Supplementary materials

The following supplementary materials provide further details on training, on the results of the different benchmarks, and more qualitative analysis and visualizations.

6. Experiments setup

6.1. Training details and hyperparameters

We trained each model using a standard stochastic gradient descent (SGD) optimizer with momentum parameter 0.937 and weight decay $5e^{-4}$. We used warm-up and cosine decay rule for training. For the NMS parameters, we used an IoU threshold of 0.65 and an object confidence threshold of 0.001. When generating pseudolabels, we used a higher confidence threshold of 0.4. We used the model definitions, as defined by the initial release of YOLOv5 [20] with last commit id '364fcfd7d'. Finally, we used a generalized IoU loss (GIoU) for localization and a focal loss for the classification loss and objectness loss for training the models.

To manage our experiments and make our results reproducible, we used the open-source tool Guildai [29]. Most hyper-parameters (Momentum, NMS, etc.) were set as defaults in YOLOv5 repo [20]. We tuned only the learning rate for each dataset. The value of hyperparameters are configured in the 'guild.yml' file. For the gradual adaptation procedure, we use a large enough number of epochs for Phase 1 to ensure the convergence of BN adaptation. We use a separate validation set to maintain the best checkpoint using the validation AP. Therefore, we initialize the phase 2 training with the best checkpoint of phase 1. It is also worth noting that our framework does not add new hyperparameters.

When training the COCO source models or the Stylize and DeepAugment baselines, we followed the training procedure in YOLOv5 [20] and trained the model from scratch using 300 epochs and a learning rate of 0.01. For Pascal and Cityscapes the source models were obtained through transfer learning from COCO pretrained weights using 100 and 200 epochs respectively. For that, we used learning rate of $4e^{-5}$ and batch size of 128. When applying our adaptation method, we also fine-tuned the source model using same learning rate of $4e^{-5}$, a batch size of 128 and 100 epochs for all models and target domains.

We did not use multi-scale training to simplify our analysis. The same image input size was used during training, pseudo-label generation and evaluation. For Sim10K/KITTI to Cityscapes, we specify the input size used to train each student and teacher model in our results. For the artistic benchmark, we use the same input size of 416 for both student and teacher models. For the image corruption benchmark, we used the same input size of 416 for Pascal-C and COCO-C whereas we used a larger size of 640 for Cityscapes-C.

For the Stylize baseline, we applied only one style for each image to keep the dataset size the same, to ensure a fair comparison. We preserved the original image dimensions and disabled the cropping. Alpha was fixed to 1 to apply the highest strength of stylization.

6.2. More details on datasets

Table 8, 9, 10, and 11 show a summary of the data splits that we used as the source or clean split versus target or stylized/augmented split for each dataset. To make a fair comparison, we keep the total number of images in the entire training data to be the same for all methods.

For Pascal-C, COCO-C, and Cityscapes-C, we generated the corrupted test set by applying each corruption to the clean test with all five severity levels. For the cross-domain adaptation benchmark, we used the test split for Clipart, Watercolor, or Comic for measuring test AP on the target domain.

- **Sim10K**: we use the SIM10k dataset as the labeled source training data and the training set of Cityscapes as unlabeled target data. The validation set of Cityscapes was used as target test set.

- **KITTI**: we use the training set of KITTI as the labeled source data and the training set of Cityscapes as unlabeled target data. The validation set of Cityscapes was used as target test set.

- **Clipart/Watercolor/Comic**: the datasets used in Source, DeepAugment, Stylize baselines for Clipart/Watercolor/Comic are exactly the same as those used for Pascal-C. Other than this, the train set of Clipart/Watercolor/Comic were used as the target domain dataset. In DT+PL experiments, we first apply the domain transfer on the union of VOC2007 trainval and VOC2012 trainval. Then, we apply the DT step on the source model using the domain-transferred dataset. Finally, we apply the PL step on the output model of DT step using the train split of of Clipart/Watercolor/Comic. Note that we do not use the ground-truth labels but use the pseudo-labels instead.

- **Pascal-C**: we used VOC2007 trainval as the source and VOC2012 trainval as the target. For the DeepAugment baseline, we augmented VOC2012 train with the CAE method and VOC2012 val with the EDSR method. We used VOC2007 test as the clean test set.

- **COCO-C**: we split COCO train2017 into two approximately equal halves and used the first half as source, the second half as target. For DeepAugment, we divided COCO train2017 in three random splits and used them for the clean split, CAE split and EDSR split respectively. COCO val2017 was used as clean test.

Method	Source / Clean split (size)	Target / Augmented split (size)
Source	VOC2007-trainval (5011)	N/A
DeepAugment	VOC2007-trainval (5011)	CAE VOC2012-train (5717) + EDSR VOC2012-val (5823)
Stylize	VOC2007-trainval (5011)	stylized VOC2012-trainval (11540)
BN-Adapt	VOC2007-trainval (5011)	VOC2012-trainval (11540)
STAC	VOC2007-trainval (5011)	VOC2012-trainval (11540)
SimROD (Ours)	VOC2007-trainval (5011)	VOC2012-trainval (11540)

Table 8. Dataset splits used for Pascal-C

Method	Source / Clean split (size)	Target / Augmented split (size)
Source	coco-train2017/first half (58458)	N/A
DeepAugment	coco-train2017/first 1/3 (39088)	CAE second 1/3 (39088) + EDSR third 1/3 (39090)
Stylize	coco-train2017/first half (58458)	stylized coco-train2017/second half (58808)
BN-Adapt	coco-train2017/first half (58808)	coco-train2017/second half (58808)
STAC	coco-train2017/first half (58458)	coco-train2017/second half (58808)
SimROD (Ours)	coco-train2017/first half (58458)	coco-train2017/second half (58808)

Table 9. Dataset splits used for COCO-C

- **Cityscapes-C**: we split the source domain and target domain by city names. We carefully chose the cities for each domain so that source and target are of approximately equal size. Of all 18 cities in cityscapes-train, 9 cities: 'cologne', 'krefeld', 'bremen', 'darmstadt', 'hanover', 'aachen', 'stuttgart', 'jena', and 'tubingen' were used as source data; the other 9 cities: 'bochum', 'ulm', 'monchengladbach', 'weimar', 'strasbourg', 'zurich', 'hamburg', 'dusseldorf', and 'erfurt' were used as target data. When training the DeepAugment baseline for Cityscapes, we further split the target domain into two splits. The first split that contains 'zurich', 'weimar', 'erfurt', and 'strasbourg' was augmented with the CAE method. The second split which contains 'bochum', 'ulm', 'monchengladbach', 'hamburg', 'dusseldorf' was augmented with the EDSR method. The validation set of Cityscapes was used as clean test.

7. More results on synthetic-to-real and cross-camera benchmarks

7.1. Full results on Sim10K/KITTI to Cityscapes

Table 12 and 13 expand on the results reported in Table 1 and 2 respectively. In particular, they show the performance of the teacher models and that of models adapted with the smaller teacher model X640.

7.2. Qualitative visualization

In Figure 5, we present qualitative results for the detection of the model S416 (i.e. yolov5s with input 416) to demonstrate the improvement brought by SimROD compared to the source model. By comparing with ground-truth labels, Figure 5 shows that the adapted model can detect most objects with good accuracy except for some highly occluded ones.

8. More results on artistic benchmark

8.1. Benchmark results on Clipart and Comic

We include the benchmarks results for Clipart and Comic in Table 14 and 15 respectively. We used only 500 unlabeled images from the target domain for Clipart and 1000 images for Comic. Similar to the results for Watercolor in Table 3, our method SimROD outperformed the baselines when compared with models that achieve same Source AP performance. Compared to DT+PL in [18], our method further improved the AP50 of the S416 model by absolute 8.35, 12 and 10.69 percentage points on Clipart, Comic and Watercolor respectively. In addition, SimROD consistently achieves high effective adaptation gains ρ between 70-97% across model sizes and benchmarks.

8.2. Data efficiency analysis on Watercolor and Comic

Next, we analyze the data efficiency of SimROD by increasing the size of unlabeled data used to adapt the models. For Watercolor and Comic, we used the extra splits, which contains extra 52.8K and 17.8K additional unlabeled images respectively. Moreover, all models use the same input size of 416. Figure 6 and 7 compare the performance of SimROD with the two pseudo-labeling baselines (STAC and DT+PL) on Watercolor and Comic respectively. All methods improved when using more unlabeled data from the target domain. For example, SimROD improves the Yolov5s model performance by absolute +3.23% and +4.69% on Watercolor and Comic respectively.

Nonetheless, SimROD could outperform baseline methods without using extra data for Yolov5s and Yolov5m models, which are adapted using the self-adapted teacher Yolov5x. In other words, our proposed method used only 1000 unlabeled images and still outperformed the baselines, which used 50$\times$ or 18$\times$ more data. For example, our

Method	Source / Clean split (size)	Target / Augmented split (size)
Source	cityscapes-train/first half (1483)	N/A
DeepAugment	cityscapes-train/first half (1483)	CAE train/second half-split 1 (732) + EDSR train/second half-split 2 (750)
Stylize	cityscapes-train/first half (1483)	stylized cityscapes-train/second half (1482)
Bn_only	cityscapes-train/first half (1483)	cityscapes-train/second half (1482)
Stac	cityscapes-train/first half (1483)	cityscapes-train/second half (1482)
Ours w/o TG	cityscapes-train/first half (1483)	cityscapes-train/second half (1482)
Ours	cityscapes-train/first half (1483)	cityscapes-train/second half (1482)

Table 10. Dataset splits used for Cityscapes-C

Method	Source / Clean split (size)	Target / Augmented split (size)
Source	VOC2007-trainval (5011)	N/A
DeepAugment	VOC2007-trainval (5011)	CAE VOC2012-train (5717) + EDSR VOC2012-val (5823)
Stylize	VOC2007-trainval (5011)	stylized VOC2012-trainval (11540)
Bn_only	VOC2007-trainval (5011)	clipart/watercolor/comic-train (500/1000/1000)
Stac	VOC2007-trainval (5011)	clipart/watercolor/comic-train (500/1000/1000)
Ours w/o TG	VOC2007-trainval (5011)	clipart/watercolor/comic-train (500/1000/1000)
Ours	VOC2007-trainval (5011)	clipart/watercolor/comic-train (500/1000/1000)

Table 11. Dataset splits used for Clipart/Watercolor/Comic

method achieved an AP50 of 42.34% on yolov5s whereas the best baseline on yolov5m has an AP50 of only 37.79%.

8.3. Qualitative comparison on Clipart, Comic and Watercolor

In Figures 8 and 9, we provide qualitative comparisons with pseudo-labeling baselines (STAC [30] and DT+PL [18]) and DeepAugment method using same Yolov5s model. These comparisons illustrates the simplicity and effectiveness of SimROD. Our proposed DomainMix augmentation and teacher-guided gradual adaptation enabled to leverage unlabeled target data and to mitigate the label noise and domain shift. In contrast to DT+PL, SimROD did not need to generate synthetic intermediate dataset and our proposed augmentation is much simpler than DeepAugment.

Method	Arch.	Backbone	Source	AP50	Oracle	τ	ρ	Reference
DAF [6]	F-RCNN	V	30.10	39.00	-	8.90	-	CVPR 2018
MAF [11]	F-RCNN	V	30.10	41.10	-	11.00	-	ICCV 2019
RLDA [22]	F-RCNN	I	**31.08**	**42.56**	68.10	**11.48**	**31.01**	ICCV 2019
SCDA [38]	F-RCNN	V	34.00	43.00	-	9.00	-	CVPR 2019
MDA [36]	F-RCNN	V	34.30	42.80	-	8.50	-	ICCV 2019
SWDA [27]	F-RCNN	V	34.60	42.30	-	7.70	-	CVPR 2019
Coarse-to-Fine [37]	F-RCNN	V	**35.00**	43.80	59.90	8.80	35.34	CVPR 2020
SimROD (self-adapt)	YOLOv5	S320	33.62	38.73	48.81	5.11	33.66	Ours
SimROD (w. teacher X640)	YOLOv5	S320	33.62	**44.70**	48.81	**11.08**	**72.93**	Ours
MTOR [4]	F-RCNN	R	39.40	46.60	-	7.20	-	CVPR 2019
EveryPixelMatters [16]	FCOS	V	**39.80**	49.00	69.70	9.20	30.77	ECCV 2020
SimROD (self adapt)	YOLOv5	S416	39.57	44.21	56.49	4.63	27.37	Ours
SimROD (w. teacher X640)	YOLOv5	S416	39.57	51.68	56.49	12.10	71.53	Ours
SimROD (w. teacher X1280)	YOLOv5	S416	39.57	**52.05**	56.49	**12.47**	**73.73**	Ours
SimROD (self-adapt)	YOLOv5	M640	55.86	60.29	71.05	4.43	29.16	Ours
SimROD (w. teacher X640)	YOLOv5	M640	55.86	62.18	71.05	6.33	41.64	Ours
SimROD (w. teacher X1280)	YOLOv5	M640	55.86	64.40	71.05	8.54	56.24	Ours
SimROD (self-adapt)	YOLOv5	X640	60.34	63.27	72.51	2.93	24.09	Ours
SimROD (self-adapt)	YOLOv5	X1280	71.66	75.94	82.90	4.28	38.08	Ours

Table 12. Results of different method/model pairs for the Sim10K-to-Cityscapes adaptation scenario. "V", "I" and "R" represent the VGG16, ResNet50, Inception-v2 backbones respectively. "S320", "M416", "X640", "X1280" represent different scales of Yolov5 model with increasing depth, width and input size. "Source" denotes that the model is trained only using source images without domain adaptation. For fair comparison, we group together method/model pairs whose "Source" performance are similar. We report the AP50 (%) performance of the adapted model and the "Oracle" model which is trained with labeled target data as well each method's absolute and effective gains (%) when available. τ and ρ are the absolute gain and the effective gain respectively as defined in (1) and (2).

Method	Arch.	Backbone	Source	AP50	Oracle	τ	ρ	Reference
DAF [6]	F-RCNN	V	30.20	38.50	-	8.30	-	CVPR 2018
MAF [11]	F-RCNN	V	30.20	41.00	-	10.80	-	ICCV 2019
RLDA [22]	F-RCNN	I	31.10	42.98	68.10	11.88	32.11	ICCV 2019
PDA [17]	F-RCNN	V	30.20	43.90	55.80	13.70	53.52	WACV 2020
SimROD (self-adapt)	YOLOv5	S416	31.61	35.94	56.15	4.33	17.65	Ours
SimROD (w. teacher X640)	YOLOv5	S416	31.61	43.55	56.15	11.94	48.66	Ours
SimROD (w. teacher X1280)	YOLOv5	S416	**31.61**	**45.66**	56.15	**14.05**	**57.27**	Ours
SCDA [38]	F-RCNN	V	**37.40**	42.60	-	5.20	-	CVPR 2019
EveryPixelMatters [16]	FCOS	R	35.30	45.00	70.40	9.70	27.64	ECCV 2020
SimROD (self adapt)	YOLOv5	M416	36.09	42.94	59.29	6.85	29.51	Ours
SimROD (w. teacher X640)	YOLOv5	M416	36.09	45.29	59.29	9.19	39.64	Ours
SimROD (w. teacher X1280)	YOLOv5	M416	36.09	**47.52**	59.29	**11.43**	**49.26**	Ours
SimROD (self-adapt)	YOLOv5	X640	45.67	50.81	72.18	5.14	19.38	Ours
SimROD (self-adapt)	YOLOv5	X1280	52.07	58.25	82.50	6.18	20.31	Ours

Table 13. Results of different method/model pairs on the KITTI to Cityscapes adaptation scenario. τ and ρ are the absolute gain and the effective gain respectively as defined in (1) and (2).

Method	Arch.	Backbone	Source	AP50	Oracle	τ	ρ	Reference
ADDA [34]	SSD	V	26.80	27.40	55.40	0.60	2.10	CVPR 2017
DT+PL [18]	SSD	V	**26.80**	**46.00**	55.40	**19.20**	**67.13**	CVPR 2018
DAF [6]	F-RCNN	V	26.20	22.40	50.00	-3.80	-15.97	CVPR 2018
DT+PL [18]	F-RCNN	V	26.20	34.90	50.00	8.70	36.55	CVPR 2018
SWDA [27]	F-RCNN	V	**27.80**	38.10	50.00	10.30	46.40	CVPR 2019
DAM [23]	F-RCNN	V	24.90	41.80	50.00	**16.90**	**67.33**	CVPR 2018
DeepAugment [12]	YOLOv5	S416	29.32	31.65	56.07	2.33	8.71	arXiv 2020
BN-Adapt [19]	YOLOv5	S416	29.32	37.43	56.07	8.11	30.32	NeurIPS 2020
Stylize [10]	YOLOv5	S416	29.32	38.80	56.07	9.48	35.44	arXiv 2019
STAC [30]	YOLOv5	S416	29.32	39.64	56.07	10.32	38.58	arXiv 2020
DT+PL [18]	YOLOv5	S416	29.32	39.49	56.07	10.17	38.02	CVPR 2018
SimROD (self-adapt)	YOLOv5	S416	29.32	41.28	56.07	11.96	44.72	Ours
SimROD (teacher X416)	YOLOv5	S416	**29.32**	**47.84**	56.07	**18.52**	**69.24**	Ours

Table 14. Benchmark results on Real (VOC) to Clipart1k domain shift

Method	Arch.	Backbone	Source	AP50	Oracle	τ	ρ	Reference
ADDA	SSD	V	24.90	23.80	46.40	-1.10	-5.12	CVPR 2017
DT	SSD	V	24.90	29.80	46.40	4.90	22.79	CVPR 2018
DT+PL	SSD	V	**24.90**	**37.20**	46.40	**12.30**	**57.21**	CVPR 2018
DAF	F-RCNN	V	21.40	23.20	-	1.80	-	CVPR 2018
DT	F-RCNN	V	21.40	29.80	-	8.40	-	CVPR 2018
SWDA	F-RCNN	V	21.40	28.40	-	7.00	-	CVPR 2019
DAM	F-RCNN	V	**21.40**	**34.50**	-	**13.10**	-	CVPR 2019
DeepAugment	YOLOv5	S416	18.19	21.39	39.81	3.20	14.80	arXiv 2020
BN-Adapt	YOLOv5	S416	18.19	25.53	39.81	7.34	33.95	NeurIPS 2020
Stylize	YOLOv5	S416	18.19	27.57	39.81	9.38	43.39	arXiv 2019
STAC	YOLOv5	S416	18.19	26.40	39.81	8.21	37.97	arXiv 2020
DT+PL	YOLOv5	S416	18.19	25.66	39.81	7.47	34.55	CVPR 2018
SimROD (self-adapt)	YOLOv5	S416	18.19	29.54	39.81	11.35	52.50	Ours
SimROD (teacher X416)	YOLOv5	S416	**18.19**	**37.65**	39.81	**19.46**	**90.01**	Ours
DeepAugment	YOLOv5	M416	23.58	27.65	49.13	4.07	15.93	arXiv 2020
BN-Adapt	YOLOv5	M416	23.58	32.04	49.13	8.46	33.11	NeurIPS 2020
Stylize	YOLOv5	M416	23.58	34.56	49.13	10.98	42.97	arXiv 2019
STAC	YOLOv5	M416	23.58	32.76	49.13	9.18	35.93	arXiv 2020
DT+PL	YOLOv5	M416	23.58	33.53	49.13	9.95	38.94	CVPR 2018
SimROD (self-adapt)	YOLOv5	M416	23.58	37.93	49.13	14.35	56.15	Ours
SimROD (teacher X416)	YOLOv5	M416	**23.58**	**42.08**	49.13	**18.50**	**72.41**	Ours

Table 15. Benchmark results on Real (VOC) to Comic domain shift

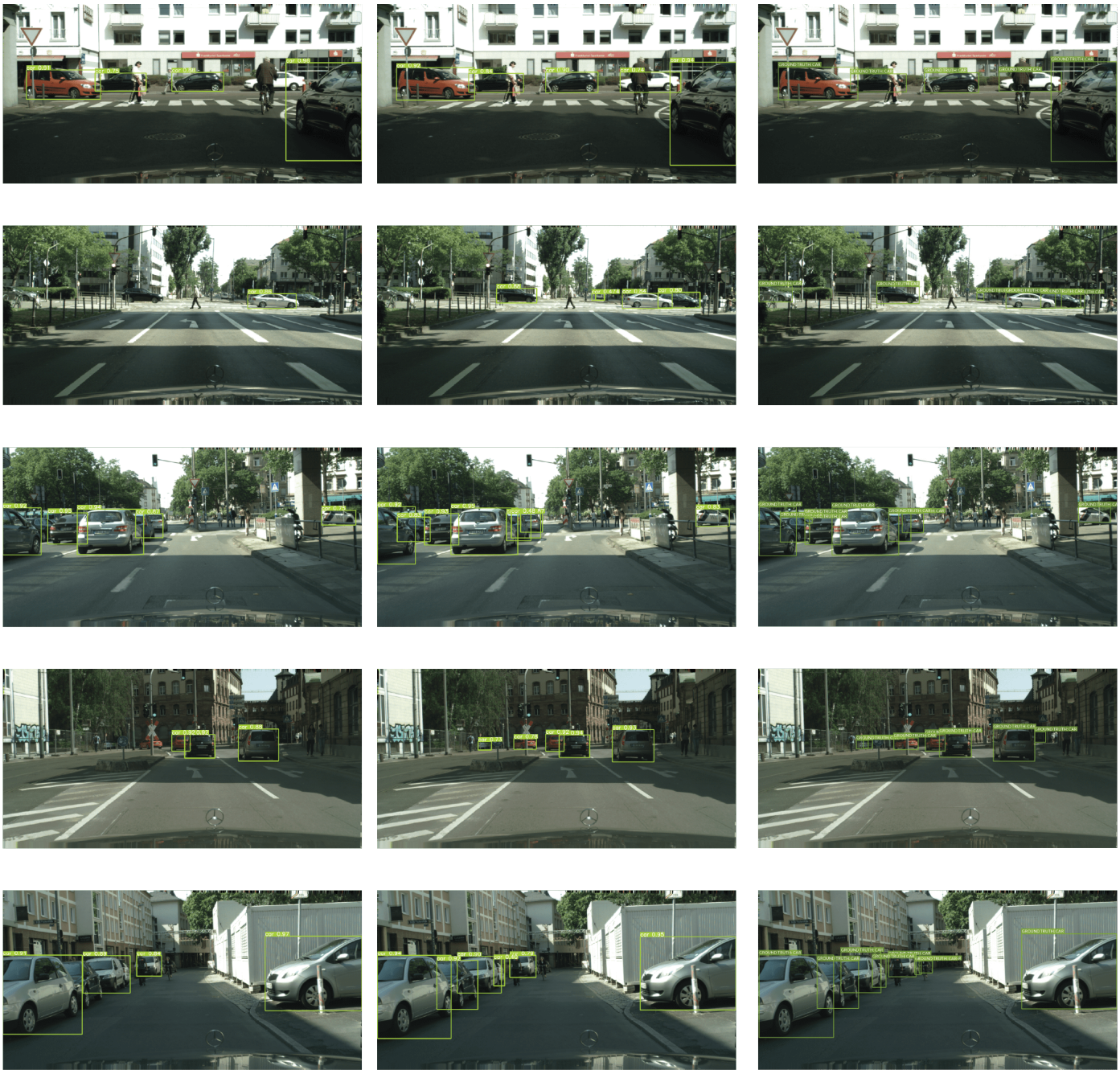

Figure 5. Examples of prediction results on Sim10K to Cityscapes. We show predictions on the target test set before and after applying SimROD as well as the ground-truth labels.

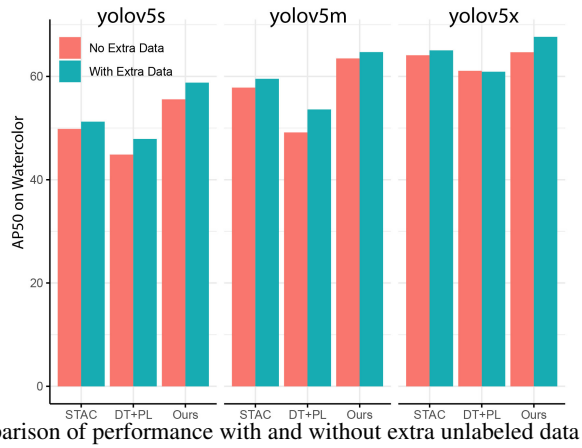

Figure 6. Comparison of performance with and without extra unlabeled data on Watercolor.

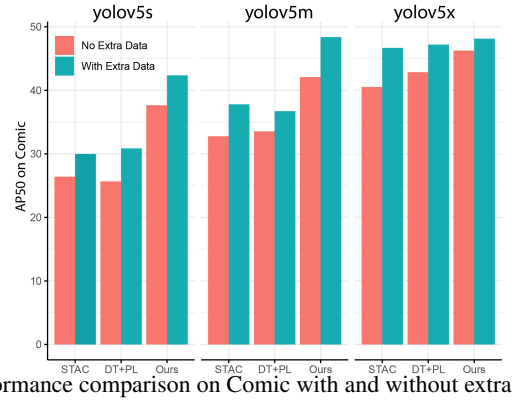

Figure 7. Performance comparison on Comic with and without extra unlabeled data.

9. More results on image corruptions

9.1. Results for different model sizes

Table 4, 5, and 6 show only the results for Yolov5m model for Pascal-C, COCO-C, and Cityscapes-C respectively. In Table 16, 17, and 18, we show that SimROD consistently achieves higher performance compared to the baselines across different model sizes and benchmarks. As expected, larger models provided extra capacity and thus higher Performance.

9.2. Per-corruption performance on Pascal-C

In the main paper, we reported the mAP, rPC, and τ_c metrics, which were averaged over 15 corruption types. Here, in Tables 19, 20, and 21, we provide a breakdown of the results for each corruption type on the Pascal-C dataset for the three YOLOv5 models.

9.3. Performance comparison with Augmix

Here, we compare our proposed method with Augmix augmentation [14] and report the results on Pascal-C in Table 22 and 23 for the models YOLOv5s and YOLOv5x respectively. When comparing the mean performance under corruption (mPC), we can see that Augmix performed the worst among all augmentation-based baselines. Interestingly, applying Augmix augmentation with DeepAugment improved the performance of DeepAugment by +3.3% AP50 and +1.03% AP50 on YOLOv5s and YOLOv5x models respectively. Nonetheless, SimROD still outperformed DeepAument+Augmix by more than +5% AP50 on both models. Although we have not tried, it is possible that applying Augmix on top of DomainMix may further improve the performance of our proposed method.

9.4. Data efficiency analysis on Pascal-C

In Figure 10 and 11, we analyzed the data efficiency of our proposed method using a YOLOv5s model and Pascal-C dataset. For that, we used a subset of training datasets and considered two scenarios. For both scenarios, we randomly generated three different sets of data, measured the performance in three runs. The average of the three runs are plotted with error bars in Figure 10 and 11.

In the first scenario, we used all the available labeled data from source domain consisting of 5011 images. On the other hand, we used only a portion of the unlabeled images available. As shown in Figure 10, our proposed method outperformed STAC by a margin of 10% AP50. Moreover, our

Figure 8. Comparing various methods on examples from the Comic dataset.

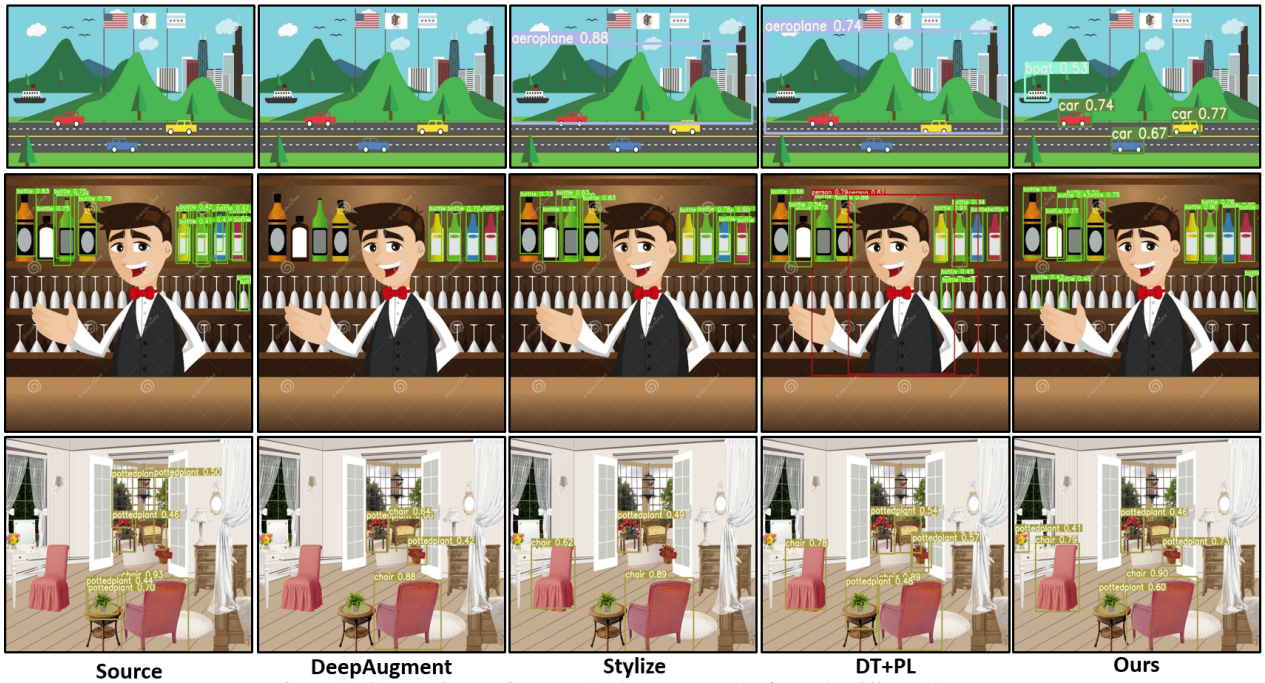

Figure 9. Comparing various methods on examples from the Clipart dataset.

method achieved a relative robustness τ_c of +21.75% AP50 and +16.61% AP50 using only 10% and 1% of unlabeled target domain images respectively. Since the data was imbalanced in this scenario, we also considered applying the

Method	AP^{50}_{clean}	mPC^{50}	rPC	τ_c
yolov5s				
Source	75.87	42.38	55.86	0.00
Stylize	77.26	52.12	67.46	9.74
BN-Adapt	74.71	53.75	71.94	11.37
DeepAugment	77.89	55.42	71.15	13.04
STAC	**80.11**	56.12	70.05	13.74
SimROD (Ours)	80.08	**67.95**	**84.85**	**25.57**
Supervised training	80.44	71.18	88.49	28.80
yolov5m				
Source	83.13	53.78	64.69	0.00
Stylize	84.79	62.92	74.21	9.14
BN-Adapt	83.01	64.60	77.82	10.82
DeepAugment	85.05	64.88	76.28	11.10
STAC	**87.00**	66.88	76.88	13.11
SimROD (Ours)	86.97	**75.40**	**86.70**	**21.63**
Supervised training	86.75	78.74	90.76	24.96
yolov5x				
Source	87.42	62.84	71.88	0.00
Stylize	87.29	69.60	79.73	6.76
BN-Adapt	86.59	71.59	82.68	8.75
DeepAugment	87.78	72.15	82.19	9.31
STAC	**89.57**	73.68	82.25	10.84
SimROD (Ours)	89.24	**78.48**	**87.95**	**15.64**
Supervised training	88.88	82.56	92.89	19.72

Table 16. Performance comparison on Pascal-C benchmark

Method	AP_{clean}	mPC	rPC	τ_c
yolov5s				
Source	**31.35**	17.68	56.40	0.00
Stylize	30.07	18.99	63.15	1.31
BN-Adapt	30.91	20.09	64.99	2.40
DeepAugment	30.37	19.87	65.44	2.19
STAC	31.25	20.00	64.02	2.32
SimROD (Ours)	31.21	**23.94**	**76.71**	**6.26**
Supervised training	30.90	25.33	81.99	7.65
yolov5m				
Source	**36.85**	22.03	59.79	0.00
Stylize	35.75	23.82	66.63	1.79
BN-Adapt	36.24	24.79	68.39	2.76
DeepAugment	35.51	24.33	68.52	2.30
STAC	36.76	24.80	67.46	2.77
SimROD (Ours)	36.79	**28.46**	**77.36**	**6.43**
Supervised training	36.23	30.16	83.26	8.13
yolov5x				
Source	41.61	26.60	63.93	0.00
Stylize	40.38	28.16	69.73	1.56
BN-Adapt	41.70	29.77	71.40	3.17
DeepAugment	41.12	29.13	70.84	2.53
STAC	**41.85**	29.69	70.93	3.09
SimROD (Ours)	41.63	**31.87**	**76.57**	**5.27**
Supervised training	41.06	34.84	84.86	8.24

Table 17. Performance benchmark on COCO-C dataset

weighted balanced sampling to STAC. Figure 10 shows that it could slightly improve the performance of STAC when the datasets were very imbalanced.

In the second scenario, we used only a given percentage of the available training data for both the source and target domain. While this scenario assumes the datasets are balanced, the total number of training images is much smaller than in the previous scenario. For example, using 1% of training data corresponds to a total of 165 images. With 1% of training data, STAC could not adapt the model. In contrast, our proposed method provided a relative robustness τ_c of +4.54% AP50 and +18.28% AP50 using only 1% and 10% of training data respectively.

These results confirm that our method was more data-efficient. In particular, our DomainMix augmentation could produce a diverse set of mixed samples even from very few training images from both domains. When more unlabeled data was available, our method could further leverage the unlabeled data and provide strong supervision for adaptation by mitigating the label noise.

Method	AP_{clean}	mPC	rPC	τ_c
yolov5s				
Source	17.08	9.50	55.62	0.00
Stylize	18.96	11.75	61.97	2.25
DeepAugment	17.24	11.39	66.07	1.89
STAC	**20.34**	12.82	63.02	3.32
SimROD (Ours)	19.82	**14.95**	**75.45**	**5.45**
Supervised training	22.30	19.35	86.77	9.85
yolov5m				
Source	19.48	11.53	59.19	0.00
Stylize	21.77	14.62	67.16	3.09
DeepAugment	20.28	14.79	72.93	3.26
STAC	**24.54**	15.39	62.71	3.86
SimROD (Ours)	24.06	**18.01**	**74.86**	**6.48**
Supervised training	26.58	23.50	88.43	11.97
yolov5x				
Source	25.65	16.63	64.83	0.00
Stylize	27.70	19.38	69.96	2.75
DeepAugment	25.12	18.80	**74.84**	2.17
STAC	**29.62**	20.98	70.85	4.35
SimROD (Ours)	29.27	**21.70**	74.15	**5.07**
Supervised training	31.48	27.66	87.87	11.03

Table 18. Performance benchmark on Cityscapes-C dataset

			Noise			Blur				Weather				Digital			
Method	AP_{clean}	mPC	Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG
Source	75.87	42.38	32.71	35.32	28.24	43.02	32.96	39.87	29.05	37.09	43.53	59.66	69.21	42.00	47.04	46.53	49.48
Stylize	77.26	52.12	41.51	44.61	37.82	49.80	48.02	47.37	35.79	49.53	57.37	67.55	74.07	51.69	59.10	56.77	60.84
DeepAugment	77.89	55.42	50.48	53.12	48.67	55.38	49.23	48.87	37.58	49.73	58.19	70.29	74.91	56.88	51.61	63.39	62.99
BN Adapt	74.71	53.75	48.07	51.22	46.00	53.23	44.34	48.60	38.63	50.56	55.80	68.50	73.34	57.18	59.32	52.86	58.55
STAC	**80.11**	56.12	46.85	49.78	44.08	58.41	45.38	51.99	41.68	53.39	59.80	74.01	78.91	59.85	61.76	56.14	59.78
SimROD (Ours)	80.08	**67.95**	**64.91**	**66.11**	**65.28**	**65.12**	**63.03**	**65.54**	**53.99**	**69.19**	**69.27**	**76.85**	**79.14**	**71.38**	**73.52**	**65.54**	**70.34**
Oracle	80.44	71.18	68.28	69.14	68.10	68.18	67.84	69.77	62.19	71.41	71.26	77.49	79.95	73.41	75.90	70.40	74.41

Table 19. Performance comparison per corruption type for YOLOv5s model on Pascal-C benchmark

9.5. Effects of corruption severity levels

To apply our method on the image corruption benchmark, we applied a corruption severity level of 3 for cre-

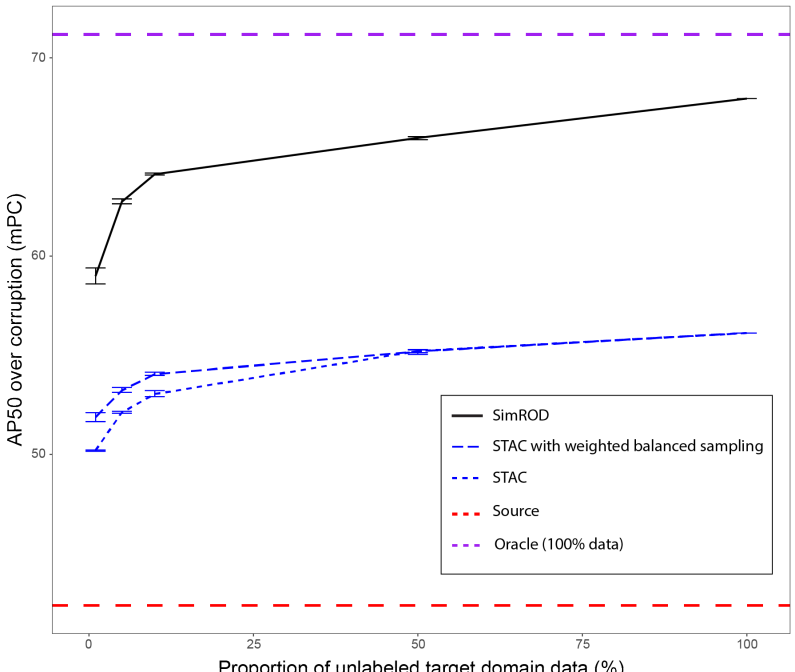

Figure 10. mPC performance of YOLOv5s on Pascal-C for a given percentage of unlabeled target data and using 100% source data.

			Noise			Blur				Weather				Digital			
Method	AP_{clean}	mPC	Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG
Source	83.13	53.78	47.44	51.35	44.98	53.87	42.17	48.61	36.64	51.77	56.29	71.74	78.82	55.81	56.43	54.52	56.17
Stylize	84.79	62.92	53.44	57.56	52.62	60.18	57.42	57.53	45.32	63.02	67.50	78.02	81.91	65.64	69.69	66.10	67.86
DeepAugment	85.05	64.88	61.75	64.06	60.64	63.74	57.95	56.18	44.75	62.31	68.27	79.36	82.69	68.34	61.92	71.40	69.78
BN Adapt	83.01	64.60	61.06	63.83	60.54	62.33	55.29	58.77	46.71	65.44	67.88	78.34	81.62	69.48	68.81	62.15	66.75
STAC	**87.00**	66.88	61.46	64.77	60.73	67.17	55.54	61.35	49.57	68.41	71.20	82.52	85.90	71.83	69.92	65.61	67.25
SimROD (Ours)	86.97	**75.40**	**72.00**	**74.11**	**73.01**	**72.65**	**70.25**	**72.85**	**60.65**	**77.81**	**77.47**	**84.03**	**86.17**	**79.66**	**80.49**	**72.54**	**77.36**
Oracle	86.75	78.74	76.35	76.68	76.42	75.63	75.12	77.10	70.31	80.07	79.56	84.25	86.15	80.60	82.88	78.73	81.22

Table 20. Performance comparison per corruption type for YOLOv5m model on Pascal-C benchmark

			Noise			Blur				Weather				Digital			
Method	AP_{clean}	mPC	Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG
Source	87.42	62.84	59.30	61.06	58.38	61.45	51.43	58.48	41.79	63.82	66.34	77.28	84.25	65.77	64.40	63.07	65.79
Stylize	87.29	69.60	62.44	65.03	62.20	67.57	63.64	65.13	50.84	70.44	74.10	82.44	85.30	74.16	74.73	72.76	73.25
DeepAugment	87.78	72.15	71.25	73.27	71.16	71.40	64.70	65.57	49.76	71.42	74.91	84.17	86.43	77.48	68.75	77.16	74.88
BN Adapt	86.59	71.59	71.05	72.63	70.94	67.90	63.70	66.55	52.41	72.79	73.91	82.38	84.62	76.03	74.96	70.51	73.43
STAC	**89.57**	73.68	71.77	73.40	71.71	72.57	64.51	69.37	52.81	76.21	77.68	85.04	88.40	78.87	75.92	72.69	74.23
SimROD (Ours)	89.24	**78.48**	**76.09**	**78.31**	**77.23**	**75.85**	**73.11**	**75.29**	**62.75**	**81.10**	**80.96**	**86.62**	**88.16**	**82.94**	**82.45**	**76.64**	**79.69**
Oracle	88.88	82.56	81.14	81.96	81.27	79.10	79.08	80.65	73.97	83.58	83.66	87.18	88.54	84.03	85.55	84.09	84.57

Table 21. Performance comparison per corruption type for YOLOv5x model on Pascal-C benchmark

ating the unlabeled target domain images. In this section, we present additional analysis to understand the effects of corruption severity of the training images on the test performance. In Fig. 12 and 13, we show the relative robustness τ and mean performance under corruption mPC of an adapted Yolov5s model using our method. Similarly, Fig. 14 and 15 show the same metrics for an adapted YOLOv5x model.

The corruption types are sorted in ascending order based on the performance of the source model on these types. For instance, the source models achieved the highest mPC on fog and lowest mPC on impulse noise. This explains that the relative robustness on fog was lower compared to those on other corruption types because the source model already achieved high mPC on fog. Notable improvements were observed on the other corruption types.

Fig. 12 and 13 show that the adapted YOLOv5s model enjoyed higher improvement on test datasets with higher severity levels. More importantly, high improvements could be achieved when the training images have severity levels similar to those of the test images. This means that using unlabeled target-domain samples is effective as long as they are representative of the actual test set.

9.6. Qualitative comparison on image corruptions

Fig. 16 illustrates how various methods handle the glass blur corruption (severity 5) on Pascal-C sample. In addition, Fig. 17 shows results of various methods across a range of severity levels for the glass blur corruption. We see that the proposed method was more effective in handling the corruptions. In contrast to the baseline methods, our adaptation method detected most objects in the images and make fewer classification errors. We could also observe that the source model completely failed to detect objects in most cases.

9.7. More detailed ablations on the components

Table 24 expands the ablation study provided in the main paper onto various model sizes.

Method	AP_{clean}	mPC
Source	75.87	42.38
Augmix	79.42	46.94
Stylize	77.26	52.12
DeepAugment	77.89	55.42
DeepAugment+Augmix	**80.85**	60.15
SimROD (Ours)	80.08	**67.95**

Table 22. Augmix comparison for YOLOv5s model on Pascal-C.

Method	AP_{clean}	mPC
Augmix	87.46	62.31
Source	87.42	62.84
Stylize	87.29	69.60
DeepAugment	87.78	72.15
DeepAugment+Augmix	88.36	73.18
SimROD (Ours)	**89.24**	**78.48**

Table 23. Augmix comparison for YOLOv5x model on Pascal-C.

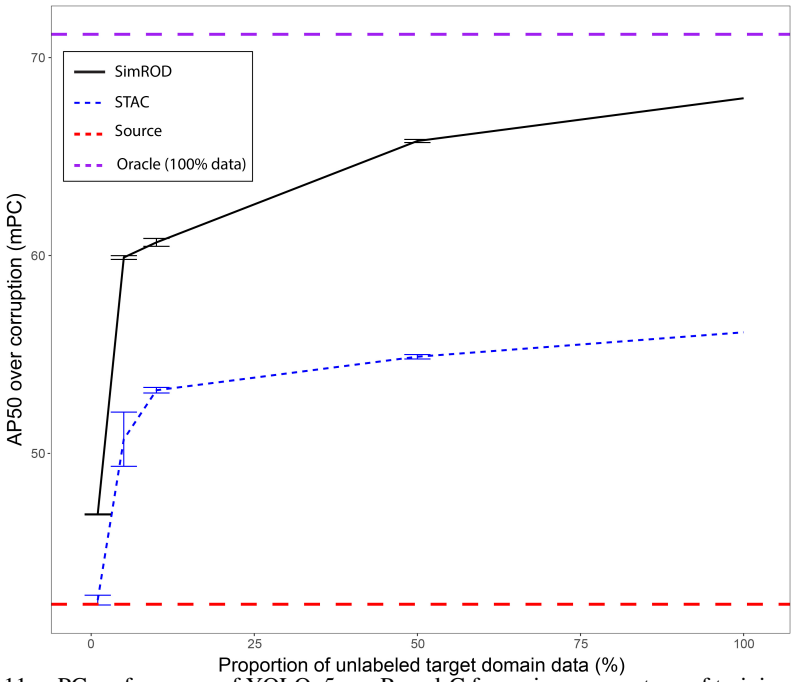

Figure 11. mPC performance of YOLOv5s on Pascal-C for a given percentage of training data (source and target).

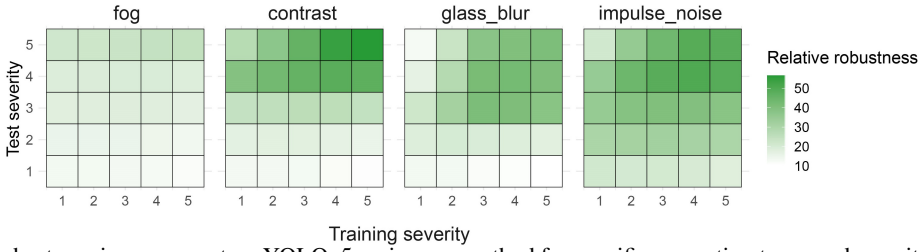

Figure 12. Relative robustness improvement on YOLOv5s using our method for specific corruption types and severity levels on Pascal-C.

Figure 13. Final mPC performance of YOLOv5s using our method for specific corruption types and severity levels on Pascal-C.

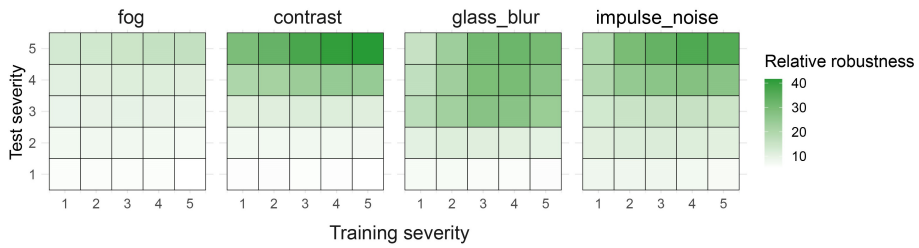

Figure 14. Relative robustness improvement on YOLOv5x using our method for specific corruption types and severity levels on Pascal-C.

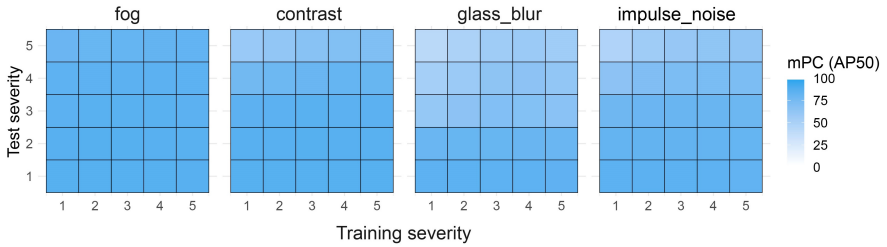

Figure 15. Final mPC performance of YOLOv5x using our method for specific corruption types and severity levels on Pascal-C.

Figure 16. Demonstration of how different methods handle glass_blur corruption (severity 5); images from Pascal-C.

10. Dataset and DomainMix visualizations

Fig. 18 and 19 show examples of the domain-mixed images produced by the DomainMix augmentation from different datasets. Note that the images used to form domain-mixed examples, are randomly cropped, and may occupy a

Model	Method	TG	DomainMix	BN-Adapt	Finetune	Corrupt AP50	τ_c
yolov5s	Source					42.38	0.00
	BN-Adapt			✓		53.75	11.37
	BN-Adapt + DomainMix		✓	✓		56.13	13.75
	SimROD (Ours) w/o Teacher Guidance		✓	✓	✓	60.35	17.97
	SimROD (Ours) w/o Gradual Adaptation	✓	✓		✓	67.87	25.49
	Our full method (SimROD)	✓	✓	✓	✓	67.95	25.57
yolov5m	Source					53.78	0.00
	BN-Adapt			✓		64.60	10.82
	BN-Adapt + DomainMix		✓	✓		66.78	13.01
	SimROD (Ours) w/o Teacher Guidance		✓	✓	✓	71.81	18.03
	SimROD (Ours) w/o Gradual Adaptation	✓	✓		✓	73.45	19.67
	Our full method (SimROD)	✓	✓	✓	✓	75.40	21.62
yolov5x	Source					62.84	0.00
	BN-Adapt			✓		71.83	8.99
	BN-Adapt + DomainMix		✓	✓		73.64	10.80
	SimROD (Ours) w/o Gradual Adaptation	✓	✓		✓	75.58	12.74
	SimROD (Ours) w/o Teacher Guidance		✓	✓	✓	78.16	15.32
	Our full method (SimROD)	✓	✓	✓	✓	78.48	15.64

Table 24. Ablation study on Pascal-C dataset

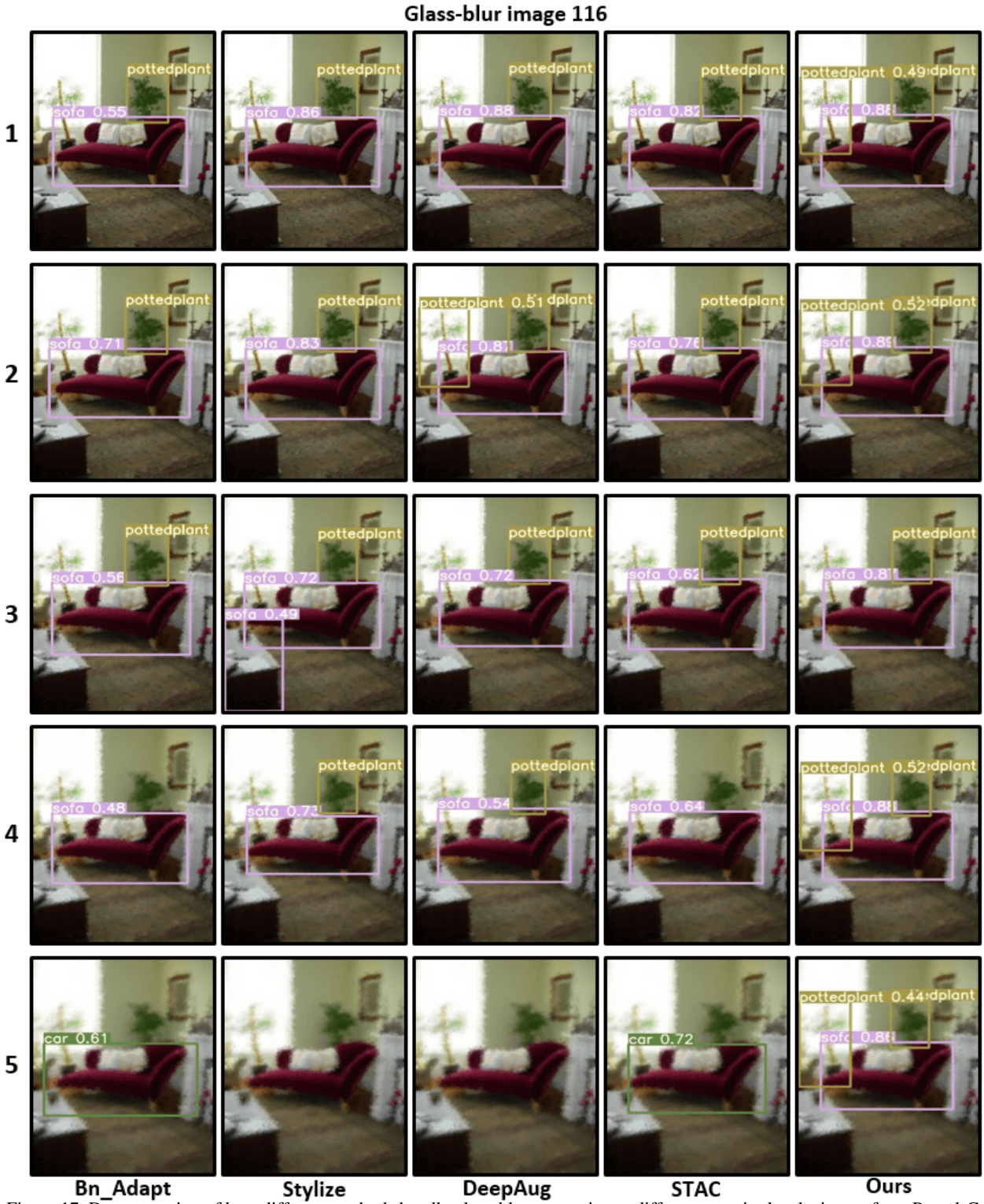

Figure 17. Demonstration of how different methods handle glass_blur corruption at different severity levels; image from Pascal-C.

different height and width of the final image.

Figure 18. Examples of DomainMix image samples on Pascal-C dataset with various corruption types.

Figure 19. Examples of DomainMix image samples on Watercolor dataset.