Bag of Tricks for Fully Test Time Adaptation

Saypraseuth Mounsaveng*, Florent Chiaroni, Malik Boudiaf, Marco Pedersoli, Ismail Ben Ayed
ÉTS Montréal, Canada

1. TTA Problem settings

In this section, we provide an overview of the different TTA settings.

Setting	Source Data	Target Data	Training Loss	Testing Loss	Offline	Online	Source Acc.
Fine-tuning	✗	x^t, y^t	$\mathcal{L}(x^t, y^t)$	-	✓	✗	NC
Continual learning	✗	x^t, y^t	$\mathcal{L}(x^t, y^t)$	-	✓	✗	M
Unsupervised domain adaptation	x^s, y^s	x^t	$\mathcal{L}(x^s, y^s) + \mathcal{L}(x^s, x^t)$	-	✓	✗	M
Test-time training	x^s, y^s	x^t	$\mathcal{L}(x^s, y^s) + \mathcal{L}(x^s)$	$\mathcal{L}(x^t)$	✗	✓	NC
Fully test-time adaptation (FTTA)	✗	x^t	✗	$\mathcal{L}(x^t)$	✗	✓	NC

Table 1. **Overview of TTA problem settings [3].** In our work, we consider the Fully Test-Time Adaptation (FTTA) scenario, which is source-free and online. In last column, NC=Not Considered and M=Maintained.

2. Technical details

In this section, we provide additional technical details.

In Tab. 2, we compare the size of the different architectures mentioned in our work. In Tab. 3, we provide the links to the source code of the methods we compare and finally in Tab. 4, we provide the links to the weights of the pretrained models used in our experiments.

Architecture	Number of parameter
ResNet50-BN	25M
ResNet50-GN	25M
ResNet-101	43M
VitBase-LN	86M
WRN28-10	36.5M
WRN40-2	2.2M

Table 2. **Number of parameters of each architecture used in our experimental setup.**

*Corresponding author: saypraseuth.mounsaveng.1@etsmtl.net
†Code is available at https://github.com/smounsav/tta_bot

Method	Code Link
Tent [9]	https://github.com/DequanWang/tent
SAR [4]	https://github.com/mr-eggplant/SAR
Delta [13]	https://github.com/bwbwzhao/DELTA
DUA [2]	https://github.com/jmiemirza/DUA
Hebbian [8]	n/a

Table 3. **Links to the source code of the methods mentioned in the article.**

Architecture	Code Link
ResNet50-BN	https://download.pytorch.org/models/resnet50–9c8e357.pth torchvision [5]
ResNet50-GN	timm [10]
ResNet-101	https://github.com/Albert0147/NRC_SFDA [12]
VitBase-LN	timm [10]
WRN28-10	RobustBench [1]
WRN40-2	RobustBench [1]
SVHN model	Pytorch-Playground [11]

Table 4. **Links to the weights of the pretrained models mentioned in the paper.**

3. Algorithms

In this section, we present the details of the algorithms used in our experiments.

In Algo. 1, we introduce the DOT [13] algorithm used in the class rebalancing scenario.

Algorithm 1 Dynamic Online reweighTing (DOT) [13]

Input: inference step $t := 0$; test stream samples $\{x_j\}$; pre-trained model $f_{\{\theta_0,a_0\}}$; class-frequency vector z_0; loss function $\mathcal{L}$; smooth coefficient λ.
while *the test mini-batch* $\{x_{m_t+b}\}_{b=1}^{B}$ *arrives* **do**
  $t \leftarrow t+1$
  // output predictions
  $\{p_{m_t+b}\}_{b=1}^{B}, f_{\{\theta_{t-1},a_t\}} \leftarrow$Forward$(\{x_{m_t+b}\}_{b=1}^{B}, f_{\{\theta_{t-1},a_{t-1}\}})$
  for $b = 1$ **to** B **do**
    // predicted label
    $k^*_{m_t+b} = arg\,max_{k\in[1,K]} p_{m_t+b}[k]$
    // assign sample weight
    $w_{m_t+b} = 1/(z_{t-1}[k^*_{m_t+b}] + \epsilon)$
  end for
  // normalize sample weight
  $\bar{w}_{m_t+b} = B.w_{m_t+b}/\Sigma_{b'=1}^{B} w_{m_t+b'}, b = 1, 2, \ldots, B$
  // combine sample weight with loss
  $l = \frac{1}{B}\Sigma_{b=1}^{B} \bar{w}_{m_t+b}.\mathcal{L}(p_{m_t} + b)$
  // update θ
  $f_{\{\theta_t,a_t\}} \leftarrow$Backward&Update$(l, f_{\{\theta_{t-1},a_t\}})$
  // update z
  $z_t \leftarrow \lambda z_{t-1} + \frac{(1-\lambda)}{B}\Sigma_{b=1}^{B} p_{m_t+b}$
end while

4. Comparison to other methods – Additional experiments

In this section, we provide additional results to extend the comparison of our selected methods to other methods.

In Tab. 5, we present results for experiments on CIFAR10-C on 3 different architectures ResNet26, WRN28-10 and WRN40-2. On ResNet26, BoT obtains the best results, whereas Hebbian Learning [8] performs best on WRN28-10 and WRN40-2. On the last 2 architectures, updating the model using hebbian learning seems more performant than using methods updating only the batch norm layers.

	Methods	gaus	shot	impul	defcs	gls	mtn	zm	snw	frst	fg	brt	cnt	els	px	jpg	Avg.
ResNet26	Source	67.7	63.1	69.9	55.3	56.6	42.2	50.1	31.6	46.3	39.1	17.1	74.6	34.2	57.9	31.7	49.2
	TTT [7]	45.6	41.8	50.0	21.8	46.1	23.0	23.9	29.9	30.0	25.1	12.2	23.9	22.6	47.2	27.2	31.4
	NORM [6]	44.6	43.7	49.1	29.4	45.2	26.2	26.9	25.8	27.9	23.8	18.3	34.3	29.3	37.0	32.5	32.9
	DUA [2]	34.9	32.6	42.2	18.7	40.2	24.0	18.4	23.9	24.0	20.9	12.3	27.1	27.2	26.2	28.7	26.8
	Hebbian [8]	33.2	30.6	38.2	17.7	41.2	20.8	17.4	24.0	27.2	20.4	13.5	21.1	28.4	23.7	28.9	25.8
	TENT [9]	39.4	38.8	47.9	19.9	45.0	23.2	20.6	28.1	32.1	24.5	16.1	26.7	32.4	30.6	35.5	30.7
	SAR [4]	**27.0**	**24.8**	35.2	14.3	34.0	15.9	14.6	**18.5**	19.3	15.5	11.5	15.5	24.0	18.5	24.7	20.9
	Delta [13]	27.6	26.0	**34.7**	13.6	33.8	16.1	13.7	18.6	19.5	15.0	10.0	13.7	24.1	17.9	24.7	20.6
	BoT	27.8	25.2	34.8	**13.4**	**33.2**	**15.2**	**13.4**	**18.5**	**19.2**	**14.8**	**9.9**	**13.6**	**23.7**	**18.1**	**24.4**	**20.3**
WRN28-10	Source	72.3	65.7	72.9	46.9	54.3	34.8	42.0	25.1	41.3	26.0	9.3	46.7	26.6	58.5	30.3	43.5
	NORM [6]	28.1	26.1	36.3	12.8	35.3	14.2	12.1	17.3	17.4	15.3	8.4	12.6	23.8	19.7	27.3	20.4
	DUA [2]	27.4	24.6	35.3	13.1	34.9	14.6	11.6	16.8	17.5	13.1	**7.6**	14.1	22.7	19.3	26.2	19.9
	Hebbian [8]	**23.6**	21.4	**30.9**	**11.0**	31.1	13.0	10.9	**14.2**	**15.5**	**13.0**	8.0	10.3	21.8	16.7	**22.4**	**17.6**
	TENT [9]	24.8	23.5	33.0	12.0	31.8	13.7	10.8	15.9	16.2	13.7	7.9	12.1	22.0	17.3	24.2	18.6
	SAR [4]	24.4	23.1	31.4	12.9	31.4	14.1	12.4	17.4	17.7	15.2	8.4	13.1	21.9	18.8	23.8	19.1
	Delta [13]	24.3	22.0	31.2	11.6	**30.9**	**12.9**	10.8	15.3	15.7	13.1	7.8	10.2	**21.6**	**16.6**	23.5	17.8
	BoT	24.1	**21.9**	31.4	11.7	31.0	**12.9**	**10.7**	15.3	15.6	13.2	7.9	**9.9**	21.7	**16.6**	23.6	17.8
WRN40-2	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
	NORM [6]	18.7	16.4	22.3	9.1	22.1	10.5	9.7	13.0	13.2	15.4	7.8	12.0	16.4	15.1	17.6	14.6
	DUA [2]	15.4	13.4	17.3	8.0	18.0	9.1	**7.7**	10.8	10.8	12.1	6.6	10.9	13.6	13.0	14.3	12.1
	Hebbian [8]	**13.4**	**12.3**	15.0	**7.5**	**16.0**	**8.7**	**7.7**	**9.1**	**9.6**	**10.1**	**6.4**	**8.2**	**13.3**	**9.3**	**13.3**	**10.7**
	TENT [9]	15.7	13.2	18.8	7.9	18.1	9.0	8.0	10.4	10.8	12.4	6.7	10.0	14.0	11.4	14.8	12.1
	SAR [4]	14.7	12.7	17.2	8.4	17.2	9.4	8.5	10.6	10.7	11.8	7.3	9.8	13.8	11.4	14.4	11.9
	Delta [13]	14.4	**12.3**	16.9	7.6	16.8	**8.7**	7.6	9.8	10.0	10.6	6.5	8.3	13.6	10.4	14.5	11.2
	BoT	14.3	**12.3**	**16.7**	7.6	16.9	8.8	**7.7**	9.8	9.9	10.8	6.5	9.1	13.7	10.7	14.4	11.3

Table 5. **Top-1 Classification Error (%) for each corruption on CIFAR10-C at the highest severity level (Level 5).** The architecture used in the experiments are ResNet26 (top), WRN28-10 (middle) and WRN40-2 (bottom). Sources to pretrained weights are available in Tab. 4. Results for TTT, NORM, DUA, TENT and Hebbian are reported from [8]. Other results were the average of 3 runs using implementations provided by respective authors and documented in Tab. 3. Batch size used is 128 and follows [8]. Best results are shown in **bold**.

	Methods	gaus	shot	impul	defcs	gls	mtn	zm	snw	frst	fg	brt	cnt	els	px	jpx	Avg.
WRN40-2	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	46.7
	NORM [6]	44.7	44.2	47.4	32.4	46.4	32.9	33.0	39.0	38.4	45.3	30.2	36.6	40.6	37.2	44.2	39.5
	DUA [2]	42.2	40.9	41.0	30.5	44.8	32.2	29.9	38.9	37.2	43.6	29.5	39.2	39.0	35.3	41.2	37.6
	Hebbian [8]	38.4	**37.1**	**36.2**	**28.4**	**41.0**	**29.3**	29.7	**32.2**	**33.1**	**36.1**	**26.4**	30.9	**36.2**	**30.8**	**38.3**	**33.6**
	TENT [9]	**40.3**	39.9	41.8	29.8	42.3	31.0	30.0	34.5	35.2	39.5	28.0	33.9	38.4	33.4	41.4	36.0
	SAR [4]	40.7	39.4	39.1	29.8	42.3	31.1	29.9	34.3	35.1	37.0	28.2	31.5	37.9	32.2	40.4	35.3
	Delta [13]	40.7	39.6	39.1	29.1	41.9	30.8	29.7	34.5	34.7	37.0	27.5	30.3	37.9	32.2	40.4	35.0
	BoT	40.5	39.1	39.1	29.1	41.8	30.7	**29.5**	34.3	34.7	36.9	27.5	**30.2**	38.1	32.1	40.3	34.9

Table 6. **Top-1 Classification Error (%) for each corruption on CIFAR100-C at the highest severity level (Level 5).** The architecture used in the experiments is WRN-40-2. Source to pretrained weights is available in Tab. 4. Results for NORM, DUA, TENT and Hebbian are reported from [8]. Other results were the average of 3 runs using implementations provided by respective authors and documented in Tab. 3. Batch size used is 128 and follows [8]. Best results are shown in **bold**.

In Tab. 6, we present results for experiments on CIFAR100-C dataset on WRN40-2 network. Hebbian learning [8] gets slightly better results than BoT models updating only the BatchNorm layers.

	Methods	MNIST	MNIST-M	USPS	Avg.
SVHN [11]	NORM [6]	39.6	52.1	41.4	44.4
	Hebbian [8]	31.2	**47.9**	**32.6**	**37.2**
	TENT [9]	45.8	56.2	48.3	50.1
	SAR [4]	36.5	54.4	43.2	44.7
	Delta [13]	54.4	48.3	48.3	50.3
	BoT	**27.9**	48.3	38.9	38.4

Table 7. **Top-1 Classification Error (%) for test-time adaptation on digit recognition.** The architecture used in the 'svhn' model from pytorch-playground repository. Source to pretrained weights is available in Tab. 4. Results for NORM, TENT and Hebbian are reported from [8]. Other results were the average of 3 runs using implementations provided by respective authors and documented in Tab. 3. Batch size used is 128 and follows [8]. Best results are shown in **bold**.

In Tab. 7, we present results for experiments on test-time adaptation for digit recognition. More precisely, we adapt a model trained on SVHN dataset to 3 different datasets, MNIST, MNIST-M and USPS. On MNIST, BoT performs best, however on the 2 other datasets, Hebbian Learning [8] performs best. On average over the 3 datasets, Hebbian Learning performs best, beating other methods updating only the BatchNorm Layers.

References

[1] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2021. 1

[2] M. Jehanzeb Mirza, Jakub Micorek, Horst Possegger, and Horst Bischof. The norm must go on: Dynamic unsupervised domain adaptation by normalization. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1, 2

[3] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shi Dong Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *International Conference on Machine Learning (ICML)*, 2022. 1

[4] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Zhiquan Wen, Yaofo Chen, Peilin Zhao, and Mingkui Tan. Towards stable test-time adaptation in dynamic wild world. In *International Conference on Learning Representations (ICLR)*, 2023. 1, 2

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach 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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 1

[6] Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving robustness against common corruptions by covariate shift adaptation. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2

[7] 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 *International Conference on Machine Learning (ICML)*, 2020. 2

[8] Yushun Tang, Ce Zhang, Heng Xu, Shuoshuo Chen, Jie Cheng, Luziwei Leng, Qinghai Guo, and Zhihai He. Neuromodulated hebbian learning for fully test-time adaptation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1, 2

[9] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno A. Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations (ICLR)*, 2021. 1, 2

[10] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. 1

[11] Aaron Xichen. Pytorch-playground. https://github.com/aaron-xichen/pytorch-playground, 2023. 1, 2

[12] Shiqi Yang, Yaxing Wang, Joost van de Weijer, Luis Herranz, and Shangling Jui. Exploiting the intrinsic neighborhood structure for source-free domain adaptation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 1

[13] Bowen Zhao, Chen Chen, and Shutao Xia. Delta: degradation-free fully test-time adaptation. In *International Conference on Learning Representations (ICLR)*, 2023. 1, 2