R2SNet: Scalable Domain Adaptation for Object Detection in Cloud-Based Robots Ecosystems via Proposal Refinement

Michele Antonazzi, Matteo Luperto, N. Alberto Borghese, Nicola Basilico

***Abstract*— We introduce a novel approach for scalable domain adaptation in cloud robotics scenarios where robots rely on third-party AI inference services powered by large pre-trained deep neural networks. Our method is based on a downstream proposal-refinement stage running locally on the robots, explotiting a new lightweight DNN architecture, R2SNet. This architecture aims to mitigate performance degradation from domain shifts by adapting the object detection process to the target environment, focusing on relabeling, rescoring, and suppression of bounding-box proposals. Our method allows for local execution on robots, addressing the scalability challenges of domain adaptation without incurring significant computational costs. Real-world results on mobile service robots performing door detection show the effectiveness of the proposed method in achieving scalable domain adaptation.**

I. Introduction

Robot-assisted services are today present in a wide range of real-world applications, including healthcare, logistics, domestic assistance, and agriculture [1]. While becoming more and more ubiquitous, autonomous mobile robots are facing a growing need to tackle increasingly complex perception and decision-making tasks for which the recent wave of AI and deep learning offers solutions of unprecedented potential, often available as very large Deep Neural Networks (DNNs) pre-trained on public or third-party datasets.

The computational capabilities that such a need brings about are at odds with the typical profiles of robots: not only are they devices with limited resources, but they need to be. Keeping affordable hardware costs and preserving energy consumption at operational time are mandatory requisites in many real-world scenarios. This is the reason why offloading the computationally demanding inference with DNNs is an emerging trend in the field, for which third-party AI services deployed in the cloud are a convenient solution. Such services have great capabilities, but, as many robotic practitioners are well aware of, also have access constraints and performance barriers. Constraints typically entail that they can only be accessed with queries. Among performance limiting factors, domain shifts are perhaps the most relevant to the field-AI paradigm that robots embody: the data distribution encountered in their target environments can significantly diverge from the distribution on which the cloud-based DNN has been trained. This discrepancy can inevitably result in substantial performance degradation.

Consider these challenges in the scope of a robotic ecosystem where multiple independent units are deployed across different environments and rely on a cloud-based DNN inference service. Assume that the robots' working environments are initially unknown and that the number of robots in the system is expected to increase by deploying new units in novel environments during its operational lifespan. Standard *domain adaptation* techniques [2], [3], where fine-tuning and feature alignment are exploited to train models that can withstand the shift to a target domain, face scalability issues in such a scenario. Qui vogliamo dire che serve 1 modello per ogni robot o che è difficile avere modello installation su ogni robot? The fields of Cloud Robotics [4], and more recently Fog Robotics [5], come in handy by studying inference-serving solutions that distribute, in an adaptive way, the computational and storage loads across robots and cloud services. However, the application of domain adaptation techniques over these architectures is subject to scalability issues. First, it requires full access to the cloud DNN; this is not always feasible if the cloud DNN is provided by an external vendor. Then, it would pose significant computational costs for re-training with target-domain data and updating the DNN; when a new robot is deployed, a new DNN is to be trained, deployed, and maintained, as a DNN model after performing domain adaptation cannot be shared by multiple robots.

In this work, we focus on scalable domain adaptation, a problem at the intersection of cloud robotics and deep learning that, despite being relevant to many real-world settings, is still largely underexplored. We focus on the task of object detection in the general scenario represented by Fig. 1: a set of robots need to carry out such a task from RGB images acquired in their respective environments. To

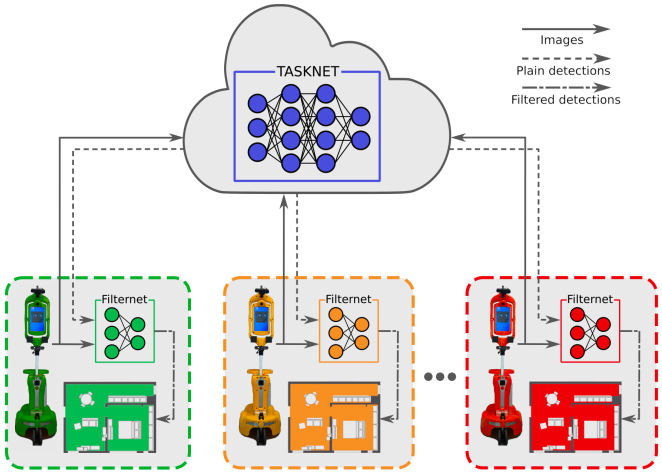

Fig. 1: A general overview of the cloud-based scenario we consider.

All authors are with the Department of Computer Science, University of Milan, Milano, Italy name.surname@unimi.it

such end, they rely on a general-purpose pre-trained DNN, referred to as TaskNet, that is provided as a third-party cloud service, making it accessible exclusively through queries.

The core contribution of our approach is to deploy adaptation as an efficient downstream proposal-refinement stage, running locally on the robots. As we shall detail in Section III-A, this strategy is inspired by the observation that state-of-the-art object detectors typically work by generating dense sets (up to thousands) of bounding-box proposals which then undergo heuristic post-processing via confidence thresholding and non-maximum suppression [6]. Our findings indicate that a substantial portion of the performance degradation due to domain shifts can be mitigated by replacing such post-processing heuristics with a proposal-refinement process adapted to the target environment. To such end, we introduce R2SNet, a novel lightweight DNN architecture for proposal refinement that focuses on three different types of corrective actions: relabeling, rescoring, and suppression of bounding boxes. To carry out such a task, R2SNet leverages the acquired images and the geometrical features of the corresponding bounding-box proposals, and it can be run downstream and locally on the robot.

We evaluate this method in a real-world testbed where mobile service robots must perform real-time *door detection*, that is identifying the location and status (open/closed) of doors/passages through visual recognition [7]. For service robots, this object detection task is key for navigation, but also one recognized as very much affected by domain shifts [8]. The obtained results show how our method enables scalable adaptation, effectively mitigating the performance losses due to domain shifts encountered with the general pre-trained model, all while avoiding the need for substantial computational costs in training and inference.

II. Related Works

Cloud Robotics [4] is an active area of research focusing on engineering the distribution of storage and computational tasks away from robotic platforms to web-enabled architectures [9]. In the last years, this area faced the wave of cyber-physical systems' increasing reliance on large, pre-trained DNNs. Such models pose substantial computational demands that fostered the development of strategies for distributing their workload to the edge, towards ecosystems where edge computing and deep learning become interlinked. Object detection represents one of the most significant testbeds against this background [10], [11].

On of the mainstream approaches for cloud-based DNN workload distribution is Model Splitting [12] where, essentially, the model is divided into two or more portions that are run collaboratively across the network. Examples of this method for object detection are [13] where YOLOv3 undergoes a process of cloud-edge distribution and [14] where inference follows a hierarchical structure from the cloud to the end device.

A series of works, falling under the umbrella of "Fog Robotics", investigated solutions based on a continuum of computing resources from robots to cloud data centers. This paradigm is becoming increasingly widespread [15], with distributed object detection (typically in synergy with grasping) being among its real-world challenges. Examples of works on this line include [5], where models are initially trained in the cloud and subsequently adapted at the edge, [16] where authors focus reducing latency by means of a Q-learning-based policy for load balancing, and [17] where object detection based on SSD [18] is served to the robot from a fog node cluster whose resources can be adapted to guarantee service quality. Other examples of similar offloading strategies for object detection in mobile robots have been proposed in [19] and [20].

Most of these works primarily focus on enhancing service quality but overlook the challenge of scalable domain adaptation in the constrained cloud setting we adopt. In this paper, we directly address this issue with an architecture related to the one proposed in [21] where cooperation between a large cloud-based model and a smaller one operating locally on the robot is exploited. Our method differentiates in the role and design of the smaller model. In such work, the smaller model is essentially a scaled-down, less accurate variant of the larger one, aimed at reducing costs. In contrast, our approach enhances the smaller model's role to not just serve as a cost-effective alternative but to specifically refine and adapt the cloud model's predictions for the robot's unique operational environment in a scalable way.

III. Method

A. Proposals Filtering in Object Detection

Object Detection (OD) amounts to identifying the location and dimension of objects in an image. Deep learning is today the leading approach to building detectors, which are typically based on architectures that analyze the input image through different stages to ensure its comprehensive coverage [22]. Two-stage detectors (such as Faster R–CNN [23]) use a Region Proposal Network (RPN) to predict, in a first stage, proposals of bounding boxes from multi–scale image embeddings. In the second stage, such proposals are classified into object categories. Differently, one–stage models (such as YOLO [24]) directly predict object classes for a set of predefined bounding boxes called *anchors*, which uniformly cover the image with multiple scales and sizes.

Both architectures share a characteristic: they produce many overlapping bounding–box proposals, typically numbering in the thousands, which are independently scored using the image's features. To distill meaningful detections from this dense set, a heuristic two–step post–processing is executed. The first step, called Non-Maximum Suppression (NMS) [6], iteratively selects pairs of proposals whose Intersection Over Union area (IoU) exceeds a threshold ρ_{iou} and suppresses the one with the lowest confidence. In the second step, any proposals with a confidence lower than a threshold ρ_c are also discarded.

For achieving scalable adaptation, we suggest relocating the post–processing step to operate locally on each robot. This entails integrating it as a downstream module of a global cloud–based object detector we call TaskNet (see

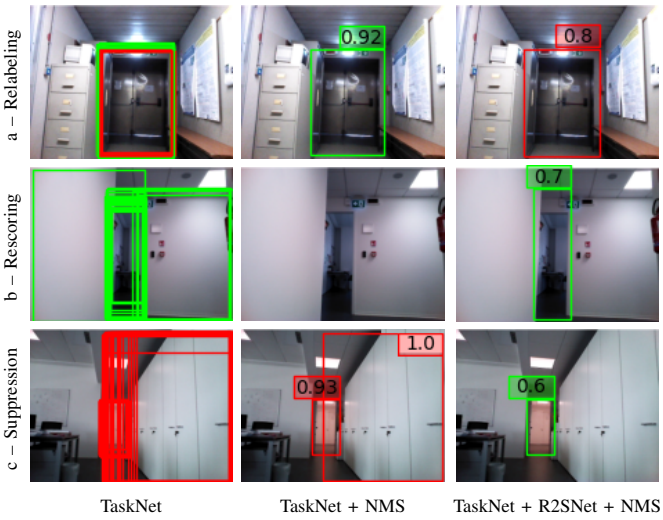

Fig. 2: Refinements obtained with R2SNet in filtering dense proposals compared to standard post–processing. Green/red bounding boxes are open/closed doors.

Fig. 1) which has been configured to return raw proposals[1]. Additionally, we augment this post–processing by incorporating R2SNet, a lightweight deep architecture tailored to the robot's target environment.

We formally denote our TaskNet as $\mathcal{T}_{net}(\cdot, \theta)$, where θ are its pre–trained weights. From an image x^i, we obtain $\hat{Y}^i = \{\hat{y}\} = \mathcal{T}_{net}(x^i, \theta)$ as the set of raw proposals, with

$$\hat{y} = \left[\hat{c}_x, \hat{c}_y, \hat{w}, \hat{h}, \hat{c}, \text{hot}(\hat{o})\right]_{1\times f} \tag{1}$$

where $\hat{c}_x, \hat{c}_y \in [0, 1]$ are the center coordinates, $\hat{w}, \hat{h} \in [0, 1]$ represent width and height, $\hat{c}$ is the confidence, $\hat{o} \in \mathcal{O}$ is an integer indicating the object category, and hot$(\cdot)$ is its one–hot encoding (so $f = 5 + |\mathcal{O}|$). Once received by the robot, the k most confident proposals, where $k \gg O$, with O indicating the maximum number of identifiable objects potentially present in an image, are given as input to R2SNet. Before presenting its architecture, we examine the three primary types of interventions along which the network is trained and used: Relabeling, Rescoring, and Suppression (hence the acronym R2SNet).

Relabeling: Frequently, TaskNet generates several overlapping proposals over the same target object. In challenging instances, it can happen that some of these proposals receive different categories. Fig. 2a shows an example where a door in the middle is both labeled as closed and open. These errors are frequent when involve objects that might resemble each other (e.g., open and closed doors, chairs and armchairs). The standard post–processing based on NMS would select the proposal with the highest confidence disregarding the correctness of its object category. In our method, we improve on this by relabeling all overlapping proposals to a single category, forcing a consensus. Also, we identify isolated proposals not overlapping with any others as spurious. Based on our empirical observations, these isolated proposals often correspond to errors in object localization. Consequently, we relabel them as `background`, an additional category introduced in this stage.

Rescoring: It is well–known that confidence scores may not consistently reflect the actual uncertainty, and thus the likelihood of correctness, of the proposals computed with TaskNet [25]. When a poorly localized proposal receives high confidence, NMS might erroneously reject nearby proposals that better match with the object. Conversely, if a properly localized proposal receives a low confidence, the thresholding step could erroneously discard it. We observed this phenomenon in challenging instances, such as the one depicted in Fig. 2b, where an open door is partially hidden behind a corner, often leading to errors. Controllare To address this, we correlate the confidence of each proposal to the IOU area they have with the best overlapping ground truth box. In this way, the iou threshold ρ_{iou} becomes the only hyper–parameter to setup the post–processing techniques.

Suppression: Another frequent error occurring with a dense set of proposals is that some may be situated in parts of the image where no objects are present. This might happen because the features in these regions mimic an object category that the detector is trained to identify. For instance, Figure 2c highlights instances where cabinets (or windows) are misclassified as doors. While relabeling partially mitigates this issue, we introduce a suppression phase to directly address it by learning a feature embedding for each image region to differentiate between background areas and those containing an object.

B. R2SNet

First, given the k most confident proposals computed by the TaskNet, R2SNet extracts a matrix of descriptors $BD^i = \left[\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k\right]_{k\times f}$ by stacking the vectors defined in Eq. 1. Additionally, it extracts an embedding feature vector for each proposal using a convolutional architecture we call BFNet (Bounding–box Feature Network, detailed in Sec. III-C), to compute a matrix of image descriptors $ID^i_{[k\times 8]}$. BD^i and ID^i can be seen as projections of the k most confident bounding boxes in two distinct spaces $\mathbb{R}^f$ and $\mathbb{R}^8$, which are meant to capture their geometrical and visual features, respectively.

Given these preliminaries, R2SNet (depicted in Fig. 3) is inspired by PointNet [26], which is designed to classify and segment dense point clouds and to be invariant to input permutation (proposals, in our setting). R2SNet processes BD and ID with two symmetric sub–networks. At first, each of them maps the input to a high–dimensional space using a Multi–Layer Perceptron (MLP) shared across the k proposal descriptors, to obtain local features $L_{[k\times l]}$. In the MLPs, the same weights are applied to each descriptor, making the size of the network fixed regardless of the number k of proposals. After this step, the local features are expanded again with another MLP and then aggregated using max (symmetric and permutation invariant), to obtain a global feature vector $G_{[1\times g]}$. This last one is concatenated with

[1]The proposal filtering is a post–processing step kept separated from the model as it parameters are task–dependent.

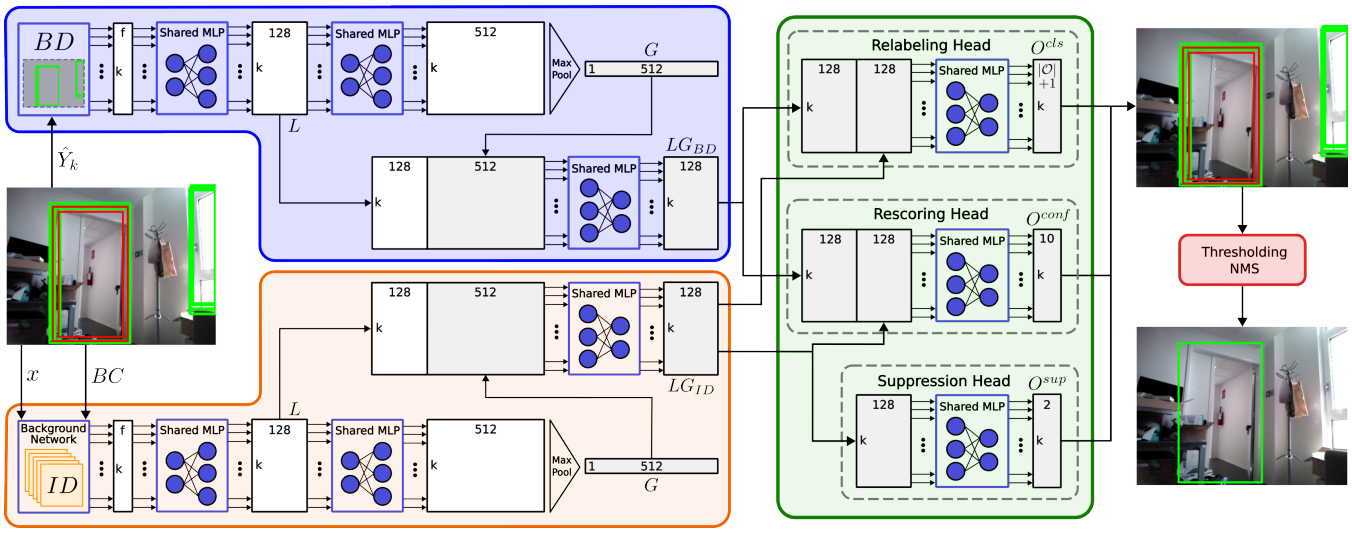

Fig. 3: The R2SNet architecture. Batch normalization and ReLU activation functions are applied to all layers of the shared MLPs.

each row of $L_{[k\times l]}$ and then mixed with a shared MLP, obtaining an embedding $LG_{[k\times 128]}$ that represents both local and global features of the k proposals. The outputs LG_{BD} and LG_{ID} of the two sub–networks are fed into three heads to handle the relabeling, rescoring, and suppression of the proposals.

We denote as Y^i the set of ground truth bounding boxes for image x_i where each $y \in Y^i$ is encoded as per Eq. 1 by setting $c = 1$. We define a matching rule to assign a proposal $\hat{y}$ to a ground–truth bounding box $\hat{y}^{GT} = \arg\max_{y\in Y^i} a_{IOU}(\hat{y}, y)$, where $a_{IOU}(\hat{y}, y)$ is the IOU area between $\hat{y}$ and y. The relabeling head, starting from the concatenation of SG_{BD} and SG_{ID}, assigns to each proposal $\hat{y}$ the probabilities for each object class in the set $\mathcal{O}\cup\{\texttt{background}\}$, producing an output $O^{cls}_{[k\times|\mathcal{O}|+1]}$. This head is trained with the following log–loss:

$$\mathcal{L}_{cls}(O^{cls}) = -\frac{1}{k}\sum_{p=1}^{k} \log\left(O^{cls}_p\right)\cdot \text{hot}(\hat{o}_p), \qquad (2)$$

where $(\cdot)$ is the dot product and $\hat{o}_p$ is the true class for the p-th proposal determined by our matching rule:

$$\hat{o}_p = \begin{cases} Class(\hat{y}^{GT}_p) & \text{if } a_{IOU}(\hat{y}^{GT}_p, \hat{y}_p) \geq \rho_{iou} \\ \texttt{background} & \text{otherwise.} \end{cases}$$

The rescoring head's objective is to align the confidence of a proposal $\hat{y}$ to its IOU area with the associated ground truth $\hat{y}^{GT}$. To achieve this, the confidence score $c \in [0, 1]$ is discretized into 10 intervals. The rescoring head is then tasked with predicting the likelihood that the confidence score falls within each of these intervals, yielding an output matrix $O^{conf}_{[k\times 10]}$. For training, we construct a target vector $v(\hat{y})$ for a proposal $\hat{y}$, whose values peak at the interval corresponding to the IoU score between $\hat{y}$ and the ground truth y, and decrease in a Gaussian-like manner on either side of the peak. In such a way, we obtain a measure of the error that increases with the distance between the predicted and true peaks. This error is adopted for the rescoring loss:

$$\mathcal{L}_{res}(O^{conf}) = \frac{1}{k}\sum_{p=1}^{k} \|O^{conf}_p - v(\hat{y}_p)\|_1. \qquad (3)$$

The confidence assigned to each $\hat{y}_p$ is $\arg\max_j O^{conf}_{p,j}$.

Finally, the suppression head is trained with a loss obtained by adapting (2) for binary classification between proposals that correspond to an object (those for which $\hat{o}_p \neq \texttt{background}$) and those falling on the background.

C. BFNet

BFNet, depicted in Fig. 4, is designed to enable the recognition of those proposals that are wrongly placed on the background (so that can be later suppressed). The reason why image descriptors ID are needed is that such a task becomes quite hard if based only on the geometrical and classification descriptors (location, confidence, and class) of a proposal. BFNet operates by partitioning the input image with a low–resolution grid mask $M_{[W\times H]}$. Then it extracts a feature encoding $IF_{[8\times W\times H]}$ (8 channels for each cell), which is mapped to each proposal's region of interest in a full end–to–end manner.

At first, BFNet extracts a multi–scale feature hierarchy of the input image using a CNN–based backbone with residual connections [27]. The last three embeddings are re–scaled with dimensions $[W \times H]$, $\left[\frac{W}{2} \times \frac{H}{2}\right]$, and $\left[\frac{W}{4} \times \frac{H}{4}\right]$ using adaptive average pooling layers. Differently from a standard Feature Pyramid Network [28], each embedding is processed by three parallel convolutional backbones and step–by–step top–down aggregated through upsampling and summation. The resulting embeddings are concatenated and mixed through convolution to generate a feature map IF.

To select the features related to each proposal IF must be sliced according to the bounding box coordinates. We efficiently obtain this exploiting a series of MLPs (called MaskNet) which produces a binary mask $M^{\hat{y}}_{[W\times H]}$ for each proposal $\hat{y}$ where an element is set to 1 (0) if inside (outside)

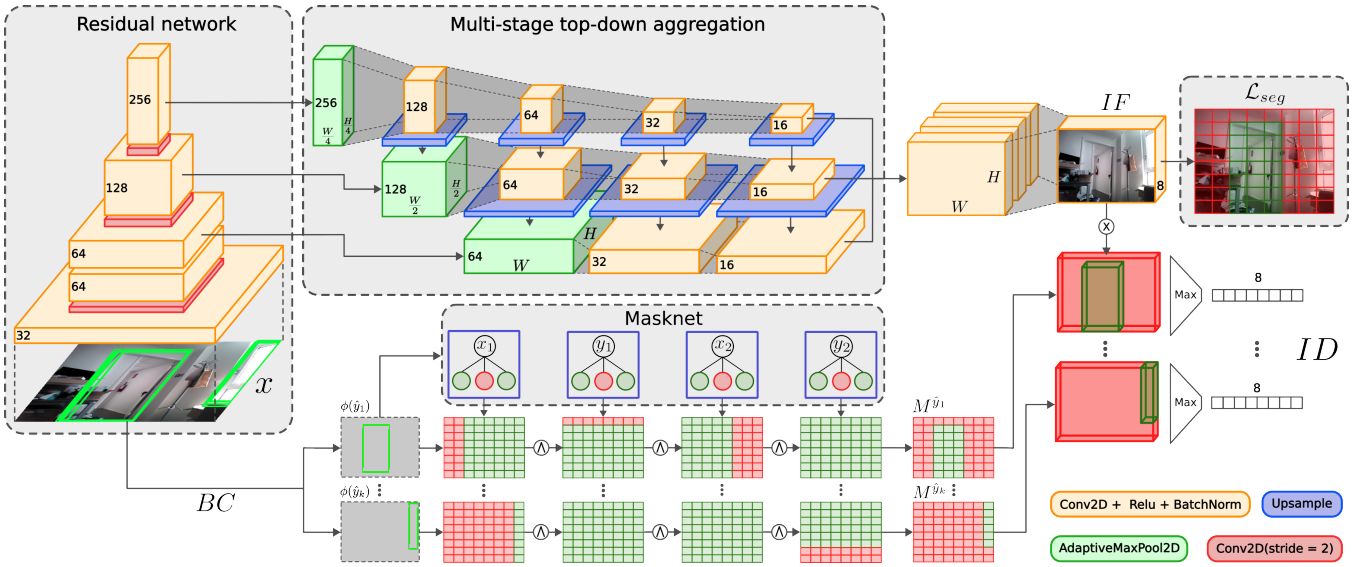

Fig. 4: The BFNet architecture.

the area of $\hat{y}$. This mask is used to suppress the features of IF exceeding the bounding box's boundaries. First, MaskNet receives in input a matrix $BC = [\phi(\hat{y}_1) \dots \phi(\hat{y}_k)]_{k\times 4}$, where $\phi : \mathbb{R}^f \rightarrow \mathbb{N}^4$ encodes an input proposal $\hat{y}$ to a vector $[x_0, y_0, x_1, y_1]$ containing the coordinates of the bottom–left and top–right corners in the grid mask $M_{[W,H]}$. It then computes, for each proposal $\hat{y}$, four binary grids defined as, for $j \in \{0, 1\}$,

$$\begin{aligned} M^{x_j} &= \mathbb{1}_{\leq}\Big((-1)^j(x_j - B)\Big) \\ M^{y_j} &= \mathbb{1}_{\leq}\Big((-1)^j(H - 1 - y_j - B^T)\Big), \end{aligned} \tag{4}$$

where $B = \text{diag}(I_n) \times [0 \dots W-1]$. We obtain the matrices using four MLPs, each comprising one input and $W \times H$ output neurons. The weights are initialized according to Eq. 4 and remain fixed during the training process. The mask of each proposal $\hat{y}$, obtained as

$$M^{\hat{y}}_{[W\times H]} = \bigwedge_{j=0}^{1} M^{x_j} \wedge M^{y_j},$$

is combined with the embedding IF to suppress the features outside the bounding box boundaries. The results are then compressed along the last two dimensions with a max operation, obtaining $ID_{[k\times 8]}$ that encodes the image descriptors of each proposal for R2SNet.

Before training the whole R2SNet, BFNet is pre–trained for addressing a low–resolution binary segmentation task. The image features $IF_{[8\times W\times H]}$ are convoluted into a binary grid mask $M^{seg}_{[2\times W\times H]}$ obtained by training on this loss:

$$\mathcal{L}_{seg}(M^{seg}) = -\frac{1}{WH}\sum_{w=1}^{W}\sum_{h=1}^{H}\log\left(M^{seg}_{w,h}\right)\cdot \text{hot}\big(l_{w,h}\big), \tag{5}$$

where the ground truth label for each cell $l_{w,h}$ is

$$l_{w,h} = \begin{cases} 1 & \text{if } \exists\ y \in Y^i \mid \phi(y) \text{ contains the cell } w, h \\ 0 & \text{otherwise.} \end{cases}$$

IV. Experimental Evaluation

A. Experimental Setting

To carry out our experiments we use 3 publicly available datasets (see [7], [8]) for door detection in RGB images. The first, called *DeepDoors2* ($\mathcal{D}_{\texttt{DD2}}$), has 3K real–world examples of doors in images taken from a human perspective while the second one, named $\mathcal{D}_{\texttt{G}}$ because acquired in 10 environments virtualized through Gibson, contains around 5K photorealistic images acquired from the viewpoint of a mobile robot. The last dataset, $\mathcal{D}_{\texttt{real}}$, is collected on the field by our Giraff–X platform (Fig. 1) equipped with an Orbecc ASTRA RGB–D camera (1 Hz at 320 × 240) [29] while exploring four indoor environments (three university facilities and an apartment, see Fig. 2 for some examples of acquired images). All datasets have $\mathcal{O} = \{\texttt{closed}, \texttt{open}\}$.

The reference TaskNet we will use for testing (assuming it will be deployed in the cloud) is based on Faster R–CNN [23] that, including its ResNet–50 [27] backbone, totalizes 41M parameters. We train it for door detection using all the public data available, namely $\mathcal{D}_{\texttt{DD2}}$ and $\mathcal{D}_{\texttt{G}}$ (60 epochs, batch size of 4). We then deploy it for inference disabling the NMS and thresholding post–processing.

R2SNet needs to be trained from scratch, requiring the expansion of our datasets. For each image, alongside the ground–truth bounding boxes of doors, we must include the corresponding TaskNet proposals. However, it is key to generate these proposals using images unseen by TaskNet during its training, ruling out the use of the reference TaskNet introduced above. To overcome this, we train 11 versions of Faster R–CNN, dividing the datasets into 11 segments (the first includes $\mathcal{D}_{\texttt{DD2}}$, while the others consist each of images from single environments of $\mathcal{D}_{\texttt{G}}$) using a leave-one-out approach for training. These models are then utilized to extract proposals from their respective unseen images. The resulting dataset is used to pre–train R2SNet: we run a first pass training only BFNet using $\mathcal{L}_{seg}$, then in a second

Exp.	e_1				e_2				e_3				e_4			
	mAP↑	**TP↑**	**FP↓**	**BFD↓**	**mAP↑**	**TP↑**	**FP↓**	**BFD↓**	**mAP↑**	**TP↑**	**FP↓**	**BFD↓**	**mAP↑**	**TP↑**	**FP↓**	**BFD↓**
TaskNet	33	41%	10%	13%	27	33%	5%	18%	13	24%	7%	34%	47	47%	6%	14%
R2S^{30}_{25}	39	50%	7%	11%	30	36%	4%	10%	20	26%	7%	12%	58	64%	4%	11%
R2S^{30}_{50}	40	49%	8%	9%	35	40%	6%	6%	22	29%	7%	10%	59	63%	5%	9%
R2S^{30}_{75}	49	54%	5%	7%	35	39%	6%	6%	26	31%	8%	10%	62	62%	1%	5%

TABLE I: R2SNet performance evaluation when trained with increasing amount of data.

pass we train the whole architecture using $\mathcal{L}_{R2S} = \mathcal{L}_{cls} + \mathcal{L}_{conf} + \mathcal{L}_{sup}$ (both passes with 60 epochs and batch size of 16, $k = 30$, and $W = H = 32$).

The pre–trained R2SNet, assumed to be deployed the robot, undergoes adaptation with data specific to its deployment environment, leveraging examples from $\mathcal{D}_{\texttt{real}}$. We label this customized version as $\text{R2S}^{\#proposals}_{\#data}$, where the superscript denotes the number of proposals per training example, and the subscript indicates the percentage of data utilized relative to the total available data in $\mathcal{D}_{\texttt{real}}$ from the deployment environment. In particular, we assessed the impact of using 10, 50, and 100 proposals, alongside 25%, 50%, and 75% of the data. We release the implementation of R2SNet and the code to run the experiments in a publicly available repository[2]

The performance metrics are the mean Average Precision [30] over all object categories (mAP) with 3 additional indicators (formally defined in [7]) measuring the percentage of doors detected with the correct (wrong) label denoted as *TP* (*FP*) and the rate between the background detections and the total number of ground truth objects, named *BFD*.

B. Results

Tab. I shows the performance increasing when the bounding boxes are refined by multiple R2SNet versions trained with an increasing amount of data. The results confirm the effectiveness of our network when compared to the standard post–processing, even when trained with the smallest amount of data. The 25% of images collected during the first robot exploration (which correspond to data acquisition at 0.25 Hz) increase both the mAP and TP while dropping the percentage of wrong proposals (FP and BFD). The performance increasing from R2SNet^{30}_{25} to R2SNet^{30}_{75} demonstrates that our architecture successes in improving the proposals refinement even keeping fixed the number of proposals k. While in e_2 the last two training steps (with data collected at 0.5 and 0.75 Hz) reach comparable performance, R2SNet^{30}_{75} maximizes the amount of correct detection (as shown by the mAP and TP) and ensures the most conservative filtering of errors expressed by FP and BFD in all the other environments.

Fig. 5 shows the performance of R2S^{k}_{75} varying the number of proposals k. The results show a mAP drop with respect to the not–refined proposals when $k = 10$. Despite this setting discards a considerable number of valuable proposals (as corroborated by the lower number of TP), R2SNet manages to reduce both FP and BFD, demonstrating its ability to filter out the spurious detection obtained from the unaltered proposals. Despite the filtering becomes challenging due to the high number of noisy proposals while increasing k, both the mAP and TP reported in Fig. 5 show an increasing trend, justifying the increment of BFD which remain comparable using only the TaskNet.

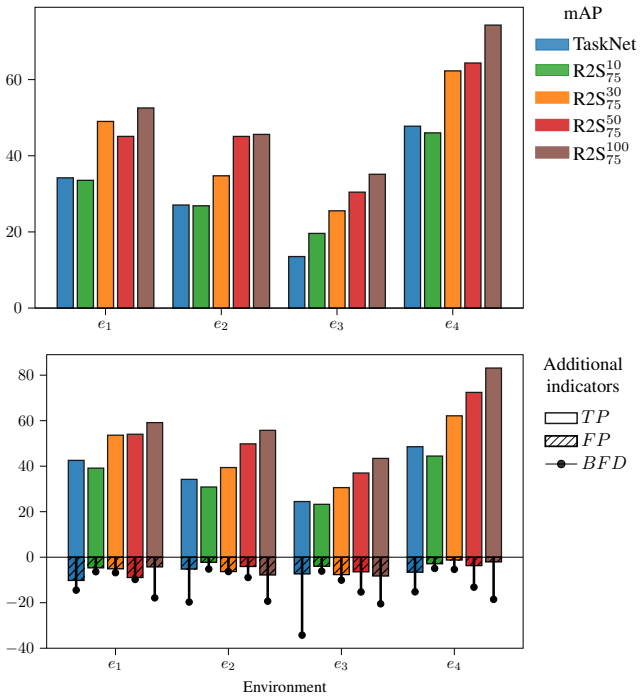

Fig. 5: R2SNet performance when varying k expressed with the mAP (top row) and the additional indicators (bottom row).

To further evaluate the contribution of each network's component, we conduct an ablation experiment considering R2SNet^{100}_{75} which shows the best performance in the previous evaluations. Tab. II reports the results averaged over the 4 environments selectively applying the relabeling, rescoring, and refinement. The worst mAP value is obtained by disabling all refinements of all heads (first row). Suppose we activate only a single head, the relabeling ensures the best mAP improvement as boosts the TP while reducing the FP and BFD. On the contrary, the rescoring head reaches the best TP improvements but cannot reduce the errors, and the suppression head manages to reduce only the number of BFD. The best mAP performance is obtained when all the heads contribute to the proposal refinement, as shown

[2] https://github.com/aislabunimi/door-detection-long-term/tree/bbox_filtering

in the last row of Tab. II. Comparable results are obtained when the suppression head is deactivated, mAP and FP are the same while the (BFD) TP (de)increase by 1%. While this outcome suggests that suppression is redundant during inference as performed by the other two heads, we prove its impact during training as it reduces the BFD by 3%.

Rel.	Res.	Sup.	Envs. averages mAP↑	TP↑	FP↓	BFD↓
			34	44%	10%	35%
✓			44	48%	4%	6%
	✓		41	54%	15%	34%
		✓	37	43%	9%	14%
✓	✓		52	61%	6%	20%
✓		✓	44	47%	4%	5%
	✓	✓	41	53%	15%	31%
✓	✓	✓	52	60%	6%	19%

TABLE II: Ablation study results

Our R2SNet comprises approximately 8 million parameters. To assess its computational demands for inference, we installed it on an NVIDIA Jetson TX2 (a common edge device equipped with both GPU and CPU) mounted on the Giraff–X robot. In this typical service robot configuration, R2SNet demonstrates remarkable efficiency, processing images at rates of 16.7 Hz on the GPU and 2.6 Hz on the CPU. For context, TaskNet processes at significantly lower frequencies of 1.1 Hz and 0.06 Hz, respectively, on the same platforms. These results align with the real–time operational standards expected of mobile robots, illustrating R2SNet's capability for efficient deployment in robots that utilize edge devices, even those without a GPU.

V. Conclusions

In this work, we present a horizontally–scalable domain adaptation solution for robotics ecosystems using a shared third–party AI object detection service. We propose R2SNet, a novel lightweight architecture to refine the proposals locally in the robot to mitigate the performance degradation caused by domain shifts. We validate our framework with real–world experiments addressing the door detection task.

In future work, we will extend our approach enabling the robots to upload domain–independent images to enhance the TaskNet performance. Furthermore, we plan to investigate techniques for adaptive learning for evolving environments.

References

[1] M. B. Alatise and G. P. Hancke, "A review on challenges of autonomous mobile robot and sensor fusion methods," *IEEE Access*, vol. 8, pp. 39 830–39 846, 2020.

[2] P. Oza, V. A. Sindagi, V. V. Sharmini, and V. M. Patel, "Unsupervised domain adaptation of object detectors: A survey," *IEEE Trans. Pattern Anal. Mach. Int.*, 2023.

[3] Y. Lee, A. S. Chen, F. Tajwar, A. Kumar, H. Yao, P. Liang, and C. Finn, "Surgical fine-tuning improves adaptation to distribution shifts," in *Proc. ICLR*, 2023.

[4] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *IEEE Netw.*, vol. 26, no. 3, pp. 21–28, 2012.

[5] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering," in *Proc. ICRA*. IEEE, 2019, pp. 4559–4566.

[6] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," in *Proc. CVPR*, July 2017.

[7] M. Antonazzi, M. Luperto, N. A. Borghese, and N. Basilico, "Development and adaptation of robotic vision in the real-world: the challenge of door detection," 2024.

[8] M. Antonazzi, M. Luperto, N. Basilico, and N. A. Borghese, "Enhancing door-status detection for autonomous mobile robots during environment-specific operational use," in *Proc. ECMR*, 2023.

[9] M. Afrin, J. Jin, A. Rahman, A. Rahman, J. Wan, and E. Hossain, "Resource allocation and service provisioning in multi-agent cloud robotics: A comprehensive survey," *IEEE Commun. Surv. Tutor.*, vol. 23, no. 2, pp. 842–870, 2021.

[10] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surv. Tutor.*, vol. 22, no. 2, pp. 869–904, 2020.

[11] Y. Guo, B. Zou, J. Ren, Q. Liu, D. Zhang, and Y. Zhang, "Distributed and efficient object detection via interactions among devices, edge, and cloud," *IEEE Trans. Multimed.*, vol. 21, no. 11, pp. 2903–2915, 2019.

[12] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, and S. Nepal, "Can we use split learning on 1d cnn models for privacy preserving training?" in *Proc. ASIACCS*, 2020, pp. 305–318.

[13] I. Chakroun, T. Vander Aa, R. Wuyts, and W. Verachtert, "Distributing intelligence for object detection using edge computing," in *Proc. CLOUD*. IEEE, 2021, pp. 681–687.

[14] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. ICDCS*. IEEE, 2017, pp. 328–339.

[15] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. Llontop, D. Xu, C. Buscaron, J. Kubiatowicz, I. Stoica, J. Gonzalez, and K. Goldberg, "Fogros2: An adaptive platform for cloud and fog robotics using ros 2," in *Proc. ICRA*, 2023, pp. 5493–5500.

[16] A. K. Tanwani, R. Anand, J. E. Gonzalez, and K. Goldberg, "Rilaas: Robot inference and learning as a service," *IEEE RA-L*, vol. 5, no. 3, pp. 4423–4430, 2020.

[17] D. Vinod and P. SaiKrishna, "Development of an autonomous fog computing platform using control-theoretic approach for robot-vision applications," *Robot. Auton. Syst.*, vol. 155, p. 104158, 2022.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Proc. ECCV*. Springer, 2016, pp. 21–37.

[19] W. J. Beksi, J. Spruth, and N. Papanikolopoulos, "Core: A cloud-based object recognition engine for robotics," in *Proc. IROS*. IEEE, 2015, pp. 4512–4517.

[20] M. Penmetcha, S. S. Kannan, and B.-C. Min, "Smart cloud: Scalable cloud robotic architecture for web-powered multi-robot applications," in *Proc. SMC*. IEEE, 2020, pp. 2397–2402.

[21] S. Chinchali, A. Sharma, J. Harrison, A. Elhafsi, D. Kang, E. Pergament, E. Cidon, S. Katti, and M. Pavone, "Network offloading policies for cloud robotics: a learning-based approach," *Auton. Robot.*, vol. 45, no. 7, pp. 997–1012, 2021.

[22] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proc. IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[23] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Adv. Neur. In.*, vol. 28, no. 6, 2015.

[24] A. Farhadi and J. Redmon, "YoloV3: An incremental improvement," 2018.

[25] T. Popordanoska, A. Tiulpin, and M. B. Blaschko, "Beyond classification: Definition and density-based estimation of calibration in object detection," in *Proc. WACV*, January 2024, pp. 585–594.

[26] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proc. CVPR*, July 2017.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778.

[28] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. CVPR*, July 2017.

[29] M. Luperto, M. Romeo, J. Monroy, J. Renoux, A. Vuono, F.-A. Moreno, J. Gonzalez-Jimenez, N. Basilico, and N. A. Borghese, "User feedback and remote supervision for assisted living with mobile robots: A field study in long-term autonomy," *Robot. Auton. Syst.*, vol. 155, p. 104170, 2022.

[30] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *Int. J. Comput. Vision*, vol. 88, pp. 303–338, 2009.