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

Cross-Architecture Auxiliary Feature Space Translation for Efficient Few-Shot Personalized Object Detection

Francesco Barbato[1,2,*], Umberto Michieli[1], Jijoong Moon[3], Pietro Zanuttigh[2,†], Mete Ozay[1]

***Abstract*— Recent years have seen object detection robotic systems deployed in several personal devices (e.g., home robots and appliances). This has highlighted a challenge in their design, i.e., they cannot efficiently update their knowledge to distinguish between general classes and user-specific instances (e.g., a *dog* vs. *user's dog*). We refer to this challenging task as *Instance-level Personalized Object Detection* (IPOD). The personalization task requires many samples for model tuning and optimization in a centralized server, raising privacy concerns. An alternative is provided by approaches based on recent large-scale Foundation Models, but their compute costs preclude on-device applications. In our work we tackle both problems at the same time, designing a *Few-Shot IPOD* strategy called AuXFT. We introduce a conditional coarse-to-fine few-shot learner to refine the coarse predictions made by an efficient object detector, showing that using an off-the-shelf model leads to poor personalization due to neural collapse. Therefore, we introduce a *Translator* block that generates an auxiliary feature space where features generated by a self-supervised model (e.g., DINOv2) are distilled without impacting the performance of the detector. We validate AuXFT on three publicly available datasets and one in-house benchmark designed for the IPOD task, achieving remarkable gains in all considered scenarios with excellent time-complexity trade-off: AuXFT reaches a performance of 80% its upper bound at just 32% of the inference time, 13% of VRAM and 19% of the model size.**

I. INTRODUCTION

Object Detection (OD) systems have been a staple in robotics and on a variety of personal devices such as home appliances (*e.g.*, robot vacuum cleaners) and personal assistants [1] for many years. Indeed, several applications require precise localization and recognition of the surrounding objects (*e.g.*, for navigation or object grasping). While most of them need only coarse-level classes (e.g., *person*, *door*, *plant*, etc.), emerging user-centric scenarios demand fine-grained detection abilities to identify personal instances (*e.g.*, *user's friend*, *kitchen door*, *chamomile*). For example, a user watching the television may ask an assistive robot to retrieve its remote, rather than the remote for the hi-fi; a robot smart vacuum cleaner to clean the floor in front of a specific piece of furniture; a lawn mower to avoid specific plants or flowers; a phone to find photos of the user's pets, and so on. We envision that OD systems should allow instance-level personalized understanding (*e.g.*, *dog* vs. *user's dog*) when deployed on resource-constrained robotic devices through a very limited set of user-provided reference samples. We define this task as *Few-Shot Instance-level Personalized Object Detection* (FS-IPOD).

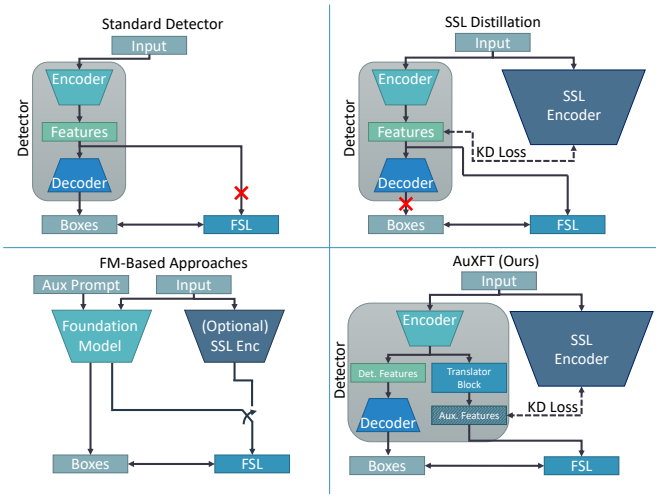


Fig. 1: We explore few-shot instance-level personalized object detection in constrained robotic applications. **Top-left:** standard pre-trained models suffer from *neural collapse*, hence the FSL personalization fails. **Top-right:** naïve knowledge distillation degrades detection performance significantly. **Bottom left:** Foundation Model-driven approaches use compute-heavy vision-text architectures as guidance for feature pooling. **Bottom right (ours):** we create an auxiliary feature space where teacher knowledge is distilled for FSL personalization, without impacting detection performance.

FS-IPOD presents various challenges, chief among which are privacy concerns related to personal data processing. Viable solutions, indeed, should update the OD system on the device itself, while learning with a few samples provided by the user for training [2]–[6]. Regrettably, deep learning based OD systems cannot efficiently update their knowledge once pre-training is complete, and fine-tuning them is typically very computationally expensive. To cope with this, several post-training few-shot learning (FSL) strategies have been proposed [2], [7], [8]. These approaches, however, require the features processed by the architecture to be descriptive enough to distinguish between instances.

As summarized in Fig. 1, existing approaches fall short in

*Research completed during internship at Samsung R&D Institute UK.

† This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

[1]Samsung R&D Institute UK (SRUK), Communications House, South St, Staines, Surrey, United Kingdom {u.michieli,m.ozay}@samsung.com
[2]University of Padova, via Gradenigo 6/b, 35131, Padova, Italy. {francesco.barbato,zanuttigh}@dei.unipd.it
[3]Samsung Research Korea, Seoul R&D Campus, 56, Seongchon-gil, Seocho-gu, Seoul, Rep. of Korea jijoong.moon@samsung.com

learning both discriminative (for OD) and descriptive (for personalization) features. Standard OD systems [9] suffer from the *Neural Collapse* (NC) phenomenon [10], [11]. More precisely, the cross-entropy (CE) objective at pre-training stage causes the models' features to collapse around class centroids, losing intra-class variance and hence their usefulness for the personalization stage. More descriptive features can be learned by models pre-trained via self-supervised learning (SSL) losses [12] instead of CE, making these models especially suited as feature extractors for FSL modules. Nonetheless, distilling knowledge from an SSL pre-trained model (*e.g.*, DINO [13], [14]), into an OD system entails several compatibility challenges only recently addressed [15]–[17] and can affect features for a downstream OD task. Other works explore personalization of large semantic segmentation Foundation Models (*e.g.*, SAM [18]); such approaches require multiple architectures and, therefore, are not suitable for resource-constrained applications [19], [20].

To restore descriptiveness in the features, we propose a cross-architecture knowledge transfer approach via an auxiliary feature space called AuXFT. It aims at improving the personalization ability of OD models while keeping the detection performance unaltered. To achieve this, we distill the knowledge of an SSL-pretrained model into an auxiliary feature space of a detector. However, construction of the auxiliary feature space is highly nontrivial due to the intrinsic architectural differences between the models of interest: in our case, a convolutional network (e.g., YOLOV8n [9]), and a vision transformer (e.g., DINOv2 [14]). The two architectures have different numbers of feature maps, channel depths, and resolution layers. To align the feature spaces of the models, we propose a *translator block* which exploits two *differential* modules. We evaluate the effectiveness of our approach for 1-shot and 5-shot tasks on 3 public datasets (PerSeg [19], iCubWorld [21], and CORe50 [22]), as well as on an in-house benchmark called Personalized Object Detection (POD), capturing personal objects in indoor scenes. [1]. AuXFT shows significant improvements in all scenarios.

The major contributions of our work are: i) We design a pipeline to personalize object detectors to instance-level classes by creating an auxiliary feature space that is used for personalization, while keeping the detection inference unaltered; ii) we introduce a knowledge translator block to align the detector's auxiliary feature space to the one of an SSL feature extractor, allowing knowledge distillation losses to bring intra-class variability into the auxiliary feature space; iii) we design a conditional coarse-to-fine FSL block that extracts representative embedding vectors from the auxiliary space via pooling and employs them for personalized instance-level retrieval with minimal computation.

The rest of the manuscript is organized as follows: Sec. II summarizes the related works; Sec. III outlines the problem statement; Sec. IV introduces each component of our approach; Sec. V reports the experimental results, whose analysis is extended in Sec. VI; and finally, Sec. VII concludes and discusses possible future venues of research.

[1]Code and data are available at: https://github.com/SamsungLabs/AuXFT.

II. Related Works

Cross-Architecture Knowledge Distillation has emerged in recent years as a generalization of hint-based knowledge distillation approaches [23]–[26] (which assume that the architectures' topology matches). The objective is to distill transformer architectures [27]–[33] into more efficient alternatives (*e.g.*, CNNs), since their impressive performance comes at a significant compute cost. The first investigation into the topic by Liu et al. [15] focused on distilling the output space of two heterogeneous architectures. In [16] the authors could improve the performance of a face recognition model introducing an ad-hoc distillation approach that used facial keypoints as hints and focused on the attention maps produced by the teacher. The most recent work on the topic, which is also the closest to our approach, is [17] where distillation on outputs and feature spaces of models is applied simultaneously, mimicking shortcut-based architectures. The core differences compared to our approach lie in the utilized distillation techniques and overall objective. [17] focuses on achieving the highest accuracy on a task shared by teacher and student models, via KL-divergence-based losses to distill teacher logits into the student's output space and projected features. Our objective is different since we consider the reconstruction of the teacher's feature space as an auxiliary task, which we want to solve without impacting the performance of the main task. Therefore, we use losses more geared toward reconstruction and apply them to an auxiliary, translated space.

Few Shot Learning task was first introduced by Miller et al. [34] and it has been extensively investigated in the past years. The seminal work by Snell et al. [2] sparked a variety of approaches [7], [8]. These techniques are based on a prototypical classification approach where class-representative vectors (i.e., prototypes) are stored in memory and used to classify an unknown query vector. Recently, FSL has seen impressive development thanks to its potential in a variety of tasks beyond image classification, especially for dense scene understanding [3]–[5], [35].

Personalized Scene Understanding task traces its roots in NLP [36] where the conversation tone tends to change dramatically for different users, and architectures should adapt accordingly. The first to investigate its application to vision tasks was Zhang et al. [37], which applied the concepts to semantic segmentation. Since then, some few-shot techniques have been introduced, thanks to the development of Foundation Models (FM) such as CLIP [38] and SAM [18], these architectures allow the use of textual prompts to guide the behavior of vision models. Some examples of this recent trend of research can be identified in PerSAM [19], Matcher [39], and SegGPT [40], where the core idea is to prompt an FM and use it to generate either segmentation masks or descriptive features, which can be used for the personalization.

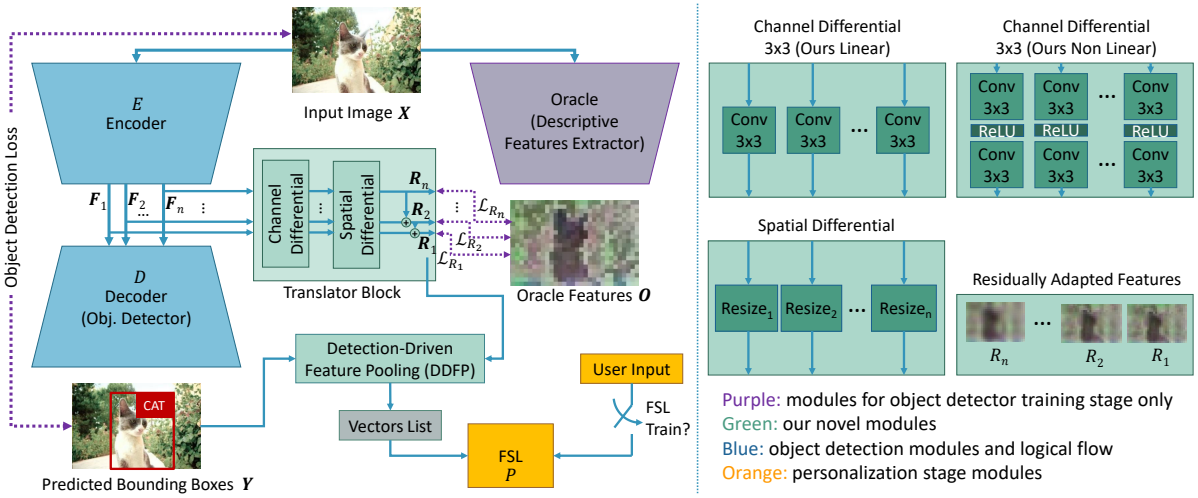


Fig. 2: Overview of our AuXFT. The descriptive features produced by the oracle network are distilled in the auxiliary space through the Translator Block during training. At inference time, the oracle is discarded. For personalization, the predicted boxes and features pass through the DDFP, whose output is fed to the FSL. User input is only needed for FSL training.

III. Problem Formulation

In this work, we focus on personalizing object detectors to instance-level classes for their significant real-world implications in various robot systems such as humanoid, service, and rescue robotics. Nonetheless, in principle, our approach can be applied in any scenario where the knowledge of a compute-heavy model needs to be distilled into a more efficient one, especially in the presence of a mismatch in the architectures or in the task performed by the networks.

In the FS-IPOD task, we assume that we have an OD model that recognizes coarse-level classes $\mathcal{C}_c$ (*e.g.*, *dog*, *cat*, *person*, etc.) and we want to expand the class set $\mathcal{C} = \mathcal{C}_c \cup \mathcal{C}_f$, to include classes more relevant to the user (*e.g.*, *dog1*, *dog2*, *cat1*, *etc.*), without forgetting the original class set [41], [42]. This task is especially useful in personal devices (*e.g.*, robotic assistants, smart vacuum cleaners, phones, *etc.*) which come with very limited computational resources and where, due to the personal nature of the final class set, learning at server-side is undesirable. This means that the refinement operations must be performed in a compute-efficient manner and with a minimal amount of labeled samples since the end-user will need to provide them.

To address these issues, we resort to FSL, where a general model is trained on a chosen scenario with very few samples available for tuning (order of 1-5 samples per class). To mimic the limited number of samples, FSL evaluation is often performed in a cross-validation manner: in each *episode* $\mathcal{T} = (\mathcal{T}_s, \mathcal{T}_q)$, $\mathcal{T}_s \cap \mathcal{T}_q = \varnothing$, a *support set* $\mathcal{T}_s$ of training samples is selected from the available samples, leaving the remaining as the validation *query set* $\mathcal{T}_q$ where the performance is evaluated. This procedure is repeated a number of times, changing the support and query set each time, to obtain the average episodic accuracy, which is a more reliable estimate of the actual system performance. When $|\mathcal{T}_s| = 1$ the scenario is referred to as 1-shot, when $|\mathcal{T}_s| = 5$ we refer to it as 5-shot. A common approach to enable FSL is feature-level prototypical classification, which allows to split the processing burden in two. A frozen and compute-heavy feature extractor can be implemented in hardware, while the variable and efficient FSL classifier in software.

In this work, we propose AuXFT: an FS-IPOD method that can refine the original (*i.e.*, coarse-level) detection class-set with a limited amount of user-labeled personal (*i.e.*, fine-level) samples. In object detection, the input space $\mathcal{X} \subset \mathbb{R}^{H \times W \times 3}$ can be identified as the set of RGB images of size $H \times W$; while the output space is a list of coordinates (x_0, y_0, x_1, y_1), class-assignments (c), and classification confidence (p): $\mathcal{Y} = \{(x_0, y_0, x_1, y_1, c, p)\}_{k=1,\dots,K} \subset (\mathbb{R}^4 \times \mathcal{C}_c \times [0,1])^K$; where K is the number of predicted detections. To enable FSL, we choose an encoder-decoder detector (in particular YOLOV8n [9]), *i.e.* $M = D \circ E : \mathcal{X} \mapsto \mathcal{Y}$, where $\circ$ is the composition operator, so that we can access the intermediate features. Given an input image $\mathbf{X} \in \mathcal{X}$, the encoder produces a set of features $\mathcal{F} \ni \{\mathbf{F}_1, \mathbf{F}_2, \dots \mathbf{F}_n\} = E(\mathbf{X})$ which we can feed to the FSL block $P : \mathcal{F} \mapsto \mathcal{C}$ to obtain the fine-grained classification.

IV. Method

Our system is based on an efficient object detector network, that is modified by our feature *Translator Block*. This block generates an auxiliary feature space where the knowledge of an oracle SSL feature extractor is distilled. The auxiliary features are used at the personalization stage to compute descriptive embeddings for the objects in the scene. Note that the oracle model is only used at training time and discarded for inference. A graphical representation of AuXFT is shown in Fig. 2.

A. Translator Block

We begin by describing our feature translator block, which we use to generate the auxiliary feature space. The block consists of two major components: a *Channel Differential* module (D_C) and a *Spatial Differential* module (D_S) that

will be detailed next. The two modules translate the original feature space into an intermediate space where all feature maps have the same resolution and channel depth. Such features are then used to produce the auxiliary features via residual chain by

$$\mathbf{R}_i = \sum_{j=i}^{n} D_S(D_C(\mathbf{F}_j)) \qquad \forall i = 1, 2, \ldots, n. \tag{1}$$

This allows the high-fidelity and low-resolution estimates generated by the channel-dense features to have their high-frequency components restored by their more shallow but higher-resolution counterparts. We remark that only $\mathbf{R}_1$ is used at inference time by the *Detection-Driven Feature Pooling* (Sec. IV-B) to generate the embedding vectors fed to the FSL block (Sec IV-C).

As common in reconstruction tasks [43], we employ the sum of ℓ_1 and ℓ_2 norms as the distillation loss by

$$\mathcal{L}_{\mathbf{R}} = \sum_{i=1}^{n} \sum_{p \in H'' \times W''} \frac{\|\mathbf{R}_i[p] - \mathbf{O}[p]\|_1 + \|\mathbf{R}_i[p] - \mathbf{O}[p]\|_2}{H'' \times W''} \tag{2}$$

where $\mathbf{O} = E_O(\mathbf{X})$ are the oracle features of an SSL extractor (e.g., DINOv2 [14]) having resolution $H'' \times W''$.

1) Channel Differential: In principle, the channel differential module can be any mapping D_C from the feature space of the encoder to the feature space of the same spatial resolution, but with channel depth matching the one of the target oracle feature extractor. That is, $D_C : \mathbb{R}^{H' \times W' \times l_E} \mapsto \mathbb{R}^{H' \times W' \times l_O}$, where l_E and l_O are the number of channels of detector and oracle, respectively. As detailed in Sec. V, we tested a variety of different projections before choosing a mid-range option offering a good compromise between few-shot performance and computational cost, that is, a 3×3 convolution.

2) Spatial Differential: After projecting the feature vectors in the teacher's channel space, we adapt the spatial dimensions via the spatial differential, to allow computation of the residual maps. In principle, this task can be performed by any mapping $D_S : \mathbb{R}^{H' \times W' \times l_O} \mapsto \mathbb{R}^{H'' \times W'' \times l_O}$. For efficiency, we opted for a parameter-free approach using classical resampling algorithms. Since the aspect ratio (W: H) of detector and oracle features is the same, we employed an adaptive strategy to select the interpolation algorithm depending on the resizing factor $\rho = \frac{H''}{H'} = \frac{W''}{W'}$ as

$$\text{Resampling alg.} = \begin{cases} \text{i) area ,} & \text{if } \rho < 1 - \delta \\ \text{ii) bilinear,} & \text{if } 1 - \delta \leq \rho < 1 + \delta \\ \text{iii) bicubic,} & \text{if } \rho \geq 1 + \delta \end{cases} \tag{3}$$

where δ is a parameter that we empirically set to 0.1. Such a choice finds root in classical signal processing [44]: i) to downsample an image without introducing re-sampling artifacts the best strategy is to average the pixels from the high-resolution source; ii) if the target resolution is close to the source one, then there is no need for large interpolation kernels and a simple linear interpolation is enough, to reduce the complexity overhead; iii) to upsample an image of a larger factor, a more complex interpolation strategy (*e.g.*, bicubic) is preferable to obtain more accurate results.

B. Detection-Driven Feature Pooling (DDFP)

A fundamental component of our AuXFT is the re-labeling of the boxes detected by M. This is done using a prototype-based FSL (Sec. IV-C), which requires embedding vectors as inputs. The task of converting the auxiliary feature $\mathbf{R}_1$ into such vectors (one for each bounding box, $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_K\}$) is handled by the DDFP block, that uses the predicted boxes to summarize the relevant regions of the map into a vector using spatial-aware average pooling:

$$\mathbf{v}_k = \frac{1}{x_1 - x_0} \frac{1}{y_1 - y_0} \sum_{x=x_0}^{x_1 - 1} \sum_{y=y_0}^{y_1 - 1} \mathbf{R}_1[x, y], \tag{4}$$
$$\forall\ \mathbf{Y}_k = (x_0, y_0, x_1, y_1, c, p) \in M(\mathbf{X}).$$

During FSL training, these vectors are accompanied by the original coarse class prediction $c \in \mathcal{C}_c$ and by a fine class assignment $c' \in \mathcal{C}_f$ provided by the user. At inference time, only the coarse class prediction is needed by the FSL module to predict a new fine-class assignment.

C. Few Shot Learner

Our FSL module is based on a prototypical network [2], but is tailored for efficiency, given the computational constraints of robotic applications. In general, a prototypical network is trained by providing a series of labeled vectors, which are used to build internal class prototypes. During inference, the distance between a query vector and all internal prototypes is computed and used to classify the vector by choosing the class of the closest prototype.

This approach scales poorly with the number of classes. Therefore, we implement our FSL in a conditional fashion, that is, we limit the search space of the fine-level class on the basis of the coarse class predicted by the detector (rather than searching among all possible options). This allows our method to be about 3 times faster than the standard non-conditional FSL, without significant impact on the re-classification accuracy, as we observe in the ablation study.

Another difference compared to standard FSL modules is the introduction of an additional vector in each of the conditional sets, which we use as a fallback when a query does not match any fine class. It is the centroid of the set and will match the vectors that are not similar enough to any of the prototypes. This addition requires the FSL to be robust and for this reason, we employed multiple distances and a majority voting approach for the classification. Given a set of distance functions $\mathcal{D} = \{d_1, d_2, \ldots\}$, a query vector $\mathbf{q}$, its coarse class prediction c and a set of prototypes $\mathcal{P}[c] = \{\mathbf{p}_1, \mathbf{p}_2, \ldots\}$, we can compute a new class assignment as follows:

$$\mathbf{d} = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \text{softmax}\left\{ \frac{1}{d(\mathbf{q}, \mathbf{p}_i)}; \forall i = 1, \ldots, |\mathcal{P}[c]| \right\}, \tag{5}$$

$$c' = \text{argmax}_{i=1,\ldots,|\mathcal{P}[c]|}\ \mathbf{d}[i]. \tag{6}$$

TABLE I: Average mAP50-95 over 100 episodes (↑). Our target is a good trade-off between performance (first four columns) and costs (last three columns). no pers.: metrics on coarse class set. Last two rows: AuXFT (3x3 lin.) score in relative terms compared to Yolo Only (last feat.) and Oracle (single fwd.), respectively. † retrieval accuracy is used instead of mAP50-95. Note that the 5-shot results in POD do not have an std, because in that scenario all training samples are used for tuning.

Arch.	Setup	PerSeg (13/39)	POD (15/45)		iCubWorld† (6/31)		CORe50† (10/50)		Time	VRAM	Size
		1-shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	[ms/im] (↓)	[MB] (↓)	[MB] (↓)
Standard	Yolo Only (last feat.)	38.5±2.7	20.9± 2.1	27.7	49.8±2.4	65.2±2.4	56.7±6.5	65.6±7.5	32.3±5.2	221.4	12.2
	Yolo Only (concat.)	41.2±2.6	23.6±2.4	30.4	51.2±3.0	68.4±2.4	57.8±6.3	67.3±7.7	50.5±5.3	221.5	12.2
	Inverse [24]	41.3±3.0	21.1±2.4	25.6	50.6±2.9	63.9±2.5	56.6±6.0	65.4±7.3	48.6±5.3	221.5	12.2
	AvgStd [20]	41.0±2.7	22.9±2.2	27.8	50.6±2.5	69.2±2.1	58.0±6.2	68.5±7.1	53.5±5.3	222.0	12.2
Translated	AuXFT (1x1 lin.)	49.6±3.1	29.4±2.5	35.4	55.1±2.7	73.4±2.1	58.4±6.0	67.3±7.6	32.7±5.1	283.7	12.9
	AuXFT (3x3 lin.)	48.8±3.4	31.5±2.7	38.8	55.0±3.0	74.5±2.5	58.8±6.4	69.3±6.8	32.6±5.1	288.7	18.4
	AuXFT (5x5 lin.)	49.5±3.3	34.1±2.5	39.1	55.0±3.0	74.3±2.3	59.3±6.0	69.8±7.1	34.6±5.1	431.6	29.4
	AuXFT (3x3 non-lin.)	50.4±3.4	32.7±2.3	40.5	52.3±2.9	72.7±2.5	59.8±5.4	70.8±5.9	36.1±5.1	342.4	49.4
Multi	Oracle (single fwd.)	61.9±3.0	40.4±2.7	47.9	67.5±2.6	84.8±2.0	65.4±7.6	75.7±8.3	102.2±4.7	2313.9	98.5
	Oracle (multiple fwd.)	62.6±3.1	46.0±3.0	62.0	73.1±2.6	87.9±1.7	74.5±6.3	84.5±5.6	357.3±12.8	338.3	98.5
	Upper Limit (no pers.)	69.9	70.0		99.8		98.9		-	-	-
	AuXFT Rel. Baseline	126.8%	150.7%	140.1%	110.4%	114.3%	103.7%	105.6%	100.9%	130.4%	150%
	AuXFT Rel. Oracle	78.8%	78.0%	81.0%	81.5%	87.9%	89.9%	91.5%	31.9%	12.5%	18.7%

Effectively, we compute an assignment probability distribution from the inverse of the distance between the query vector and the prototypes for each function (for the results we used cosine, ℓ_1 and ℓ_2 distances, see Sec. VI). We then average over the distances and select as the fine class the index corresponding to the maximum.

D. Datasets

We pre-train all detectors on a subset of the OpenImages dataset [45] with 31 coarse classes, covering the ones present in the personalization datasets, personalization classes absent from OpenImages were discarded.

PerSeg [19] dataset contains 212 images in 39 instance-level classes (13 coarse). To the best of our knowledge, it is the only dataset proposed for few-shot personalized dense tasks. PerSeg was originally proposed for semantic segmentation, so we converted the annotations for OD. Some instances have a limited number of samples (as low as 4), precluding the 5-shot scenario and the coarse-to-fine mapping is unbalanced, that is, some classes have only one personal example, while others have up to eight.

POD (Personalized Object Detection) dataset is an in-house benchmark of everyday objects in indoor scenes (*e.g.*, glasses, shoes, bottles, etc.) developed to solve the limitations of PerSeg. We capture a total of 5 training images for each of the 45 instances considered (3 for each of the 15 coarse classes) for a total of 225 images. In the validation set, we capture images both in light and dark conditions. We acquired 3 validation images per instance: alone in the scene, mixed with the other same-class instances, and mixed with instances from different classes (for a total of 150 images).

iCubWorld [21] dataset focuses on robotic views and has been acquired using video sequences. We select 15 samples per sequence (31 fine classes, 6 coarse) in the FSL split. The bounding boxes have noticeable padding around the objects making them incompatible with the pre-training. Therefore, we fine-tuned the decoder before training the FSL and used the retrieval accuracy instead of OD metrics.

CORe50 [22] dataset provides a variety of object instances captured in variable-background scenes (10 classes, 5 instances per class). We selected a frame with a valid bounding box every thirty from the source videos. The boxes were generated algorithmically and are squared with significant padding around the object. Therefore, we fine-tune the detector and use the accuracy in this dataset as well. To align with the other domains, we set up the episodes on this dataset to have a single scene inside them.

E. Implementation Details

We train using images (394,619 in total) with resolution 672x672px, Adam [46] optimizer for 50 epochs adopting a cosine annealing with linear warmup (1000 iterations) scheduler for both learning rate (max 1×10^{-3}) and weight decay (max 1×10^{-4}). We follow the same policy as [9] loading weights of a detector pre-trained on the COCO dataset [47] using an exponential moving average on the weights for a more stable training evolution. Maximum VRAM and inference time were measured on an NVIDIA 1080Ti GPU.

V. Experimental Results

We validate AuXFT on four benchmarks against some baseline architectures and compute-heavy upper bounds (we ignore FM-based strategies since their cost makes them unviable on-device). We choose two metrics for the evaluation, both averaged over 100 episodes, as common practice in FSL strategies: mAP50-95 (mean Average Precision 50-95) and retrieval accuracy. mAP50-95 is the average of the IoU-thresholded mAPs from mAP50 to mAP95 (in steps of 5), each measuring the ratio of correctly predicted boxes with at least N% IoU [9], [47]. The retrieval accuracy instead measures the fraction of boxes correctly classified, regardless of localization. We remark that both are computed on the

TABLE II: Results when pretraining class set matches the deployment set. Average mAP50-95 over 100 episodes (↑).

Setup	PerSeg (13/39)	POD (15/45)	
	1-shot	1-shot	5-shot
Yolo Only (last feat.)	34.8±3.0	22.1±2.2	32.0
Yolo Only (concat.)	38.2±2.9	25.6±2.4	30.3
AuXFT (3x3 lin.)	49.5±3.2	33.0±2.7	40.6
Oracle (single fwd.)	60.4±3.2	42.3±2.8	48.2
Oracle (multiple fwd.)	61.2±3.2	47.6±2.9	61.9
AuXFT Rel. Baseline	142.2%	149.3%	126.9%
AuXFT Rel. Oracle	82.0%	78.0%	84.2%

TABLE III: Results when both fine and coarse classes are predicted in FSL. Average mAP50-95 over 100 episodes (↑).

Setup	PerSeg (13/39)	POD (15/45)	
	1-shot	1-shot	5-shot
Yolo Only (last feat.)	19.9±2.7	12.5±1.9	19.3
Yolo Only (concat.)	22.4±2.8	14.1±2.3	23.3
AuXFT (3x3 lin.)	34.5±3.2	22.9±2.7	34.1
Oracle (single fwd.)	46.7±3.0	26.9±2.6	38.5
Oracle (multiple fwd.)	49.9±2.9	33.6±2.9	50.1
AuXFT Rel. Baseline	173.4%	183.2%	176.7%
AuXFT Rel. Oracle	73.8%	85.1%	88.6%

fine-grained class set, which on average, contains 5 times more classes than the coarse set in the considered datasets.

The quantitative results for one- and five-shot learning are reported in Table I, together with the average throughput, maximum VRAM usage, and model size. Here we report five groups of strategies: i) two baselines (non-distilled YOLOv8n); ii) two competing approaches (similar to hint-based knowledge distillation); iii) four implementations of our approach (changing D_C); iv) and two oracle upper bounds (joint use the object detector and the SSL pretrained extractor). Additionally, we report a global upper limit, which is the performance of the detector on the coarse class-set, and the relative performance of our method when compared to the *Yolo Only (last feat.)* baseline and the *Oracle (single fwd.)* upper bound. The difference between the two baselines lies in how the vectors input to the FSL are generated: in *last feat.* we use YOLO's last feature map only, while in *concat.* we use all three feature maps and concatenate the results. In the *single fwd.* oracle setup we use the FSL in the same way as our approach, but substitute $\mathbf{R}_1$ with $\mathbf{O}$. In the other scenario, *multiple fwd.*, we use the detector predictions to cut the original RGB image, rather than the feature maps, these sections are then fed into the oracle network to extract a vector embedding. Note that this strategy needs as many forward passes as the number of detected boxes.

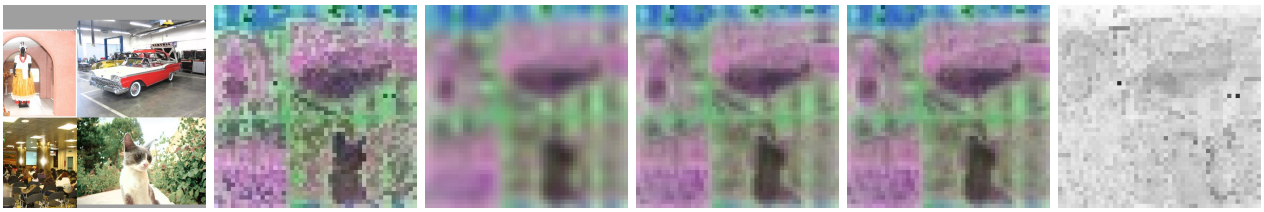
Table I shows how our strategy significantly improves performance over the baseline strategies and competitors using the unmodified feature space, regardless of the actual channel differential block implementation. Compared to the baseline, AuXFT gains an average of 10.5 mAP in PerSeg, 9.7 mAP in POD 1-shot scenario, and 9.4 in POD 5-shot. This performance is a very close match with the oracle upper bound, where we achieve a consistent 80% relative performance with less than a $1/3$ of the inference time and VRAM usage. Conversely, due to the scenario mismatch involved in iCubWorld and CORe50, the gains in retrieval accuracy are less impressive in absolute terms, but much more significant when compared to the upper bound. In these datasets, we achieve an astonishing 87.7% of the oracle performance. Another interesting point regards the efficiency of our approach: with just a small increase of 0.3 ms/im in throughput and 65MB of VRAM compared to the baseline, AuXFT *1x1 lin.* as well as *3x3 lin.* increase the mAP by about 10 points on PerSeg and POD. Given that VRAM and inference time of the two approaches are nearly identical, we choose the *3x3 lin.* as our reference implementation.

We remark that the performance in POD is lower than PerSeg due to the inherently more difficult setup. Indeed, POD has more classes and contains dark scenes. As reported in Table I the global upper limit is 70.0 mAP, but the number rises to 79.0 when only bright scenes are considered (and it decreases to 61.5 for dark scenes only). This difference is reflected in the personalization as well, where our approach can achieve 33.9 (46.9) in bright scenes, on 1- (5-) shot, respectively. In dark scenes the performance is lower, falling to 31.0 (35.0).

To further explore our approach we report some additional experiments in Tables II and III. In the former we study the results when dataset-specific pretraining is employed, that is, the OpenImages subset is selected to match exactly the deployment class set. In the latter, we explore the performance of our approach when coarse-level fallback is enabled in the FSL.

Table II shows that using dataset-specific checkpoints for our approach gains on average 1.3 mAP compared to the unified pretraining results (Table I). This confirms the viability of pretraining the detector once for all considered scenarios. Even more, in the baseline and oracle runs the improvement is inconsistent: in the PerSeg dataset using a restricted class set leads to an average performance decrease of 2.4 mAP, conversely, in POD, we see an improvement of 1.4. This is probably due to the different number of personal instances per coarse class in the two datasets.

The results in Table III confirm the effectiveness of our strategy, even in the challenging scenario where the FSL predictions belong to $\mathcal{C}$, which contains 51 and 60 classes, for PerSeg and POD, respectively. In general, the experiments are consistent with all previous cases showing how our approach bridges the gap between the baseline and oracle architectures. Interestingly, in this scenario, the relative gain compared to the baseline is much more significant, with an average of 177.8%. Comparing with the oracle we see a different behavior: in POD the performance increases (compared to results in Table I) by an average of 7.4%, while in PerSeg we see a loss of 5%. We suppose this depends on the unbalanced nature of PerSeg's class set, which disproportionally penalizes our method when the coarse classes are enabled in the FSL. We remind the reader that PerSeg has multiple classes with only one instance (see Sec. IV-D).



(a) RGB Input (b) Oracle ($\mathbf{O}$) (c) Partial I ($\mathbf{R}_3$) (d) Partial II ($\mathbf{R}_2$) (e) Full ($\mathbf{R}_1$) (f) Cosine Sim.

Fig. 3: Auxiliary Features visualization. To generate the colors, during training, we fit an embedding of the oracle features $\mathbf{O}$ in $[0,1]^3$ (RGB space), the same mapping is used to color $\mathbf{R}_i$. The cosine similarity is computed pointwise.

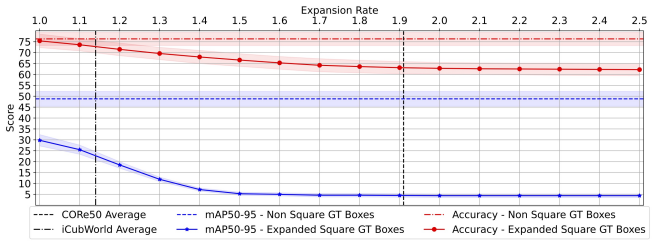


Fig. 4: Personalization performance varying the ground truth boxes padding size on PerSeg.

TABLE IV: Effect of different distances in the FSL accuracy. **Best** in bold, second best underlined.

cos	ℓ_1	ℓ_2	mAP (↑) Coarse ✗	mAP (↑) Coarse ✓
✓	✗	✗	48.3±3.3	**34.6**±3.2
✗	✓	✗	48.6±3.2	33.8±3.1
✗	✗	✓	**48.9**±3.3	34.2±3.2
✓	✓	✗	48.4±3.3	**34.6**±3.2
✓	✓	✓	48.8±3.4	34.5±3.2

VI. Ablation

In this section we analyze in more detail some of the design choices made for the proposed architecture: i) in Table IV we focus on the distance functions involved in the FSL, showing how their combined use leads to a more stable performance; ii) in Table V we show the effect of changing the FSL, comparing our conditional majority voting strategy to other protonet-based strategies; and iii) in Table VI we show the performance of the FSL when different residual maps $\mathbf{R}_i$ are used (as opposed to $\mathbf{R}_1$). For this study, in Fig. 3 we also provide some qualitative results visually showing the improvement in reconstruction. We show a colorized version of the oracle ($\mathbf{O}$) and auxiliary ($\mathbf{R}_i$) feature spaces, together with the original input image (augmented according to the training policy) and the pointwise cosine similarity between $\mathbf{R}_1$ and $\mathbf{O}$. The color is generated according to a non-linear embedding which is fit on the oracle features at training time. The channel-dense features into $[0,1]^3$ (RGB space) are mapped using a 1x1 convolution with sigmoid activation function, the embedding is then expanded to the original space with another 1x1 convolution and a reconstruction loss between the original and reconstructed features is minimized.

For the analysis in Table IV we consider three distance functions: cosine, ℓ_1 and ℓ_2. The table shows how different distance functions are better depending on whether the coarse classes are considered or not (cosine is best when they are enabled, ℓ_2 otherwise). For this reason, we employ all three distances together, achieving a good compromise (second-best performance in both cases).

TABLE V: Comparisons between different Few Shot Learners.

Model	mAP (↑)	Time [ms/im] (↓)
Protonet-cos [2]	50.8±3.2	144.8±6.7
Protonet-ℓ_2 [2]	**51.0**±3.5	96.1±5.9
SimpleShot [7]	50.8±3.2	94.1±5.8
Ours	48.8±3.4	**32.6**±5.1

TABLE VI: Results on residual chain shortcuts.

Features	mAP (↑)
Partial I ($\mathbf{R}_3$)	48.4±3.5
Partial II ($\mathbf{R}_2$)	48.7±3.4
Full ($\mathbf{R}_1$)	**48.8**±3.4

We compare the proposed FSL strategy with state-of-the-art competitors in Table V, measuring both personalization accuracy and time complexity. The table shows how our conditional approach leads to a slight decrease in performance (2.2 mAP compared to the best strategy), but improves the computation time about 3×. Such a trade-off is fundamental for robotic applications where many vision tasks require real-time performance.

Fig. 4 reports a study on the padding of GT boxes, which shows how sensitive the mAP is to its presence. The results justify the choice of retrieval accuracy made for iCubWorld and CORe50, given that the metric is much more resilient to changes in box size. For further confirmation, we measured the mAP of iCubWorld and obtained 20.9. This is very close to the expected value from the figure (i.e., around 22.5).

The final ablation study we report refers to the residual chain of the auxiliary features: we investigated the performance attained by our method when the partial reconstructions $\mathbf{R}_i$ were used instead of $\mathbf{R}_1$. The results shown in Table VI confirm the improvements brought by the chain: using only $\mathbf{R}_n$ ($n = 3$ in our setup) results in a total of 48.4 mAP which increases to 48.7 when two maps are merged. The best performance is improved again when $\mathbf{R}_1$ is also used, slightly raising to 48.8.

VII. Conclusions

In this work, we presented *AuXFT* a Few-Shot Instance-Level Personalized Object Detection architecture, which translates the features of a CNN-based detector into an auxiliary feature space where cross-architecture knowledge translation is possible. We have designed a framework for personalizing coarse-level detection to instance-level classes using a prototypical-based FSL that receives input vectors generated by our detection-driven feature pooling. We have validated the performance of our architecture on four

different personalized object detection datasets, achieving significant gains in all of them. In the future, we plan to investigate the approach using different detectors and oracle extractors to verify the generalizability of the strategy and its suitability for deployment on resource-constrained robotic devices. We also plan to test new mapping functions in the channel differential block and whether some parametric resampling strategies (such as transposed convolutions) can improve performance when added to the spatial differential module.

REFERENCES

[1] L. Sharma and N. Lohan, "Internet of things with object detection: Challenges, applications, and solutions," in *Handbook of Research on Big Data and the IoT*. IGI Global, 2019, pp. 89–100.

[2] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," *NeurIPS*, 2017.

[3] A. Ayub and A. R. Wagner, "Tell me what this is: Few-shot incremental object learning by a robot," in *IROS*, 2020, pp. 8344–8350.

[4] S. Kim, C. Wang, B. Li, and S. Scherer, "Robotic interestingness via human-informed few-shot object detection," in *IROS*, 2022.

[5] Y. Shukla, B. Kesari, S. Goel, R. Wright, and J. Sinapov, "A framework for few-shot policy transfer through observation mapping and behavior cloning," in *IROS*, 2023, pp. 7104–7110.

[6] K. Paramonov, J.-X. Zhong, U. Michieli, and M. Ozay, "Swiss dino: Efficient and versatile vision framework for on-device personal object search," *IROS*, 2024.

[7] Y. Wang, W.-L. Chao, K. Q. Weinberger, and L. Van Der Maaten, "Simpleshot: Revisiting nearest-neighbor classification for few-shot learning," *ArXiv:1911.04623*, 2019.

[8] I. Ziko, J. Dolz, E. Granger, and I. B. Ayed, "Laplacian regularized few-shot learning," in *ICML*, 2020, pp. 11 660–11 670.

[9] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[10] V. Papyan, X. Han, and D. L. Donoho, "Prevalence of neural collapse during the terminal phase of deep learning training," *PNAS*, 2020.

[11] H. Tiomoko Ali, U. Michieli, and M. Ozay, "Deep neural network models trained with a fixed random classifier transfer better across domains," in *ICASSP*, 2024.

[12] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon, "A survey on contrastive self-supervised learning," *Technologies*, 2020.

[13] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging Properties in Self-Supervised Vision Transformers," *ArXiv:2104.14294*, 2021.

[14] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, *et al.*, "DINOv2: Learning robust visual features without supervision," *ArXiv:2304.07193*, 2023.

[15] Y. Liu, J. Cao, B. Li, W. Hu, J. Ding, and L. Li, "Cross-architecture knowledge distillation," in *ACCV*, 2022, pp. 3396–3411.

[16] W. Zhao, X. Zhu, Z. He, X.-Y. Zhang, and Z. Lei, "Cross architecture distillation for face recognition," *ArXiv:2306.14662*, 2023.

[17] Z. Hao, J. Guo, K. Han, Y. Tang, H. Hu, Y. Wang, and C. Xu, "One-for-All: Bridge the Gap Between Heterogeneous Architectures in Knowledge Distillation," *NeurIPS*, 2024.

[18] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *ArXiv:2304.02643*, 2023.

[19] R. Zhang, Z. Jiang, Z. Guo, S. Yan, J. Pan, H. Dong, Y. Qiao, P. Gao, and H. Li, "Personalize segment anything model with one shot," in *ICLR*, 2024.

[20] U. Michieli and M. Ozay, "Online continual learning for robust indoor object recognition," in *IROS*, 2023, pp. 3849–3856.

[21] G. Pasquale, C. Ciliberto, F. Odone, L. Rosasco, and L. Natale, "Teaching icub to recognize objects using deep convolutional neural networks," in *MLIS*. PMLR, 2015, pp. 21–25.

[22] V. Lomonaco and D. Maltoni, "Core50: a new dataset and benchmark for continuous object recognition," in *ACRL*, 2017, pp. 17–26.

[23] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *ArXiv:1412.6550*, 2014.

[24] L. Zhang, Y. Shi, Z. Shi, K. Ma, and C. Bao, "Task-oriented feature distillation," *NeurIPS*, pp. 14 759–14 771, 2020.

[25] N. Passalis, M. Tzelepi, and A. Tefas, "Heterogeneous knowledge distillation using information flow modeling," in *CVPR*, 2020, pp. 2339–2348.

[26] S. Luo, X. Wang, G. Fang, Y. Hu, D. Tao, and M. Song, "Knowledge amalgamation from heterogeneous networks by common feature learning," *ArXiv:1906.10546*, 2019.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, 2017.

[28] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *ArXiv:2010.11929*, 2020.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *ArXiv:1810.04805*, 2019.

[30] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *ICCV*, 2021, pp. 10 012–10 022.

[31] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions," *ArXiv:2102.12122*, 2021.

[32] F. Barbato, G. Rizzoli, and P. Zanuttigh, "Depthformer: Multimodal positional encodings and cross-input attention for transformer-based segmentation networks," in *ICASSP*, 2023, pp. 1–5.

[33] G. Rizzoli, F. Barbato, and P. Zanuttigh, "Multimodal semantic segmentation in autonomous driving: A review of current approaches and future perspectives," *Technologies*, vol. 10, no. 4, p. 90, 2022.

[34] E. G. Miller, N. E. Matsakis, and P. A. Viola, "Learning from one example through shared densities on transforms," in *CVPR*, 2000, pp. 464–471.

[35] K. Wang, J. H. Liew, Y. Zou, D. Zhou, and J. Feng, "Panet: Few-shot image semantic segmentation with prototype alignment," in *ICCV*, 2019, pp. 9197–9206.

[36] S. Mirkin, S. Nowson, C. Brun, and J. Perez, "Motivating personality-aware machine translation," in *Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1102–1108.

[37] Y. Zhang, C.-B. Zhang, P.-T. Jiang, M.-M. Cheng, and F. Mao, "Personalized image semantic segmentation," in *ICCV*, 2021, pp. 10 549–10 559.

[38] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," *ArXiv:2103.00020*, 2021.

[39] Y. Liu, M. Zhu, H. Li, H. Chen, X. Wang, and C. Shen, "Matcher: Segment anything with one shot using all-purpose feature matching," *ArXiv:2305.13310*, 2024.

[40] X. Wang, X. Zhang, Y. Cao, W. Wang, C. Shen, and T. Huang, "Seggpt: Segmenting everything in context," *ArXiv:2304.03284*, 2023.

[41] D. Shenaj, F. Barbato, U. Michieli, and P. Zanuttigh, "Continual coarse-to-fine domain adaptation in semantic segmentation," *Image and Vision Computing*, vol. 121, p. 104426, 2022.

[42] F. Barbato, E. Camuffo, S. Milani, and P. Zanuttigh, "Continual road-scene semantic segmentation via feature-aligned symmetric multi-modal network," *ArXiv:2308.04702*, 2023.

[43] H. Fu, M. K. Ng, M. Nikolova, and J. L. Barlow, "Efficient minimization methods of mixed l2-l1 and l1-l1 norms for image restoration," *SIAM Journal on Scientific computing*, pp. 1881–1902, 2006.

[44] S. K. Mitra, *Digital signal processing: a computer-based approach*. McGraw-Hill Higher Education, 2001.

[45] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *IJCV*, 2020.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ArXiv:1412.6980*, 2017.

[47] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," *ArXiv:1405.0312*, 2015.