Token Boosting for Robust Self-Supervised Visual Transformer Pre-training

Tianjiao Li†† Lin Geng Foo†† Ping Hu2 Xindi Shang3 Hossein Rahmani4 Zehuan Yuan3 Jun Liu††
1Singapore University of Technology and Design 2Boston University 3ByteDance 4Lancaster University
{tianjiao_li, lingeng_foo}@mymail.sutd.edu.sg, pinghu@bu.edu, shangxindi@bytedance.com
h.rahmani@lancaster.ac.uk, yuanzehuan@bytedance.com, jun_liu@sutd.edu.sg

Abstract

Learning with large-scale unlabeled data has become a powerful tool for pre-training Visual Transformers (VTs). However, prior works tend to overlook that, in real-world scenarios, the input data may be corrupted and unreliable. Pre-training VTs on such corrupted data can be challenging, especially when we pre-train via the masked autoencoding approach, where both the inputs and masked “ground truth” targets can potentially be unreliable in this case. To address this limitation, we introduce the Token Boosting Module (TBM) as a plug-and-play component for VTs that effectively allows the VT to learn to extract clean and robust features during masked autoencoding pre-training. We provide theoretical analysis to show how TBM improves model pre-training with more robust and generalizable representations, thus benefitting downstream tasks. We conduct extensive experiments to analyze TBM’s effectiveness, and results on four corrupted datasets demonstrate that TBM consistently improves performance on downstream tasks.

1. Introduction

Having rapidly risen in popularity in recent years, Vision Transformer (ViT) [16] and its variants [19] have shown impressive performance across various computer vision tasks, such as image classification, video recognition and 3D action analysis [2, 20, 31, 51, 60]. Amongst this wave of research on Visual Transformers (VTs), there has emerged a popular paradigm – self-supervised VT pre-training [2, 6, 20, 60] – which has attracted a lot of attention in the research community. These works [2, 6, 20, 60] generally pre-train VTs on a large dataset in a self-supervised manner, allowing them to extract semantically meaningful and generalizable features without the need for annotated data. These pre-trained VTs are practical and effective, showing good downstream performance with only slight fine-tuning, and have quickly become an important research direction.

Among the various self-supervised pre-training strategies, masked autoencoding [2, 16, 20] is a prominent approach that has been widely explored. Masked autoencoding [2, 11, 16, 20, 33, 60, 61, 71] works by randomly masking a portion of input tokens or patches, and letting the VT reconstruct them. Thus, in order to successfully reconstruct the masked tokens or patches, the VT is driven to learn the underlying structure and semantic information of the input data. Due to its natural compatibility with the token-wise representation in VTs, masked autoencoding has been explored for pre-training VTs on data across many fields, such as RGB images [2, 20, 61], pose data [11, 33] and 3D data [71].

However, in many real-world scenarios, the input data can be of low quality and can be unreliable, which is often overlooked. For example, in adverse weather conditions [22], the quality of captured images can be quite bad, with many corrupted pixels that degrade the image. Depth images, which are commonly used for 3D analysis, also often contain errors and noticeable perturbations due to measurement errors introduced by the depth camera sensors [26]. Skeleton data can also have distortions in the joint coordinates [14, 37, 59] due to noise introduced by the sensors, e.g., the pose tracking function of Kinect v2 often introduces average offsets of 50-100 mm per joint [59]. Examples of such corrupted data are depicted in Fig. 1.

In many cases, we have access to large-scale unlabeled data containing such corruptions and noise, and would like to exploit them to pre-train a VT that can be effectively adapted to a wide range of downstream tasks [2, 11, 20, 61]. Yet, this can be rather challenging since the data corruptions can interfere with the learning of representations during self-supervised pre-training, leading to unreliable features and predictions in downstream tasks. This is the case
in particular for masked autoencoding approaches, where both the inputs and the “ground truth” targets can be corrupted at the same time during pre-training. Hence, we aim to tackle the key question: How can we improve the VT’s ability to extract robust and reliable features during self-supervised pre-training on corrupted data? Once this is achieved, the pre-trained capability can then be transferred to downstream tasks, where the VTs can reliably extract generalizable features and perform well even when the testing data is unreliable.

To this end, we propose a novel Token Boosting Module (TBM) to improve the robustness of VTs against unreliable and corrupted data during the pre-training process. TBM is a plug-and-play module that can improve the performance of VTs by boosting the reliability of tokens, making them cleaner such that we can feed robust features to the subsequent layers. We also provide analysis to theoretically demonstrate that our TBM can indeed encourage the learning of cleaner features and tokens during masked autoencoding pre-training.

In summary, our contributions are:

- We design a novel Token Boosting Module (TBM) to learn to extract reliable features from corrupted and unreliable data during the self-supervised pre-training process of VTs. The TBM can be easily incorporated into multiple layers of VTs, and trained in an end-to-end manner to improve robustness and performance on corrupted data.
- We provide theoretical analysis to show that the robustness capabilities of TBM can be learned during the self-supervised masked autoencoding pre-training.
- We conduct experiments on multiple tasks including RGB image classification, 3D skeleton action recognition, and depth image classification with corrupted data. Experiments consistently show that our TBM provides significant improvements in downstream tasks.

2. Related Work

Self-supervised pre-training. As a practical way to train models without the need for annotated data, self-supervised pre-training has received a lot of attention. Recently, masked autoencoding has become a very popular approach for self-supervised Visual Transformer (VT) pre-training [2,7,16,20,35,45,60,61], together with other alternatives including contrastive learning [8,9,23,47,68,72,74] and self-distillation [6,18,28]. Due to the popularity of pre-training methods and their potential widespread applications in real-world scenarios [2,11,20,33,35,71], it becomes important to improve their robustness to corrupted data, but this is often overlooked by previous methods. For instance, during masked autoencoding pre-training on corrupted data, both inputs and “ground truths” can be unreliable, making it a tricky situation. Therefore, we explore if, under these challenging circumstances, we can allow VTs to learn to extract robust feature representations.

Robustness of VTs. Visual Transformers have recently been a popular and hot research direction, and their robustness has also attracted attention. Some existing works [1,3,42,77] investigate the test-time robustness of VTs, while some others explore the adversarial robustness of VTs [24,41,42]. Different from these works, we aim to improve the VTs’ capability to extract clean and robust features during self-supervised pre-training. This capability can benefit downstream tasks and improve their robustness against corrupted input data at test time.

Handling of Corrupted Data. Improving deep models’ performance on corrupted data is important for many practical tasks [5,14,37,43,52,73]. Previous efforts have been made to handle corrupted input data for supervised settings in tasks like skeleton action recognition [10,12,14,37,50,65], depth image classification [5,73], and face recognition [52]. Alternatively, some other approaches [43,55,62,64] can be performed during data pre-processing to reconstruct a “clean” input sample before using them for corresponding tasks. Many of these methods [43,48,55,62,64] require an additional stage of training, and tend to be specially designed for specific types of data or corruptions only. Here, we investigate a different problem from the above works, and instead explore the handling of corrupted data during self-supervised VT pre-training.

3. Method

In many real-world scenarios, the data (e.g., depth images, images in adverse weather conditions) can be cor-
lected with substantial corruptions. When such corrupted data is used for self-supervised pre-training, they can lead to unreliable features and predictions, interfering with the learning of generalizable and robust features. In order to improve VTs’ performance on corrupted and unreliable data, we introduce a Token Boosting Module (TBM) for VTs that can learn to boost the reliability of features during self-supervised pre-training.

TBM is a plug-and-play module that can be placed between layers in the VT, and can be trained in an end-to-end manner. When placed in the intermediate layers, TBM can make features in VTs cleaner and more reliable. To further improve performance, we incorporate TBM into multiple layers of the VT, such that contextual information at multiple feature levels are used for boosting the quality of the features. This enhanced capability of VTs, achieved during self-supervised pre-training, can thus be transferred to downstream tasks to achieve good performance even when facing corrupted inputs.

In this section, we first give a brief background of masked autoencoding. Then, we explain the feature boosting technique and describe TBM in detail. Lastly, we theoretically show why our TBM can benefit pre-trained VTs with more robust and reliable representations.

3.1. Masked Autoencoding Pre-training

Previous works [2,16,20] have explored masked autoencoding to train Transformers, which we follow. In masked autoencoding pre-training [2,16,20], a part of the input is often masked, and the Transformer (with a decoder head that is only used during pre-training) is tasked to reconstruct those masked inputs. Driven by the reconstruction losses, the Transformer will learn to capture semantically meaningful information within the input, producing features that tend to generalize well to other downstream tasks. However, this masked autoencoding approach can be challenging when performed with corrupted data, since both the input and masked target could be corrupted, impairing downstream tasks with biased/unreliable feature representations.

3.2. Feature Boosting Technique

In this subsection, inspired by previous works [43,63], we describe a simple technique to extract robust and reliable features from data corrupted by noise. Specifically, for an unreliable feature \( F \) that is going to be fed into a VT layer, this technique allows us to obtain a reliable boosted feature \( \hat{R} \). The robust feature \( \hat{R} \) (instead of the unreliable feature \( F \)) can then be fed into the VT layer for better performance.

Let \( R \) represent the ground truth reliable features. We are unable to directly observe the reliable feature \( R \), but are able to observe the unreliable feature \( F \) (which are extracted from a real input sample). Importantly, this means that we are unable to simply train a module with direct supervision to produce \( R \) from \( F \). We assume that the difference between \( R \) and \( F \) can be modelled by a feature perturbation \( P \) which is randomly drawn from a distribution \( \mathcal{P} \), i.e., we have \( F = R + P \). Below, we show how, with only access to the unreliable feature \( F \), we can produce \( \hat{R} \) (which statistically speaking is an unbiased conditional expectation [54,56]) that estimates the reliable feature \( R \).

Given \( F \), we first create an intermediate representation, \( I = F + Q \) where \( Q \) is a synthetic corruption drawn from \( \mathcal{P} \) as well, as shown in Fig. 2. Importantly, as \( P \) and \( Q \) are i.i.d., the two sources of corruptions in \( I \) will be indistinguishable to an observer, i.e., we will not be able to tell how much of the corruption comes from \( P \) or \( Q \). We then train an autoencoder \( g \) (parameterized by \( \theta \)) to predict \( F \) given \( I \) as its input, and we denote the output of \( g \) as \( \hat{F} \) (an estimate of \( F \)), as shown in Fig. 2. \( \hat{F} \) will later be used to construct the boosted features \( \hat{R} \).

Specifically, the autoencoder \( g \) is trained with an L2 loss with \( F \) as the target, and will be optimized to reconstruct good estimates of \( F \) given \( I \) as input, i.e., \( F = \mathbb{E}[F|I] \). Notably, in order to reproduce \( F \) perfectly, the autoencoder needs to distinguish between \( P \) and \( Q \), but since it never observes \( P \) and \( Q \) independently, it cannot reproduce \( F \) fully, and instead can only produce its best estimate \( \hat{F} \). Expanding \( \hat{F} \), we observe that \( \hat{F} = \mathbb{E}[F|I] = \mathbb{E}[R + P|I] = \mathbb{E}[R|I] + \mathbb{E}[P|I] \).

Next, as \( Q \) and \( P \) are assumed to be independently drawn from the same distribution, we get \( \mathbb{E}[Q|I] = \mathbb{E}[P|I] \). Intuitively, this can be understood as \( P \) and \( Q \) being indistinguishable from one another, and a simple proof has been provided in the Supplementary. Multiplying the above equation by 2 and performing substitution, we get \( 2\mathbb{E}[F|I] = \mathbb{E}[R|I] + \mathbb{E}[R|I] + \mathbb{E}[P|I] + \mathbb{E}[Q|I]) = \mathbb{E}[R|I] + \mathbb{E}[R + P + Q|I] = \mathbb{E}[R|I] + \mathbb{E}[F + Q|I] = \mathbb{E}[R|I] + I \).

By rearranging, we get \( \mathbb{E}[R|I] = 2\mathbb{E}[F|I] - I \). We can thus obtain \( \hat{R} \) using an estimate \( \hat{F} \) as:

\[
\hat{R} = 2\hat{F} - I. \tag{1}
\]

In summary, by using this technique, we can still estimate the reliable feature \( \hat{R} \) even if we do not have ground truth feature \( R \). This is important, because we do not have access to any ground truth clean data or features (\( R \)).

3.3. Token Boosting Module

Next, we present the Token Boosting Module (TBM) to improve reliability of VTs by incorporating the technique described above. One practical issue we face, is how to sample synthetic corruption \( Q \), which requires us to make assumptions on the distribution \( \mathcal{P} \). Previous methods [27,32,52,70] handling corrupted data show that modelling perturbations in the deep feature space as Gaussian distributed can be very effective for learning approximate complicated and intractable distributions underlying...
Then, we utilize Eq. 1 to obtain a boosted feature \( \hat{R} \), which is used as the input to the next layer of the VT.

**TBM for features.** We insert TBM between self-attention layers in VTs, such that the tokens at the feature layers can be boosted before generating the next set of key, query and value embeddings. In order to further improve the robustness of VTs, we incorporate TBM into multiple layers of the VT. This allows us to enhance the intermediate features of VTs in the presence of different levels of contextual information, which leads to better robustness capabilities.

**L2 Reconstruction Loss.** To train the autoencoder parameters \( \theta \), we introduce an L2 reconstruction loss \( L_{recon} \):

\[
L_{recon}(F, \hat{F}) = \lambda \sum_{k=1}^{K} [F_k - \hat{F}_k]^2
\]

where \( \lambda \) is a hyperparameter that controls the weightage of the loss with respect to the end-to-end pre-training objective. This loss \( L_{recon} \) trains the autoencoder \( g \) to produce good estimates \( \hat{F} \), and can be combined with the end-to-end objective in masked autoencoding pre-training to drive TBM parameters \( \theta \) and \( \alpha \) towards improved robustness capabilities, as described in the next sub-section.

### 3.4. TBM in Masked Autoencoding Pre-training

Our main goal in this paper is to improve the capability of VTs to extract robust features on corrupted and unreliable data during masked autoencoding pre-training, which is a popular and effective paradigm for pre-training VTs. However, one key question remains: can the TBM be meaningfully trained under the masked autoencoding pre-training objective? This is not immediately straightforward, especially since both the inputs and the reconstruction targets are unreliable. In this section, we answer the above question and theoretically show how \( \alpha \) and \( \theta \) can be meaningfully trained with the masked autoencoding objective.

We first show our theoretical motivations in more detail using a linear example for the sake of simplicity. Specifically, we explain how a simple linear regression model, tasked to regress some masked inputs \( V \in \mathbb{R}^{N_V} \) from the unmasked inputs \( U \in \mathbb{R}^{N_U} \), will potentially be able to achieve a lower loss from the use of our TBM. This shows that gradients from the loss will lead to meaningful training of the TBM parameters. In this example, we assume that the best-fit linear relationship between \( U \) and \( V \) can be modelled as \( V = \beta U + c + \epsilon \), where \( \beta \in \mathbb{R}^{N_V \times N_U}, c \in \mathbb{R}^{N_V}, \epsilon \in \mathbb{R}^{N_V} \) respectively refer to the coefficient, intercept and random error coming from a Gaussian with standard deviation of \( \gamma \). We additionally assume that at least some elements in \( \beta \) are non-zero, which means that there is at least some correlation between \( U \) and \( V \).

---

![Figure 2. Illustration of the TBM placed at a VT layer. Firstly, TBM takes in the unreliable input feature \( F \), i.e., all tokens. Synthetic corruption \( Q \) is produced by scaling Gaussian noise \( S \) with learned parameters \( \alpha \). An intermediate representation \( I = F + Q \) is produced, and fed into the autoencoder \( g \) to reconstruct \( \hat{F} \), an estimate of \( F \). Lastly, we apply Eq. 1 with \( \hat{F} \) and \( I \) as input, producing \( \hat{R} \), which is our boosted output feature. \( \hat{R} \) is then fed as input to the next VT layer.

The data. Following them, we apply the Gaussian distribution to model the feature perturbation \( P \sim \mathcal{P} \), with a mean of 0 and unknown standard deviations. Notably, the Gaussian assumption [27, 32, 52, 70] allows us to "parameterize" corruptions in feature space, and we can then optimize the distribution of the added synthetic corruption \( Q \) to be similar to the underlying corruption distribution \( \mathcal{P} \), that fulfills the conditions in Sec. 3.2. We empirically observe this works effectively for handling various types of corruptions and tasks.

Let the input feature of TBM be \( F \in \mathbb{R}^K \). As shown in Fig. 2, our TBM consists of two sets of learnable parameters. Firstly, we have the scaling parameters \( \alpha \in \mathbb{R}^K_{>0} \) that scale the added corruptions. This is necessary to learn the optimal standard deviation parameters of the synthetic corruptions, where a different standard deviation is learned for each of \( K \) feature dimensions. Importantly, our training loss will optimize \( \alpha \) towards making the distribution similar to the natural corruption’s distribution \( \mathcal{P} \), therefore the synthetic corruptions and the natural corruptions can be seen as i.i.d.\(^1\). Secondly, we have the autoencoder \( g \) containing learnable parameters \( \theta \). This autoencoder \( g \) takes in \( F \) with added corruptions \( Q \), and reconstructs \( \hat{F} \), an estimate of \( F \).

We show an illustration of the TBM in Fig. 2. First, we sample a Gaussian noise vector \( S \in \mathbb{R}^K \) with a mean of 0 and standard deviation of 1. The sampled Gaussian noise \( S \) is scaled using the learned parameters \( \alpha \) in an element-wise manner (denoted by \( \circ \)). Note that \( \alpha \circ S \) corresponds to sampling \( K \) Gaussians, with the standard deviation of the Gaussian at each index \( k \) to be \( \alpha_k \). We use \( \alpha \circ S \) as synthetic corruption \( Q \), and add it to \( F \) to form \( I \) as follows:

\[
I = F + (\alpha \circ S).
\]  

Next, the autoencoder \( g \) is used to reconstruct the estimate of \( F \) from \( I \):

\[
\hat{F} = g(I).
\]

\(^1\)We provide detailed analysis that distribution of \( Q \) can be trained to approximate distribution of \( P \) in Sec. 5 of Supplementary.
When trained with adequate data, the fitted linear model (comprising of parameter estimates \( \beta \) and \( c \)) will converge to \( \beta \) and \( c \), with the MSE error as follows:

\[
\mathbb{E}
\left[
\frac{1}{N_V} \sum_{j=1}^{N_V} [V_j - (\beta U + c)]^2
\right] = \frac{1}{N_V} \sum_{j=1}^{N_V} \mathbb{E} [\epsilon_j^2] = \gamma^2,
\]

where the detailed steps are shown in Eq. 23-28 of Supplementary. This (\( \gamma^2 \)) represents the best possible result that the linear model can achieve. However, when “natural” corruptions are added towards all inputs in \( U \) and \( V \) to get \( \tilde{U} \) and \( \tilde{V} \) (where \( \tilde{U} = U + S_U \) and \( \tilde{V} = V + S_V \), with \( S_V \) and \( S_U \) containing elements drawn from a Gaussian with standard deviation \( \sigma_N \)), the MSE will be increased to:

\[
\mathbb{E}
\left[
\frac{1}{N_V} \sum_{j=1}^{N_V} [\tilde{V}_j - (\beta \tilde{U} + c)]^2
\right] = \frac{1}{N_V} \sum_{j=1}^{N_V} \mathbb{E} [\epsilon_j^2] + \mathbb{E} [(S_V)^2_j] + \mathbb{E} [(\beta S_U)^2_j]
\]

\[
= \gamma^2 + \sigma_N^2 + \frac{1}{N_V} \sum_{j=1}^{N_V} \sum_{k=1}^{N_U} \beta_{jk}^2 \sigma_N^2.
\]

Refer to Eq. 29-40 of Supplementary for more details of the above derivations. The increase in MSE from Eq. 5 to Eq. 8 represents the extra difficulty in conducting inference, when there are corruptions in the input data \( U \) (contributing \( \frac{1}{N_V} \sum_{j=1}^{N_V} \sum_{k=1}^{N_U} \beta_{jk}^2 \sigma_N^2 \)), as well as corruptions in the “ground truth” inputs \( V \) (contributing \( \sigma_N^2 \)). We can thus potentially reduce the MSE to \( \gamma^2 + \sigma_N^2 \) by boosting \( \tilde{U} \) (which represents the corrupted inputs) and feeding to the model \( \tilde{U} \approx U \), which our TBM, if placed before our linear regression model, will be optimized to do. In other words, if we can boost \( \tilde{U} \) into a clean version \( \hat{U} \approx U \), our expected loss (with the full proof in Eq. 41-50 of Supplementary) will be:

\[
\mathbb{E}
\left[
\frac{1}{N_V} \sum_{j=1}^{N_V} [\hat{V}_j - (\beta \hat{U} + c)]^2
\right] = \frac{1}{N_V} \sum_{j=1}^{N_V} \mathbb{E} [\epsilon_j^2] + \mathbb{E} [(S_V)^2_j]
\]

\[
= \gamma^2 + \sigma_N^2,
\]

which is lower than the MSE if \( \tilde{U} \) is directly used (shown in Eq. 8). Thus, during the optimization process, when our TBM is updated via the loss gradients that aim to minimize the pre-training loss, our TBM will tend to learn feature boosting capabilities in order to achieve a smaller loss. This enables the feature boosting capabilities (i.e. learned parameters \( \alpha \) and \( \theta \)) of our TBM to be trained, such that it can function as described in Sec. 3.2.

Although the above is just a linear example, we remark that the concepts involved can generalize to non-linear models such as VTs. In general, even for non-linear cases, unreliable inputs will lead to difficulties in masked patch prediction, which will lead to a higher pre-training loss than reliable inputs. Thus, in order to minimize this loss, the gradients will optimize TBM parameters \( \alpha \) and \( \theta \) towards learning how to boost the features, i.e., produce good estimates \( \hat{R} = R \), which the TBM’s design as described in Sec. 3.2 explicitly allows it to accomplish. Our experimental results also demonstrate the efficacy of such a design.

Note that when trained with supervision labels, TBMs can also be trained to boost the reliability of features as well, as input corruptions are likely to negatively affect performance of the supervised task (e.g. 3D action recognition [10, 17, 39, 51], depth image classification [40, 46, 73]) such that the gradient for \( \alpha \) and \( \theta \) will tend to encourage the learning of feature boosting capabilities.

4. Experiments

We first conduct extensive experiments on image classification with added synthetic corruptions, which allows us to systematically vary the type of added corruptions for a more detailed analysis. Then, we test our method on two real-world settings, namely 3D action recognition and depth image classification, where we attain good performance. More experiments can be found in Supplementary.

4.1. Training Setups

In order to evaluate our method more comprehensively, we conduct experiments with two different training setups: the Self-supervised Training setup and the Supervised Training setup, which we explain below.

Self-supervised Training. Following previous works [2, 16, 20], we add a decoder on top of our VT encoder (inserted with some TBM modules) to mask input patches and train our VT encoder+decoder by reconstructing these masked patches using MSE loss. After the end-to-end self-supervised training, we follow the linear probe setting [20] to train a classifier for testing purposes. Specifically, the decoder is replaced by a classification head, and we train the classification head while keeping the parameters in our VT encoder (including our TBMs) frozen. Then, we report the results using the predictions from the VT encoder+classification head.

Supervised Training. We also aim to evaluate the capability of TBM when trained in a supervised setting. Therefore, we train our VT encoder (including the TBM modules), in a normal supervised manner, where a classification head is attached to the end of the VT encoder and the full
model is trained in an end-to-end manner. The classification head is reused at test time for evaluation.

### 4.2. Image Classification

**Dataset.** We conduct both self-supervised and supervised training on ImageNet [15] with the synthetic corruptions from ImageNet-C [21]. ImageNet-C is a corrupted version of the ImageNet test set, and is a commonly used benchmark for robustness evaluation, employing 15 types of synthetically generated corruptions, and some examples can be seen in Fig. 1. ImageNet-C allows us to comprehensively and systematically evaluate the effectiveness of TBM on various types of common corruptions. Specifically, during both supervised and self-supervised training, we follow the procedure for ImageNet-C [21] and add synthetic corruptions for 50% of input images, where the corruption types are randomly selected (among the 15 types). During testing, we evaluate our method on the corrupted ImageNet-C [21] benchmark.

**Network Architecture.** To comprehensively evaluate the efficacy of the proposed TBM, we perform experiments using three different sizes of ViT [16]: ViT-Huge, ViT-Large and ViT-Base, as well as DeiT [58] and Swin Transformer (Swin) [38]. We use the Base versions of DeiT and Swin. Following previous works [2, 20], the pre-training decoders are composed of stacked transformer blocks. The autoencoder $g$ in our proposed TBM is comprised of 3 fully-connected+ReLU layers with hidden layers having dimensionality $input\_dim/2$, where $input\_dim$ depends on the architecture of the VT. Scaling values $\alpha$ have the same shape as the input feature map $F$ to the TBM, which depends on the architecture. Our TBM is inserted at 3 different feature levels, i.e., bottom layer, middle layer, and top layer in our ViT, DeiT and Swin backbones, in order to enhance feature robustness with different levels of contextual information.

**Training Details.** In the self-supervised training phase, we follow existing works [20] and use the AdamW optimizer to train the VTs (for all ViTs, DeiT and Swin) for 400 epochs, where 75% of input patches are randomly masked. The initial learning rate is set to 0.01 and gradually decay to 0 following the cosine rule. In the supervised training phase, we follow [20] to use LARS optimizer for ViT and we use AdamW optimizer for DeiT and Swin. All experiments are conducted on Nvidia A100 GPUs, and $\beta$ is set to 1.

**Results.** Our results are shown in Tab. 1. Overall, our TBM provides significant performance gains for all five baseline VTs in both self-supervised and fully-supervised settings. This shows that TBM is capable of improving the learning on corrupted data.

Next, we conduct more in-depth analysis using ViT-Huge as the backbone model. We evaluate the effects of TBM on various types of corruptions in Tab. 2. We observe that TBM improves performance consistently across all corruption types (noise, blue, weather, digital) used in ImageNet-C. We remark that this includes a 3.1% increase on images corrupted with snow, and a 3.6% increase on images corrupted with motion blur, which are examples of practical yet complicated corruptions to tackle. These results suggest that TBM has the capability to generalize to various types of corruptions. We also conduct experiments on another setting ViT-Huge+TBM (Individual), where a single model is trained to deal with each individual type of corruption. This setting leads to a slight improvement as expected, since this setting allows our TBM to fully specialize in tackling a single type of corruption at a time. Nevertheless, our significant improvements over the baseline show that our method (which boosts the features at multiple levels using TBM) is capable of handling these various synthetic corruptions simultaneously, which is a complicated task.

**Evaluation with data pre-processing methods.** We also want to explore if data pre-processing techniques can be applied to effectively improve the quality of the data before performing self-supervised pre-training. We evaluate the performance of some common smoothing methods, namely the Median Filter and the Gaussian Filter, as well as three image-based pre-processing methods. As shown in Tab. 3, we find that performance improvements are small. This can be because there are many different types of corruptions, and it can be difficult for a single pre-processing technique to tackle them effectively. It can be more effective to perform the boosting and “cleaning” at the feature level, like our TBM does. Furthermore, we also explore using these five data pre-processing techniques in conjunction with our method ([Pre-processing] + ViT-Huge + TBM), where we apply these methods to the input image before feeding them into our ViT-Huge+TBM model. The results when we use these pre-processing techniques in conjunction with our method are comparable to directly using our TBM, suggesting that it is sufficient to directly use our TBM, which indicates the efficacy of our method.

### 4.3. Networks Architecture

**Training Details.** In the self-supervised training phase, we follow existing works [20] and use the AdamW optimizer to train the VTs (for all ViTs, DeiT and Swin) for 400 epochs, where 75% of input patches are randomly masked. The initial learning rate is set to 0.01 and gradually decay to 0 following the cosine rule. In the supervised training phase, we follow [20] to use LARS optimizer for ViT and we use AdamW optimizer for DeiT and Swin. All experiments are conducted on Nvidia A100 GPUs, and $\beta$ is set to 1.

**Results.** Our results are shown in Tab. 1. Overall, our TBM provides significant performance gains for all five baseline VTs in both self-supervised and fully-supervised settings. This shows that TBM is capable of improving the learning on corrupted data.

Next, we conduct more in-depth analysis using ViT-Huge as the backbone model. We evaluate the effects of TBM on various types of corruptions in Tab. 2. We observe that TBM improves performance consistently across all corruption types (noise, blue, weather, digital) used in ImageNet-C. We remark that this includes a 3.1% increase on images corrupted with snow, and a 3.6% increase on images corrupted with motion blur, which are examples of practical yet complicated corruptions to tackle. These results suggest that TBM has the capability to generalize to various types of corruptions. We also conduct experiments on another setting ViT-Huge+TBM (Individual), where a single model is trained to deal with each individual type of corruption. This setting leads to a slight improvement as expected, since this setting allows our TBM to fully specialize in tackling a single type of corruption at a time. Nevertheless, our significant improvements over the baseline show that our method (which boosts the features at multiple levels using TBM) is capable of handling these various synthetic corruptions simultaneously, which is a complicated task.

**Evaluation with data pre-processing methods.** We also want to explore if data pre-processing techniques can be applied to effectively improve the quality of the data before performing self-supervised pre-training. We evaluate the performance of some common smoothing methods, namely the Median Filter and the Gaussian Filter, as well as three image-based pre-processing methods. As shown in Tab. 3, we find that performance improvements are small. This can be because there are many different types of corruptions, and it can be difficult for a single pre-processing technique to tackle them effectively. It can be more effective to perform the boosting and “cleaning” at the feature level, like our TBM does. Furthermore, we also explore using these five data pre-processing techniques in conjunction with our method ([Pre-processing] + ViT-Huge + TBM), where we apply these methods to the input image before feeding them into our ViT-Huge+TBM model. The results when we use these pre-processing techniques in conjunction with our method are comparable to directly using our TBM, suggesting that it is sufficient to directly use our TBM, which indicates the efficacy of our method.

### 4.3. Networks Architecture

**Training Details.** In the self-supervised training phase, we follow existing works [20] and use the AdamW optimizer to train the VTs (for all ViTs, DeiT and Swin) for 400 epochs, where 75% of input patches are randomly masked. The initial learning rate is set to 0.01 and gradually decay to 0 following the cosine rule. In the supervised training phase, we follow [20] to use LARS optimizer for ViT and we use AdamW optimizer for DeiT and Swin. All experiments are conducted on Nvidia A100 GPUs, and $\beta$ is set to 1.

**Results.** Our results are shown in Tab. 1. Overall, our TBM provides significant performance gains for all five baseline VTs in both self-supervised and fully-supervised settings. This shows that TBM is capable of improving the learning on corrupted data.

Next, we conduct more in-depth analysis using ViT-Huge as the backbone model. We evaluate the effects of TBM on various types of corruptions in Tab. 2. We observe that TBM improves performance consistently across all corruption types (noise, blue, weather, digital) used in ImageNet-C. We remark that this includes a 3.1% increase on images corrupted with snow, and a 3.6% increase on images corrupted with motion blur, which are examples of practical yet complicated corruptions to tackle. These results suggest that TBM has the capability to generalize to various types of corruptions. We also conduct experiments on another setting ViT-Huge+TBM (Individual), where a single model is trained to deal with each individual type of corruption. This setting leads to a slight improvement as expected, since this setting allows our TBM to fully specialize in tackling a single type of corruption at a time. Nevertheless, our significant improvements over the baseline show that our method (which boosts the features at multiple levels using TBM) is capable of handling these various synthetic corruptions simultaneously, which is a complicated task.

**Evaluation with data pre-processing methods.** We also want to explore if data pre-processing techniques can be applied to effectively improve the quality of the data before performing self-supervised pre-training. We evaluate the performance of some common smoothing methods, namely the Median Filter and the Gaussian Filter, as well as three image-based pre-processing methods. As shown in Tab. 3, we find that performance improvements are small. This can be because there are many different types of corruptions, and it can be difficult for a single pre-processing technique to tackle them effectively. It can be more effective to perform the boosting and “cleaning” at the feature level, like our TBM does. Furthermore, we also explore using these five data pre-processing techniques in conjunction with our method ([Pre-processing] + ViT-Huge + TBM), where we apply these methods to the input image before feeding them into our ViT-Huge+TBM model. The results when we use these pre-processing techniques in conjunction with our method are comparable to directly using our TBM, suggesting that it is sufficient to directly use our TBM, which indicates the efficacy of our method.

### Layers to add TBM

In our method, we apply TBM across...
s several layers spanning from the bottom to the top of the VT. Here, we ablate this decision by evaluating several alternative designs on ImageNet-C, with results shown in Tab. 4. **Top layer**, **Mid layer** and **Bot layer** are settings where the TBM is only applied to the top layer, the middle layer, or the bottom layer. Compared to using it in any single layer, our incorporation into multiple layers performs better, demonstrating the efficacy of exploiting context information at different feature levels.

Table 4. Evaluation of TBM applied to different layers. We also conduct testing with TBM incorporated into more layers, and do not find further improvement.

<table>
<thead>
<tr>
<th>Settings</th>
<th>Baseline [16]</th>
<th>Top layer</th>
<th>Mid layer</th>
<th>Bot layer</th>
<th>Ours (All 3 layers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc. (%)</td>
<td>35.2</td>
<td>37.2</td>
<td>37.1</td>
<td>37.3</td>
<td>38.6</td>
</tr>
</tbody>
</table>

**Weightage of losses.** We conduct ablation studies on the impact of \( \lambda \) in Tab. 5. \( \lambda = 1 \) performs the best. When \( \lambda \) is too low, the TBM is not trained well, and when \( \lambda \) is too high, the VT may lose focus on the main objective. Specifically, when we set \( \lambda = 0 \), we do not train with the reconstruction loss, which effectively disables our boosting mechanism, leading to sub-optimal performance. Notably, when \( \lambda = 0 \), the model also contains the same number of model parameters as our main ViT-Huge+TBM setting reported in Tab. 1, but performs significantly worse, which indicates that the benefits of our TBM do not merely come from the slightly (~ 5%) increased number of parameters.

**Evaluation on clean ImageNet.** We also evaluate the accuracy of ViT-Huge+TBM to be 85.5% on the clean test set of ImageNet, while the baseline ViT-Huge achieves an accuracy of 85.1%. As accuracy remains about the same as the original ViT-Huge, we conclude that the benefits of TBM mainly lie in improved robustness against corrupted data.

### 4.3. 3D Skeleton Action Recognition

In 3D skeleton action recognition, models are tasked to classify the performed action using 3D skeleton sequences as input. However, there are often errors in the 3D skeleton data [37], which will negatively affect performance. Thus, this is a real-world setting where we can evaluate the effectiveness of our TBM.

**Dataset.** NTU RGB+D 60 (NTU60) [48] is a large dataset that has been widely used for 3D action recognition, containing approximately 56k RGB-D sequences from 60 activity classes collected with Kinect v2 sensors. NTU RGB+D 120 (NTU120) [36] is an extension of NTU60, and is currently the largest RGB+D dataset for 3D action analysis, containing more than 114k skeletal sequences across 120 activity classes. We follow the standard evaluation protocols of [36, 48] for 3D action recognition, where the Cross-Subject (xsub) and Cross-View (xview) protocols are tested for NTU60, while the Cross-Subject (xsub) and Cross-Setup (xset) protocols are tested for NTU120.

**Network Architecture.** We perform experiments using DSTA-Net [51], which is specifically designed for processing skeleton data, and currently achieves state-of-the-art performance among VTs in this field. Following previous works [13, 69], we design the decoder (used during pre-training) as a stack of GCN blocks. Our TBM settings here are similar to the image classification experiments, where autoencoder \( g \) consists of 3 fully-connected+ReLU layers, and TBM is inserted at 3 different feature levels, from low to high.

**Pre-training Settings.** For masked autoencoding pre-training for 3D sequences, we randomly mask “spatial” patches as well as “temporal” patches, and task DSTA-Net to reconstruct them. Thus, DSTA-Net will be trained to encode information about both the spatial aspect of the human body and the temporal aspect of motions, which will lead to

<table>
<thead>
<tr>
<th>Methods</th>
<th>Noise</th>
<th>Blur</th>
<th>Weather</th>
<th>Digital</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-Huge [16]</td>
<td>36.7</td>
<td>35.6</td>
<td>35.0</td>
<td>34.3</td>
<td>32.5</td>
</tr>
<tr>
<td>ViT-Huge+TBM</td>
<td>39.5</td>
<td>39.2</td>
<td>38.7</td>
<td>37.6</td>
<td>30.4</td>
</tr>
<tr>
<td>ViT-Huge+TBM (individual)</td>
<td>39.8</td>
<td>39.4</td>
<td>39.1</td>
<td>37.9</td>
<td>30.7</td>
</tr>
</tbody>
</table>

Table 2. Performance comparison (%) of TBM on different types of corruptions on ImageNet-C.
its features capturing generalizable information that benefits downstream tasks. More details are in Supplementary.

**Training Details.** In the self-supervised training phase, we train our model for 200 epochs with batch size of 256 using the AdamW optimizer. We set the initial learning rate to $0.01$, and set it to gradually decay to $0$ following the cosine rule. $\lambda$ is set to $1$. In the supervised training phase, we use SGD optimizer following [51].

**Results on Self-Supervised Training.** We compare with the current state-of-the-art self-supervised models in Tab. 6. It can be observed that adding our TBM to DSTA-Net and then conducting pre-training improves the performance of DSTA-Net consistently across all settings. This also includes a significant $6.0\%$ and $4.0\%$ improvement over DSTA-Net baselines on NTU60 xsub and xview respectively. We also highlight that DSTA-Net+TBM achieves good results on all reported settings.

**Table 6. Performance comparison (%) on 3D Skeleton Action Recognition.** Models are pre-trained in a self-supervised manner, and tested with a linear probe. We attain good performance on all settings.

<table>
<thead>
<tr>
<th>Methods</th>
<th>NTU60</th>
<th>NTU120</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>xsub</td>
<td>xview</td>
</tr>
<tr>
<td>LongT GAN [76]</td>
<td>39.1</td>
<td>52.1</td>
</tr>
<tr>
<td>MS2L [34]</td>
<td>52.6</td>
<td>-</td>
</tr>
<tr>
<td>P&amp;C FS-AEC [57]</td>
<td>50.6</td>
<td>76.3</td>
</tr>
<tr>
<td>P&amp;C Fw-AEC [57]</td>
<td>50.7</td>
<td>76.1</td>
</tr>
<tr>
<td>SeBiReNet [44]</td>
<td>-</td>
<td>79.7</td>
</tr>
<tr>
<td>EntGAN-PoseRNN [25]</td>
<td>68.6</td>
<td>77.8</td>
</tr>
<tr>
<td>T-Colorization [67]</td>
<td>71.6</td>
<td>79.9</td>
</tr>
<tr>
<td>TS-Colorization [67]</td>
<td>74.6</td>
<td>82.6</td>
</tr>
<tr>
<td>TSP-Colorization [67]</td>
<td>75.2</td>
<td>83.1</td>
</tr>
<tr>
<td>SkeletonCLR [29]</td>
<td>68.3</td>
<td>76.4</td>
</tr>
<tr>
<td>3s-SkeletonCLR [29]</td>
<td>75.0</td>
<td>79.8</td>
</tr>
<tr>
<td>CroSCLR [29]</td>
<td>68.3</td>
<td>76.4</td>
</tr>
<tr>
<td>3s-CroSCLR [29]</td>
<td>77.8</td>
<td>83.4</td>
</tr>
<tr>
<td>DSTA-Net [51]</td>
<td>73.1</td>
<td>81.5</td>
</tr>
<tr>
<td>DSTA-Net+TBM</td>
<td>79.1</td>
<td>85.5</td>
</tr>
</tbody>
</table>

**Results on Supervised Training.** In Tab. 7, we show the results of training DSTA-Net in a supervised manner for 3D skeleton action recognition. Applying TBM to DSTA-Net provides improvement gains over the baseline DSTA-Net over all reported settings, including a $1.6\%$ gain on NTU60 xsub and $2.5\%$ gain on NTU120 xsub. This allows our DSTA-Net+TBM model to achieve good results for all reported settings.

**Table 7. Performance comparison (%) on 3D Skeleton Action Recognition with models trained in a fully-supervised manner.** We follow the setting of [36, 48].

<table>
<thead>
<tr>
<th>Methods</th>
<th>NTU60</th>
<th>NTU120</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>xsub</td>
<td>xview</td>
</tr>
<tr>
<td>ST-LSTM+TG [37]</td>
<td>69.2</td>
<td>77.7</td>
</tr>
<tr>
<td>ST-GCN [65]</td>
<td>81.5</td>
<td>88.3</td>
</tr>
<tr>
<td>AS-GCN [30]</td>
<td>86.8</td>
<td>94.2</td>
</tr>
<tr>
<td>2s-AGCN [50]</td>
<td>88.5</td>
<td>95.1</td>
</tr>
<tr>
<td>DGNN [49]</td>
<td>89.9</td>
<td>96.1</td>
</tr>
<tr>
<td>SGN [75]</td>
<td>89.0</td>
<td>94.5</td>
</tr>
<tr>
<td>Shift-GCN [12]</td>
<td>90.7</td>
<td>96.5</td>
</tr>
<tr>
<td>MS-G3D [39]</td>
<td>91.5</td>
<td>96.2</td>
</tr>
<tr>
<td>FGCN [9]</td>
<td>90.2</td>
<td>96.3</td>
</tr>
<tr>
<td>CTR-GCN [10]</td>
<td>92.4</td>
<td>96.8</td>
</tr>
<tr>
<td><strong>DSTA-Net</strong></td>
<td>79.1</td>
<td>85.5</td>
</tr>
<tr>
<td><strong>DSTA-Net+TBM</strong></td>
<td>93.1</td>
<td>97.0</td>
</tr>
</tbody>
</table>

**4.4. Depth Image Classification**

In depth image classification, models are tasked to classify images using only the depth image as input (without RGB data). This is challenging as the depth cameras often induce corruptions in the data, which leads to noticeable perturbations and noise. Note that here, we only use the depth image for classification.

**Table 8. Performance comparison (%) on Depth Image Classification using WRGBD dataset based on depth data only.**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Supervised</th>
<th>Self-supervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-RNN [53]</td>
<td>78.9</td>
<td>-</td>
</tr>
<tr>
<td>DECO [5]</td>
<td>84.0</td>
<td>-</td>
</tr>
<tr>
<td>VGGf-RNN [4]</td>
<td>84.0</td>
<td>-</td>
</tr>
<tr>
<td>RCFusion [40]</td>
<td>85.9</td>
<td>-</td>
</tr>
<tr>
<td>HP-CNN [73]</td>
<td>85.0</td>
<td>-</td>
</tr>
<tr>
<td>MMFLAN [46]</td>
<td>84.0</td>
<td>-</td>
</tr>
<tr>
<td>ViT-Huge [16]</td>
<td>82.3</td>
<td>69.7</td>
</tr>
<tr>
<td>ViT-Huge+TBM</td>
<td>86.1</td>
<td>71.5</td>
</tr>
</tbody>
</table>

**Dataset.** The Washington RGB-D Object (WRGBD) dataset [26] contains 300 common household objects from 51 different categories. These objects are filmed using RGB-D cameras from multiple angles.

**Implementation Details.** We perform experiments using ViT-Huge [16], and follow the architecture and training details from our image classification experiments.

**Results.** We evaluate our method on depth image classification in Tab. 8. TBM provides a $3.8\%$ improvement on the supervised setting and a $1.8\%$ improvement on the self-supervised setting. This further verifies the efficacy of TBM at tackling tasks with corrupted data.

**5. Conclusion**

In this work, we propose a novel TBM for robust self-supervised VT pre-training on corrupted data. We also theoretically motivate how TBM can learn its feature boosting capabilities using the masked autoencoding pre-training task. When incorporated into VTs and trained in a self-supervised or fully-supervised manner, our TBM leads to performance improvements on corrupted data.

**Acknowledgments.** This work is supported by MOE AcRF Tier 2 (Proposal ID: T2EP20222-0035), National Research Foundation Singapore under its AI Singapore Programme (AISG-100E-2020-065), and SUTD SKI Project (SKI 2021_02_06). This work is also supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.
References


[20] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021. 1, 2, 3, 5, 6


[24] Ziyu Jiang, Tianlong Chen, Ting Chen, and Zhangyang Wang. Robust pre-training by adversarial contrastive learn-


[33] Yaqian Liang, Shanshan Zhao, Baosheng Yu, Jing Zhang, and Fazhi He. Meshmae: Masked autoencoders for 3d mesh data analysis, 2022.


[71] David Junhao Zhang, Kunchang Li, Yali Wang, Xunpeng Chen, Shashwat Chandra, Yu Qiao, Luqi Liu, and

