BiasAdv: Bias-Adversarial Augmentation for Model Debiasing
- *Supplementary Material* -

Jongin Lim[1*] Youngdong Kim[1] Byungjai Kim[1] Chanho Ahn[1]
Jinwoo Shin[2] Eunho Yang[2] Seungju Han[1]
[1]Samsung Advanced Institute of Technology (SAIT)
[2]Korea Advanced Institute of Science and Technology (KAIST)

This document presents additional descriptions and results that are not included in the manuscript due to the page limit. In Section A, we present pseudo codes of the training procedures when the proposed BiasAdv is applied to ERM, JTT [9], and LfF [11], respectively. In Section B, we present sensitive analysis on the hyper-parameters of BiasAdv. In Section C, we provide additional qualitative results. In Section D, we describe further details of our experimental setup.

A. Training with BiasAdv

In this work, we apply BiasAdv to three different methods to verify its effectiveness and general applicability: a vanilla ERM (ERM + BiasAdv), JTT [9] (JTT + BiasAdv), and LfF [11] (LfF + BiasAdv). In the case of ERM + BiasAdv, since the vanilla ERM does not have an auxiliary model, we introduce an auxiliary model and train it with the GCE [15] loss, as in [6, 11]. For JTT and LfF, BiasAdv is seamlessly integrated by utilizing the auxiliary model used in their method. BiasAdv does not change the architecture or algorithm of LfF and JTT. The pseudo codes of the training procedure for ERM + BiasAdv, JTT + BiasAdv, and LfF + BiasAdv are presented in Algorithm 1, Algorithm 2, and Algorithm 3, respectively. The newly added parts by BiasAdv are marked in blue. Note that BiasAdv is easy to implement with just a few lines that are easily applicable to any debiasing methods based on the biased auxiliary model. Therefore, BiasAdv can be a very practical and effective solution for learning debiased representations.

Algorithm 1 Training procedure of ERM + BiasAdv

Require: Training dataset $\mathcal{D}$, neural networks f_θ and g_ϕ, cross entropy loss $\mathcal{L}$, generalized cross entropy loss $\mathcal{L}_g$, adversarial perturbation ϵ, hyper-parameters λ and β, learning rate η, batch size B, number of iterations T.

Initialize two networks $f_\theta(x;\theta)$ and $g_\phi(x;\phi)$ ▷ f_θ: debiased model, g_ϕ: biased model
for $t = 1, 2, \cdots, T$ **do**
  Draw a mini-batch $\mathcal{B} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{B}$ from $\mathcal{D}$
  for $i = 1, 2, \cdots, B$ **do**
    $x_{\text{adv}}^{(i)} \leftarrow \underset{\tilde{x}^{(i)}:=x^{(i)}+\epsilon}{\text{argmax}} \left[\mathcal{L}(\tilde{x}^{(i)}, y^{(i)};\phi) - \lambda \cdot \mathcal{L}(\tilde{x}^{(i)}, y^{(i)};\theta)\right]$ ▷ Adversarial attack using PGD [10]
  end for
  $\mathcal{R}_g(\phi) \leftarrow \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}_g(x^{(i)}, y^{(i)};\phi)$
  $\phi \leftarrow \phi - \eta\nabla_\phi \mathcal{R}_g(\phi)$ ▷ Update g_ϕ
  $\mathcal{R}_f(\theta) \leftarrow \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}(x^{(i)}, y^{(i)};\theta) + \frac{\beta}{B}\sum_{i=1}^{B} \mathcal{L}(x_{\text{adv}}^{(i)}, y^{(i)};\theta)$
  $\theta \leftarrow \theta - \eta\nabla_\theta \mathcal{R}_f(\theta)$ ▷ Update f_θ
end for

*Corresponding author: jonny.lim@samsung.com

Algorithm 2 Training procedure of JTT + BiasAdv

Require: Training dataset $\mathcal{D}$, neural networks f_θ and g_ϕ, cross entropy loss $\mathcal{L}$, adversarial perturbation ϵ , hyper-parameters λ and β , learning rate η, batch size B, number of up-sampling N_{up}, number of iterations T_1 and T_2.

Initialize two networks $f_\theta(x;\theta)$ and $g_\phi(x;\phi)$ ▷ f_θ: debiased model, g_ϕ: biased model

for $t = 1, 2, \cdots, T_1$ **do** ▷ Train g_ϕ on $\mathcal{D}$ via ERM for T_1 steps

  Draw a mini-batch $\mathcal{B} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{B}$ from $\mathcal{D}$

  $\mathcal{R}_g(\phi) \leftarrow \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}(x^{(i)}, y^{(i)}; \phi)$

  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{R}_g(\phi)$ ▷ Update g_ϕ

end for

$E \leftarrow \{(x_i, y_i) \text{ s.t. } g_\phi(x_i) \neq y_i\}$ ▷ Construct an error set E

Construct up-sampled dataset $\mathcal{D}_{\text{up}}$ containing examples in the error set N_{up} times and all other examples once.

for $t = 1, 2, \cdots, T_2$ **do** ▷ Train f_θ on $\mathcal{D}_{\text{up}}$ via ERM for T_2 steps

  Draw a mini-batch $\mathcal{B} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{B}$ from $\mathcal{D}_{\text{up}}$

  for $i = 1, 2, \cdots, B$ **do**

    $x_{\text{adv}}^{(i)} \leftarrow \underset{\tilde{x}^{(i)}:=x^{(i)}+\epsilon}{\text{argmax}} \left[\mathcal{L}(\tilde{x}^{(i)}, y^{(i)}; \phi) - \lambda \cdot \mathcal{L}(\tilde{x}^{(i)}, y^{(i)}; \theta)\right]$ ▷ Adversarial attack using PGD [10]

  end for

  $\mathcal{R}_f(\theta) \leftarrow \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}(x^{(i)}, y^{(i)}; \theta) + \frac{\beta}{B}\sum_{i=1}^{B} \mathcal{L}(x_{\text{adv}}^{(i)}, y^{(i)}; \theta)$

  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{R}_f(\theta)$ ▷ Update f_θ

end for

Algorithm 3 Training procedure of LfF + BiasAdv

Require: Training dataset $\mathcal{D}$, neural networks f_θ and g_ϕ, cross entropy loss $\mathcal{L}$, generalized cross entropy loss $\mathcal{L}_g$, adversarial perturbation ϵ , hyper-parameters λ and β , learning rate η, batch size B, number of iterations T.

Initialize two networks $f_\theta(x;\theta)$ and $g_\phi(x;\phi)$ ▷ f_θ: debiased model, g_ϕ: biased model

for $t = 1, 2, \cdots, T$ **do**

  Draw a mini-batch $\mathcal{B} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{B}$ from $\mathcal{D}$

  for $i = 1, 2, \cdots, B$ **do**

    $\omega_x^{(i)} \leftarrow \frac{\mathcal{L}(x^{(i)}, y^{(i)}; \phi)}{\mathcal{L}(x^{(i)}, y^{(i)}; \phi) + \mathcal{L}(x^{(i)}, y^{(i)}; \theta)}$

    $x_{\text{adv}}^{(i)} \leftarrow \underset{\tilde{x}^{(i)}:=x^{(i)}+\epsilon}{\text{argmax}} \left[\mathcal{L}(\tilde{x}^{(i)}, y^{(i)}; \phi) - \lambda \cdot \mathcal{L}(\tilde{x}^{(i)}, y^{(i)}; \theta)\right]$ ▷ Adversarial attack using PGD [10]

    $\omega_{\text{adv}}^{(i)} \leftarrow \beta \cdot \left(1 - \omega_x^{(i)}\right)$

  end for

  $\mathcal{R}_g(\phi) \leftarrow \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}_g(x^{(i)}, y^{(i)}; \phi)$

  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{R}_g(\phi)$ ▷ Update g_ϕ

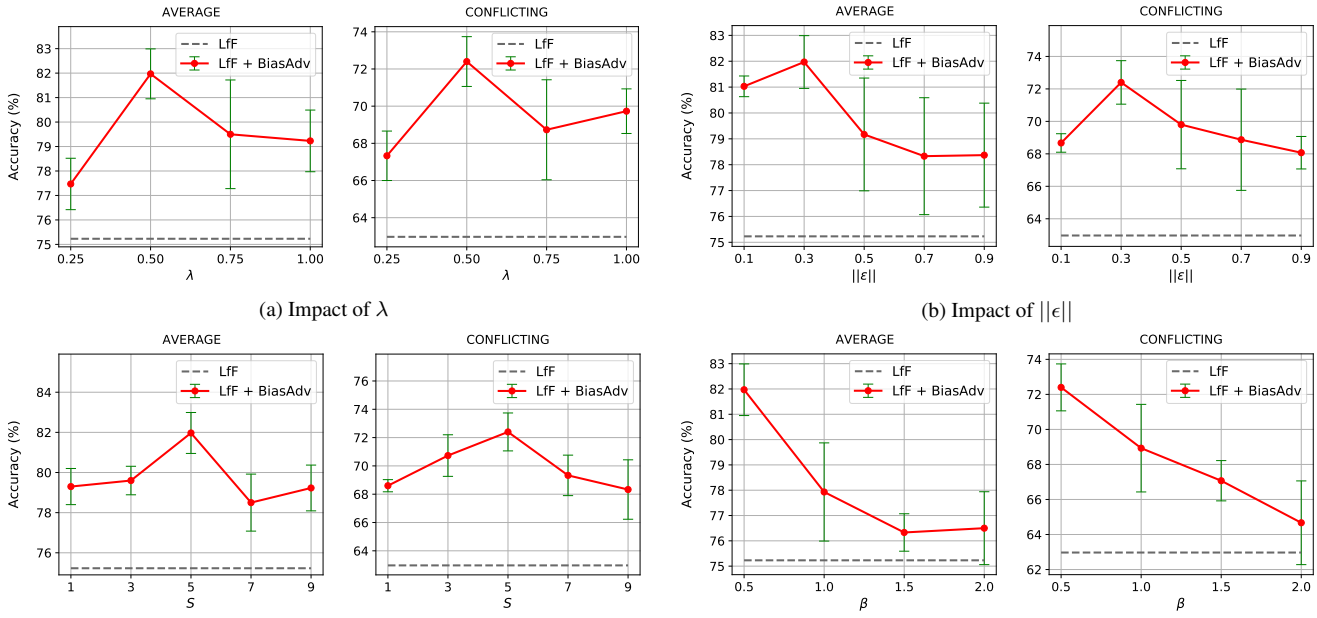
  $\mathcal{R}_f(\theta) \leftarrow \frac{1}{B}\sum_{i=1}^{B} \omega_x^{(i)} \cdot \mathcal{L}(x^{(i)}, y^{(i)}; \theta) + \frac{1}{B}\sum_{i=1}^{B} \omega_{\text{adv}}^{(i)} \cdot \mathcal{L}(x_{\text{adv}}^{(i)}, y^{(i)}; \theta)$

  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{R}_f(\theta)$ ▷ Update f_θ

end for

B. Sensitivity Analysis

We analyzed the impact of hyper-parameters of BiasAdv using the BFFHQ dataset. Specifically, we investigated the impacts of λ in Eq. (3) of the manuscript, the size of the adversarial perturbation $||\epsilon||$, the number of attack steps S, and the weights of adversarial images β by varying the values to $\lambda \in \{0.25, 0.5, 0.75, 1\}$, $||\epsilon|| \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, $S \in \{1, 3, 5, 7, 9\}$, and $\beta \in \{0.5, 1, 1.5, 2\}$, respectively. For all experiments, we applied BisAdv to LfF [11] and evaluated the AVERAGE and CONFLICTING accuracies (%). We ran three independent trials and reported the mean and the standard deviation. Figure 1 summarizes the results. Overall, BiasAdv demonstrated reliable and robust performance regardless of the hyper-parameter choices. In particular, BiasAdv significantly improved the baseline performance (gray dashed line) in all cases, suppporting the effectiveness of BiasAdv. The best performance was achieved when $\lambda = 0.5$, $||\epsilon|| = 0.3$, $S = 5$, and $\beta = 0.5$, respectively.



(a) Impact of λ

(b) Impact of $||\epsilon||$

(c) Impact of S

(d) Impact of β

Figure 1. **Impact of hyper-parameters of BiasAdv.** We plot the AVERAGE and CONFLICTING accuracies (%) on the BFFHQ dataset with different hyper-parameter choices.

C. Qualitative Results

In Figure 6 of the manuscript, we presented Grad-CAM [14] of the test set images of the BFFHQ dataset. To demonstrate the consistency of our results, we present more qualitative results in Figure 2. The results clearly support our claim that applying BiasAdv contributes to learning more generalizable representations, attending on discriminative regions for the target class (age), yet neutral from the bias attribute (gender).

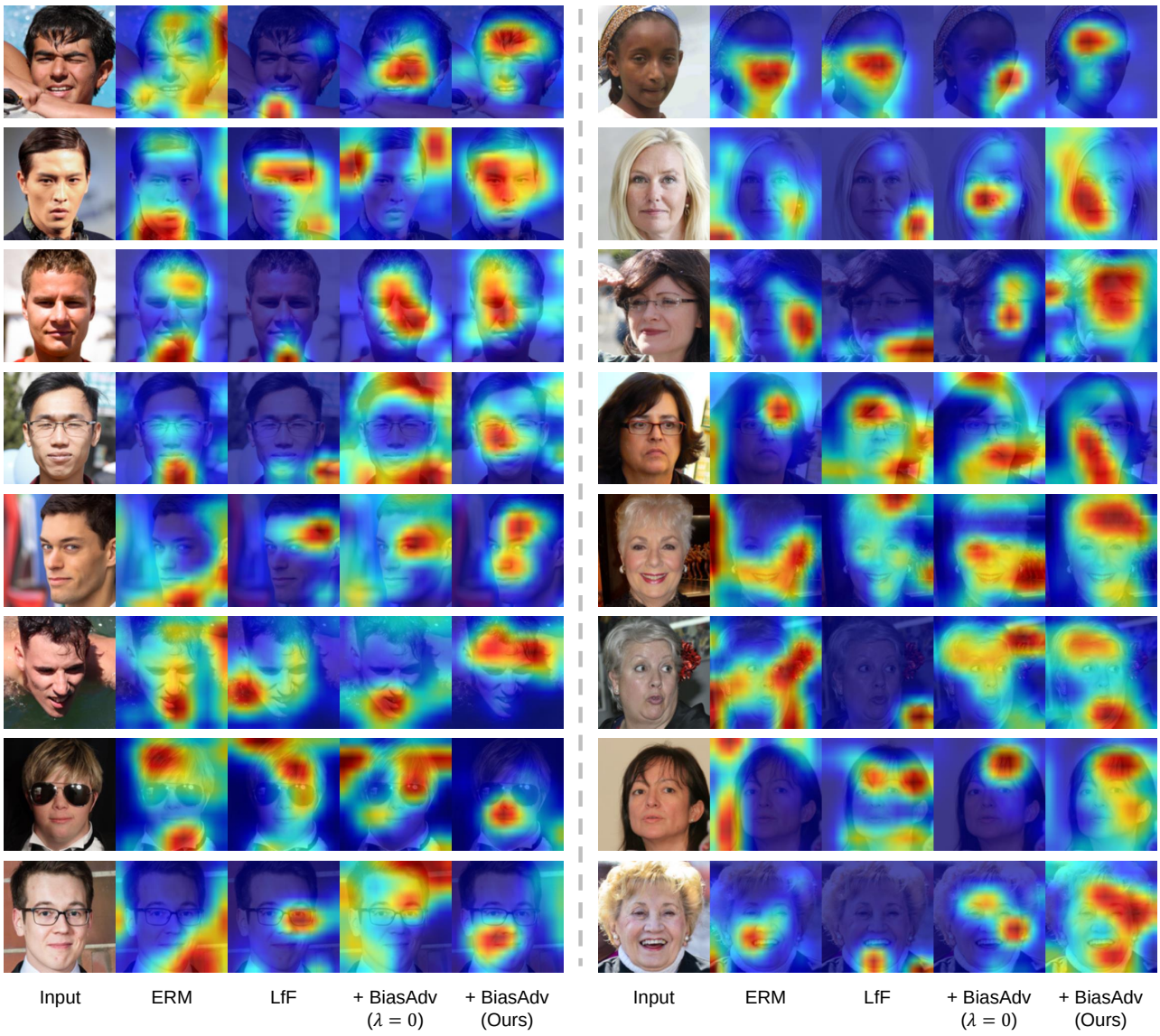


Figure 2. **Grad-CAM [14] comparison of ablation models.** We analyze the class activation maps of the test set images on the BFFHQ dataset.

D. Experimental Details

This section provides further details of our experimental setup. We first provide a detailed explanation of the datasets used in our experiments in Section D.1. Then, we describe the implementation details in Section D.2. In Section D.3, we present the detailed settings of the input corruptions used in the analysis on model robustness (Table 6 of the manuscript).

D.1. Datasets

In this work, we evaluated the proposed method on four benchmark datasets: CIFAR-10C [6], BFFHQ [6], BAR [11], and MetaShift [7]. For a fair and meaningful comparison with the existing methods, we evaluated the proposed method under the unified experimental setup following the previous state-of-the-art methods of each dataset. For CIFAR-10C and BFFHQ, we used the datasets provided in the official repository of DFA [6]. While in DFA, only the accuracy of bias-conflicting samples was reported for BFFHQ, we measured both AVERAGE and CONFLICTING accuracies on the entire unbiased test set. In the original setup of BAR, the training set consisted only of bias-guiding samples while the test set consisted only of bias-conflicting samples. In our experiments, we set the ratio of bias-conflicting samples in the training set to $p \in \{1\%, 5\%\}$, following IRMCon-IPW [13]. To this end, we randomly selected 17 and 94 bias-conflicting samples from the test set and moved them to the training set. The BAR dataset is available in the official repository of IRMCon-IPW. For MetaShift, we used "Cat vs. Dog", a subset of MetaShift for evaluating subpopulation shifts, and followed the default settings, which is available in the official repository of MetaShift.

D.2. Implementation

Table 1, 2, 3, and 4 summarize the full list of implementation details for experiments on CIFAR-10C, BFFHQ, BAR, and MetaShift, respectively.

- f_θ **and** g_ϕ**:** We used PyTorch [12] for implementation. For all datasets, we used the same ResNet-18 [3] architecture for both f_θ and g_ϕ. For BAR and MetaShift, we started learning from the pre-trained weights on ImageNet [1], following the prior works [7, 11]. Except for the experiments on BAR and MetaShift, we trained the models from scratch. For ERM + BiasAdv and LfF + BiasAdv, we trained g_ϕ with the GCE [15] loss to amplify the reliance on the biased features. The GCE hyper-parameter $q = 0.7$ was simply taken from the original paper [15].

- **BiasAdv:** To generate adversarial images, we employed PGD [10] as the default attacker for all experiments. The hyper-parameters of BiasAdv (*i.e.*, λ in Eq. (3) of the manuscript, the size of the adversarial perturbation $||\epsilon||$, the number of attack steps S, and the weights of adversarial images β) were determined empirically (see Section B).

- **Training details:** To apply BiasAdv to JTT and LfF, we used the codes provided by the authors, and the training hyper-parameters, such as batch size, epoch, learning rate, and weight decay, were mainly selected according to the settings of [6, 9, 11]. For JTT, we used grid-search to choose the training epoch of g_ϕ and the up-sampling parameter N_{up}. When applying Biasadv, all the training details, except the hyper-parameters of BiasAdv, were maintained the same for a fair comparison. During training, input images were augmented with random crop and horizontal flip transformations following the convention. Throughout all the experiments, we used Adam [5] optimizer and applied StepLR for learning rate scheduling.

Table 1. **Implementation details for experiments on the CIFAR-10C dataset.**

Module	Name	ERM + BiasAdv	JTT + BiasAdv	LfF + BiasAdv
f_θ	Architecture	ResNet-18	ResNet-18	ResNet-18
	Pre-trained	False	False	False
	Loss	CE	CE	CE
	Epoch	200	50	200
g_ϕ	Architecture	ResNet-18	ResNet-18	ResNet-18
	Pre-trained	False	False	False
	Loss	GCE (q = 0.7)	CE	GCE (q = 0.7)
	Epoch	200	20	200
BiasAdv	Attacker	PGD	PGD	PGD
	λ	1	1	0.5
	$\|\|\epsilon\|\|$	1	0.5	0.5
	S	5	5	5
	β	2	1	1
Training	N_{up}	-	50	-
	Batch size	128	128	128
	Learning rate	1e-3	1e-3	1e-3
	Weight decay	1e-4	1e-4	1e-4
	Optimizer	Adam	Adam	Adam
	Scheduler	StepLR (40/0.5)	StepLR (10/0.5)	StepLR (40/0.5)

Table 2. **Implementation details for experiments on the BFFHQ dataset.**

Module	Name	ERM + BiasAdv	JTT + BiasAdv	LfF + BiasAdv
f_θ	Architecture	ResNet-18	ResNet-18	ResNet-18
	Pre-trained	False	False	False
	Loss	CE	CE	CE
	Epoch	150	50	150
g_ϕ	Architecture	ResNet-18	ResNet-18	ResNet-18
	Pre-trained	False	False	False
	Loss	GCE (q = 0.7)	CE	GCE (q = 0.7)
	Epoch	150	9	150
BiasAdv	Attacker	PGD	PGD	PGD
	λ	0.5	1	0.5
	$\|\|\epsilon\|\|$	0.3	0.5	0.3
	S	5	5	5
	β	0.5	1	0.5
Training	N_{up}	-	20	-
	Batch size	64	64	64
	Learning rate	1e-4	1e-4	1e-4
	Weight decay	1e-4	1e-4	1e-4
	Optimizer	Adam	Adam	Adam
	Scheduler	StepLR (30/0.5)	StepLR (10/0.5)	StepLR (30/0.5)

Table 3. **Implementation details for experiments on the BAR dataset.**

Module	Name	ERM + BiasAdv	JTT + BiasAdv	LfF + BiasAdv
f_θ	Architecture	ResNet-18	ResNet-18	ResNet-18
	Pre-trained	True	True	True
	Loss	CE	CE	CE
	Epoch	20	20	20
g_ϕ	Architecture	ResNet-18	ResNet-18	ResNet-18
	Pre-trained	True	True	True
	Loss	GCE (q = 0.7)	CE	GCE (q = 0.7)
	Epoch	20	10	20
BiasAdv	Attacker	PGD	PGD	PGD
	λ	1	1	0.5
	$\|\|\epsilon\|\|$	0.1	0.1	0.3
	S	9	3	7
	β	1.5	0.5	0.5
Training	N_{up}	-	10	-
	Batch size	64	64	64
	Learning rate	1e-3	1e-3	1e-3
	Weight decay	1e-4	1e-4	1e-4
	Optimizer	Adam	Adam	Adam
	Scheduler	StepLR (10/0.5)	StepLR (10/0.5)	StepLR (10/0.5)

Table 4. **Implementation details for experiments on the MetaShift dataset.**

Module	Name	ERM + BiasAdv	JTT + BiasAdv	LfF + BiasAdv
f_θ	Architecture	ResNet-18	ResNet-18	ResNet-18
	Pre-trained	True	True	True
	Loss	CE	CE	CE
	Epoch	10	1	10
g_ϕ	Architecture	ResNet-18	ResNet-18	ResNet-18
	Pre-trained	True	True	True
	Loss	GCE (q = 0.7)	CE	GCE (q = 0.7)
	Epoch	10	1	10
BiasAdv	Attacker	PGD	PGD	PGD
	λ	1	0.5	1
	$\|\|\epsilon\|\|$	0.5	0.7	0.5
	S	3	1	3
	β	0.5	1.5	0.5
Training	N_{up}	-	10	-
	Batch size	32	32	32
	Learning rate	2e-5	2e-5	2e-5
	Weight decay	1e-4	0	1e-4
	Optimizer	Adam	Adam	Adam
	Scheduler	-	-	-

D.3. Input Corruptions

This section describes the detailed setup of the input corruptions used in the experiments presented in Table 6 of the manuscript. To make corrupted input, we used *imgaug* [4], a python library for image augmentation, following [8]. Specifically, we considered eight input corruptions that were not used for training: Gaussian, Salt & Pepper, Cutout, Dropout, Rotation, Perspective, JPEG-compression, and Gaussian blur. Figure 3 illustrates the resulting images for each corruption on the image from the BFFHQ dataset. The detailed settings for each corruption are listed below.

- **Gaussian:** We added Gaussian noise to an image. For each pixel, the noise was sampled from a normal distribution $\mathcal{N}(0, s)$, where s was sampled per image and varies between 0 and 0.01*255.
- **Salt & Pepper:** We replaced 0.1% of all pixels in an image with the Salt & Pepper noise.
- **Cutout:** We randomly replaced two random rectangle areas of each image with grayish pixels. The size of each rectangle was set to 20% of the input image size. For more details, please refer to [2].
- **Dropout:** We dropped 0 to 5% of all pixels by converting them to black pixels. Specifically, we applied Coarse Dropout [4] that leads to random rectangular areas being dropped.
- **Rotation:** We rotated images by a random value between -30° and 30°. Empty pixels due to rotation were filled with symmetrical padding.
- **Perspective:** We applied random four point perspective transformations to images. Specifically, we used a random scale between 0.05 and 0.15 per image, where the scale is roughly a measure of how far the perspective transformation's corner points may be distanced from the image's corner points.
- **JPEG-compression:** We removed high frequency components in images via JPEG-compression with a compression strength between 80 and 95 (randomly and uniformly sampled per image).
- **Gaussian blur:** We blurred each image with a Gaussian kernel with a random sigma s that was sampled per image and varied between 1 and 3.

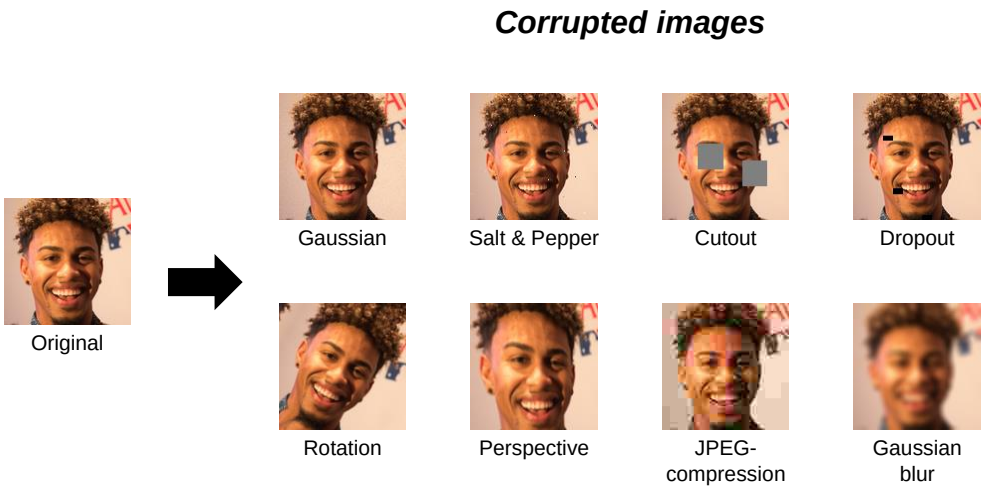


Figure 3. **Examples of input corruptions used in the experiment.** We visualize the original image and the example results of eight input corruptions: Gaussian, Salt & Pepper, Cutout, Dropout, Rotation, Perspective, JPEG-compression, and Gaussian blur. For more details, please refer to *imgaug* [4].

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[2] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 8

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[4] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. https://github.com/aleju/imgaug, 2020. Online; accessed 01-Feb-2020. 8

[5] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[6] Jungsoo Lee, Eungyeup Kim, Juyoung Lee, Jihyeon Lee, and Jaegul Choo. Learning debiased representation via disentangled feature augmentation. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 25123–25133. Curran Associates, Inc., 2021. 1, 5

[7] Weixin Liang and James Zou. Metashift: A dataset of datasets for evaluating contextual distribution shifts and training conflicts. *arXiv preprint arXiv:2202.06523*, 2022. 5

[8] Jongin Lim, Sangdoo Yun, Seulki Park, and Jin Young Choi. Hypergraph-induced semantic tuplet loss for deep metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 212–222, 2022. 8

[9] Evan Z Liu, Behzad Haghgoo, Annie S Chen, Aditi Raghunathan, Pang Wei Koh, Shiori Sagawa, Percy Liang, and Chelsea Finn. Just train twice: Improving group robustness without training group information. In *International Conference on Machine Learning*, pages 6781–6792. PMLR, 2021. 1, 5

[10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 1, 2, 5

[11] Junhyun Nam, Hyuntak Cha, Sungsoo Ahn, Jaeho Lee, and Jinwoo Shin. Learning from failure: De-biasing classifier from biased classifier. *Advances in Neural Information Processing Systems*, 33:20673–20684, 2020. 1, 3, 5

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 5

[13] Jiaxin Qi, Kaihua Tang, Qianru Sun, Xian-Sheng Hua, and Hanwang Zhang. Class is invariant to context and vice versa: On learning invariance for out-of-distribution generalization. In *ECCV*, 2022. 5

[14] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 4

[15] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018. 1, 5