This WACV 2023 paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

TTTFlow: Unsupervised Test-Time Training with Normalizing Flow

David Osowiechi* , Gustavo A. Vargas Hakim*,
Mehrdad Noori , Milad Cheraghalikhani , Ismail Ben Ayed , and Christian Desrosiers

LIVIA, ÉTS Montréal, Canada
International Laboratory on Learning Systems (ILLS),
McGILL - ETS - MILA - CNRS - Université Paris-Saclay - CentraleSupélec, Canada
david.osowiechi.1@ens.etsmtl.ca, gustavo-adolfo.vargas-hakim.1@ens.etsmtl.ca,
mehrdad.noori.1@ens.etsmtl.ca,milad.cheraghalikhani.1@ens.etsmtl.ca,
ismail.benayed@etsmtl.ca, christian.desrosiers@etsmtl.ca

Abstract

A major problem of deep neural networks for image classification is their vulnerability to domain changes at test-time. Recent methods have proposed to address this problem with test-time training (TTT), where a two-branch model is trained to learn a main classification task and also a self-supervised task used to perform test-time adaptation. However, these techniques require defining a proxy task specific to the target application. To tackle this limitation, we propose TTTFlow: a Y-shaped architecture using an unsupervised head based on Normalizing Flows to learn the normal distribution of latent features and detect domain shifts in test examples. At inference, keeping the unsupervised head fixed, we adapt the model to domain-shifted examples by maximizing the log likelihood of the Normalizing Flow. Our results show that our method can significantly improve the accuracy with respect to previous works.

1. Introduction

Deep learning has become increasingly effective for computer vision tasks such as segmentation or classification. Nevertheless, these achievements are often made under the assumption that training and test data share the same distribution, which is not always the case in practice. Furthermore, a small distribution shift between the training and test data can lead to an important drop in model performance [20]. Two types of methods were proposed to increase the robustness of the model to distributional change: Domain Generalization and Domain Adaptation. Domain Generalization (DG) [24, 19, 26] involves training on a large set of source data from several domains to help the model be more robust and generalize to unseen domains. However, DG requires a large amount of data from different domains, which can be difficult to obtain, and there is no guarantee that the new model can generalize well to an unseen domain at test-time. On the other hand, Domain Adaptation (DA) [3, 14, 27] aims to avoid performance degradation of a model trained on a source domain when used on a test set from a different domain. The distribution shift in this context is reduced without prior training on different domains, but in some cases requires access to the labeled source samples.

Test-Time Adaptation (TTA) [13, 25, 22, 2] is an emerging field that studies approaches to quickly adapt a pretrained deep network to domain shifts during test-time. Unlike DG, the source training typically involves a single domain. Moreover, in contrast to DA, it is possible to fine-tune the network at test-time. This task remains challenging as it is expected that the source data is not available during test-time, hence directly measuring the domain discrepancy becomes complicated. However, finding a solution for TTA is an attractive endeavor, as it promises a more widely useful deployment of deep networks in real-world contexts. Recent TTA techniques have explored adapting batch statistics in the feature extractor of deep networks [25]. While this strategy provides some robustness, it is often suboptimal, as a sufficiently diverse batch of samples is needed in order to capture enough information to adapt the weights of the network. Another strategy, inspired by self-supervised learning, is to include additional tasks [22, 15]. Although this strategy has been used successfully for TTA, it is sensitive to the chosen proxy task and requires pseudo-labels.

*Equal contribution

In this paper, we present TTTFlow, a novel approach for unsupervised Test-Time Adaptation, which makes use of a Normalizing Flow as a domain shift detector. Our method does not require access to the source data during inference, can measure domain discrepancy in a tractable way, and does not require special proxy tasks to be solved along with the source training. Moreover, TTTFlow could be built on top of any off-the-shelf network without additional technical adjustments.

Specifically, the contributions of this work can be summarized as follows:

- We introduce an unsupervised method under the test-time training paradigm of TTA. Our approach is designed to directly measure the domain shift between target and source images, without the need of an extra task.

- To the best of our knowledge, this is the first work that employs Normalizing Flows to measure domain shift in Test-Time Adaptation. While they have been recently investigated for domain alignment, their application in tasks related to Domain Adaptation remains unexplored.

The remainder of this paper is organized as follows. Section 2 presents prior work on Test-Time Adaptation. Section 3 then introduces the proposed TTTFlow method. Section 4 describes the experimental setup used to evaluate our method, and Section 5 reports the results.

2. Related Work

Normalizing Flows In the field of generative modeling, Normalizing Flows have gained important traction due to their capabilities of learning tractable distributions in the latent space [11, 18]. Their goal is to transform a generally unknown data distribution into a known one, typically the normal distribution, from which we can easily sample new data points and measure their exact likelihood. The transformation of the flow model is guaranteed to be bidirectional by using an invertible and differentiable architecture. Although Normalizing Flows have not been formally used in the field of Domain Adaptation, recent works suggest that they can be an effective tool for Domain Alignment [6, 23], where the domain of two different datasets must be fitted with indistinguishable distributions. In this work, we take a step further by proposing Normalizing Flows as an alternative to learn and codify a domain, so that it can later be used at test-time.

Test-Time Adaptation These methods allow using off-the-shelf models without any additional training. In general terms, test-time adaptation focuses on adapting models that were not trained with a special configuration prior to being used at inference. One of the first approaches of this category, called TENT [25], requires to be given the model and target data. It then updates the model layers containing normalization statistics by minimizing the Shannon entropy of predictions. The authors of [16] improve TENT by using a log-likelihood ratio instead of entropy, and by estimating the statistics of the target batch. In SHOT [14], the entire feature extractor is fine-tuned using a mutual information loss along with pseudo-labels to correct inaccurate predictions from the pretrained model. LAME [2] is an adaptation method that do not alter the network layers, but just focuses on a post-hoc adaptation of the softmax predictions through Laplacian regularization.

Batch Norm Adaptation To increase robustness of a model, Batch Normalization (BN) can be used for a faster convergence and increased stability during training. Nevertheless, a shift in the distribution causes the statistics to change, which is why some papers suggest adapting normalization statistics to improve performance. For instance, Prediction Time Batch Normalization [17] proposes to use the mean and variance from the batch of test samples as statistics in the batch norm layer. However, this estimation can be inaccurate due to a small number of data samples. To avoid this, the authors of [21] compute a new mean and variance, which is a mix of the BN statistics computed at training and the new estimation at test time. The same approach is used by SITA [9] to estimate statistics, with the difference that it can be used on a single data example. To achieve this, SITA generates a pseudo-batch by randomly augmenting this example and then computes the statistics on this pseudo-batch.

Test-Time Training Methods based on Test-Time Training (TTT) [22] update the model at inference, but use a Y-shaped architecture with a main task and a self-supervised task which are learned at training time. The model is trained by jointly minimizing the losses in both branches. After the model has been trained, the parameters of the main task branch are frozen. At test-time, the parameters of the shared encoder are updated so to minimize the self-supervised loss. Following this approach, [22] uses rotation prediction [5] as self-supervised task, where images are randomly rotated by multiples of 90° (0°, 90°, 180°, 270°) and the model should recover this rotation. A major problem of this approach is the choice of the self-supervised loss, which should be related to both source and target datasets. Inspired by TTT, TTT++ [15] adds a loss which promotes online feature alignment by comparing the statistics of the source data with those of the current batch. For the self-supervised task, the rotation prediction loss is replaced by a contrastive loss which encourages the encoded features for two different augmentations of the same image to be similar, and the ones of different images to be dissimilar. Lastly, the authors of MT3 [1] use meta training at inference on the

second task to improve the performance of TTT.

3. Method

We start by defining the problem of Test-Time Training and then present our TTTFlow method for this problem.

3.1. Problem definition

In the context of classification, we denote the domain as the joint distribution P_{XY} between the input space $\mathcal{X}$ and the label space $\mathcal{Y}$, and define the marginal distribution of the inputs as P_X. At training time, a deep network learns from the data out of a source domain $(\mathcal{X}_s, \mathcal{Y}_s)$, whilst at test-time, the network must be adapted to a new target domain $(\mathcal{X}_t, \mathcal{Y}_t)$, such that $P_{X_s} \neq P_{X_t}$. Both domains share the same label space ($\mathcal{Y}_s = \mathcal{Y}_t$), but the labels for the target inputs are unknown. The goal of Test-Time Training is to learn a function $g_f : \mathcal{X}_t \rightarrow \mathcal{Y}_t$ on the basis of an already known function $f : \mathcal{X}_s \rightarrow \mathcal{Y}_s$.

3.2. Proposed framework

Our framework exploits a multi-head architecture where a Normalizing Flow is used to encode domain-specific information from a pretrained feature extractor. An important benefit of this configuration is that it can be applied on any pretrained network without the need of a special training on the source data. In what follows, we describe the different components of TTTFlow and the mechanisms that allow test-time training with domain shift. The overall scheme of the model can be seen in Fig. 1.

3.3. Source Training

Although our TTFlow method can be used on top of any architecture, in this work we consider a CNN as the classification backbone. This CNN can be divided in a feature extractor $f_{\boldsymbol{\theta}}$ (parameterized by $\boldsymbol{\theta}$) followed by a classifier head $h_{\boldsymbol{\varphi}}$ (parameterized by $\boldsymbol{\varphi}$). Let $\mathbf{x} = f_{\boldsymbol{\theta}}(\mathbf{I}) \in \mathbb{R}^{c \times h \times w}$ be the 2D feature map from an input image $\mathbf{I}$, and $\hat{\mathbf{y}} = h_{\boldsymbol{\varphi}}(\mathbf{x}) \in [0,1]^K$ be the softmax predictions from the classifier, where K is the number of classes. The source training of the network is performed in a supervised way using the cross-entropy loss. However, and as shown in further sections, using a more robust source training (e.g., adding contrastive learning) can help to achieve better adaptation results.

3.4. Normalizing flows as a domain shift detector

Once the CNN is trained, we need a way to encode the source domain distribution, such that domain shifts can be detected at test time. We propose using Normalizing Flows [11, 4, 10] for this purpose, because of their ability to model complex, high-dimensional distributions effectively. Normalizing Flows are generative models capable of transforming data from a complex and often unknown distribution into a latent space with a well-defined and tractable distribution. In this study, the feature map $\mathbf{x}$ follows an unknown distribution $P(\mathbf{x})$, which is related to P_X. A function $g_{\boldsymbol{\phi}}$ transforms the feature map into its latent representation $\mathbf{z} = g_{\boldsymbol{\phi}}(\mathbf{x}) \in \mathbb{R}^{c \times h \times w}$ with $\mathbf{z} \sim P_{\boldsymbol{\phi}}(\mathbf{z})$, such that $P_{\boldsymbol{\phi}}(\mathbf{z})$ is a tractable distribution (e.g. standard multivariate Gaussian distribution) with a known probability density function. The flow-based function $g_{\boldsymbol{\phi}}$ should meet two requirements: (1) being invertible, i.e. $\mathbf{x} = g_{\boldsymbol{\phi}}^{-1}(\mathbf{z})$, and (2) being differentiable w.r.t. the input in both directions. Furthermore, a higher representation power can be achieved if a composition of invertible and differentiable functions is used: $g_{\boldsymbol{\phi}} = g_1 \circ g_2 \circ \cdots g_M$. Knowing that $\mathbf{x}$ is transformed into $\mathbf{z} \sim P_{\boldsymbol{\phi}}$, the likelihood of the original variable can then be computed exactly using the change of variable rule,

$$\begin{aligned} \log P(\mathbf{x}) &= \log P_{\boldsymbol{\phi}}(\mathbf{z}) + \log \left| \det \left(\frac{d\mathbf{z}}{d\mathbf{x}} \right) \right| \\ &= \log P_{\boldsymbol{\phi}}(\mathbf{z}) + \sum_{i=1}^{M} \log \left| \det \left(\frac{dg_i}{dg_{i-1}} \right) \right|, \end{aligned} \quad (1)$$

where $\log |\det(dg_i/dg_{i-1})|$ is the logarithm of the Jacobian matrix determinant.

Affine coupling layers are a popular choice to build Normalizing Flows [4, 10], so that the resulting Jacobian matrix is upper triangular and its determinant is easily computed as the product of its diagonal elements. The model can be trained by minimizing the negative log-likelihood in Eq. (2):

$$\mathcal{L}_{\text{uns}} = -\log P(\mathbf{x}). \quad (2)$$

A Normalizing Flow based on RealNVP [4] is placed on top of the frozen feature extractor $f_{\boldsymbol{\theta}}$ to learn the latent space of $\mathbf{z}$ from the source inputs $\mathbf{x} \sim \mathcal{X}_s$ in an unsupervised way (i.e., using Eq. (1) directly). We hypothesize that this model captures the domain information from the source data, thus can be used to measure domain shift in the target data.

3.5. Test-time training with flow-based model

At test-time, the pretrained network must adapt its parameters to unlabeled inputs from an also unknown target domain X_t. We achieve this by only focusing on the extractor parameters $\boldsymbol{\theta}$, similarly to [22, 15]. The frozen Normalizing Flow transforms each new test image feature map $\mathbf{x}_t$ into its latent representation $\mathbf{z}_t = g_{\phi}(\mathbf{x}_t)$ to compute its log-likelihood using Eq. (2). Note again that the log-likelihood is measured with respect to the multivariate Gaussian distribution, into which the unsupervised head transforms the features' distribution. This value provides information of the domain shift, as a feature map that is closer to the latent space of the source data should have a higher log-likelihood than a feature map that is farther away. Hence, negative

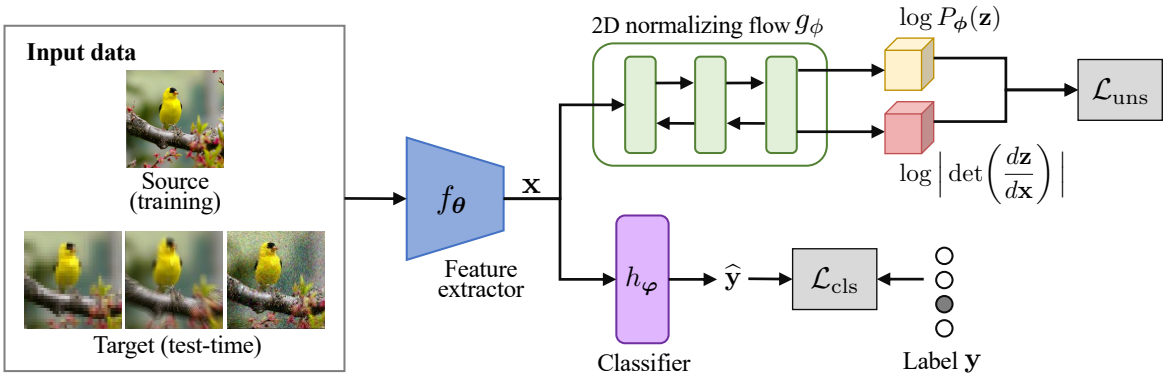


Figure 1. Architecture of TTTFlow. The feature maps from a pretrained network are transformed into an isotropic Gaussian distribution through a Normalizing Flow g_{ϕ}. The log-likelihood of an input $\mathbf{x}$ is measured based on the likelihood in the latent space $\mathbf{z} = g_{\phi}(\mathbf{x})$. At test-time, the frozen unsupervised head is then used as a domain shift detector to fine-tune the extractor. Target images show examples of pixelate, zoom blur and Gaussian noise corruptions (from left to right) from the CIFAR-10-C dataset.

log-likelihood can be once again used as the loss function to adapt the extractor for the target input.

4. Experimental setup

We evaluate our TTTFlow method on two popular test-time adaptation benchmarks based on the CIFAR-10 dataset [12], CIFAR-10-C [8] and CIFAR-10.1 [20], and compare its performance against state-of-art approaches for this task. As explained in Section 3.2, the first step is to train a CNN on source data from CIFAR-10, a natural image classification dataset consisting of 10,000 images for training and 2,000 images for testing, with 10 different classes.

Once the CNN is trained, the training of the Normalizing Flow (NF) is performed as explained in Section 3.4. We adopted RealNVP [4] as it has become a standard tool for flow modeling. Our compact version is made of three coupling layers with two resblocks in each of them. The *checkerboard* coupling was found to be more effective than its channelwise counterpart. Similar to [15], the flow model is placed on top of the second layer of the ResNet50's feature extractor based on the common assumption that domain information is mostly located at the early stages of feature extraction while class information is encoded at later stages [28]. More implementation details can be found in the supplementary material, as well as the corresponding ablation study on the flow architecture.

We use CIFAR-10 images without any label information for this step, as the NF model is trained in an unsupervised manner. The training was performed for 100 epochs using SGD with an initial learning rate of 0.1 and a cosine annealing scheduler.

Following previous work, ResNet50 [7] is chosen as the main architecture. The model is trained for 350 epochs with SGD, using a batch size of 128 images and an initial learning rate of 0.1 which is reduced by a factor of 10 at epochs 150 and 250.

Method		**Accuracy (%)**
TTT [22]	Separate	61.18
	JT as in [22]	57.96
TTTFlow	Separate	62.75
	JT ($\beta = 0.01$)	58.16
	JT ($\beta = 0.001$)	58.44

Table 1. Comparison of joint versus separate training for TTT and TTTFlow on CIFAR-10-C data with Level 5 Gaussian Noise Corruption.

After the two-step source training, the NF model is used to detect domain shift through negative log-likelihood and update the part of the network from where the features are collected (i.e., up until second layer). For all the experiments at test-time, we keep the batch size of 128 images and use a learning rate of 0.001, along with SGD as the main optimizer. At each new batch, we initialize our feature extractor with the weights of the learning part. This is to avoid computing on an error made by the optimization, and is based on the assumption that each batch can have different corruptions as made by [22] in their offline mode.

We use the pretrained CNN as baseline, and compare our results against TENT [25], TTT [22], and TTT++ [15]. For a fair comparison, we reproduced these previous methods under the same experimental conditions as in TTTFlow, i.e. using the same hyperparameters such as batch size, number of adaptation epochs, and so on. Our codebase can be found in `https://github.com/GustavoVargasHakim/TTTFlow.git`.

5. Results and discussion

We first perform ablation and comparison experiments on the CIFAR-10-C dataset containing different types of image corruption, and then extend our evaluation to natural domain shift using the CIFAR-10.1 dataset.

5.1. Object recognition on corrupted images

Our first experiments evaluate TTTFlow on the CIFAR-10-C dataset which comprises 15 different algorithmic corruptions (e.g. Gaussian noise, zoom blurring, etc.) with 10,000 images each (see Fig. 1 for examples). Each corruption has five severity levels, with Level 1 corresponding to mild corruptions and Level 5 to strongest ones. Unless specified otherwise, we evaluate TTTFlow on Level 5, as it represents the most challenging adaptation scenario.

Joint vs separate training As a first step, we compare our method, which learns the NF on top of a frozen classifier (separate training), with the Joint Training (JT) approach training the classification task and unsupervised task at the same time by minimizing

$$\mathcal{L}_{\text{JT}} = \mathcal{L}_{\text{cls}} + \beta \mathcal{L}_{\text{uns}}, \tag{3}$$

where hyperparameter β controls the trade-off between the two losses. This JT approach follows previous work on test-time training [22, 15]. An important problem with this approach is the need to retrain the main classification network when learning occurs along with the secondary task. To avoid this issue and to exploit the weights of any pretrained backbone, we freeze all the parameters of the CNN, except the batch norm statistics of the feature extractor, and proceed to train the NF independently. This enables using any backbone without retraining.

Table 1 gives the accuracy of our method using separate training or JT with $\beta = 0.01$ or $\beta = 0.001$. As can be seen, placing the NF model on a pretrained encoder yields better performance than performing joint training. We conjecture that the Normalizing Flow is particularly sensitive to the joint training, as it is forced to learn a Gaussian distribution out of the continually changing feature maps distribution, as they are being modified for classification. Moreover, the information needed to encode the domain of examples seems different from the information needed to classify them. The same analysis is performed for TTT [22], which has a rotation prediction semi-supervised loss in addition to the classification loss. As reported in Table 1, we find once again that training the TTT model in two separate steps is better than the JT approach in [22]. For remaining experiments, we therefore use the separate training strategy for TTTFlow and TTT.

Number of adaptation iterations We compare the accuracy of TTTFlow for different iterations of adaptation at test-time. As we can see in Fig. 2 and 3, our model's accuracy typically increases monotonically with a greater number of iterations. Furthermore, in most cases, a maximum accuracy is reached after about 20 iterations. Beyond this point, performing more iterations increases runtime without any significant gain in performance. For some corruptions like Snow, which severely degrade the image, we find

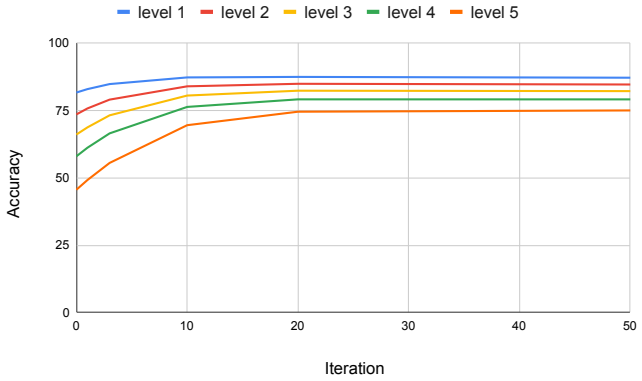


Figure 2. Evolution of accuracy over iterations of the average of each Level

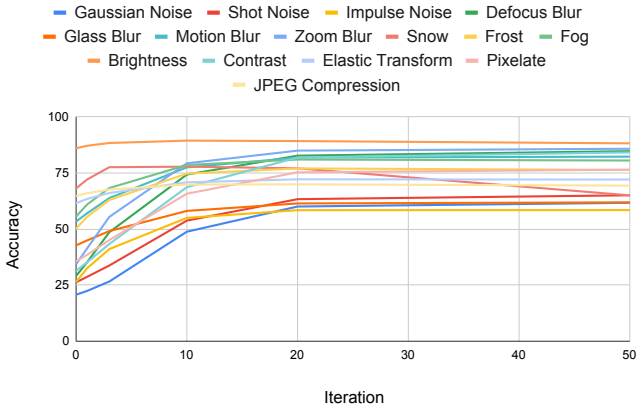


Figure 3. Evolution of accuracy for every corruption on Level 5.

that performance actually drops when doing more adaptation iterations. When testing other approaches, we do not observe the same stability of performance with respect to the number of iterations. To have a fair comparison, for all methods, we thus compute the accuracy for 1, 3, 10, 20 and 50 iterations and report the maximum accuracy.

Comparison to methods using a classifier trained with only $\mathcal{L}_{\text{cls}}$ As shown in Table 2, TTTFlow achieves an average accuracy improvement of 14.54% with respect to the pretrained ResNet50 Baseline. Significant improvements in accuracy are obtained for all corruption types except JPEG compression and Elastic transform. Moreover, TENT yielded a very low accuracy for all corruption types, due to collapsing predictions. Compared to TTT, using the same classifier trained with only $\mathcal{L}_{\text{cls}}$, our method obtains an improvement in average accuracy of 0.84%. These results support our hypothesis that the NF can be used to measure domain shifts and improve the extractor accordingly in an unsupervised manner. As said in the article and in addition with these results, it confirms our hypothesis that the NF can be used to measure domain shifts and improve the extractor accordingly in an unsupervised manner.

Comparison with TTT++ on baseline trained with $\mathcal{L}_{\text{cls}}$

and $\mathcal{L}_{\text{ssl}}$ As discussed earlier, the unsupervised head of TTTFlow can be placed on top of any pretrained feature extractor. In TTT++ [15], the pre-training of the CNN is based on a similar Y-shaped architecture, where the secondary task is a self-supervised classification task using the contrastive loss. The joint learning process is then performed using Eq. 4 as the final loss function:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda \mathcal{L}_{\text{ssl}} \tag{4}$$

where λ is a hyperparameter. Using contrastive learning as an auxiliary task yields to a more robust network, and in consequence, a stronger feature extractor. For this reason, we propose to train the NF model using the TTT++ pre-training. As seen in Table 2, our method outperforms TTT++ in all but two corruption types (Impulse noise and Fog) and gives an average accuracy 2.10% higher than this state-of-art approach.

Visualization of adaptation To visualize the result of our adaptation method, we show in Figure 4 the t-SNE plots of features at the end of extractor, before and after adaptation. As can be seen, TTTFlow allows each sample to be separated for better interpretation and prediction. However, a collapse of feature vectors to the same point in space is visible in the top right of Figure 4 (b) and (d). Since our NF-based method pushes representations toward the mode of the distribution, this could be a side effect of making too many iterations for adaptation. Nevertheless, in our experiments, accuracy generally remains stable and may even increase after performing many adaptation iterations.

5.2. Object recognition on natural domain shift

We also evaluate TTTFlow when natural domain shift is present. For this purpose, the CIFAR-10.1 dataset [20] is used as the second benchmark. CIFAR-10.1 consists of 2,000 images sampled from the original CIFAR-10 set with the objective of maximizing domain shift with respect to the source data. TTTFlow is once again compared with previous methods, and the standard pretrained CNN is used as baseline. Results are also compared with previous methods.

As reported in Table 3, TTT++ achieves a better performance than TTTFlow in this case, with a 1.75% improvement in accuracy. However, when looking at the accuracy for the different adaptation iteration, we find that the better performance of TTT++ only occurs for the first few iterations. Compared to our method, which remains stable, TTT++'s accuracy degrades beyond 3 iterations. The superior performance of TTT++ for this CIFAR10.1 could be explained by the nature of this dataset, in which the distribution shift is smaller compared to the corruptions found in CIFAR-10-C. The distribution shift in this dataset, which is more related to semantic content, may not be fully captured by our NF model applied on the second ResNet50 layer. As shown in the t-SNE plots of Figure 5, the features obtained by our model at the end of the extractor are very similar for CIFAR-10 and CIFAR-10.1, which supports that the adaptation over the iterations (Table 4) for TTTFlow is stable.

6. Conclusion

This work follows the line of former research on test-time training, which develop techniques to adapt models at test-time when distribution shifts are prevalent. To tackle some limitations of previous works, we proposed using Normalizing Flows as domain shift detector that can be plugged into the feature extractor of any pretrained architecture, and that can be trained in an unsupervised manner under maximum likelihood.

Our method, TTTFlow, provided of substantial accuracy gains to the source model, also in comparison with the *state-of-the-art* methods in test-time adaptation. Besides the practical advantage of being compatible without any model and not requiring a special joint training, it has been shown that TTTFlow can also enhance the performance of strongly trained source models, such as the one of a similar work, TTT++.

Future work includes tackling the perceived limitations of TTTFlow, which include: (a) sensitivity to the Normalizing Flow architecture, where a lower representation power could yield underfitting to the domain information, and a higher one could lead to a class collapse. (b) Depending on the type of domain shift, different layers in the encoder can be more useful to capture domain specific information, for which further studies on the effects of the chosen shared stage of the extractor are encouraged. (c) So far, TTTFlow depends on the use of batches, whilst adapting for a single sample is highly desirable. Devising a criterion to select which samples the model should adapt to would have an important impact both in performance and computational costs.

	Encoder trained with $\mathcal{L}_{\text{cls}}$ only				Encoder trained with $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{ssl}}$	
	Baseline	TENT [25]	TTT [22]	TTTFlow	TTT++ [15]	TTTFlow
Gaussian Noise	53.25	46.65 ±0.12	61.29 ±0.07	**61.73 ±0.35**	75.87 ±5.05	**79.58 ±0.09**
Shot Noise	57.71	46.31 ±0.25	64.37 ±0.10	**65.08 ±0.14**	77.18 ±1.36	**80.20 ±0.03**
Impulse Noise	43.79	37.95 ±0.15	**58.97 ±0.20**	58.48 ±0.12	**70.47 ±2.18**	67.30 ±0.08
Defocus Blur	51.80	59.77 ±0.29	83.80 ±0.11	**84.75 ±0.17**	86.02 ±1.35	**90.96 ±0.06**
Glass Blur	54.69	41.24 ±0.18	61.23 ±0.29	**61.93 ±0.12**	69.98 ±1.62	**71.54 ±0.09**
Motion Blur	64.97	56.40 ±0.33	76.86 ±0.13	**82.31 ±0.10**	85.93 ±0.24	**85.95 ±0.07**
Zoom Blur	61.62	59.23 ±0.35	84.67 ±0.08	**85.82 ±0.17**	88.88 ±0.95	**91.90 ±0.05**
Snow	74.12	55.93 ±0.21	75.63 ±0.1	**77.84 ±0.19**	82.24 ±1.69	**84.28 ±0.12**
Frost	67.98	46.44 ±0.20	**77.17 ±0.17**	77.05 ±0.10	82.74 ±1.63	**85.88 ±0.05**
Fog	63.67	52.70 ±0.20	**81.15 ±0.12**	81.02 ±0.25	**84.16 ±0.28**	74.02 ±0.05
Brightness	87.16	66.34 ±0.18	88.84 ±0.09	**89.45 ±0.17**	89.07 ±1.20	**92.38 ±0.01**
Contrast	22.89	49.03 ±0.45	**84.79 ±0.12**	84.20 ±0.18	86.60 ±1.39	**92.20 ±0.10**
Elastic Transform	**76.96**	50.27 ±0.36	72.45 ±0.09	72.20 ±0.24	78.46 ±1.83	**80.47 ±0.08**
Pixelate	48.22	52.52 ±0.25	74.71 ±0.09	**76.50 ±0.13**	82.53 ±2.01	**88.84 ±0.05**
Jpeg Compression	**81.42**	56.78 ±0.30	69.75 ±0.24	69.95 ±0.11	81.76 ±1.58	**87.95 ±0.03**
Average	60.68	51.84	74.38	**75.22**	81.46	**83.56**

Table 2. Accuracy (%) on CIFAR-10-C dataset with Level 5 corruption for TTTFlow compared to ResNet50, TENT, TTT, and TTT++ with different encoders . Mean and standard deviation are reported over 5 runs

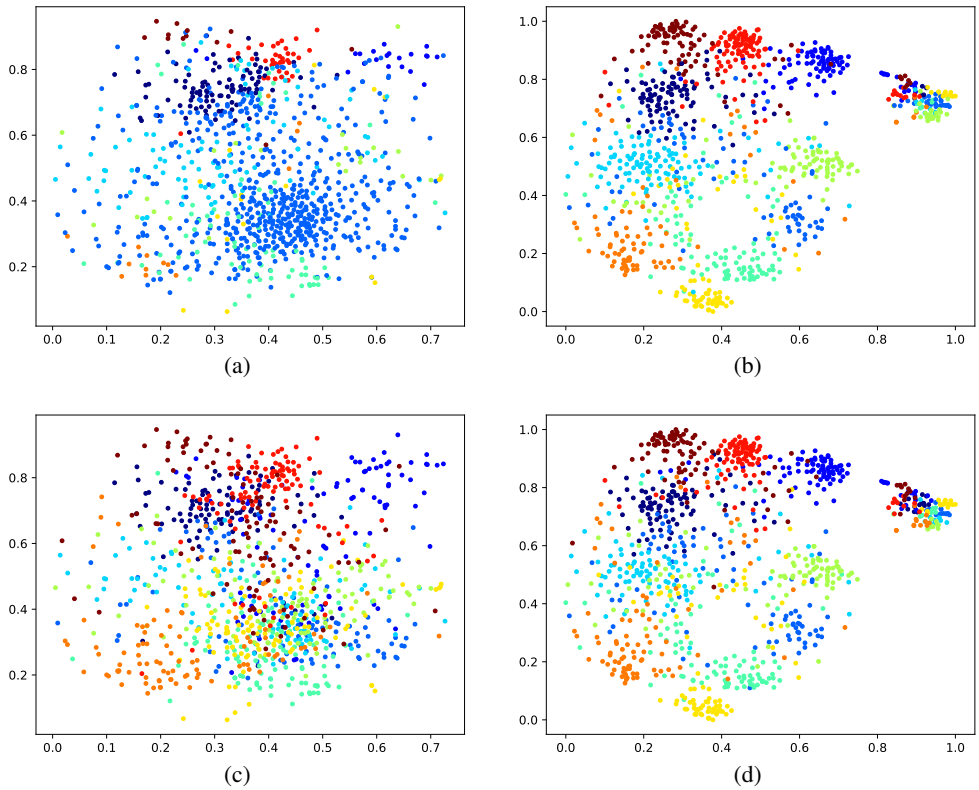
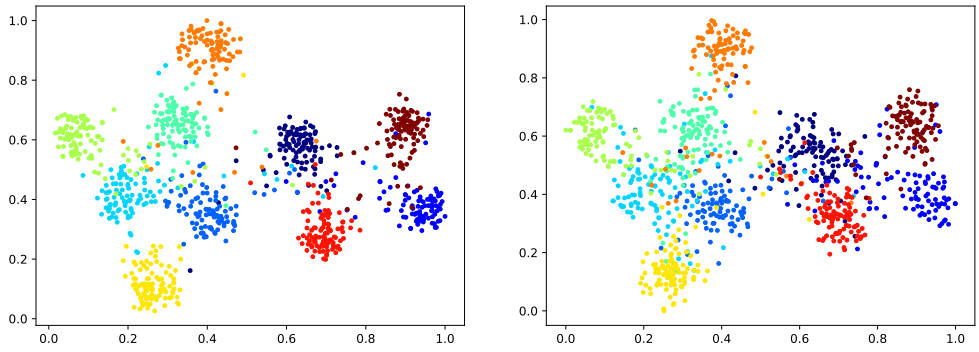


Figure 4. t-SNE plots on defocus blur for the features at the output of the extractor from TTTFlow. (a) is the prediction of the model without adaptation. (b) is the prediction of the model after 50 iterations. (c) is the ground truth of the model without adaptation. (d) is the ground truth of the model after 50 iterations.



(a) (b)

Figure 5. t-SNE plots for the features at the output of the extractor from TTTFlow. Comparison between CIFAR-10 and CIFAR-10.1. (a) is the ground truth of the model without adaptation for CIFAR-10. (b) is the ground truth of the model without adaptation for CIFAR-10.1.

Method	**Accuracy (%)**
Baseline	84.70
TENT [25]	58.98 $\pm$0.12
TTT [22]	84.49 $\pm$0.15
TTTFlow ($\mathcal{L}_{\text{cls}}$)	**85.11** $\pm$**0.30**
TTT++ [15]	**88.24** $\pm$**0.17**
TTTFlow ($\mathcal{L}_{\text{cls}}$ + $\mathcal{L}_{\text{ssl}}$)	86.49 $\pm$0.02

Table 3. Accuracy of compared methods on the CIFAR-10.1 dataset containing natural domain shift.

Iterations	**Accuracy (%)**	
	TTT++ [15]	TTTFlow ($\mathcal{L}_{\text{cls}}$ + $\mathcal{L}_{\text{ssl}}$)
1	88.19 $\pm$0.09	86.49 $\pm$0.02
3	88.24 $\pm$0.17	86.41 $\pm$0.13
10	86.49 $\pm$0.22	86.36 $\pm$0.07
20	85.03 $\pm$0.99	86.29 $\pm$0.08
50	80.43 $\pm$0.34	86.48 $\pm$0.07

Table 4. Comparison of accuracy over iterations for TTT++ [15] and TTTFlow on th CIFAR-10.1 dataset.

References

[1] Alexander Bartler, Andre Bühler, Felix Wiewel, Mario Döbler, and Bin Yang. MT3: Meta test-time training for self-supervised test-time adaption. *CoRR*, abs/2103.16201, 2021.

[2] Malik Boudiaf, Romain Mueller, Ismail Ben Ayed, and Luca Bertinetto. Parameter-free online test-time adaptation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8344–8353, 2022.

[3] Boris Chidlovskii, Stephane Clinchant, and Gabriela Csurka. Domain adaptation in the absence of source domain data. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 451–460, 2016.

[4] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[5] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *CoRR*, abs/1803.07728, 2018.

[6] Aditya Grover, Christopher Chute, Rui Shu, Zhangjie Cao, and Stefano Ermon. Alignflow: Cycle consistent learning from multiple domains via normalizing flows. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4028–4035, 2020.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

[9] Ansh Khurana, Sujoy Paul, Piyush Rai, Soma Biswas, and Gaurav Aggarwal. SITA: single image test-time adaptation. *CoRR*, abs/2112.02355, 2021.

[10] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[11] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, 2021.

[12] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[13] Yanghao Li, Naiyan Wang, Jianping Shi, Xiaodi Hou, and Jiaying Liu. Adaptive batch normalization for practical domain adaptation. *Pattern Recognition*, 80:109–117, 2018.

[14] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? Source hypothesis transfer for unsupervised domain adaptation. *arXiv:2002.08546 [cs]*, June 2021. arXiv: 2002.08546.

[15] 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? *Neural Information Processing Systems (NeurIPS)*, 2021.

[16] Chaithanya Kumar Mummadi, Robin Hutmacher, Kilian Rambach, Evgeny Levinkov, Thomas Brox, and Jan Hendrik Metzen. Test-time adaptation to distribution shift by confidence maximization and input transformation, 2021.

[17] Zachary Nado, Shreyas Padhy, D. Sculley, Alexander D'Amour, Balaji Lakshminarayanan, and Jasper Snoek. Evaluating prediction-time batch normalization for robustness under covariate shift. *arXiv:2006.10963 [cs, stat]*, Jan. 2021. arXiv: 2006.10963.

[18] George Papamakarios, Eric T Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *J. Mach. Learn. Res.*, 22(57):1–64, 2021.

[19] Aayush Prakash, Shaad Boochoon, Mark Brophy, David Acuna, Eric Cameracci, Gavriel State, Omer Shapira, and Stan Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7249–7255. IEEE, 2019.

[20] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do CIFAR-10 classifiers generalize to cifar-10? *CoRR*, abs/1806.00451, 2018.

[21] Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving robustness against common corruptions by covariate shift adaptation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11539–11551. Curran Associates, Inc., 2020.

[22] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning (ICML)*, 2020.

[23] Ben Usman, Avneesh Sud, Nick Dufour, and Kate Saenko. Log-likelihood ratio minimizing flows: Towards robust and quantifiable neural distribution alignment. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21118–21129. Curran Associates, Inc., 2020.

[24] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. *Advances in neural information processing systems*, 31, 2018.

[25] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: fully test-time adaptation by entropy minimization. *arXiv:2006.10726 [cs, stat]*, Mar. 2021. arXiv: 2006.10726.

[26] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and Philip Yu. Generalizing to unseen domains: A survey on domain generalization. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[27] Garrett Wilson and Diane J Cook. A survey of unsupervised deep domain adaptation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(5):1–46, 2020.

[28] Kaiyang Zhou, Yongxin Yang, Yu Qiao, and Tao Xiang. Domain generalization with mixstyle. In *International Conference on Learning Representations*, 2020.