Mixup for Test-Time Training

Bochao Zhang
bochao@comp.hkbu.edu.hk

Rui Shao
rui.shao@ntu.edu.sg

Jingda Du
jingda@comp.hkbu.edu.hk

PC Yuen
pcyuen@comp.hkbu.edu.hk

1 Hong Kong Baptist University
Hong Kong SAR, China

2 Nanyang Technological University
Singapore

Abstract

Test-time training provides a new approach solving the problem of domain shift. In its framework, a test-time training phase is inserted between training phase and test phase. During test-time training phase, usually parts of the model are updated with test sample(s). Then the updated model will be used in the test phase. However, utilizing test samples for test-time training has some limitations. Firstly, it will lead to overfitting to the test-time procedure thus hurt the performance on the main task. Besides, updating part of the model without changing other parts will induce a mismatch problem. Thus it is hard to perform better on the main task. To relieve above problems, we propose to use mixup in test-time training (MixTTT) which controls the change of model’s parameters as well as completing the test-time procedure. We theoretically show its contribution in alleviating the mismatch problem of updated part and static part for the main task as a specific regularization effect for test-time training. MixTTT can be used as an add-on module in general test-time training based methods to further improve their performance. Experimental results show the effectiveness of our method.

1 Introduction

Usually in deep learning based methods, training phase and test phase are strictly separated. E.g. a normal dataset will be divided into three parts: training set, validation set and test set. Models are trained with training set data, while hyperparameters of the model are chosen with respect to the performance on validation set. Then the well trained model is evaluated on the test set. However, the demand of a good model is not restricted to good performance on the test set from the same dataset but also from other datasets. Namely, under distribution shifts whether the model still enjoys a high generalization ability and keeps its high performance. This problem is specifically studied in many fields with different settings like domain adaptation, domain generalization, adversarial learning etc. Test-time training opens a new learning paradigm and provides a new strategy to counter this problem.

Test-time training inserts a test-time training phase between the normal training phase and test phase. The model would be well trained in the training phase. Then when a test
sample(s) comes for inference, it carries in itself information that could be utilized in test-time training phase. Such information includes domain information and visual information etc. Usually test-time training phase will fine-tune the model or important statistics (e.g. prototypes) as completing the auxiliary unsupervised task. Finally inference on the main task will be performed with the updated model (statistics) in test phase. From the setting view, test-time training obeys the rule of no access to test data during training. It coincides with domain generalization. From the view of test-time training process, whole or part of the model is updated to adapt to the test sample. In this way it coincides with domain adaptation. So test-time training basically puts very little requirement for the data and model, yet can counter certain degree of domain shift in its process. The early network structure of test-time training from [12] is a multitask training framework. One main task and one self-supervised auxiliary task share a feature extractor and each task has an independent classifier, which is called head. Self-supervised task rotation [1] is chosen as auxiliary task for the test-time procedure. Simple multi-task training is used for the first phase. In test-time training phase of [12], a test sample(s) will be rotated to perform the auxiliary task and update the feature extractor. In test phase inference will be performed on the updated feature extractor and the original classifier. This network structure is considered as one of the most classical network structures for test-time training methods. Therefore, we conduct our theoretical analysis about MixTTT and ordinary TTT based on this framework.

Test-time training aims that optimizing auxiliary task will immediately improve the main task through the updated shared model. This could be realized with two approaches. The first one relies that auxiliary task can minimize the domain discrepancy between the training set and test samples. Naturally when performing the main task, more accurate results could be obtained. Most test-time training methods follow this approach. However in such cases, a batch of test samples are demanded. The second approach depends on the cooperation of the main task and auxiliary task on certain datasets. It is expected that auxiliary task can help dig the inherent properties of the test sample. E.g. by performing the auxiliary task, visual information of test sample could be better extracted. Under such situation, optimizing auxiliary task will immediately optimize the main task. This approach normally does not restrict the number of test samples. However it requires more delicate test-time update process to keep the good relation of two tasks, especially on unseen samples. As we mentioned above, uncontrolled optimization on the auxiliary task will cause overfitting. Besides, with much change of some parts of the model the static part for the main task will not work well on top of it.

In this paper we propose to utilize mixup between training data and the test sample(s) to mitigate the above problems in test-time procedure. Our method can be applied on both the first approach setting and the second approach setting without specific requirement about the number of test sample(s). As a result, our add-on module allows test-time training to improve the main task with the full strength. In summary, our contribution is three-folded:

- We identify an important problem in test-time training, which is model mismatch between the updated part and static part when accomplishing the auxiliary task.
- We show from theoretical analysis that the effect using mixup in test-time training brings implicit control in model change beyond original test-time training.
- MixTTT can be seen as an add-on module without specific requirement about the number of test samples which can further boost the performance of existing test-time training related methods.
2 Related Work

2.1 Test-time training

Test-time training [12] opens a new learning paradigm to solve domain shift problem. In its belief, when the test sample comes for inference, it is a waste to not explore it. The core idea is to utilize the test sample information to optimize some parts of the model with the unsupervised auxiliary task and then evaluate the test sample with the updated model for the main task. Following its idea, appears test-time adaptation, for which usually many test samples are required to perform adaptation. Under the setting of test-time adaptation, many domain adaptation related techniques could be borrowed and utilized. Namely in such methods the aim of auxiliary task is to minimize the discrepancy between the source and target domains. [13] estimates normalization statistics and updates affine transformations to reduce entropy in the auxiliary task. [8] performs contrastive learning on the test sample batch with its augmented versions and on the other hand aims to align the mean and variance of features from the test sample batch with the training set. For now most test-time methods belong to test-time adaptation, there is rather limited paper solving single-test-sample based test-time training problem. Our method can fit in with both multiple-test-sample-based and single-test-sample-based test-time procedure.

Test-time training itself could also be used as an add-on module in domain generalization methods to improve the performance. [4] does not literally update the model during test, instead it chooses to update prototypes in a memory bank. At the end, the test sample is classified based on the distance of its feature vector with adjusted prototype representations. [13] does not explicitly mention test-time training, but they share similar idea in utilizing the test sample to do optimization for better inference. It aims to project the target sample to the source domain manifold through an inference-time procedure thus get more accurate inference outcome.

Approaches from other fields like medical semantic segmentation [3, 5] and face anti-spoofing [11, 15] also utilize the idea of test-time training together with specific domain knowledge to solve the domain shift problem.

2.2 Mixup

The core idea of mixup [20] is that convex combination of sample pair and their labels forms a general vicinal distribution. Previous experience shows that samples drawn from vicinal distribution increase the amount of training samples and relieve overfitting. [13] claims that manifold mixup as a regularization method gives smoother decision boundary and better regularization during training stage. More mixup-based augmentation methods gradually appear as well e.g. [19] further improves the performance on localization. [6] gives a more effective strategy on how to cut and mix sample pairs.

Mixup as a data processing method is also used to solve domain generalization and domain adaptation problems. [16] separates two kinds of mixup: mixing samples from two different domains and mixing samples from all domains. The second one shows good performance in visual decathlon benchmark [10]. [17, 18] both utilize the concepts of mixup and adversarial training for domain adaptation but in different ways. [18] chooses to mix source domain data with the target domain data, thus filling the gap with mixed samples between two domains. [17] instead mix up samples within the source domain and the target domain.
There also exist papers proving efficacy of mixup through theoretical analysis. e.g. [21] expands the mixup loss through Taylor expansion and boils it down to the standard loss plus some regularization terms. It explicitly shows mixup improves robustness with respect to certain adversarial attacks and exhibits data-adaptive regularization effect for generalization. Our method, however, explains the regularization effect of mixup during test-time training from the perspective of stabilizing the feature extractor thus maintaining the good cooperation of it with classifier.

3 Test-Time Training under Model Mismatch

In this section, we will first describe the rise of model mismatch and its influence in test-time training then give a strategy solving this problem. We explore the mixup based test-time training loss (MixTTT loss) and ordinary test-time training loss. Through Taylor expansion it can be noticed that besides ordinary loss, MixTTT loss includes a regularization term that constrains the model update for test-time procedure.

3.1 Model Mismatch

Test-time training utilizes information carried in the test sample before its final inference. During this procedure, parts of the model are updated while other parts remain static (untouched). This procedure is dynamic and there exist samples showing the following property. As completing the auxiliary task, the information contained in the test sample is initially utilized in the positive direction. With the different designs of auxiliary task, the positive effect could be letting the model be more sensitive to the the visual structure of the test sample or minimizing the domain discrepancy between the training data and test data etc. However, when the test-time procedure goes further, the model would be overfitting to the auxiliary unsupervised task thus neglecting the true goal of test-time training: the main task. One obvious overfitting consequence is model mismatch between the updated part and static part for the main task.

In order to witness this mismatch problem more clearly, we visualize the embedding space of 1000 test sample points from CIFAR-10-C[2] with the [12] framework and see its change during different test-time training (TTT) stage. Overall we want to show that it’s hard to use the original main task head (classifier) to classify features from an overly changed feature extractor and MixTTT makes this task easier. Figure 1(a), 1(b), 1(c) show the embedding space variation of 1000 test samples after performing ordinary TTT. Each of them shows the embedding space after 10, 20, 30 steps of iteration in the test-time procedure. Figure 1(d), 1(e), 1(f) show the embedding space of the same data points after performing MixTTT with each corresponding to 10, 20, 30 steps of iterations during the test-time optimization. All above whole test-time procedure utilizes the same learning rate 1e-3. It can be noticed that in the embedding space of ordinary TTT clusters are initially separated as shown in Figure 1(a) with each class basically forming one cluster. As the test-time procedure goes, the clusters become chaotic and start to diffuse into other clusters as shown in Figure 1(b). In Figure 1(c) the embedding space is more messy, which puts more difficulty for the original classifier to distinguish these samples. MixTTT rather produces a less chaotic embedding space, which will ease the burden for classifier. In a nutshell, with an overly changed feature extractor, clusters are scattered and overlapped. Thus original classifier is not suitable to
classify such messy embedding space. This problem is considered as model mismatch in test-time training.

### 3.2 Mixup for Test-Time Training

For this section, we will briefly give the mixup based test-time training approach (MixTTT) and show how it can be used as an add-on module on existing test-time training based methods. Next we prove the efficacy of MixTTT in controlling the model change during test-time training with the framework from [12].

For each test-time optimization step, test sample(s) will be mixed up with randomly selected training samples using different mixup ratio drawn from Uniform distribution and form a batch of mixed data. Here fixed training set samples and fixed mixup ratio will all deteriorate the diversity of mixed samples and reduce fusion of the test sample with the training set distribution and thus restrict the regularization effect of mixup based test-time training. Then the mixed data is to perform auxiliary task optimization according to the setting of each test-time training method. We give an example of TTT-R[12] with MixTTT. The whole process is shown in Algorithm 1.

Let us define the notations of different variables for convenience. We consider a training set \( S_{tr} = \{(x_1, y_1), (x_2, y_2), \ldots (x_i, y_i), \ldots, (x_n, y_n)\} \), where \( x_i \) represents one training sample and \( y_i \) the corresponding main task label. One single test is denoted as \( x_t \) and its corresponding auxiliary self-supervised task label \( y_t \). We denote \( \lambda \) as the mixup ratio and one mixed sample \( \tilde{x}_{i,t} = \lambda x_i + (1 - \lambda) x_t \). Let \( f \) denote the shared feature extractor and \( h_m, h_s \) two independent classifiers of the main task and auxiliary self-supervised task. We package the whole auxiliary model as one function, denoted as \( g_\theta \). We denote \( L_{mt} \) as the loss with mixed samples as input in test-time training and \( L_t \) as the original test-time training loss. \( C \) is the function calculating auxiliary self-supervised loss.

From the start of equation (4) we show that how MixTTT loss \( L_{mt} \) is beyond original test-time training loss \( L_t \). The two loss functions are shown in equation (4) and (5). Here we
Algorithm 1 MixTTT

**Input:** one test-time training based method $f$; single test sample $x_t$, training set images, $\{x_1, x_2, \ldots, x_n\}$, shared feature extractor $f_\theta$, head for main task $hm_{\phi_1}$, head for auxiliary task $hs_{\phi_2}$, Loss function for auxiliary task $C$

**Parameter:** learning rate $\alpha$, mixup ratio list $\lambda_L$

**Output:** inference result

1: **while** iterationstep $\leq$ totalsteps **do**
2:   sample a mixup ratio $\lambda$
3:   sample a batch of training images $X$
4:   **for** each training sample in the batch **do**
5:     mix the training sample with the test sample and get the mixed batch $X_m$
6:   **end for**
7:   calculate auxiliary loss with the following equation:
   \[
   L_{aux} = C(X_m; hs_{\phi_2}, f_\theta)
   \] (1)
8:   update the shared feature extractor as follows:
   \[
   f_\theta \leftarrow f_\theta - \nabla_\theta L_{aux}
   \] (2)
9: **end while**

10: **inference** on the test sample with the following equation:
    \[
    \hat{y}_t = hm_{\phi_1} \circ f_\theta(x_t)
    \] (3)

11: **return** inference result

consider the classification as the auxiliary self-supervised task and its corresponding label $y_t$.

\[
L_t = -y_t^T \cdot \log(g_\theta(x_t))
\] (4)

\[
L_{mt} = -y_t^T \cdot \log(g_\theta(\tilde{x}_{i,t})) = -y_t^T \cdot \log(g_\theta(\lambda \cdot x_i + (1 - \lambda) \cdot x_t))
\] (5)

Next we write the mixup loss with Taylor expansion at mixup ratio $\lambda$ equals 0.

\[
L_{mt} = -y_t^T \cdot \log(g_\theta(x_t)) + \nabla_\lambda L_{mt}(0)\lambda + O(\lambda^2) = L_t + \nabla_\lambda L_{mt}(0)\lambda + O(\lambda^2)
\] (6)

From equation (6) we notice that mixup in test-time training in its loss function provides more regularization terms besides original test-time training loss. Now we focus on the regularization term and denote it as $L_r$.

\[
L_r = \nabla_\lambda L_{mt}(0)\lambda
\] (7)

\[
\nabla_\lambda L_{mt}(\lambda) = -\{(y_t \odot g_\theta(\tilde{x}_{i,t}) - 1)^T \left[ \frac{\partial g_\theta(x)}{\partial x} (x_i - x_t) \right] \}^T
\] (8)

From another perspective, we can write the gradient of original test-time training loss over input $x$ as follows:

\[
\nabla_x L_t(x) = -\{(y_t \odot g_\theta(\tilde{x}_{i,t}) - 1)^T \left[ \frac{\partial g_\theta(x)}{\partial x} \right] \}^T
\] (9)
Since we expand the loss at $\lambda$ equals zero, then $x$ equals $x_t$. We can get $L_r = (x_i - x_t)^T \nabla_x L_t(x_t) \lambda$ and $L_{mt}$ as follows:

$$L_{mt} = L_t + (x_i - x_t)^T \nabla_x L_t(x_t) \lambda + O(\lambda^2)$$  \hfill (10)

$$\nabla_x L_t = \frac{\partial f_{\text{eat}}}{\partial x} \frac{\partial L_t}{\partial f_{\text{eat}}}$$  \hfill (11)

$$\|\nabla_\theta L_t\| = \|\frac{\partial f_{\text{eat}}}{\partial \theta} \frac{\partial L_t}{\partial f_{\text{eat}}}\|$$  \hfill (12)

Equation (11) is to measure how much the feature extractor is supposed to change in back propagating such loss. By manually select $x_i$ bigger than $x_t$, we minimize the loss is to minimize $\nabla_x L_t$. Since feature is determined by two factors $\theta$ and $x$, for back propagation $x$ is fixed and will not be updated. So the derivative of feature over $x$ is fixed. So minimizing $\nabla_x L_t$ minimizes the $\frac{\partial L_t}{\partial f_{\text{eat}}}$ which is a shared part of $\|\nabla_\theta L_t\|$. Thus the change of feature extractor is intermediately minimized in Equation (11).

4 Experiments

4.1 Datasets and Experimental Settings

We evaluate the effectiveness of MixTTT on existing test-time training based methods following their own test-time auxiliary task. Our chosen methods are TTT-R [12], Tent [14] and TTT++[8]. We also choose CIFAR-10-C and CIFAR-100-C [2] for evaluation. The dataset setting of CIFAR-10-C and CIFAR-100-C are similar in many ways. They both contain $15 \times 5 \times 10000$ colour images and the size of all images is $32 \times 32$. 15 corresponds to 15 different kinds of corruption and 5 corresponds to 5 levels of severities. These 15 kinds of corruption reflects common situations that images will be under e.g. weather change, different lighting conditions, compression etc. Therefore it can widely verify the robustness of algorithms. We choose the most severe level to carry experiments. CIFAR-100-C is a 100-way classification dataset, while CIFAR-10-C is a 10-way classification dataset. Since the difference lies in the difficulty for these two datasets, their experimental results actually verify the problem we identified, which will be discussed in the next part.

We introduce the experimental setting of test-time procedure with MixTTT. In TTT-R [12], we train a network with ResNet50 as the backbone. Single test sample is to mixup with training samples and form a batch of mixed samples. These mixed samples are then rotated with a certain degree. and then to perform the auxiliary task. Considering the iteration step is limited, mixup ratio is selected with bigger proportion for the test sample, so we sample it from a uniform distribution $U[0.7, 1]$. The learning rate for CIFAR-10-C is set as $1e^{-3}$ and the iteration steps is 10. For CIFAR-100-C learning rate is set as $1e^{-4}$ and the iteration steps is 5. In Tent [14] test samples are to mix with training samples. Since the label for test samples are unknown, we choose to mix them with high proportion test images to not severely disturb the class label. We sample the mixup ratio from $U[0.95, 1]$. The mixed images are to perform entropy minimization to adjust the affine parameters in batch normalization layers. The experimental results demonstrate that MixTTT influence the test-time procedure in a positive way. We implement it using the tent code part from [8] and borrow the well trained checkpoints and hyperparameter setting from it for the test-time procedure.
In TTT++ the test-time procedure is to perform contrastive learning and feature alignment. When applying MixTTT in contrastive learning, the goal is set as maximizing the agreement between augmented test images and mixed images. To maintain the class label of test images, the mixup ratio is set for bigger proportion test images $U_{[0.9, 1]}$. We also use the setting, code and checkpoints from [8]. All the experiments are carried on a Nvidia V100 GPU.

### 4.2 Experimental Results

Table 1 shows the error rate of existing methods and its application with MixTTT on CIFAR-10-C. The initial model checkpoints for corresponding existing methods and its application with MixTTT are the same for a fair comparison. It can be noticed that MixTTT helps improve the main task performance with different auxiliary tasks in test-time procedure e.g. rotation, entropy minimization and contrastive learning. Although mixup may introduce some bias for entropy minimization, within a range it could still counter model mismatch and benefit the test-time procedure. For CIFAR-10-C, test-time procedure with MixTTT performs better than original test-time procedure for almost all kinds of corruption. For CIFAR-100-C, test-time procedure with MixTTT performs better than original test-time procedure for all kinds of corruption. It can be also noticed from Table 2 that for different kinds of unsupervised test-time procedure, our add-on module is effective and helpful and can suit situations with different number of test samples.

Overall the 10-way classification on CIFAR-10-C is easier than the task on CIFAR-100-C which has 100 classes. The difference is not just the number of classes. If using the same backbone for feature extraction, then the load of classifier for 100-way classification is heavier than 10-way classification. That is to say, 100-way classification is prone to be influenced by model mismatch than 10-way classification. This also explains task on CIFAR-100-C benefits more than task on CIFAR-10-C.

| Methods | gauss | shot | impul | defoc | glass | motn | zoom | snow | frost | fog | birt | contr | elast | pixel | jpeg | avg |
|---------|-------|------|-------|-------|-------|------|------|------|------|-----|------|-------|-------|-------|-----|-----|
| TTT-R[12] | 26.42 | 23.77 | 33.50 | 12.66 | 31.02 | 17.36 | 12.36 | 15.06 | 15.60 | 19.75 | 9.40 | 16.08 | 23.79 | 21.50 | 22.83 | 20.07 |
| SHOT[7] | 15.66 | 14.67 | 24.84 | 8.96 | 23.13 | 12.42 | 7.42 | 14.07 | 12.40 | 17.06 | 7.79 | 7.67 | 17.88 | 10.68 | 12.98 | 13.84 |
| Tent[8] | 14.47 | 13.10 | 21.80 | 8.34 | 21.08 | 11.06 | 6.98 | 12.06 | 11.71 | 14.04 | 7.11 | 7.02 | 16.70 | 10.19 | 12.05 | 12.51 |
| TTT++[8] | 14.10 | 12.28 | 12.12 | 8.74 | 16.98 | 9.91 | 7.17 | 10.15 | 10.60 | 9.22 | 5.84 | 6.24 | 14.04 | 16.98 | 11.59 | 11.06 |
| +MixTTT | 25.32 | 22.54 | 32.67 | 11.60 | 29.51 | 16.00 | 11.30 | 15.01 | 14.34 | 18.16 | 8.77 | 8.99 | 22.34 | 20.82 | 21.81 | 18.61 |
| +MixTTT | 14.28 | 12.99 | 22.09 | 8.27 | 20.57 | 10.83 | 7.14 | 11.52 | 11.52 | 13.05 | 6.98 | 7.84 | 16.27 | 9.69 | 12.05 | 12.33 |
| [8]+MixTTT | 12.72 | 11.55 | 11.85 | 7.96 | 17.04 | 8.95 | 6.97 | 9.12 | 9.37 | 8.45 | 5.36 | 5.43 | 13.50 | 8.88 | 11.06 | 9.88 |

Table 1: Error rate (%) on CIFAR-10-C with level 5 corruption.

| Methods | gauss | shot | impul | defoc | glass | motn | zoom | snow | frost | fog | birt | contr | elast | pixel | jpeg | avg |
|---------|-------|------|-------|-------|-------|------|------|------|------|-----|------|-------|-------|-------|-----|-----|
| TTT-R[12] | 78.07 | 75.79 | 89.34 | 32.99 | 79.37 | 46.22 | 32.45 | 48.21 | 52.06 | 54.52 | 26.50 | 45.16 | 53.45 | 65.78 | 53.83 | 55.58 |
| SHOT[7] | 42.73 | 40.89 | 52.20 | 31.04 | 50.23 | 35.99 | 29.18 | 42.08 | 39.27 | 45.60 | 29.38 | 29.86 | 43.19 | 33.45 | 35.75 | 38.72 |
| Tent[8] | 40.45 | 38.55 | 48.50 | 29.03 | 46.78 | 32.71 | 28.02 | 38.50 | 36.64 | 37.86 | 27.60 | 27.93 | 40.44 | 31.47 | 34.33 | 35.92 |
| TTT++[8] | 36.76 | 34.76 | 39.90 | 27.98 | 40.67 | 29.75 | 25.86 | 31.71 | 33.42 | 32.24 | 24.11 | 25.03 | 35.72 | 29.58 | 31.82 | 31.95 |
| +MixTTT | 72.45 | 68.91 | 80.23 | 35.93 | 72.73 | 45.17 | 34.32 | 46.34 | 47.93 | 52.87 | 28.80 | 44.90 | 53.24 | 61.39 | 52.31 | 53.16 |
| +MixTTT | 40.17 | 38.64 | 48.50 | 28.85 | 46.44 | 32.18 | 27.42 | 37.99 | 36.55 | 36.96 | 27.24 | 29.56 | 40.47 | 31.09 | 33.79 | 35.72 |
| [8]+MixTTT | 34.72 | 32.58 | 39.14 | 26.84 | 39.29 | 28.54 | 24.94 | 31.42 | 31.85 | 30.80 | 23.54 | 23.58 | 30.09 | 27.92 | 30.30 | 30.36 |

Table 2: Error rate (%) on CIFAR-100-C with level 5 corruption.


5 Conclusion and Discussion

Through analyzing the whole test-time training process, we witness model mismatch as one of the biggest problems TTT is facing. We propose MixTTT as an add-on module for test-time training based methods and show from theoretical view how mixup in test-time training controls the model change thus further control model mismatch while completing the test-time auxiliary task. This add-on module can suit both the single-test-sample-based and multiple-test-sample-based test-time procedure following the ordinary optimization process in such procedure.

Model mismatch is not the only problem for test-time training. As the two approaches we mentioned above, how to dig and utilize the information from test sample(s) is worth more attention and could deliver extra gain for the main task. The development of more informative unsupervised or self-supervised task will further improve test-time training. The development of training mechanisms that can entangle the auxiliary task and the main task more closely will also improve test-time training related methods.

Besides, since test-time training is a new paradigm, unified evaluation concerning computation cost, test-time training time, access of data should be proposed and followed.

References

[1] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In International Conference on Learning Representations, 2018.

[2] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In International Conference on Learning Representations, 2018.

[3] Minhao Hu, Tao Song, Yujun Gu, Xiangde Luo, Jieneng Chen, Yinan Chen, Ya Zhang, and Shaoting Zhang. Fully test-time adaptation for image segmentation. In International Conference on Medical Image Computing and Computer-Assisted Intervention, pages 251–260. Springer, 2021.

[4] Yusuke Iwasawa and Yutaka Matsuo. Test-time classifier adjustment module for model-agnostic domain generalization. Advances in Neural Information Processing Systems, 34, 2021.

[5] Neerav Karani, Ertunc Erdil, Krishna Chaitanya, and Ender Konukoglu. Test-time adaptable neural networks for robust medical image segmentation. Medical Image Analysis, 68:101907, 2021.

[6] Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In International Conference on Machine Learning, pages 5275–5285. PMLR, 2020.

[7] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In Proceedings of the 37th International Conference on Machine Learning, pages 6028–6039, 2020.
[8] Yuejiang Liu, Parth Kothari, Bastien van Delft, Baptiste Bellot-Gurlet, Taylor Morgan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? *Advances in Neural Information Processing Systems*, 34, 2021.

[9] Prashant Pandey, Mrigank Raman, Sumanth Varambally, and Prathosh AP. Generalization on unseen domains via inference-time label-preserving target projections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12924–12933, June 2021.

[10] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 506–516, 2017.

[11] Rui Shao, Bochao Zhang, Pong C Yuen, and Vishal M Patel. Federated test-time adaptive face presentation attack detection with dual-phase privacy preservation. *arXiv preprint arXiv:2110.12613*, 2021.

[12] 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*, pages 9229–9248. PMLR, 2020.

[13] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR, 2019.

[14] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2020.

[15] Jingjing Wang, Jingyi Zhang, Ying Bian, Youyi Cai, Chunmao Wang, and Shiliang Pu. Self-domain adaptation for face anti-spoofing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2746–2754, 2021.

[16] Yufei Wang, Haoliang Li, and Alex C Kot. Heterogeneous domain generalization via domain mixup. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3622–3626. IEEE, 2020.

[17] Yuan Wu, Diana Inkpen, and Ahmed El-Roby. Dual mixup regularized learning for adversarial domain adaptation. In *European Conference on Computer Vision*, pages 540–555. Springer, 2020.

[18] Minghao Xu, Jian Zhang, Bingbing Ni, Teng Li, Chengjie Wang, Qi Tian, and Wenjun Zhang. Adversarial domain adaptation with domain mixup. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6502–6509, 2020.

[19] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsek Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.
[20] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[21] Linjun Zhang, Zhun Deng, Kenji Kawaguchi, Amirata Ghorbani, and James Zou. How does mixup help with robustness and generalization? In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=8yKEo06dKNo.