NanoAdapt: Mitigating Negative Transfer in Test Time Adaptation with Extremely Small Batch Sizes

Shiji Zhao, Shao-Yuan Li *, Sheng-Jun Huang

College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
MIIT Key Laboratory of Pattern Analysis and Machine Intelligence
{zhaosj, lisy, huangsj}@nuaa.edu.cn

Abstract

Test Time Adaptation (TTA) has garnered significant attention in recent years, with the research focus on addressing distribution shifts during test time. As one fundamental component of many TTA methods, the Batch Normalization (BN) layer plays a crucial role in enabling the model adaptability. However, existing BN strategies can prove detrimental when the batch size is (extremely) small. In numerous real-world scenarios, limited hardware resources or just-in-time demand often necessitates adjusting models with very small batch sizes, making existing methods less practical. In this paper, we first showcase and thoroughly analyze the negative transfer phenomenon in previous TTA methods encountering extremely small batch sizes. Subsequently, we propose a novel batch size-agnostic method called NanoAdapt to effectively mitigate the negative transfer even with batch size 1. NanoAdapt is composed of three key components: a dynamic BN calibration strategy that leverages historical information and the Taylor series to refine the statistics estimations, an entropy-weighted gradient accumulation strategy that uses the entropy of each sample’s label prediction to weigh and accumulate the loss for backpropagation, and a novel proxy computation graph to capture the sample interactions. Extensive experiments are conducted to validate the superiority of NanoAdapt, showing its consistent efficacy in improving existing TTA methods.

1 Introduction

Deep neural networks have demonstrated remarkable performance across various machine learning problems, particularly in image classification. However, they frequently exhibit brittleness and vulnerability to challenges arising from data distribution shifts. Instances of these challenges include a rapid decline in accuracy for deep image classifiers when confronted with input perturbations, such as noise or blur [Hendrycks and Dietterich, 2019]. Consequently, the robustification of deep models against such test shifts constitutes a crucial and actively researched area.

Test time adaptation (TTA) is an emerging frontier that addresses the challenge of adjusting pre-trained models to unforeseen shifts in distribution. It is particularly crucial as it enables models to perform effectively on unlabeled data encountered during deployment. In the pursuit to adapt models during testing, various methodologies have been explored, including test time normalization [Nado et al., 2020; Schneider et al., 2020], entropy minimization [Wang et al., 2021], self-supervised learning [Sun et al., 2020a; Liu et al., 2021], contrastive learning [Chen et al., 2022], data augmentation [Zhang et al., 2022a], uncertainty-aware optimization [Niu et al., 2022], etc.

Recently, some studies have focused on addressing the challenges of TTA in intricate test environments, particularly in Out-of-Distribution scenarios [Sun et al., 2020b; Li et al., 2023; Zhou et al., 2023], as well as in online continual adaptation [Boudiaf et al., 2022; Wang et al., 2022; Niu et al., 2022; Zhang et al., 2023]. However, there is a noticeable gap in the literature concerning the issue of TTA in resource-constrained scenarios. As depicted in Fig. 1, resource-constrained industrial environments with limited hardware resources such as mobile devices, or just-in-time adaptation needs for individualized models, often necessitate
adjusting models with very small batch sizes. However, existing TTA methods relying on the Batch Normalization layer encounter significant challenges in such situations, as the conventional BN relies on a sufficiently large enough batch size to accurately estimate the data distribution statistics. When the batch sizes are small, they tend to degrade drastically, leading to severe negative transfer. Fig. 1 shows the noticeable performance decline of one representative pioneering TTA approach TENT [Wang et al., 2021] as the batch size decreases from N=200 to N=1. With extremely small batch sizes N=2, 1, the model collapses with error rates exceeding 97%.

Previously only few works have explored the small batch size challenge of BN in TTA. SAR [Niu et al., 2023] highlighted that existing TTA methods struggle to handle small batch sizes. It sidestepped the issues associated with BN by opting for Group Normalization (GN) over BN in the pre-trained training model. However, what if a pre-trained model opting for Group Normalization (GN) over BN in the pre-trained training model. However, what if a pre-trained model already includes BN modules and how do we address this issue? Delta [Zhao et al., 2023] introduced test time renormalization to refine the statistical estimation in BN. While this proves helpful in addressing situations with small batch sizes, it still falls short in extremely small batch size cases. Besides, gradient accumulation is a common technique during the training phase to tackle challenges arising from small batch sizes. It entails accumulating gradients computed across multiple batches and updating them collectively. However, simple gradient accumulation alone does not mitigate the negative transfer phenomenon in TTA methods when confronted with small batch sizes.

To the best of our knowledge, our study is the first to investigate TTA with extremely small batch sizes. We first conduct a thorough analysis to comprehend why previous approaches relying on Batch Normalization struggle to adapt the model during test time. We found that when the batch size is particularly small, there is significant discrepancy between the BN statistics of the current batch and the entire test dataset. Additionally, the gradient computed from small batches is also highly unreliable. Then we introduce our NanoAdapt approach to mitigate this issue. NanoAdapt comprises three main components: Dynamic BN Calibration (DBC), Entropy-Weighted Gradient Accumulation (EWGA), and Proxy Computation Graph (PCG). DBC harnesses historical information and employs the Taylor series to correct the statistics estimations. EWGA uses the entropy of each sample’s label prediction to weigh the loss and accumulates them for backpropagation. PCG employs different data augmentations to construct a proxy computation graph for the forward process, capturing the sample interactions for backpropagation.

In summary, our contributions are as follows:

- We showcase and give a thorough analysis of the negative transfer phenomenon in prior TTA methods with extremely small batch sizes, showing that inaccurate statistical estimates and the unreliable gradients collectively contribute to this phenomenon.
- We have designed three powerful tools: DBC, EWGA, and PCG to respectively solve these issues. DBC aims to correct BN statistics at each time step. EWGA focuses on reduce noise in gradients and PCG is designed to capture sample interactions in gradients.
- We evaluate NanoAdapt on three distribution shift benchmark datasets, showing that it consistently mitigates negative transfer for existing TTA methods with (extremely) small batch sizes.

2 Related Works

Unsupervised Domain Adaptation (UDA) Unsupervised domain adaptation (UDA) seeks to enhance models’ capacity for generalization to target domain data, even when no labeled data is available in the target domain. UDA commonly involves methods such as feature alignment [Zellinger et al., 2017; Long et al., 2017], adversarial training [Ganin and Lempitsky, 2015; Yi-Hsuan Tsai, 2018; Ganin et al., 2016] and self-supervised training [Hoyer et al., 2022; Zou et al., 2018]. In more stringent scenarios, such as Source-Free Domain Adaptation (SFDA), it is not feasible to acquire source domain data, and only the source domain model is available [Liang et al., 2020; Qiu et al., 2021; Ding et al., 2022]. However, SFDA still faces limitations in addressing a more realistic scenario where the distribution of the target domain is unknown before testing commences.

Domain Generalization (DG) Domain generalization (DG) focuses on developing models capable of learning from multiple domains and performing well on unseen testing domains. Techniques such as meta-learning [Li et al., 2019; Li et al., 2018a], data augmentation [Prakash et al., 2019; Nam et al., 2021], and domain alignment [Li et al., 2018b], style augmentation [Li et al., 2022; Kang et al., 2022; Zhang et al., 2022b] are employed. However, DG is not capable of generalizing to all target domains without any prior knowledge of the specific target domain.

Test Time Adaptation (TTA) Test Time Adaptation (TTA) focuses on dynamically adapting a pre-trained model in real-time. This adaptation occurs as a test data stream in batches, offering a more agile and online approach to model refinement. Pred BN [Nado et al., 2020] replaces the normalization statistics computed during training with those derived from the test mini-batch. TENT [Wang et al., 2021] optimizes the affine parameters in Batch Normalization through entropy minimization during testing. Long-term test time adaptation in dynamically changing environments is explored by EATA [Niu et al., 2022] and CoTTA [Wang et al., 2022]. AdaContrast [Chen et al., 2022] employs contrastive learning to enhance feature learning, incorporating a pseudo label refinement mechanism. LAME [Boudiaf et al., 2022] adjusts the model’s output probabilities via the Laplacian adjusted maximum-likelihood estimation. SAR [Niu et al., 2023] replaces BN with GN in the training phase and introduces a sharpness-aware and reliable entropy minimization method. OWTTT [Li et al., 2023] develops adaptive strong OOD pruning and proposes a method to dynamically expand prototypes to represent robust OOD samples. In various real-world scenarios, the demand for constrained hardware resources or just-in-time adaptation often necessitates adapting
models with small batch sizes. However, prior research has largely overlooked the challenges associated with adaptation under small batch sizes. Delta [Zhao et al., 2023] introduces a test time batch renormalization technique to enhance estimated normalization statistics, but it falls short of fully resolving this issue. Our proposed method, NanoAdapt, effectively addresses this problem and mitigates the occurrence of the negative transfer phenomenon.

3 Problem Statement and Analysis

3.1 TTA Problem Definition

In the context of Test Time Adaptation, we denote the training data as \( D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N_{\text{train}}} \sim P_{\text{train}}(X, Y) \), where \( x \in X \) represents the input, and \( y \in Y \) represents the label. We denote the test data as \( D_{\text{test}} = \{(x_j, y_j)\}_{j=1}^{N_{\text{test}}} \sim P_{\text{test}}(X, Y) \), where the labels \( \{y_j\} \) are not available. It should be noted that the test data is delivered in a streaming fashion, organized into batches. We consider the common covariate distribution shift, which can be formalized as \( P_{\text{train}}(X) \neq P_{\text{test}}(X) \) while \( P_{\text{train}}(Y|X) = P_{\text{test}}(Y|X) \). A pre-trained model \( f_{\{\theta_0, s_0\}} \) is characterized by model weights \( \theta_0 \) and normalization statistics \( s_0 \) learned from \( D_{\text{train}} \). It requires updates at each time step \( t \) to attain \( f_{\{\theta_t, s_t\}} \) and generate label predictions for samples \( x_t \). In real-world scenarios, the batch size of incoming test data may vary at each time step. In the most extreme scenario, each batch may comprise only one sample.

3.2 The Negative Transfer Phenomenon With Small Batch Sizes

We are familiar with the forward process of BN layers, which unfolds as follows:

\[
\begin{align*}
\hat{x}_i &= \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \\
\hat{y}_i &= \gamma \hat{x}_i + \beta,
\end{align*}
\]

where \( x \) is the input and \( y \) is the output of the BN layer. \( \mu \) and \( \sigma \) are the normalization statistics and \( \gamma \) and \( \beta \) are the affine parameters.

Traditionally, each BN layer records \( \{\mu_0, \sigma_0\} \) in the training phase and directly applies them in the testing phase:

\[
\begin{align*}
\mu &= \mu_0, \\
\sigma &= \sigma_0.
\end{align*}
\]

In previous TTA methods, notable works have implemented treatments specifically tailored for Batch Normalization layers. Taking the two most representative works as examples, Pred BN [Nado et al., 2020] entails substituting the normalization statistics computed during training with those obtained from the test mini-batch. Meanwhile, TENT [Wang et al., 2021] further minimize entropy and selectively updates only the affine parameters of the BN layers. Their BN statistics are all calculated using the information from the current batch at time step \( t \):

\[
\begin{align*}
\mu &= \mu_t, \\
\sigma &= \sigma_t.
\end{align*}
\]

When dealing with small batch sizes, the calculated statistics are not accurate, and the computed gradient is also unreliable.

As illustrated in Fig. 2, when the batch size exceeds 100, both Pred BN and Tent perform well. However, as the batch size decreases to 1, their performance declines. In the most severe scenarios when \( N=1 \), their error rate reaches 99%. Since Tent requires gradient updates, we employ gradient accumulation to assist it, which helps reduce the classification error but falls short of avoiding the negative transfer phenomenon. With the implementation of NanoAdapt, the negative transfer phenomenon can be effectively circumvented.

4 The Proposed Method: NanoAdapt

4.1 Dynamic BN Calibration

To address inaccurate statistics within each testing batch, a straightforward solution arises: employing the Exponential Moving Average (EMA) to update the statistics in each time step \( t \):

\[
\begin{align*}
\hat{\mu}_t(x_{1:t}) &= m \hat{\mu}_{t-1}(x_{1:t-1}) + (1 - m) \hat{\mu}_t(x_t), \\
\hat{\sigma}_t^2(x_{1:t}) &= m \hat{\sigma}^2_{t-1}(x_{1:t-1}) + (1 - m) \hat{\sigma}^2_t(x_t),
\end{align*}
\]

where \( m \) denotes the smoothing coefficient, \( \hat{\mu}_t(x_t) \) represents the mean value of each input \( x_t \) before the BN layers at time step \( t \). \( \hat{\mu}_{t-1}(x_{1:t-1}) \) represents the aggregated mean estimates from past time steps \( 1 \) to \( t-1 \). \( \hat{\sigma}^2_t(x_{1:t}) \) represents the final mean estimation with current model \( \theta_t \). The notation for the corresponding variance \( \hat{\sigma}^2 \) has a similar interpretation.

Due to the second-order nature of \( \hat{\sigma}^2 \), a more effective approach is to maintain the mean values of both \( x \) and \( x^2 \) for each feature map before the BN layer at time step \( t \) and calculate \( \hat{\sigma}^2_t \) based on the corresponding relationship:

\[
\begin{align*}
\hat{\mu}_t(x_{1:t}) &= m \hat{\mu}_{t-1}(x_{1:t-1}) + (1 - m) \hat{\mu}_t(x_t), \\
\hat{\mu}_t(x^2_{1:t}) &= m \hat{\mu}_{t-1}(x^2_{1:t-1}) + (1 - m) \hat{\mu}_t(x^2_t), \\
\hat{\sigma}^2_t(x_{1:t}) &= \hat{\mu}_t(x^2_{1:t}) - \hat{\mu}^2_t(x_{1:t}).
\end{align*}
\]

Nonetheless, a notable issue arises with this computation that both \( \hat{\mu}_{t-1}(x_{1:t-1}) \) and \( \hat{\mu}_{t-1}(x^2_{1:t-1}) \) are computed using the previous model weights \( \theta_{t-1} \). For a more accurate estimation, these values should be computed under the current
The final dynamic BN calibration can be formalized as the gradients of layers preceding the BN layer [Yao l et al., 2021]. So we focus solely on computing the gradients of the current model weights \( \theta_{t-1} \) under the current model weights \( \theta_{t} \).

Given that the model weights \( \theta_{t} \) are updated through gradient descent, when the learning rate is sufficiently small, the alterations in parameters at each time step \( t \) can be perceived as a continuous function. Hence, the Taylor polynomials can be employed to approximately compute \( \mu_{\theta_{t-1}}(x_{t-1}, x_{t}) \) and \( \mu_{\theta_{t}}(x_{t}, x_{t}) \) derived from past model weights \( \theta_{t-1} \) under the current model weights \( \theta_{t} \).

\[
\begin{align*}
\hat{\mu}_{\theta_{t}}(x_{1:t-1}) &= \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}) + \frac{\partial \hat{\mu}_{\theta_{t-1}}(x_{1:t-1})}{\partial \theta_{t-1}} (\theta_{t} - \theta_{t-1}) \\
&\quad + O(||\theta_{t} - \theta_{t-1}||^2), \\
\hat{\mu}_{\theta_{t}}(x_{1:t-1}^2) &= \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}^2) + \frac{\partial \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}^2)}{\partial \theta_{t-1}} (\theta_{t} - \theta_{t-1}) \\
&\quad + O(||\theta_{t} - \theta_{t-1}||^2),
\end{align*}
\]

(6)

where \( O(\cdot) \) represents higher-order terms.

To expedite computation, we neglect higher-order terms. Additionally, empirical observations suggest a swift decline in the gradients of layers preceding the BN layer [Yao et al., 2021]. So we focus solely on computing the gradients of the convolutional layer preceding the BN layer. We simplify Eq. (6) as follows:

\[
\begin{align*}
\hat{\mu}_{\theta_{t}}(x_{1:t-1}) &\approx \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}) + \frac{\partial \hat{\mu}_{\theta_{t-1}}(x_{1:t-1})}{\partial \theta_{t-1}} (\theta_{t} - \theta_{t-1}), \\
\hat{\mu}_{\theta_{t}}(x_{1:t-1}^2) &\approx \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}^2) + \frac{\partial \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}^2)}{\partial \theta_{t-1}} (\theta_{t} - \theta_{t-1}),
\end{align*}
\]

(7)

where the superscript \( l \) indicates that only the parameters of the layer preceding the BN layer \( l \) are considered. Therefore, the final dynamic BN calibration can be formalized as the following:

\[
\begin{align*}
\hat{\mu}_{\theta_{t}}(x_{1:t}) &= \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}) + \frac{\partial \hat{\mu}_{\theta_{t-1}}(x_{1:t-1})}{\partial \theta_{t-1}} (\theta_{t} - \theta_{t-1}) \\
&\quad + (1 - m) \mu_{\theta_{t}}(x_{t}), \\
\hat{\mu}_{\theta_{t}}(x_{1:t}^2) &= \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}^2) + \frac{\partial \hat{\mu}_{\theta_{t-1}}(x_{1:t-1}^2)}{\partial \theta_{t-1}} (\theta_{t} - \theta_{t-1}) \\
&\quad + (1 - m) \mu_{\theta_{t}}(x_{t}^2), \\
\hat{\sigma}_{\theta_{t}}^2(x_{1:t}) &= \hat{\mu}_{\theta_{t}}(x_{1:t}^2) - \hat{\mu}_{\theta_{t}}(x_{1:t}).
\end{align*}
\]

(8)

The implementation is depicted in Figure 2. In each time step \( t \), for each BN layer \( l \), we apply the correction outlined in Eq. (8) to obtain estimates of the statistics \( \hat{\mu}_{\theta_{t}}(x_{1:t}) \) and \( \hat{\sigma}_{\theta_{t}}^2(x_{1:t}) \). These estimates are then applied to the BN layer, and the estimates at the current time step \( t \) are integrated into the subsequent time step \( t+1 \).

4.2 Entropy-Weighted Gradient Accumulation with Proxy Computation Graph

To deal with unreliable gradient with small batch sizes, we propose to employ an entropy-weighted gradient accumulation strategy that leverages entropy of each sample’s label prediction to assess the reliability of the gradient computed by the loss. Subsequently, these gradients are accumulated for backpropagation. This can be formulated as follows:

\[
L = \sum_{i=1}^{S} \min \left( \frac{1}{H(x)}, 1 \right) \cdot L_{\alpha}(x),
\]

(9)

where \( H(x) \) denotes the entropy of the sample’s label prediction, \( L_{\alpha} \) represents the original loss computed by TTA methods, \( S \) denotes the steps for gradient accumulation, \( L \) represents the corrected loss of \( x \) for gradient descent, and \( r \) is em-
ployed to progressively increase the weighting of subsequent step computations since their BN statistics are more accurate.

When combined with dynamic BN calibration proposed in Section 4.1, this approach is sufficiently effective in achieving results nearly comparable to those obtained with larger batch sizes. However, this success diminishes when the batch size is extremely small, such as N=1. In the following, we will give a formal analysis, and propose our proxy computation graph strategy to compensate this issue.

Analysis for Batch Size 1 Due to the forward process of BN layers shown in Eq. (1), we can compute the gradients across the BN layers as follows:

\[
\frac{\partial L}{\partial y_i} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_i}, \quad \frac{\partial L}{\partial \gamma_i} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_i}, \quad \frac{\partial L}{\partial \beta_i} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_i},
\]

\[
\frac{\partial L}{\partial \sigma_i^2} = \sum_{i=1}^{N} \frac{\partial L}{\partial \sigma_i^2} \cdot (x_i - \mu) \cdot \frac{1}{2} (\sigma^2 + \epsilon)^{-\frac{3}{2}},
\]

\[
\frac{\partial L}{\partial \mu} = \frac{\partial L}{\partial \sigma^2 \sigma_i^2} \cdot \frac{\partial L}{\partial \sigma_i^2} \cdot \frac{1}{m} + \sum_{i=1}^{N} \frac{\partial L}{\partial \sigma_i^2} - \frac{1}{N} \frac{\partial L}{\partial \sigma^2} + \epsilon,
\]

\[
\frac{\partial L}{\partial x_i} = \frac{1}{\sqrt{\sigma^2 + \epsilon}} \frac{\partial L}{\partial \sigma_i^2} + \frac{1}{N} \frac{\partial L}{\partial \mu} + \frac{2 (x_i - \mu)}{N} \frac{\partial L}{\partial \sigma^2}.
\]

We can further simplify the gradient over \( x_i \) as follows:

\[
\frac{\partial L}{\partial x_i} = \frac{1}{N} \sqrt{\frac{\partial L}{\partial \sigma_i^2} \sum_{j=1}^{N} \frac{\partial L}{\partial \sigma_j} - \hat{x}_i \sum_{j=1}^{N} \frac{\partial L}{\partial \sigma_j} \hat{x}_j}.
\]

The gradient includes the interaction term \( \hat{x}_i \sum_{j=1}^{N} \frac{\partial L}{\partial \sigma_j} \hat{x}_j \) between different samples within one batch. When N=1, the interaction term disappears, indicating that even if we employ dynamic BN calibration and accumulate entropy-weighted gradients, the computation graph remains different.

Proxy Computation Graph Strategy To build a proxy computation graph with sample interaction terms, we propose employing data augmentation to generate various views of a single sample \( x \). As illustrated in Fig. 3, in the forward process, we randomly select \( k \) distinct data augmentations including rotation, random crop, flipping, grayscale, saturation to generate \( \{ \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_k \} \) and combine them with the original image \( x \) to feed into the network and construct the proxy computation graph. Then we employ the dynamic BN calibration to exclusively replace the statistics while preserving the forward proxy computation graph.

\[
\hat{\mu}, \hat{\sigma} = DBC(\mu, \sigma)
\]

In other words, we fool the model into believing that the statistics are computed by the proxy computation graph. During the backpropagation process, the gradient will traverse through the substituted statistics in the proxy computation graph.

5 Experiments

5.1 Experimental Setup

Datasets We conduct experiments on three distribution shift datasets: Cifar10-C, Cifar100-C and ImageNet-C [Hendrycks and Dietterich, 2019], which are commonly used to evaluate the performance of test time adaptation. These datasets cover a comprehensive range of 15 distinct corruption types. For each corruption type, there are 5 severity levels, providing a nuanced evaluation of the models under varying degrees of data corruption.

Baselines Pred BN [Nado et al., 2020] and Tent [Wang et al., 2021] represent pioneering approaches in TTA, specifically addressing the adaptation of BN layers. Notably, Pred BN does not require gradient updates, while Tent necessitates gradient updates. Subsequent TTA works often build upon these two studies to make modifications in other directions. Our paper specifically addresses the issue that TTA methods relying on BN may exhibit negative transfer when the batch size is small. Therefore, we choose these two as baselines. Furthermore, we compare with Delta [Zhao et al., 2023], the latest and so far the only method that addresses the inaccuracy of BN statistics.

Implementation Details We evaluate the results at the most severe corruption level (level 5). We use the ResNet-50 [He et al., 2016] architecture as the pre-trained model for our experiments. The performance of ResNet-50 on each dataset is denoted as Source. For optimization, we utilize the Adam optimizer [Kingma and Ba, 2014] with a fixed learning rate of 0.001. In the dynamic BN calibration, we set the smoothing coefficient to \( m = 0.98 \). For the construction of the proxy computation graph, we opt for a value of \( k = 3 \) as the number of augmentations.

5.2 Comparative Analysis

Table 1 - 3 respectively report the results on the three distribution shift datasets. As observed in Table 1 and Table 2, when N=200, Pred BN and Tent perform well, and their error rates exhibited no significant change when augmented with Delta or NanoAdapt. However, with reduced batch size (N=10), both Pred BN and Tent experience a decline in performance, albeit still outperforming Source (without any adaptation). Applying Delta results in a partial improvement in performance, while applying NanoAdapt can nearly match or even slightly surpass the performance with N=200. When N=1, both Pred BN and Tent exhibit a negative transfer phenomenon. Delta proves effective in reducing the error rate with Pred BN, as it doesn’t require gradient updates. However, its impact is limited with Tent, as the gradient from a single sample may introduce excessive noise. On the other hand, when NanoAdapt is applied, both Pred BN and Tent experience a substantial decrease in error rate, surpassing the Source results. This indicates that NanoAdapt can effectively address the negative transfer phenomenon even with a minimal batch size. ImageNet-C is a more challenging dataset. As shown in Table 3, NanoAdapt consistently outperforms Delta. The only pity is that when N=1, NanoAdapt can only assist Tent in reducing its error rate from 99.97% to 74.10%, which is still slightly lower than the Source. This suggests that on
## Cifar10-C error rate on various batch sizes with corruption level 5

<table>
<thead>
<tr>
<th>Method</th>
<th>gauss</th>
<th>shot</th>
<th>impulse</th>
<th>defocus</th>
<th>glass</th>
<th>motion</th>
<th>zoom</th>
<th>snow</th>
<th>frost</th>
<th>fog</th>
<th>bright</th>
<th>contrast</th>
<th>elastic</th>
<th>pixel</th>
<th>jpeg</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>28.77</td>
<td>22.95</td>
<td>26.18</td>
<td>9.45</td>
<td>20.59</td>
<td>10.56</td>
<td>9.25</td>
<td>14.15</td>
<td>15.27</td>
<td>17.49</td>
<td>7.60</td>
<td>14.73</td>
<td>41.31</td>
<td>14.67</td>
<td>18.26</td>
<td></td>
</tr>
<tr>
<td>Pred BN</td>
<td>18.52</td>
<td>16.17</td>
<td>22.27</td>
<td>8.97</td>
<td>21.85</td>
<td>10.47</td>
<td>9.68</td>
<td>12.80</td>
<td>13.33</td>
<td>15.02</td>
<td>7.56</td>
<td>16.33</td>
<td>15.00</td>
<td>17.46</td>
<td>14.49</td>
<td></td>
</tr>
<tr>
<td>Tent</td>
<td>15.60</td>
<td>13.24</td>
<td>18.78</td>
<td>7.93</td>
<td>18.08</td>
<td>8.98</td>
<td>8.01</td>
<td>10.39</td>
<td>10.88</td>
<td>12.28</td>
<td>6.64</td>
<td>10.05</td>
<td>11.40</td>
<td>14.75</td>
<td>12.07</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1:** Error rate on Cifar10-C. Bold value represents the best result.

## Cifar100-C error rate on various batch sizes with corruption level 5

<table>
<thead>
<tr>
<th>Method</th>
<th>gauss</th>
<th>shot</th>
<th>impulse</th>
<th>defocus</th>
<th>glass</th>
<th>motion</th>
<th>zoom</th>
<th>snow</th>
<th>frost</th>
<th>fog</th>
<th>bright</th>
<th>contrast</th>
<th>elastic</th>
<th>pixel</th>
<th>jpeg</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>65.58</td>
<td>60.05</td>
<td>59.10</td>
<td>32.05</td>
<td>32.53</td>
<td>33.56</td>
<td>42.39</td>
<td>45.15</td>
<td>51.38</td>
<td>32.53</td>
<td>31.63</td>
<td>40.28</td>
<td>59.72</td>
<td>42.45</td>
<td>46.76</td>
<td></td>
</tr>
<tr>
<td>Pred BN</td>
<td>44.31</td>
<td>44.03</td>
<td>47.35</td>
<td>32.11</td>
<td>45.85</td>
<td>32.84</td>
<td>33.03</td>
<td>38.36</td>
<td>37.90</td>
<td>45.38</td>
<td>29.86</td>
<td>36.48</td>
<td>40.58</td>
<td>39.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+Delta</td>
<td>44.31</td>
<td>43.52</td>
<td>46.81</td>
<td>33.79</td>
<td>46.90</td>
<td>33.28</td>
<td>32.65</td>
<td>38.90</td>
<td>38.00</td>
<td>46.68</td>
<td>29.85</td>
<td>37.52</td>
<td>41.03</td>
<td>39.73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+NanoAdapt</td>
<td>41.28</td>
<td>40.92</td>
<td>43.92</td>
<td>31.73</td>
<td>43.56</td>
<td>31.05</td>
<td>29.77</td>
<td>35.99</td>
<td>34.22</td>
<td>43.13</td>
<td>33.75</td>
<td>38.71</td>
<td>41.29</td>
<td>36.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tent</td>
<td>40.11</td>
<td>39.50</td>
<td>41.95</td>
<td>29.65</td>
<td>41.85</td>
<td>30.72</td>
<td>29.67</td>
<td>34.45</td>
<td>34.76</td>
<td>39.04</td>
<td>27.43</td>
<td>32.99</td>
<td>37.55</td>
<td>40.23</td>
<td>35.52</td>
<td></td>
</tr>
<tr>
<td>+Delta</td>
<td>39.72</td>
<td>40.30</td>
<td>32.00</td>
<td>32.07</td>
<td>43.60</td>
<td>32.07</td>
<td>30.57</td>
<td>35.73</td>
<td>35.37</td>
<td>41.76</td>
<td>29.26</td>
<td>35.36</td>
<td>39.01</td>
<td>37.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+NanoAdapt</td>
<td>38.51</td>
<td>38.29</td>
<td>39.00</td>
<td>29.77</td>
<td>40.62</td>
<td>30.04</td>
<td>28.31</td>
<td>33.47</td>
<td>33.12</td>
<td>36.98</td>
<td>26.59</td>
<td>30.97</td>
<td>36.53</td>
<td>38.67</td>
<td>34.29</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1:** Error rate on Cifar100-C. Bold value represents the best result.
more challenging datasets like ImageNet-C, NanoAdapt prevents Tent from experiencing negative transfer but may not enable it to achieve nearly equivalent results to larger batch sizes.

5.3 Ablation Study of Different Components
We conducted a comprehensive ablation study on each component within NanoAdapt. As illustrated in Table 4, Tent performs poorly as the batch size decreases in the absence of NanoAdapt. Both Dynamic BN Calibration (DBC) and Entropy-Weighted Gradient Accumulation (EWGA) prove to be beneficial, although they may not assist in extreme situations when N=1. Combining DBC and EWGA effectively resolves the negative transfer phenomenon. Furthermore, incorporating the Proxy Computation Graph (PCG) results in improved performance.

5.4 Hyperparameter Sensitivity
As depicted in Table 5, we evaluate Tent+NanoAdapt with N=1, varying DBC smoothing factor \( m \) from 0.9, 0.92, 0.94, 0.96 to 0.98. In our experiments, \( m = 0.98 \) yields optimal performance. Additionally, we explore different selections of augmentation views \( k \), ranging from 1 to 4. Notably, we observe that when \( k \) reaches 3, the performance peaks.

<table>
<thead>
<tr>
<th>m</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>52.74</td>
</tr>
<tr>
<td>0.92</td>
<td>50.11</td>
</tr>
<tr>
<td>0.94</td>
<td>47.28</td>
</tr>
<tr>
<td>0.96</td>
<td>44.32</td>
</tr>
<tr>
<td>0.98</td>
<td>40.05</td>
</tr>
</tbody>
</table>

Table 5: Average Error Rate on Cifar100-C (Corruption level 5, N=1) with Tent+NanoAdapt under different hyperparameters

6 Conclusion
In this paper, we showcase the presence of negative transfer in previous test time adaptation methods with extremely small batch sizes. We attribute this issue to inaccurate BN statistics estimates and unreliable gradient updates. To address these challenges, we propose NanoAdapt, comprising three key components: dynamic BN calibration, entropy-weighted gradient accumulation and proxy computation graph. Dynamic BN calibration utilizes historical information to refine the estimation of Batch Normalization statistics, incorporating the Taylor series for additional correction. Entropy-weighted gradient accumulation employs entropy to weigh the loss and accumulate the gradients for backpropagation. The proxy computation graph is constructed to capture sample interaction terms in backpropagation, enhancing NanoAdapt’s capabilities. We evaluate NanoAdapt on three distribution shift datasets, demonstrating its efficiency in mitigating the negative transfer phenomenon with minimal batch sizes.
Acknowledgments
This work is supported by National Science and Technology Major Project (2022ZD0114801).

References


