accuracy of 95%, 0.5 FPS cannot meet the speed requirement for real-time applications. Nevertheless, this implementation is still usable in different contexts. These results are obtained on the dataset described in Section 4; however, similar values have been collected with the target data provided by the industrial partner. For comparison, we tried naive Yolact on NCS2, which got only 0.09 fps, while our model got 0.5 FPS.

Possible future works include exploring the utilization of more powerful and recent edge devices and a comprehensive analysis of contemporary instance segmentation models.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from Project "Ecosistema dell'innovazione - Rome Technopole" financed by the EU in the NextGenerationEU plan through MUR Decree n. 1051 23.06.2022 - CUP B83C22002820006

REFERENCES

Arnab, A. and Torr, P. H. S. (2017). Pixelwise instance segmentation with a dynamically instantiated network. *CoRR*, abs/1704.02386.

Avola, D., Cinque, L., Fagioli, A., Foresti, G. L., Marini, M. R., Mecca, A., and Pannone, D. (2022). Medicinal boxes recognition on a deep transfer learning augmented reality mobile application. In Sclaroff, S., Distante, C., Leo, M., Farinella, G. M., and Tombari, F., editors, *Image Analysis and Processing – ICIAP 2022*, pages 489–499, Cham. Springer International Publishing.

Avola, D., Cinque, L., Marini, M. R., Princic, A., and Venanzi, V. (2023). Keyrtual: A lightweight virtual musical keyboard based on rgb-d and sensors fusion. In *Computer Analysis of Images and Patterns: 20th International Conference, CAIP 2023, Limassol, Cyprus, September 25–28, 2023, Proceedings, Part II*, page 182–191, Berlin, Heidelberg. Springer-Verlag.

Bai, M. and Urtasun, R. (2016). Deep watershed transform for instance segmentation. *CoRR*, abs/1611.08303.

Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. (2019). YOLACT: real-time instance segmentation. *CoRR*, abs/1904.02689.

Cheng, H., Zhang, M., and Shi, J. Q. (2024). A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10558–10578.

Cong, P., Li, S., Zhou, J., Lv, K., and Feng, H. (2023). Research on instance segmentation algorithm of greenhouse sweet pepper detection based on improved mask rcnn. *Agronomy*, 13(1).

Dai, J., He, K., Li, Y., Ren, S., and Sun, J. (2016). Instance-sensitive fully convolutional networks. *CoRR*, abs/1603.08678.

Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., and Sun, J. (2021). Repvgg: Making vgg-style convnets great again. *CoRR*, abs/2101.03697.

Fontana, F., Lanzino, R., Marini, M. R., Avola, D., Cinque, L., Scarcello, F., and Foresti, G. L. (2024). Distilled gradual pruning with pruned fine-tuning. *IEEE Transactions on Artificial Intelligence*, 5(8):4269–4279.

Fu, C., Shvets, M., and Berg, A. C. (2019). Retinamask: Learning to predict masks improves state-of-the-art single-shot detection for free. *CoRR*, abs/1901.03353.

Gao, N., Shan, Y., Wang, Y., Zhao, X., Yu, Y., Yang, M., and Huang, K. (2019). SSAP: single-shot instance segmentation with affinity pyramid. *CoRR*, abs/1909.01616.

Gonizzi Barsanti, S., Marini, M. R., Malatesta, S. G., and Rossi, A. (2024). Evaluation of denoising and voxelization algorithms on 3d point clouds. *Remote Sensing*, 16(14):2632.

Gómez-Zamanillo, L., Galán, P., Bereciartúa-Pérez, A., Picón, A., Moreno, J. M., Berns, M., and Echazarra, J. (2024). Deep learning-based instance segmentation for improved pepper phenotyping. *Smart Agricultural Technology*, 9:100555.

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626.

Hariharan, B., Arbeláez, P. A., Girshick, R. B., and Malik, J. (2014). Hypercolumns for object segmentation and fine-grained localization. *CoRR*, abs/1411.5752.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. (2017a). Mask R-CNN. *CoRR*, abs/1703.06870.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

He, Y., Zhang, X., and Sun, J. (2017b). Channel pruning for accelerating very deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406.

Khayya, E. K., Oirrak, A. E., and Datsi, T. (2024). A survey on rgb images classification using convolutional neural network (cnn) architectures: applications and challenges. In *2024 International Conference on Circuit, Systems and Communication (ICCSC)*, pages 1–8.

Kirillov, A., Levinkov, E., Andres, B., Savchynskyy, B., and Rother, C. (2016). Instancecut: from edges to instances with multicut. *CoRR*, abs/1611.08272.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. *CoRR*, abs/1608.08710.

Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2022). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019.

Libutti, L. A., Igual, F. D., Pinuel, L., De Giusti, L., and Naiouf, M. (2020). Benchmarking performance and power of usb accelerators for inference with mlperf. In *Proc. 2nd Workshop Accelerated Mach. Learn.(AccML)*, pages 1–15.

Liu, H., Soto, R. A. R., Xiao, F., and Lee, Y. J. (2020). Yolactedge: Real-time instance segmentation on the edge (jetson AGX xavier: 30 fps, RTX 2080 ti: 170 FPS). *CoRR*, abs/2012.12259.

Liu, S., Jia, J., Fidler, S., and Urtasun, R. (2017). Sgn: Sequential grouping networks for instance segmentation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3516–3524.

Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018). Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534.

Pinheiro, P. H. O., Collobert, R., and Dollár, P. (2015). Learning to segment object candidates. *CoRR*, abs/1506.06204.

Shao, L., Zuo, H., Zhang, J., Xu, Z., Yao, J., Wang, Z., and Li, H. (2021). Filter pruning via measuring feature map information. *Sensors*, 21(19).

Sultana, F., Sufian, A., and Dutta, P. (2020). Evolution of image segmentation using deep convolutional neural network: A survey. *Knowledge-Based Systems*, 201:106062.

Vahidian, S., Morafah, M., and Lin, B. (2021). Personalized federated learning by structured and unstructured pruning under data heterogeneity. *CoRR*, abs/2105.00562.

Wang, P., Chen, P., Yuan, Y., Liu, D., Huang, Z., Hou, X., and Cottrell, G. W. (2017). Understanding convolution for semantic segmentation. *CoRR*, abs/1702.08502.

Wang, Y., Guo, W., and Yue, X. (2020a). Cpac-conv: Cp-decomposition to approximately compress convolutional layers in deep learning. *CoRR*, abs/2005.13746.

Wang, Y., Yue, X., et al. (2020b). Cpac-conv: Cp-decomposition to approximately compress convolutional layers in deep learning. *arXiv preprint arXiv:2005.13746*.

Xie, E., Sun, P., Song, X., Wang, W., Liu, X., Liang, D., Shen, C., and Luo, P. (2019). Polarmask: Single shot instance segmentation with polar representation. *CoRR*, abs/1909.13226.

Yang, L., Fan, Y., and Xu, N. (2019). Video instance segmentation. *CoRR*, abs/1905.04804.