ActMAD: Activation Matching to Align Distributions for Test-Time-Training Supplementary Material

M. Jehanzeb Mirza[1,2] Pol Jané Soneira[3] Wei Lin[1,4] Mateusz Kozinski[1]
Horst Possegger[1] Horst Bischof[1,2]

[1]Institute for Computer Graphics and Vision, TU Graz, Austria.
[2]Christian Doppler Laboratory for Embedded Machine Learning.
[3]Institute of Control Systems, KIT, Germany.
[4]Christian Doppler Laboratory for Semantic 3D Computer Vision.

In the following, we present additional continuous online adaptation scenarios in which we test ActMAD (Section 1), provide all implementation details for baselines and our experiments to support reproducibility (Section 2), present additional ablation studies (Section 3), and more comparisons and insights for our ActMAD (Section 4).

1. Continuous Online Adaptation

Our location-aware feature alignment is especially helpful for the task of object detection. The fine-grained supervision helps ActMAD to adapt to different weather scenarios in a dynamic and efficient manner. Moreover, ActMAD also shows promising results while adapting to different weather scenarios in a continuous manner. In the main manuscript (Figure 1), we show such a simulated continuous adaptation scenario (Clear→Fog→Snow→Rain→Clear). We further test ActMAD in similar scenarios where the weather conditions are encountered in a different sequence. In Figure 1a and 1b, we present results for adaptation with ActMAD in two such scenarios.

The results show that ActMAD can achieve impressive results in different scenarios and the sequence of weather conditions which are encountered does not matter. This is close to the requirements in the real-world, where different weather conditions can occur in any order during driving. We also see from the results that ActMAD can even improve the performance of a KITTI [5] pre-trained YOLOv3 [19] on the validation split from the dataset (which represents the *Clear* weather condition). For example, in Figure 1b, where order of the weather conditions is *Clear*→*Clear*→*Fog*→*Snow*→*Rain*→*Clear*, ActMAD achieves 1.0% higher Mean Average Precision while adapting from *Clear*→*Clear* at the beginning of the adaptation cycle. This is because even though the adaptation is performed from *Clear*→*Clear*, the activation statistics could be different between the training and inference, *e.g.*, due to slight change in the environmental conditions or scenery. Our ActMAD helps to rectify this mismatch to obtain an increase in performance.

2. Implementation Details

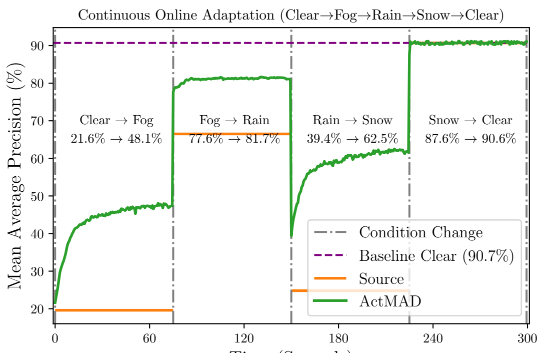
Hardware: All our experiments are performed on a single NVIDIA® GeForce® RTX 3090 and a GPU server consisting of four Quadro RTX 8000.

CIFAR-10/100C: We use an AugMix pre-trained [10] Wide-Resnet-40 [30] for our main results on CIFAR-10/100C. For experiments with ViT-B/16 [4] we finetune the ViT backbone (pre-trained on ImageNet-21k) for 10000 iterations on the training sets of CIFAR-10/100 [15] with a learning rate of $3\mathrm{e}{-2}$.

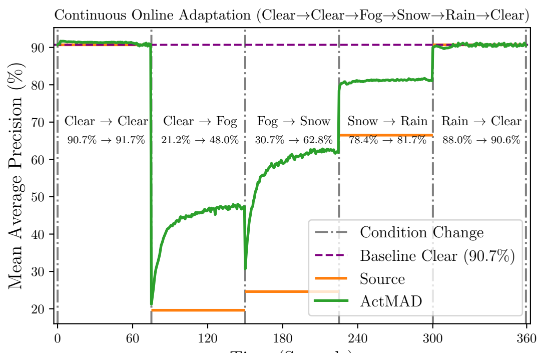
ImageNet-C: All ImageNet-C experiments use ImageNet pre-trained models. For the main results with ResNet-18 [7] we use the pre-trained weights from the PyTorch [18] model zoo. For experiments with ResNet-50 using Group Normalization [28] (Section 3.1), we use pre-trained weights from a third-party open source repository[1]. For experiments with a DeepAug [8] pre-trained ResNet-50 (Section 3.2), we use the pre-trained weights from Robust Bench[2]. For experiments on ImageNet-C with ViT-B/16 model (Section 3.4), we use the pre-trained weights from the open source *timm* library[3].

KITTI: All KITTI [5] experiments start with a model pre-trained on the original KITTI dataset. As it is collected

[1]github.com/ppwwyyxx/GroupNorm-reproduce, commit: 883f694
[2]robustbench.github.io/
[3]github.com/rwightman/pytorch-image-models, release: 0.6.11



(a) Clear→Fog→Snow→Rain→Clear



(b) Clear→Clear→Fog→Snow→Rain→Clear

Figure 1. Mean Average Precision (mAP@50) for two simulated continuous online adaptation scenarios with different weather sequences. *Source* refers to the model trained on clear weather condition and tested on the changing weather without adaptation.

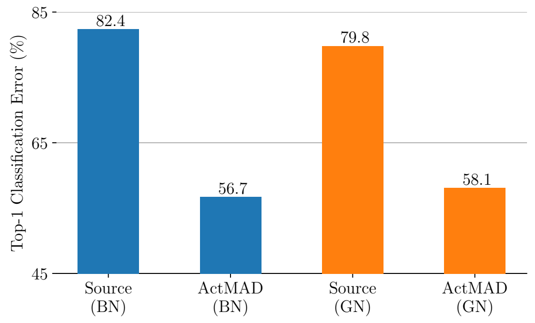


Figure 2. Mean Top-1 Classification Error (%) over all corruptions in ImageNet-C (Level 5) using a ResNet-50 with different normalization layers. BN: Batch Normalization, GN: Group Normalization. *Source* refer to results obtained without adaptation.

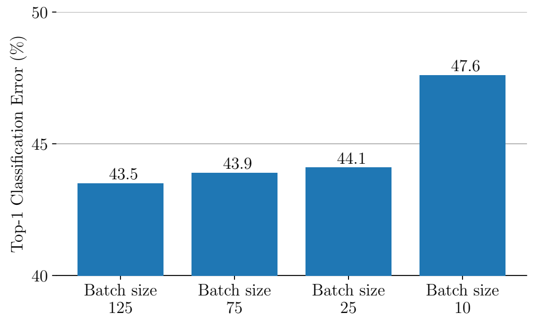


Figure 3. Mean Top-1 Classification Error (%) over all corruptions in ImageNet-C (Level 5) while decreasing the batch size. The backbone used is an AugMix [10] pre-trained ResNet-50. Here, *Source* (without adaptation) Error is 61.1%.

mostly in clear (*i.e.* sunny) weather conditions, we refer to it as KITTI-Clear. We adapt this model to degrading weather conditions, *i.e.* fog [6], rain [6] and snow [9]. For KITTI-Clear, we re-train an MS-COCO [13] pre-trained YOLOv3 in a supervised manner on the train split of the KITTI dataset. The model is trained for 100 epochs with a batch size of 30, while all the other training details are kept constant as in the PyTorch implementation of YOLOv3[4].

Baselines: Since all the baselines used to compare our results are primarily tested for object recognition on CIFAR-10/100C [9] and ImageNet-C [9], we use their official open-source codes and use all the hyper-parameters, which they report in their paper.

For object detection experiments, only DUA [16] provides official results[5]. Therefore, we implement NORM [22] for object detection using their official repository[6]. Since NORM is a gradient-free approach, it does not rely on any hyper-parameters. We use a batch size of 30 to obtain all the results for NORM.

TTT [23] uses rotation prediction as an auxiliary task for test-time training. TTT is also primarily tested for object recognition by its authors. We implement TTT for object detection for the purpose of comparison in our paper. We use the same batch size (*i.e.* 30), as for ActMAD. For joint-training with the auxiliary rotation prediction task we take the output from the feature encoder of YOLOv3 and use a single-layer linear classifier head to predict the rotation of the input image. The hyperparameters used for the KITTI pre-training (for ActMAD experiments) and joint-training are kept the same. However, we find that training a network jointly for the classification task of rotation prediction and object detection requires special design choices. For example, we use a loss scaling factor of 0.4 for the self-supervised task during joint-training to make the network converge. Similarly, the same loss scaling factor is used during test-time training. The evaluation protocol is kept

[4]github.com/ultralytics/yolov3, commit: d353371
[5]https://github.com/jmiemirza/DUA, commit: d6e5398
[6]https://github.com/bethgelab/robustness, commit: aa0a679

Corruptions:	Gauss	Shot	Impul	Defcs	Gls	Mtn	Zm	Snw	Frst	Fg	Brt	Cnt	Els	Px	Jpg	Mean	
								Level 4									
Source	24.1	17.1	16.4	6.6	23.5	8.4	7.4	12.2	11.5	8.3	6.2	9.2	10.6	19.4	13.1	12.9	
ActMAD	11.7	9.8	11.7	5.9	16.0	7.1	6.2	9.1	7.9	6.5	5.4	6.1	9.2	7.1	11.8	8.8	$\pm 7e-3$
								Level 3									
Source	20.4	14.6	9.7	5.4	12.9	8.6	6.5	9.9	11.4	6.3	5.5	7.2	7.4	9.6	12.1	9.8	
ActMAD	10.5	8.8	8.7	5.3	10.6	7.2	5.9	8.0	7.9	5.9	5.1	5.6	6.9	6.1	10.9	7.6	$\pm 7e-3$
								Level 2									
Source	13.4	8.8	8.0	5.1	14.2	6.5	5.8	9.2	8.5	5.3	5.3	6.1	6.5	7.8	10.9	8.1	
ActMAD	8.4	6.6	7.3	5.0	10.9	6.2	5.6	7.1	6.7	5.1	5.0	5.3	6.4	5.8	10.0	6.8	$\pm 8e-3$
								Level 1									
Source	8.7	6.5	6.2	4.9	14.1	5.5	5.9	6.4	6.5	4.9	5.0	5.0	6.9	5.8	8.7	6.7	
ActMAD	6.4	5.7	6.0	4.9	10.3	5.5	5.6	5.9	5.6	4.8	4.9	4.9	6.8	5.3	7.8	6.0	$\pm 9e-3$

Table 1. Top-1 Classification Error (%) for all corruptions in CIFAR-10C (level 4 – 1), highest severity is reported in the main manuscript. Lower is better. The results were obtained by adapting a **WRN-40-2 backbone**, trained on CIFAR-10, to CIFAR-10C. ActMAD results are averaged over 10 runs, and we report the mean error and its standard deviation.

Corruptions:	Gauss	Shot	Impul	Defcs	Gls	Mtn	Zm	Snw	Frst	Fg	Brt	Cnt	Els	Px	Jpg	Mean	
								Level 4									
Source	60.7	51.6	47.9	27.1	54.4	30.3	28.9	37.4	39.0	35.4	27.2	35.9	34.4	39.0	40.1	39.3	
ActMAD	38.0	34.9	35.3	26.1	41.1	28.0	27.1	32.8	30.8	31.0	24.7	26.6	31.9	28.4	37.2	31.6	$\pm 4e-3$
								Level 3									
Source	55.2	45.9	36.9	25.7	39.9	30.5	27.4	33.3	38.1	29.5	25.5	30.5	28.6	30.3	38.0	34.4	
ActMAD	36.6	32.9	30.5	24.8	34.1	28.2	26.3	30.3	30.4	27.9	24.1	25.5	27.6	26.4	35.7	29.4	$\pm 3e-3$
								Level 2									
Source	44.6	34.5	30.7	24.3	41.5	27.7	26.2	32.7	31.8	26.8	24.4	27.5	27.9	28.0	36.5	31.0	
ActMAD	32.4	28.5	27.4	24.0	33.7	26.5	25.4	28.9	28.0	25.4	23.7	24.8	27.6	26.1	34.4	27.8	$\pm 2e-3$
								Level 1									
Source	34.4	29.6	26.9	23.8	42.9	25.6	26.1	26.1	27.4	24.0	23.8	24.3	28.4	25.2	32.4	28.1	
ActMAD	28.5	26.7	25.6	23.5	33.7	24.9	25.1	25.4	25.6	23.7	23.5	23.8	28.2	24.8	31.0	26.3	$\pm 2e-3$

Table 2. Top-1 Classification Error (%) for all corruptions in CIFAR-100C (level 4 – 1), highest severity is reported in the main manuscript. Lower is better. The results were obtained by adapting a **WRN-40-2 backbone**, trained on CIFAR-100, to CIFAR-100C. ActMAD results are averaged over 10 runs, and we report the mean error and its standard deviation.

consistent for all baselines and our method.

We also implemented TTT++ [14] on YOLOv3 in order to use it as a baseline for object detection experiments. We used the same prediction head, hyper-parameters and the augmentations (for creating two augmented views for SimCLR [2] training) used by the original TTT++ paper for their experiments on CIFAR-10/100. We experimented with several design choices for joint-training (object detection and contrastive SimCLR task), each for a total of 500 epochs on the KITTI train set, starting from the pre-trained weights on MS-COCO. However, despite the best of our efforts we could not make the joint-training converge. A potential issue could be the requirement of using larger batch sizes for training the contrastive learning task. Initially, we tested with a batch size of 100 for joint training by keeping the original aspect ratio of KITTI images (*i.e.* 370×1224). However, realizing that we might require larger batch sizes, we had to downscale the KITTI images equal to the aspect ratio of ImageNet images (*i.e.* 224×224), to be able to increase the batch size to 300, but in this case the regression loss for object detection did not converge and we get low performance on the validation set of KITTI. We suspect that it might be because of the drastic decrease in the image size, as compared to the original aspect ratio of images in the KITTI

Corruptions:	Gauss	Shot	Impul	Defcs	Gls	Mtn	Zm	Snw	Frst	Fg	Brt	Cnt	Els	Px	Jpg	Mean
Level 4																
Source	93.2	94.7	94.3	84.5	89.4	85.3	77.2	83.4	79.4	72.8	44.5	88.1	63.4	71.2	58.8	78.7
ActMAD	65.7	70.1	68.6	68.4	67.0	64.8	58.4	64.6	60.7	48.3	37.7	57.3	41.9	48.1	48.4	$58.0 \pm 2e-2$
Level 3																
Source	80.9	82.7	82.9	74.1	85.4	73.9	71.9	73.6	77.8	66.2	39.6	65.3	51.1	56.7	49.3	68.8
ActMAD	55.2	56.4	57.7	58.8	61.5	53.3	54.0	56.6	59.7	45.3	35.2	43.5	37.9	41.0	41.5	$50.5 \pm 1e-2$
Level 2																
Source	63.6	67.8	74.6	59.4	65.4	57.7	65.2	77.5	67.0	56.3	36.4	51.5	60.9	43.3	46.3	59.5
ActMAD	46.7	48.3	52.8	47.9	47.5	43.7	49.7	56.5	52.0	41.4	33.4	38.6	52.0	35.8	39.3	$45.7 \pm 9e-3$
Level 1																
Source	50.5	53.1	61.9	51.3	52.4	45.2	55.9	55.5	49.7	49.0	34.2	44.0	40.4	41.3	42.6	48.5
ActMAD	40.6	41.4	46.6	42.2	40.6	37.7	44.6	44.1	41.4	38.8	32.1	36.2	37.8	35.1	36.9	$39.7 \pm 6e-3$

Table 3. Top-1 Classification Error (%) for all corruptions in ImageNet-C (level 4 – 1), highest severity is reported in the main manuscript. Lower is better. The results were obtained by adapting a **ResNet-18 backbone**, trained on ImageNet, to ImageNet-C. ActMAD results are averaged over 10 runs, and we report the mean error and its standard deviation.

Corruptions:	Gauss	Shot	Impul	Defcs	Gls	Mtn	Zm	Snw	Frst	Fg	Brt	Cnt	Els	Px	Jpg	Mean
CIFAR-10C																
Source	23.6	21.9	28.7	5.2	27.9	9.1	4.4	5.5	8.4	14.2	2.4	15.2	19.9	25.5	13.3	15.0
ActMAD	12.9	11.0	11.7	4.1	11.2	5.8	2.7	4.5	4.4	9.3	2.0	6.1	7.2	4.4	10.9	7.3 ± 0.1
CIFAR-100C																
Source	55.0	52.9	57.8	18.0	60.5	23.6	16.0	22.3	27.5	34.2	11.9	35.3	34.8	43.3	33.7	35.1
ActMAD	31.8	29.7	28.5	18.5	28.3	20.1	14.3	19.2	17.9	26.8	11.9	21.3	23.7	15.4	29.5	22.4 ± 0.1
ImageNet-C																
Source	53.1	52.4	53.1	57.3	65.8	49.6	55.3	43.1	47.4	43.5	23.9	68.2	53.3	34.5	34.0	49.0
ActMAD	44.0	41.9	42.4	47.7	46.9	40.5	40.3	32.6	36.2	31.5	20.6	39.0	33.4	27.0	30.6	36.9 ± 0.3

Table 4. Top-1 Classification Error (%) for all corruptions in CIFAR-10/100C and ImageNet-C (level 5). Lower is better. All results are obtained by using a **ViT-B/16 backbone**. ActMAD results are averaged over 10 runs, and we report the mean error and its standard deviation.

dataset, which is unsuitable for the object detection task on the KITTI dataset.

3. Additional Results

In this section we provide additional ablation studies, results on lower severity levels for ActMAD, detailed results for the vision transformer backbone [4] and the Continuous Adaptation experiment.

3.1. Different Normalization Techniques

Recent test-time training approaches which adapt the network statistics, *e.g*. NORM [26] and DUA [16], critically rely on batch normalization [11]. However, if the network is equipped with other forms of normalization, *e.g*. Instance [24], Layer [1] or Group Normalization [28], these approaches are not applicable. In contrast, our ActMAD is agnostic to the architecture and can work with any type of normalization applied in the network. We already showed in the main manuscript (Table 3) that it can be applied to a Transformer-based architecture, which uses Layer Norm. In order to test the performance of ActMAD with different normalization layers inside the convolution-based architectures, we test ActMAD with a ResNet-50 equipped with Group Normalization. Figure 2 shows the adaptation results of a ResNet-50 with both Batch Normalization (BN) and Group Normalization (GN). Using GN, our ActMAD performance decreases by only 1.8% over the same architecture using BN.

3.2. Robustifying an Already Robust Model

ActMAD can be effectively used with any off-the-shelf pre-trained model. To test this, we apply ActMAD for test-time adaptation of an already robust, AugMix [10] pre-trained ResNet-50. We further decrease the batch size for

Corruptions:	Gauss	Shot	Impul	Defcs	Gls	Mtn	Zm	Snw	Frst	Fg	Brt	Cnt	Els	Px	Jpg	Mean	
								CIFAR-10C									
Source	28.8	22.9	26.2	9.5	20.6	10.6	9.3	14.2	15.3	17.5	7.6	20.9	14.7	41.3	14.7	18.3	
CoTTA	**12.9**	13.8	**12.0**	9.1	**14.3**	**9.2**	9.2	12.0	**10.0**	**12.1**	7.0	14.3	14.1	**9.9**	**12.7**	11.5	± 0.1
ActMAD	13.3	**11.6**	15.7	**8.7**	16.7	9.9	**8.3**	**10.5**	**10.0**	12.8	**6.9**	**10.6**	**13.3**	**9.9**	13.4	**11.4**	± 0.1
								CIFAR-100C									
Source	65.7	60.1	59.1	32.0	51.0	33.6	32.4	41.4	45.2	51.4	31.6	55.5	40.3	59.7	42.4	46.8	
CoTTA	40.8	39.2	**40.8**	**29.9**	**40.9**	**30.7**	**29.8**	35.9	34.9	43.4	**26.9**	38.2	**36.6**	**30.9**	**36.8**	**35.7**	± 0.4
ActMAD	**39.6**	**38.3**	41.4	31.8	41.7	32.7	30.9	**34.6**	**33.6**	**42.9**	27.4	**36.3**	37.2	31.9	38.6	35.9	± 0.5

Table 5. Top-1 Classification Error (%) for all corruptions in CIFAR-10/100C (level 5), while continuously adapting to corruptions. Lower is better. These results were obtained by adapting a **Wide-ResNet-40-2 backbone**, trained on CIFAR-10/100, to CIFAR-10/100C. ActMAD results are averaged over 10 runs, and we report the mean error and its standard deviation. The lowest error is shown in bold, the second best is underlined.

TENT	EATA	ActMAD	TENT+ActMAD	EATA+ActMAD
67.2	**64.9**	66.0	64.1	**62.7**

Table 6. Top-1 Error (%) averaged over 15 corruptions in the ImageNet-C dataset while adapting an ImageNet pre-trained ResNet-18 from PyTorch model zoo.

	Source-only	MemCLR	ActMAD
mean Avg. Prec. (mAP)	25.2	29.8	**34.5**

Table 7. Mean Average Precision (mAP@50) for a Faster R-CNN pre-trained on the Cityscapes dataset and adapted to the FoggyCityScapes test split.

CFA	DUA	TENT	SHOT	TTT++	ActMAD$^{\text{full}}$	ActMAD$^{\text{affine}}$
5.3	2.1	5.2	6.1	6.8	5.8	5.2

Table 8. Total runtime (seconds) for CIFAR-10C, batch size 128 (see Table 1, main paper). ActMAD$^{\text{affine}}$ results are provided in Table 7, main paper.

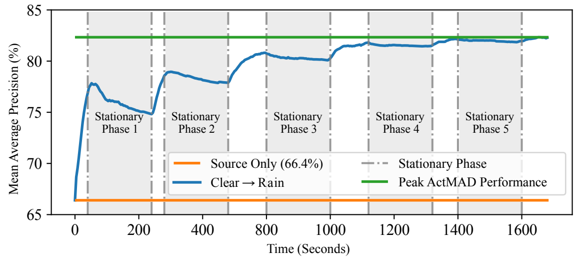


Figure 4. Mean Average Precision (mAP@50) for ActMAD adaptation from KITTI-Clear → KITTI-Rain for a simulated scenario where the car is assumed to be stationary. The gray patches represent the randomly chosen 'stationary' intervals.

adaptation on this backbone. Results for this experiment (highest severity) are provided in Figure 3. We find that ActMAD decreases the error significantly even for an already robust model. For example, the Source (without adaptation) top-1 error (%) over the 15 corruptions on ImageNet-C is 61.1%, while after ActMAD adaptation the error decreases to merely 43.5%. Furthermore, we also see that while applying ActMAD to a robust model, the decrease in performance with decreasing batch size is also minimal (similar to the ablation study in the main manuscript, Figure 4, with ResNet-18 backbone). For example, while decreasing the batch size from 125 to 10, the error increases by only 4.1%.

3.3. Lower Severity Results

In the main manuscript we provide results for the highest severity (Level 5) in the Robust Bench [9]. Our ActMAD achieves impressive gains in performance even for lower severity levels. In Table 1 and Table 2 we provide the mean and standard deviation of results (over 10 random runs) obtained by ActMAD on lower severities (*i.e.* Levels 4 – 1) on CIFAR-10/100C. We use the Wide-ResNet-40 model for these evaluations. Furthermore, in Table 3, we provide the mean and standard deviation of results over 10 random runs on the large scale ImageNet-C (with ResNet-18) dataset for lower severities (Levels 4 – 1). The results show that ActMAD can even be applied when the distribution shift between the train and test data is not too large.

3.4. Vision Transformer

ActMAD can also be seamlessly applied to the Vision Transformer backbone. In the main manuscript we provide mean results over the 15 corruptions for CIFAR-10/100C (Level 5). For reference, in Table 4 we provide results for each individual corruption for the highest severity in the CIFAR-10/100C. Additionally, we also provide ActMAD results for the highest severity on the ImageNet-C benchmark.

3.5. Continuous Adaptation to Perturbations

In the main manuscript (Table 4) we compare with CoTTA [27] on their continuous adaptation benchmark and provide the mean error over the 15 corruptions (highest severity) in the CIFAR-C benchmark. In Table 5 we provide detailed results, averaged over 10 random runs, for each individual corruption for documentation purposes.

4. Additional insights

Here, we provide comparisons with EATA [17] (for image classification) and MemCLR [25] (for object detection), and test ActMAD in more realistic scenarios.

4.1. Comparison with additional baselines

Image Classification: In Table 6 we compare our ActMAD with two other entropy based methods, EATA [17] and TENT [26]. For these results, an ImageNet pre-trained ResNet-18 is adapted to ImageNet-C. We see that the recent state-of-the-art method for the entropy based adaptation EATA, outperforms ActMAD. However, we find that the logit-level supervision provided by EATA and TENT is complementary to the test-time supervision provided by ActMAD. Thus, combining the entropy based objectives with activation distribution matching objective from ActMAD, results in substantial performance gains. It is also worth highlighting, that EATA (like TENT) is also reliant on minimizing the entropy of the output distribution for test-time adaptation, rendering it unsuitable for application to regression tasks (*e.g*. Object Detection). Whereas, ActMAD is free from such constraints and is task-agnostic in nature.

Object Detection: In Table 7 we compare our ActMAD with MemClr [25]. Our ActMAD outperforms MemCLR comfortably in their evaluation setup. To be comparable with the results reported in their paper, we re-train a Faster-RCNN [20] implemented in the Detectron-2 framework [29] (using their default settings) on the train split of the CityScapes dataset [3]. For TTT, we adapt this pre-trained model to the FoggyCityScapes [21] test split by using a batchsize of 4 and a learning rate of $5\mathrm{e}{-4}$.

4.2. ActMAD in stationary scenarios

In practice such scenarios can be encountered where a vehicle is stationary for a certain period of time. For example, when a vehicle is parked, it can happen that the sensors (*e.g*., camera), mounted on the vehicle is recording a similar type of data for the stationary period. Ideally, the test-time training algorithm should not diverge catastrophically while adapting on this similar type of data being recorded. To test ActMAD in such a scenario, we choose random stationary intervals while adapting a YOLOv3 object detector from KITTI-Clear $\rightarrow$ KITTI-Rain. To simulate the stationary intervals, we adapt on the same batch of data for this randomly chosen interval. These online adaptation results are plotted in Figure 4. We see that during the stationary intervals, the performance of ActMAD suffers a minor degradation, however, when the diverse batches of data are encountered, ActMAD recovers quickly. In practice, to counter the minor degradation in performance during the stationary intervals, adaptation can be limited to batches with sufficient diversity. One way to achieve this can be to threshold the difference between the statistics of the intermediate activation responses from the batches received in an online manner at test-time.

4.3. Runtime comparison with baselines

Contrary to test-time training approaches (*e.g*. [12, 26]) which only adapt the affine parameters at test-time, ActMAD proposes to optimize the entire parameter vector for test-time training. In order to compare the runtime for adaptation at test-time, we adapt ActMAD and other baselines to a single distribution shift in CIFAR-10C and provide the results in Table 8. The runtimes are on-par for ActMAD and all other approaches, except for DUA (which only adjusts the running mean and variance estimates, without backpropagation).

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer Normalization. *arXiv*, 2016. 4

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In *Proc. ICML*, 2020. 3

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. CVPR*, 2016. 6

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proc. ICLR*, 2020. 1, 4

[5] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *IJR*, 32(11):1231–1237, 2013. 1

[6] Shirsendu Sukanta Halder, Jean-François Lalonde, and Raoul de Charette. Physics-based Rendering for Improving Robustness to Rain. In *Proc. ICCV*, 2019. 2

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proc. CVPR*, 2016. 1

[8] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization. In *Proc. ICCV*, 2021. 1

[9] Dan Hendrycks and Thomas Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *Proc. ICLR*, 2019. 2, 5

[10] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *Proc. ICLR*, 2020. 1, 2, 4

[11] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. ICML*, 2015. 4

[12] Takeshi Kojima, Yutaka Matsuo, and Yusuke Iwasawa. Robustifying Vision Transformer without Retraining from Scratch by Test-Time Class-Conditional Feature Alignment. *Proc. IJCAI*, 2022. 6

[13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Proc. ECCV*, 2014. 2

[14] Yuejiang Liu, Parth Kothari, Bastien van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. TTT++: When Does Self-Supervised Test-Time Training Fail or Thrive? In *NeurIPS*, 2021. 3

[15] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning Transferable Features with Deep Adaptation Networks. In *Proc. ICML*, 2015. 1

[16] M Jehanzeb Mirza, Jakub Micorek, Horst Possegger, and Horst Bischof. The Norm Must Go On: Dynamic Unsupervised Domain Adaptation by Normalization. In *Proc. CVPR*, 2022. 2, 4

[17] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient Test-Time Model Adaptation without Forgetting. In *Proc. ICML*, 2022. 6

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019. 1

[19] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv*, 2018. 1

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NeurIPS*, 2015. 6

[21] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Semantic Foggy Scene Understanding with Synthetic Data. *IJCV*, 2018. 6

[22] Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving Robustness Against Common Corruptions by Covariate Shift Adaptation. In *NeurIPS*, 2020. 2

[23] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-Time Training with Self-Supervision for Generalization under Distribution Shifts. In *Proc. ICML*, 2020. 2

[24] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv*, 2016. 4

[25] Vibashan VS, Poojan Oza, and Vishal M Patel. Towards Online Domain Adaptive Object Detection. In *Proc. WACV*, 2023. 6

[26] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully Test-time Adaptation by Entropy Minimization. In *Proc. ICLR*, 2020. 4, 6

[27] Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual Test-Time Domain Adaptation. In *Proc. CVPR*, 2022. 6

[28] Yuxin Wu and Kaiming He. Group Normalization. In *Proc. ECCV*, 2018. 1, 4

[29] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019. 6

[30] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. In *Proc. BMVC*, 2016. 1