Gain from Neighbors: Boosting Model Robustness in the Wild via Adversarial Perturbations Toward Neighboring Classes

Supplementary Material

A. Sample Number vs. Similarity Rank k

Here, we provide a more detailed explanation of some of the results presented in Sec. 3.1.

A.1. Similarity Rank k

First, we followed common practices to define class embeddings as the vectors corresponding to the weight matrix of the final fully connected layer in the classification model. Assuming the [CLS] token in ViT has a dimension of $\mathbb{R}^{1\times768}$ and ImageNet-1K contains 1000 classes, the weight matrix of the head layer, i.e., the fully connected layer, is $W \in \mathbb{R}^{768\times1000}$. Then, the class embedding for the c-th class is defined as $\boldsymbol{z}_c = W[:, c]$. And the similarity matrix $\mathcal{M} = W^T W \in \mathbb{R}^{1000\times1000}$, where $\mathcal{M}_{ij} = \boldsymbol{z}_i \cdot \boldsymbol{z}_j$. Below is the pseudocode for calculating the similarity rank k between the mispredicted class p and the ground truth class c:

Algorithm 1 Compute Similarity Rank k Between Prediction p and Ground Truth c.

Input: Class embeddings $\{\boldsymbol{z}_i\}_{i=1}^{C} \in \mathbb{R}^d$, ground truth class c, predicted class p.
Output: Similarity rank k of prediction p with respect to ground truth c.

```
%Step 1: Compute similarity matrix M.
for i ∈ {1, ..., C} do
    for j ∈ {1, ..., C} do
        M_ij ← z_i · z_j^T
    end for
end for
%Step 2: Extract similarities for ground truth class c.
m ← M[c, :] ← {M_c1, M_c2, ..., M_cC}
%Step 3: Sort similarities in descending order.
m_sorted ← Sort(m, descending)
%Step 4: Find rank k for prediction p.
k ← Index(M_cp, m_sorted)
return k ∈ {1, ..., C}
```

Similarity Matrix $\mathcal{M}$: The similarity between classes is computed as the dot product of their embeddings.

Extract Similarities for Class c: The similarity values for the ground truth class c are retrieved.

Rank Computation: The similarities are sorted in descending order, and the similarity rank of the mispredicted class p is determined.

This pseudocode outlines the key steps for computing the similarity rank k.

A.2. Histogram of Rank k

Next, we analyze the relationship between k and the number of samples in wrong set $\mathcal{S}$ described in Sec. 3.1. We compute each sample's similarity rank and obtain a frequency distribution histogram. The pseudocode for this process is shown below.

Algorithm 2 Compute Frequency Distribution of Similarity Rank k

Input: Wrong set $\mathcal{S} = \{\boldsymbol{x}_i, p_i, y_i\}_{i=1}^{N}$, where ground truth labels $\{y_i\}$ and predicted incorrect labels $\{p_i\}$.
Output: Frequency distribution histogram of similarity rank k.

```
Initialize histogram H_k ← 0 for all ranks k ∈ {1, ..., C}.
Compute similarity matrix M as in Algorithm 1
for each sample (x_i, p_i, y_i) in S do
    Extract similarities for label y_i: m ← M[y_i, :]
    Obtain the sorted similarities: m_sorted
    Compute similarity rank k of predicted class p_i as in Algorithm 1: k ← Index(M_{y_i,p_i}, m_sorted)
    Increment histogram: H_k ← H_k + 1
end for
return Histogram H_k
```

We conducted this process across all 19 types of degradations in ImageNet-C, including Noise (gaussian_noise, shot_noise, impulse_noise), Blur (defocus_blur, glass_blur, motion_blur, zoom_blur), Weather (snow, frost, fog, brightness), Digital (contrast, elastic_transform, pixelate, jpeg_compression), and Others (speckle_noise, gaussian_blur, spatter, saturate), and the resulting distribution plots are shown in Fig. 7. Observing these plots, we can conclude that *misclassifications caused by image degradations are often concentrated in categories with high similarity to the ground truth labels.*

B. Pixel Intensity Under Various Degradation

We also investigated the impact of degradations (distribution shift) on pixel intensity as described in Sec. 3.1. Here,

we provide additional and more detailed visualization results. Specifically, we randomly selected five images and applied three randomly chosen types of degradation from the 19 available in ImageNet-C, each with severity levels ranging from 1 to 5. The resulting changes in pixel intensity distributions are illustrated in Fig. 8. From these images, it can be observed that the degradations primarily affect the overall intensity of the pixel values without significantly altering the underlying structure of the images. The underlying structure represents the essential features of objects within the image. The model's outputs could become unpredictable if the structure undergoes significant changes. In contrast, the subtle overall distributional shifts observed here (as illustrated in Fig. 8) often lead the model to predict semantically similar neighboring classes, further corroborating our previous conclusions.

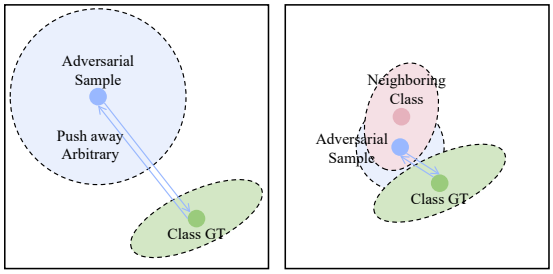
C. The Difference Between Our Method and Standard Adversarial Training

In Sec. 3.2, we introduced a method for computing the classification loss of neighboring classes on input data and applying perturbations based on the corresponding gradients, which can be written as Eq. (5):

$$\boldsymbol{x'} = \boldsymbol{x} - \eta * sign(\nabla_{\boldsymbol{x}} \mathcal{L}_{ce}(\boldsymbol{\theta}, \boldsymbol{x}, y_p)). \tag{5}$$

The standard adversarial training described in Sec. 2.1 in Eq. (3) is also included here for ease of comparison:

$$\boldsymbol{x'} = \boldsymbol{x} + \eta * sign(\nabla_{\boldsymbol{x}} \mathcal{L}_{ce}(\boldsymbol{\theta}, \boldsymbol{x}, y)). \tag{3}$$



(a) Standard adversarial training (b) Our neighboring perturbation

Figure 6. The difference between our method and standard adversarial training.

The fundamental difference between the two approaches lies in the target class used to compute the adversarial perturbations. Based on the direction and meaning of the gradients, our method in Eq. (5) perturbs the input towards neighboring class y_p, aiming to mimic the subtle ambiguities between visually or semantically related classes and performs adversarial training to distinguish them, thereby enhancing the model's robustness clearly. In contrast, standard adversarial training in Eq. (3) perturbs the input away from the ground-truth class y, aiming to defend against extreme misclassifications caused by large deviations under adversarial conditions. In other words, standard adversarial training is tailored to counter adversarial attacks, while our method is more suited for handling degraded images encountered *in the wild*.

Additionally, from an optimization perspective, the difference between the two methods can be visualized in Figs. 1(a) and 1(b), which are simplified into the schematic illustration left Fig. 6.

As shown in this figure, in standard adversarial training, the optimization process tends to be more challenging because the adversarial perturbations push the input samples towards regions far from the decision boundary, often into highly non-linear areas of the loss landscape. This results in increased complexity for the model to effectively minimize the adversarial loss.

In contrast, we focus on perturbing the input towards neighboring classes, which are closer to the decision boundary and semantically similar. This approach creates a smoother and more constrained optimization landscape, reducing the convergence difficulty while effectively improving the model's robustness against real-world degradations.

Moreover, we present the detailed implementation of our perturbation and training method in Algorithm 3. Where for details about $\boldsymbol{m}$ and $\mathcal{M}$, please refer to Algorithm 1.

Algorithm 3 Gradient Perturbation Adversarial Training

Input: Training data $\{\boldsymbol{x}, y\}$, learning rate lr, class similarity matrix $\mathcal{M}$, model f with parameters $\boldsymbol{\theta}$, similarity rank threshold k, step size η of adversarial perturbation.
Output: Updated model parameters $\boldsymbol{\theta}^*$

$\boldsymbol{x}.requires\_grad = True$
for each minibatch $\{\boldsymbol{x}_B, y_B\}$ in training data **do**
  for each sample $\{\boldsymbol{x}_i, y_i\}$ in the minibatch **do**
    `% Step 1: Obtain the class p.`
    $\boldsymbol{m} \leftarrow \{\mathcal{M}_{y_i 1}, \ldots, \mathcal{M}_{y_i C}\} \leftarrow \mathcal{M}[y_i, :]$
    $value, index \leftarrow \boldsymbol{m}.topk(k)$
    $p \leftarrow index[random.randint(0, k-1)]$
    `% Step 2: Generate adversarial samples.`
    for every step in total perturbation steps **do**
      $\boldsymbol{x'}_i \leftarrow \boldsymbol{x}_i - \eta * sign(\nabla_{\boldsymbol{x}_i} \mathcal{L}_{ce}(\boldsymbol{\theta}, \boldsymbol{x}_i, y_p))$
      $\boldsymbol{x}_i \leftarrow \boldsymbol{x'}_i$
    end for
  end for
  `% Step 3: Perform adversarial training.`
  $\mathcal{L} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}_{ce}(f(\boldsymbol{\theta}, \boldsymbol{x'}_i), y_i)$
  $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta} - lr \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}$
end for
return Updated model parameters $\boldsymbol{\theta}^*$.

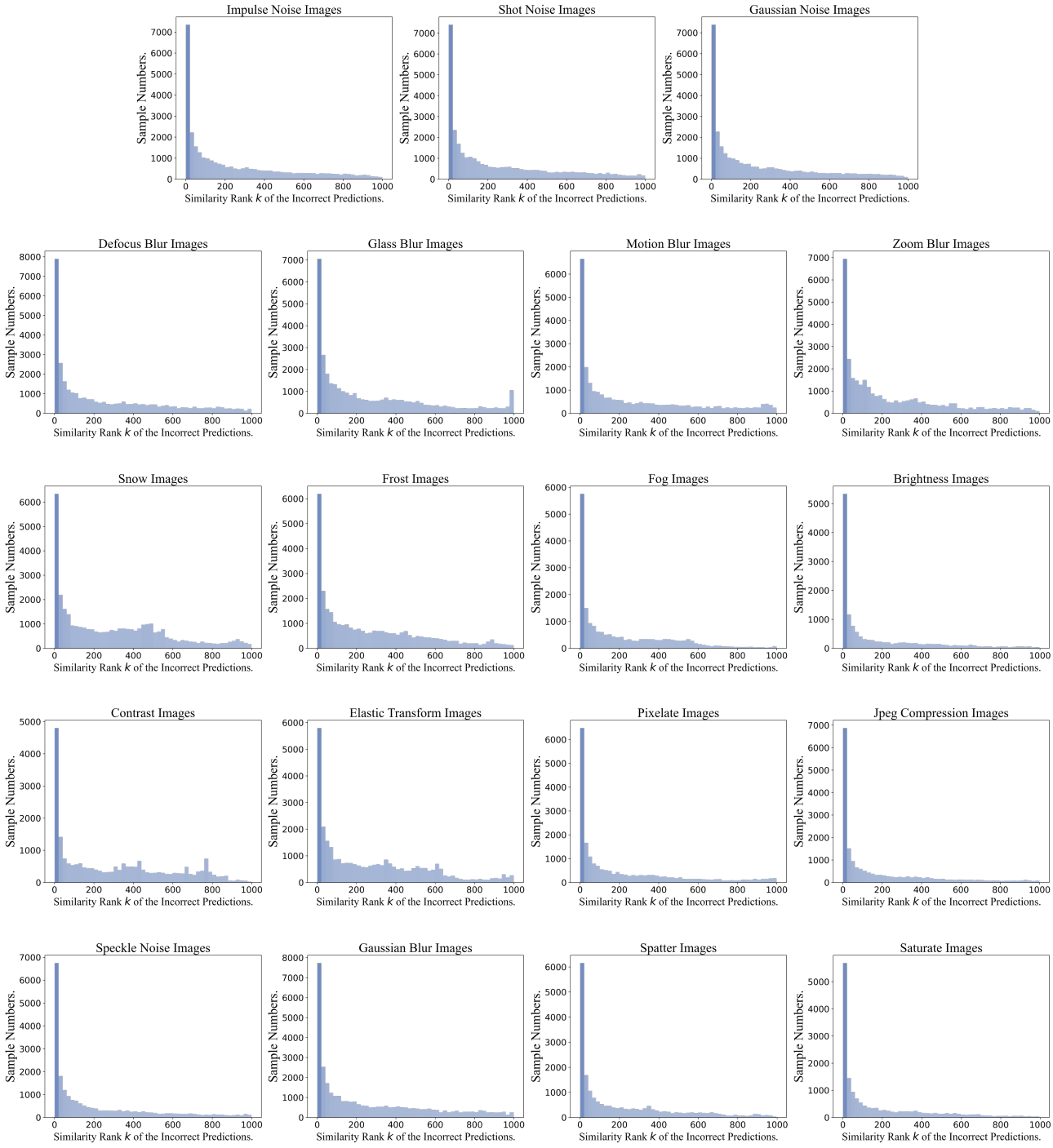
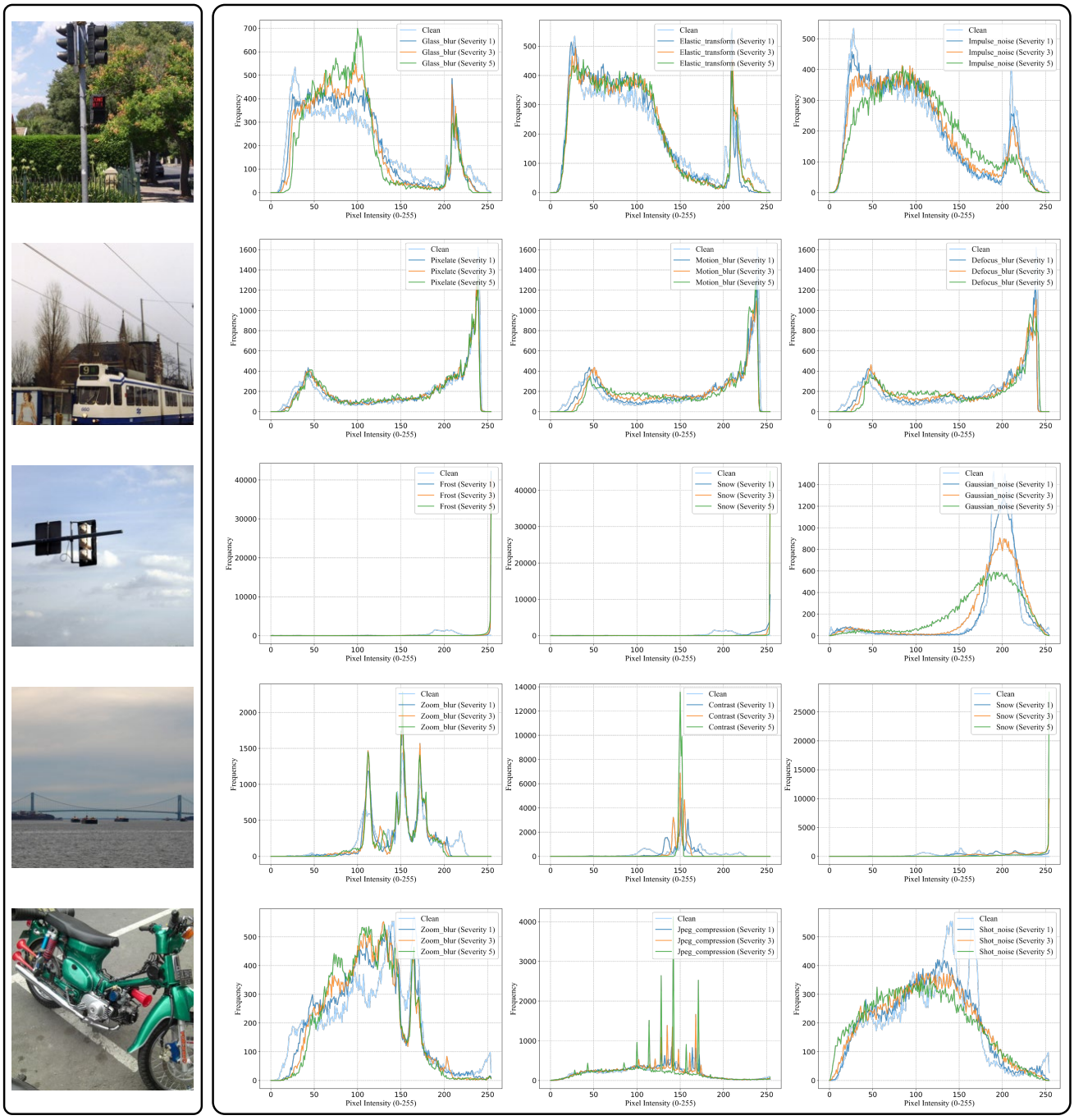


Figure 7. The histogram of sample numbers vs. similarity rank k of incorrect prediction p and ground-truth label c.



(a) Clean Image (b) Pixel Intensity

Figure 8. The pixel intensity distribution changes of clean images and their degraded versions.

Method	Average	Noise			Blur				Weather				Digital				Others			
		Gaussian	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Brightness	Contrast	Elastic	Pixelate	JPEG	Speckle Noise	Gaussian Blur	Spatter	Saturate
ViT-Base	59.20	59.27	56.39	58.44	51.20	45.29	58.39	46.01	43.09	44.78	69.55	73.06	71.41	58.22	67.27	66.67	63.91	54.06	66.68	71.06
FAN-B-H	65.75	67.70	67.18	67.98	57.95	47.20	62.58	56.11	64.79	63.34	69.76	78.86	70.69	60.07	62.10	69.29	71.23	60.82	73.64	77.99
TAPADL(FAN)	66.71	**67.83**	**68.94**	**68.73**	57.69	48.19	65.34	57.73	**65.71**	**64.87**	70.92	79.25	71.52	60.48	65.71	70.26	**71.76**	60.13	**74.25**	**78.10**
Ours(ViT-B)	**67.56**	64.79	63.05	63.55	**63.21**	**56.21**	**67.39**	**58.31**	61.75	61.51	**74.49**	**79.78**	**73.50**	**64.17**	**74.53**	**72.17**	70.61	**65.27**	72.19	77.19

Table 6. Detailed top-1 accuracy performance on 19 types of degraded images in ImageNet-C.

D. Additional Experimental Results

Here, we present the accuracy results of our method compared to state-of-the-art approaches on 19 types of degradations in ImageNet-C, as listed in Tab. 6. As shown in the table, the results demonstrate that our method consistently outperforms state-of-the-art approaches across various degradations, highlighting its effectiveness in improving robustness. Notably, our method does not introduce any additional parameters. The superior performance of TAPADL [14] on noisy images can be attributed to its incorporation of additional convolutional layers for smoothing. In contrast, our approach significantly enhances the performance of ViT-Base without adding any extra parameters, achieving an improvement, for instance, from 59% to 67%. Similarly, our method also reduces the mCE (mean Corruption Error, where lower is better) of FAN from 46% to 39%, as shown in Tab. 1 of the main text. This demonstrates the effectiveness and versatility of our approach across different architectures.