2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

R2SNet: Scalable Domain Adaptation for Object Detection in Cloud–Based Robotic 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, exploiting 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 are at odds with the typical profiles of mobile 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 the number of robots in the system is expected to increase by deploying new units in novel environments. 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. 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, performing domain adaptation on the full DNN model would pose significant costs; each time a new robot is deployed, a new DNN is to be trained, deployed, and maintained, as it cannot be shared by multiple robots after performing domain adaptation.

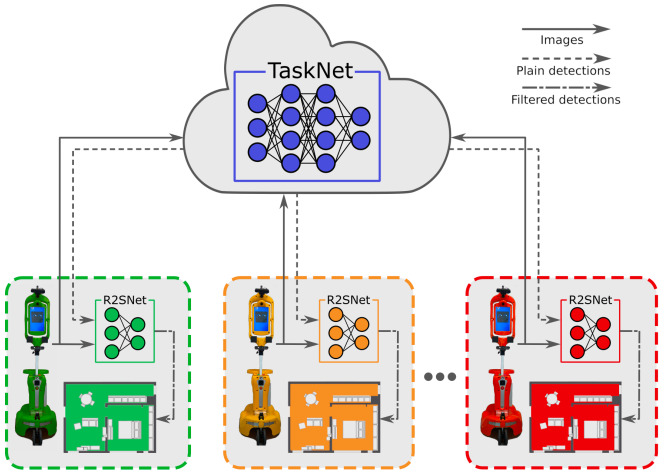

Fig. 1: A general overview of the cloud–based scenario we consider.

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 such end, they rely on a general–purpose pre–trained DNN, called here TaskNet, which is provided as a third–party cloud service, making it accessible exclusively through queries.

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

The core contribution of our approach is to perform domain 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 introducing before such post–processing heuristics a proposal–refinement step 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 [10]. Object detection represents one of the most significant testbeds against this background [11], [12].

One of the mainstream approaches for cloud–based DNN workload distribution is Model Splitting [13] 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 [14] where YOLOv3 undergoes a process of cloud–edge distribution and [15] 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 [16], 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, [17] where authors focus on reducing latency by means of a Q–learning–based policy for load balancing, and [18] where object detection based on SSD [19] 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 [20] and [21].

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 [22] 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 [23]. Two–stage detectors (such as Faster R–CNN [24]) 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 [25]) 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 commonly 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 proposal with a confidence lower than a threshold ρ_c is 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. 1) which has been configured to return raw proposals by modifying hyperparameters. Additionally, we augment this

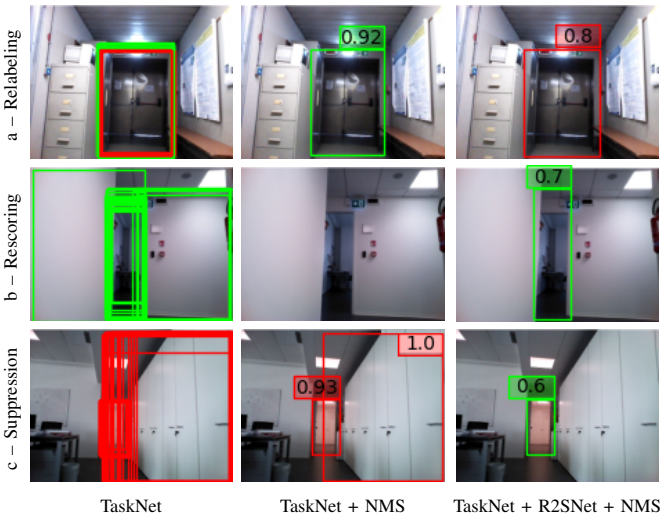

Fig. 2: R2SNet refinements in filtering dense proposals, compared to standard post–processing. Green/red bounding boxes are open/closed doors.

post–processing by incorporating R2SNet, a lightweight deep architecture tailored to the robot's target environment.

From an image x, we obtain from the TaskNet a set of raw proposal $\hat{Y} = \{\hat{y}\}$, 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 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). In the remainder of the paper, we focus on a specific object detection task, *door detection*, as it is particularly significant for this task, as discussed in the examples below. However, our considerations are general to other detection tasks for autonomous robots.

Relabeling: Frequently, a TaskNet generates several overlapping proposals over the same target object; some of these proposals often have contrasting labels. Fig. 2a shows an example of different overlapping proposals that label the same door both as closed and open. These errors are frequent when involve objects that might resemble each other (e.g., open and closed doors or 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 [26]. 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 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. 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 hyperparameter for the post–processing techniques.

Suppression: Other frequent errors occur when dense sets of proposals are 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, Fig. 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 from the image portion of each proposal 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 Bounding–box Descriptors $BD = \left[\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k\right]_{k\times f}$ by stacking the vectors defined in Eq. 1. Additionally, it extracts a feature vector from the portion of the image x corresponding to 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 $ID_{[k\times 8]}$. BD and ID 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.

Given these preliminaries, R2SNet (depicted in Fig. 3) is inspired by PointNet [27], 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 $\max$, to obtain a global feature vector $G_{[1\times g]}$. This last one is concatenated with 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

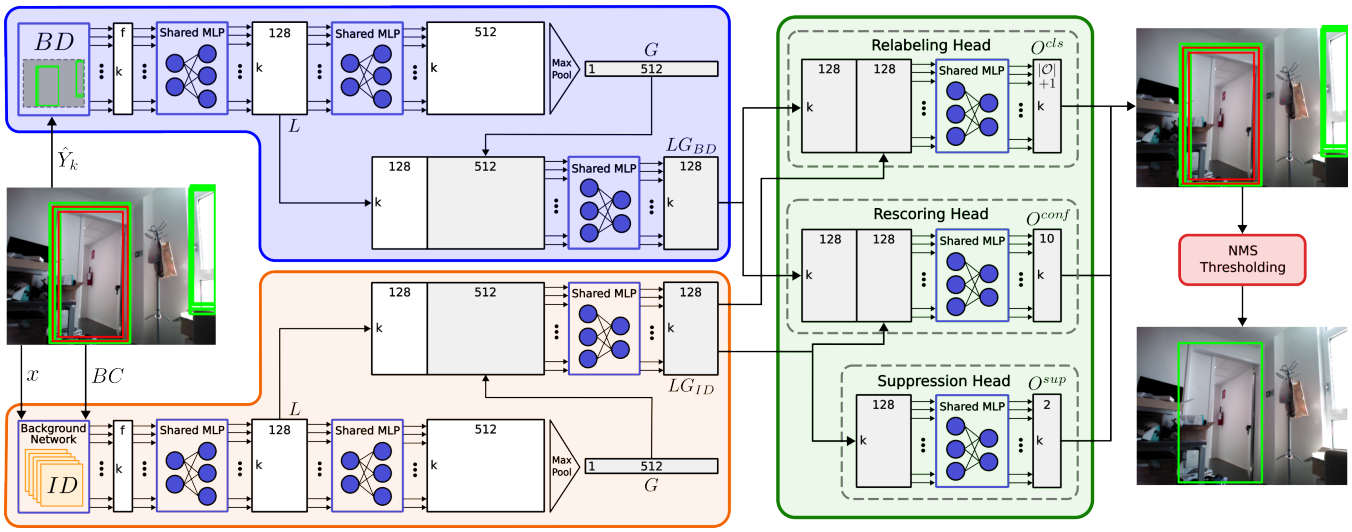

Fig. 3: The R2SNet architecture. Batch normalization and ReLU activation functions are applied to all layers of the shared MLPs.

to handle the relabeling, rescoring, and suppression of the proposals.

We denote as Y the set of ground truth bounding boxes for image x where each $y \in Y$ 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 $\hat{y}^{GT} = \arg\max_{y \in Y} a_{IoU}(\hat{y}, y)$, where $a_{IoU}(\hat{y}, y)$ is the IoU area between $\hat{y}$ and y.

The relabeling head, starting from the concatenation of LG_{BD} and LG_{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 aligns the confidence of a proposal $\hat{y}$ to its IoU area with its 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 its corresponding ground truth $\hat{y}^{GT}$, 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 Eq 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, shown in Fig. 4, guides R2SNet to identify those proposals that are wrongly placed on the background and that can be suppressed. This task is challenging when only the descriptors BD (location, confidence, and class) are used, thus image descriptors ID are needed. BFNet partitions 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.

More precisely, BFNet extracts a multi–scale feature hierarchy of the input image using a CNN–based backbone with residual connections [28]. 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. To aggregate features at different scales, Feature Pyramid Networks [29] (FPN) are commonly used. Differently from what is done in FPNs, to have more descriptive features, 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.

Then, we need to obtain the portion of IF covered by each proposal. Rather than iteratively slicing IF according to each bounding box coordinates, we perform a faster parallel end–to–end mask generation process. More precisely, we use a series of MLPs that 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) the area of $\hat{y}$. This mask is used to suppress the features of IF exceeding the bounding box's boundaries. First, BFNet 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

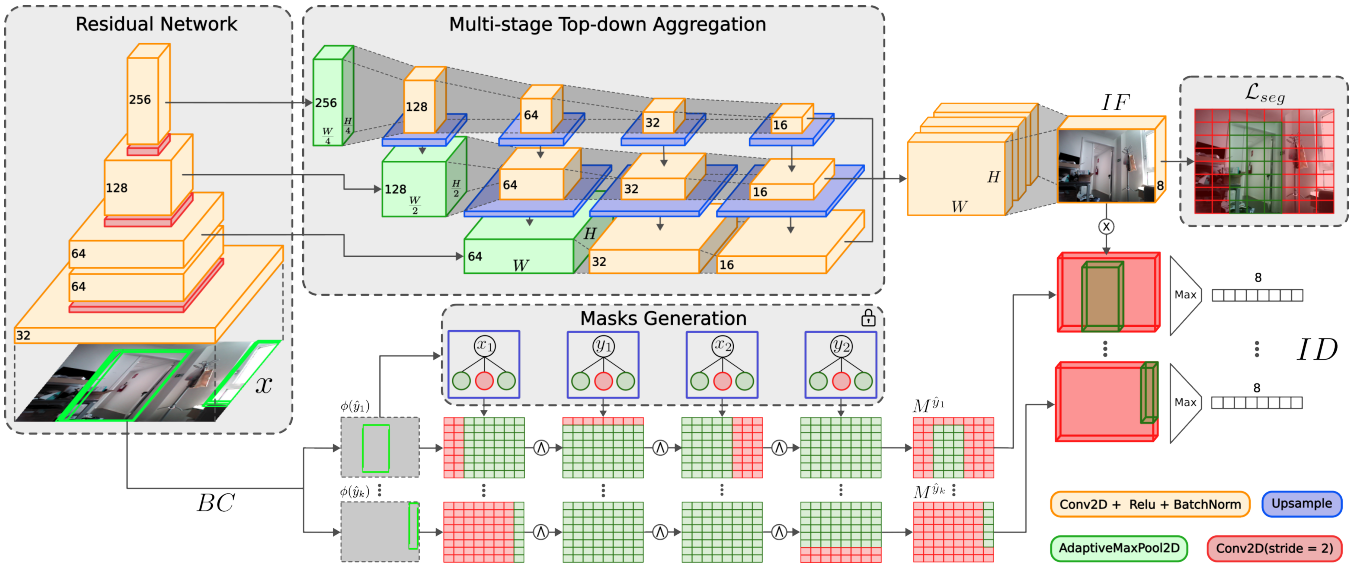

Fig. 4: The BFNet architecture.

and top–right corners in the grid mask $M_{[W\times H]}$. It then computes, for each proposal $\hat{y}$, four binary grids defined as

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

for $j \in \{0, 1\}$, where $A = \text{diag}(I_H) \times [0 \dots W-1]$ and $B = \text{diag}(I_W) \times [H-1 \dots 0]$. 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:

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

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

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

IV. Experimental Evaluation

A. Experimental Setting

Training is performed using 2 publicly available datasets (see [7], [8] for more details) for door detection in RGB images. The first one, *DeepDoors2* ($\mathcal{D}_{\texttt{DD2}}$) [30], has 3K real–world images containing doors, taken from a human perspective. The second one, $\mathcal{D}_{\texttt{G}}$, is taken in 10 environments of Gibson [31] and contains around 5K photorealistic images acquired from the viewpoint of a mobile robot. The experimental evaluation is performed on a third, real–world, dataset, $\mathcal{D}_{\texttt{real}}$, collected with our Giraff–X robot (Fig. 1) [32]; it contains four runs collected when different indoor environments e_* (three university facilities and an apartment, see Fig. 2) have been fully mapped. All datasets have $\mathcal{O} = \{\texttt{closed}, \texttt{open}\}$. We split each run in each environment of $\mathcal{D}_{\texttt{real}}$ in $75/25\%$ for training/testing R2SNet.

We use a Faster R–CNN [24] as TaskNet due to its widespread use; including its ResNet–50 [28] backbone, it has 41M parameters. We train the TaskNet on the full $\mathcal{D}_{\texttt{DD2}}$ and $\mathcal{D}_{\texttt{G}}$ for 60 epochs with a batch size of 4. We then deploy it for inference disabling the NMS and thresholding.

R2SNet needs to be trained from scratch on a large sample of dense proposals obtained by a TaskNet on unseen data; to do so, we augment $\mathcal{D}_{\texttt{DD2}}$ and $\mathcal{D}_{\texttt{G}}$ as in the following. For each image, alongside the ground–truth bounding boxes of doors, we must include the corresponding TaskNet proposals. To do this, generating these proposals using images unseen by TaskNet during its training is key, ruling out the use of the reference TaskNet trained on the full $\mathcal{D}_{\texttt{DD2}}$ and $\mathcal{D}_{\texttt{G}}$. To overcome this, we train 11 versions of Faster R–CNN, dividing the datasets into 11 segments (the first includes $\mathcal{D}_{\texttt{DD2}}$, the others contain images from one of the 10 environments of $\mathcal{D}_{\texttt{G}}$), using a leave–one–out approach. Each TaskNet, trained on 10 segments, is used to extract proposals from the remaining, unseen, one. R2SNet is thus pre–trained with the dataset obtained by combining the 11 segments. We run a first pass training only BFNet using $\mathcal{L}_{seg}$,

Exp.	e_1				e_2				e_3				e_4				$\overline{e}$			
	mAP↑	TP↑	FP↓	BFD↓	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%	30	36%	7%	20%
R2S$^{30}_{25}$	39	50%	7%	11%	30	36%	4%	10%	20	26%	7%	12%	58	64%	4%	11%	37	44%	6%	11%
R2S$^{30}_{50}$	40	49%	8%	9%	35	40%	6%	6%	22	29%	7%	10%	59	63%	5%	9%	39	45%	6%	9%
R2S$^{30}_{75}$	49	54%	5%	7%	35	39%	6%	6%	26	31%	8%	10%	62	62%	1%	5%	43	46%	5%	7%

TABLE I: R2SNet performance evaluation when trained with an increasing amount of data. $\overline{e}$ is the average.

then in a second 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 is adapted with data specific to its deployment environment using examples from $\mathcal{D}_{\texttt{real}}$. We label this customized version as R2S$^{\#proposals}_{\#data}$, where the superscript denotes the number of proposals k 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 (which correspond to $\approx$ 80, 160, and 240 training images, respectively). The remaining 25% of the data are used for testing the R2SNet, using the TaskNet as a baseline. We apply a NMS step to the TaskNet and TaskNet+R2SNet proposals, choosing a set of conservative thresholds: $\rho_{IoU} = 50\%$ and $\rho_c = 75\%$ for the former, and $\rho_{IoU} = \rho_c = 50\%$ for the latter. Note that the domain adaptation of R2SNet is performed once, using the data acquired during the initial deployment of the robot, and is used later on for the whole operative life of the robot.

The performance metrics are the mean Average Precision [33] overall 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 (i.e., false positives detections placed on the background) and the total number of ground truth objects, named *BFD*. We release the implementation of R2SNet and the code to run the experiments in a publicly available repository[1].

B. Results

Fig. 5 shows the performance of R2S$^k_{75}$ varying the number of proposals k. A value $k \in \{30, 50\}$ improves the mAP of $\approx$ 45%, while also reducing both FP and BFD, showing the effectiveness of our R2SNet. Despite the filtering becomes challenging due to the high number of noisy low–confidence proposals, a higher value of $k = 100$ results in a further increase of the mAP, at the expense of higher BFD (that are still close to those obtained by the TaskNet). Interestingly, also a low value of $k = 10$ has some positive aspects; while it does not improve the mAP, it still reduces the FP and BFD. Fig. 5 shows the benefits brought by the R2S$^{100}_{75}$ refinement to the TaskNet raw proposals.

Another interesting remark, confirming the solidity of our approach, comes from evaluating the results when a different and increasing number of samples obtained in the robot target environment are used to train R2SNet, as shown in Table I. For this test, we used $k = 30$. Even with a few examples, R2S$^{30}_{25}$ increases both the mAP and TP of $\approx$ 20% while halving the percentage of BFD. Of course, exposing the R2SNet to a higher number of examples, as in R2S$^{30}_{50-75}$, increases performances, but our findings suggest that a few examples are enough for R2SNet to improve the TaskNet performance on the target environment.

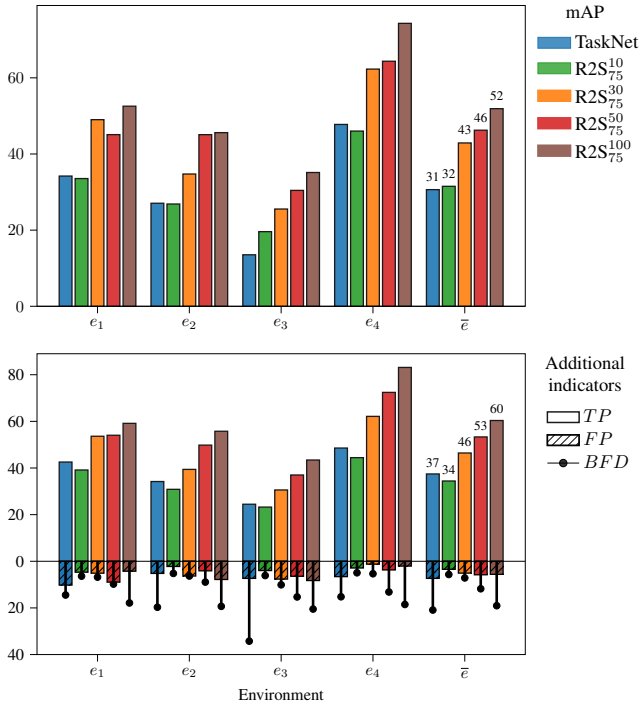

Fig. 5: R2S$^k_{75}$ performance when varying k expressed with the mAP (top row) and the additional indicators (bottom row).

To further evaluate the contribution of each network's component, we conduct an ablation study where the relabeling, rescoring, and refinement heads are incrementally activated. To do this, we use as a reference R2S$^{100}_{75}$, which has the best performances. Table II reports the results averaged over the 4 environments. From this analysis, it can be seen how the relabeling head alone ensures a significant mAP and TP improvement while reducing the FP and BFD. The use of the rescoring head similarly improves the mAP and TP, without reducing FP and BFDs. The best overall performance is shown when all three heads are used. However, the suppression head has less impact on the performance than the other two heads; still, it is needed for training, as disabling it increases the BFD of the 3%.

Our R2SNet has 8M parameters. To assess its computational demands for inference, we installed it on an edge

[1] https://aislab.di.unimi.it/research/r2snet

Rel.	Res.	Sup.	$\overline{e}$ 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.

device, an NVIDIA Jetson TX2, mounted on the Giraff–X robot. On this hardware, commonly available in service robots, R2SNet demonstrates remarkable efficiency, processing images at a rate of 16.7 Hz on the GPU and 2.6 Hz on the CPU. As a reference, TaskNet processes at significantly lower frequencies of 1.1 Hz and 0.06 Hz, respectively, on the same platforms. The R2SNet training with a RTX 3090 GPU took a few minutes. These results show how our method can be used in real–time on a mobile robot, illustrating R2SNet's capability for efficient deployment in robots that utilize edge devices, even those without a GPU.

V. Conclusions

This paper presented a scalable domain adaptation solution for robotic ecosystems relying on cloud–based object detection. We propose R2SNet, a novel lightweight architecture to refine the proposals locally in the robot to mitigate the performance degradation caused by domain shifts.

In future work, we will validate our results with other TaskNet architectures and 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.

Acknowledgements

This work was partially funded by Grant Number G53D23002860006 - PRIN2022, by the Italian Ministery of Research and University.

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] B. Liu, L. Wang, and M. Liu, "RoboEC2: A novel cloud robotic system with dynamic network offloading assisted by amazon EC2," *IEEE T AUTOM SCI ENG*, pp. 1–15, 2023.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk *et al.*, "Fogros2: An adaptive platform for cloud and fog robotics using ros 2," in *Proc. ICRA*, 2023, pp. 5493–5500.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

[29] 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.

[30] J. Ramôa, V. Lopes, L. Alexandre, and S. Mogo, "Real-time 2d–3d door detection and state classification on a low-power device," *SN Appl. Sci.*, 2021.

[31] F. Xia, A. R. Zamir *et al.*, "Gibson env: Real-world perception for embodied agents," in *Proc. CVPR)*, 2018.

[32] M. Luperto, M. Romeo, J. Monroy, J. Renoux, A. Vuono *et al.*, "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.

[33] 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.