BAdd: Bias Mitigation through Bias Addition

Supplementary Material

1. Model Architecture

In experiments involving the MNIST-based datasets, namely Biased-MNIST and FB-Biased-MNIST, we utilize a simple Convolutional Neural Network (CNN) architecture outlined in [4], which comprises four convolutional layers with 7×7 kernels and a classification head. For the experiments involving the Biased-UTKFace, Corrupted-CIFAR10, and CelebA datasets, we adopt the ResNet-18 architecture [14]. For Waterbirds and UrbanCars datasets, we use ResNet-50 networks.

2. Implementation Details

We employ the Adam optimizer with a 0.001 initial learning rate, which is divided by 10 every 1/3 of the training epochs. Batch size is fixed at 128 and weight decay is set to 10^{-4}. Following previous works [16, 22, 28], we train the models on Biased-MNIST and FB-Biased-MNIST datasets for 80 epochs. For Biased-UTKFace and CelebA, training duration is set to 20 and 40 epochs, respectively. As for the Corrupted-CIFAR10 dataset, models are trained for 100 epochs using a cosine annealing scheduler. For the Waterbirds and UrbanCars datasets, we do not use a learning rate scheduler, and the models are trained for 300 and 100 epochs, respectively. Following the initial training phase, the classification head of all models is fine-tuned for an additional 20 epochs. Note that this fine-tuning stage yields only minor performance gains (i.e., less than 1% improvement in accuracy) and can therefore be considered optional. Regarding the bias-capturing models, for Biased-MNIST, FB-Biased-MNIST, Waterbirds, UrbanCars, and CelebA datasets, they are trained on the same dataset as the main model using the attributes introducing bias as target attributes. For Biased-UTKFace we employ the pre-trained bias-capturing classifiers provided by [30], which is trained on the FairFace [35] dataset. Finally, for Corrupted-CIFAR10 we just project the one-hot vectors representing the texture labels (i.e., the attribute introducing the bias) to the feature space of the main model without using a trainable bias-capturing model. All experiments were conducted on a single NVIDIA RTX-3090 Ti GPU and repeated for 5 different random seeds.

3. Ablation Study

In this section, we explore the ways of integrating bias-capturing features into the training process. Tab. 11 presents a comparison of BAdd's performance when the bias-capturing features are added to the main features versus when they are concatenated with them. As one may observe, the concatenation approach is much less effective than the addition. This is anticipated, as relying on $\mathbf{b}$, with non-zero corresponding weights, would perform poorly on balanced settings (random background color), while not relying on it, with ~ 0 corresponding weights, would be equivalent to the sub-optimal vanilla training. Furthermore, Tab. 12 demonstrates how the selection of layer to incorporate the bias-capturing features affects the performance of BAdd. The penultimate layer yields the most favorable performance, as the shallower the selected layer, the fewer layers remain independent of the protected attributes.

Table 11. Addition vs Concatenation: Biased-MNIST performance comparison between different approaches of integrating bias-capturing features.

Method	q			
	0.99	0.995	0.997	0.999
Concatenation	91.5	81.7	70.3	36.5
Addition	**98.1**	**97.3**	**96.3**	**91.7**

Table 12. BAdd performance on Biased-MNIST with $q = 0.99$ when considering different layers for incorporating the bias capturing features.

Method	Layer			
	1st	2nd	3rd	4th
BAdd	74.6	85.8	97.7	**98.1**

Also, we explore how BAdd performs when used on datasets with a very limited degree of bias. To assess this, we utilize the Biased-MNIST dataset with low q values - specifically, 0.1, 0.3, 0.5, and 0.7. As shown in Tab. 13, BAdd maintains model performance consistently (i.e., 99.3%) across all the levels of data bias.

Table 13. BAdd accuracy on fair (i.e., $q = 0.1$) or slightly biased data (i.e., $q = \{0.3, 0.5, 0.7\}$).

Method	q			
	0.1	0.3	0.5	0.7
Vanilla	0.993	0.992	0.991	0.989
BAdd	0.993	0.993	0.993	0.993

Furthermore, in Section 5, we show that BAdd can be combined with either a trained bias capturing model or a

Table 14. Bias capturing model vs projection: Performance on Biased-MNIST.

Method	q			
	0.99	0.995	0.997	0.999
Vanilla	90.8±0.3	79.5±0.1	62.5±2.9	11.8±0.7
BAdd w/ projection	97.4±0.2	94.8±0.6	90.1±1.7	65.4±4.4
BAdd w/ bias capturing model	**98.1**±0.2	**97.3**±0.2	**96.3**±0.2	**91.7**±0.6

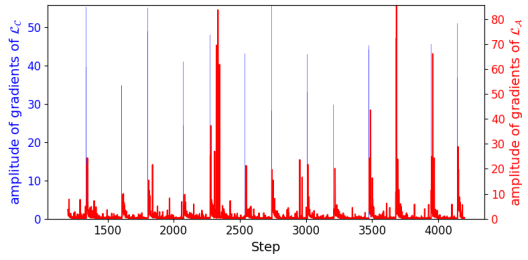
Table 15. Bias capturing model vs projection: Performance on Biased-UTKFace.

Method	Bias			
	Race		Age	
	Unbiased	Bias-conflicting	Unbiased	Bias-conflicting
Vanilla	87.4±0.3	79.1±0.3	72.3±0.3	46.5±0.2
BAdd w/ projection	89.7±2.6	88.7± 4.5	78.3±1.1	61.8±3.1
BAdd w/ bias capturing model	**92.2**±0.2	**93.3**±0.2	**80.3**±0.8	**73.6**±1.0

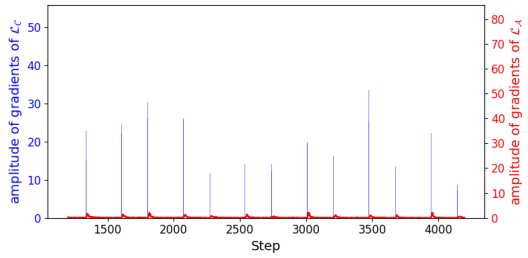
projection of one-hot vectors representing the biased attribute labels to the space of $\mathbf{h}$ for deriving $\mathbf{b}$. When employing a bias-capturing classifier, a deep learning model is specifically trained to predict the protected attribute, such as race, gender, hair color, or background. This process encourages the model to learn richer and more diverse latent features associated with these attributes. By focusing on predicting these protected attributes, the model captures subtle, underlying patterns in the data that may be overlooked if solely relying on labeled attributes. Such comprehensive representations can improve the model's understanding of complex visual features, ultimately enhancing the efficacy of bias mitigation. Conversely, the approach of projecting one-hot encoded labels is computationally less intensive and easier to implement as it does not require any additional training steps. However, this method may not effectively capture the intricate visual features that a dedicated bias-capturing classifier can uncover. Tab. 14 and Tab. 15 provide a comparison of these two BAdd variants on Biased-MNIST and Biased-UTKFace datasets, respectively. The flexibility in choosing between these approaches allows practitioners to balance implementation simplicity with the richness of feature representation. Utilizing a trained bias-capturing model may lead to more effective bias mitigation, especially in datasets where the complexity of visual features plays a critical role.

4. Learning Dynamics

As demonstrated in the main manuscript, bias-conflicting samples trigger spikes in the gradients of the bias-aligned loss in subsequent training steps. Fig. 5 illustrates that these spikes are mitigated when using BAdd, with the gradients of $\mathcal{L}_{\mathcal{A}}$ staying near zero. This is a direct result of the injection of $\mathbf{b}$ into the learning process.



(a) Vanilla



(b) BAdd

Figure 5. BAdd prevents the spikes of the gradients of bias-aligned loss triggered by the bias-conflicting samples of Biased-MNIST. Blue bars indicate the steps where bias-conflicting samples occur, with their height representing the amplitude of gradients.

5. Qualitative Results

Fig. 6 visualizes the GradCam activations of a model trained on UrbanCars with BAdd compared to a vanilla model. BAdd effectively shifts the model's focus to the object of

interest, with only minor activations in the background that, however, are reflected in the model's performance (i.e., -4.3 BG Gap).

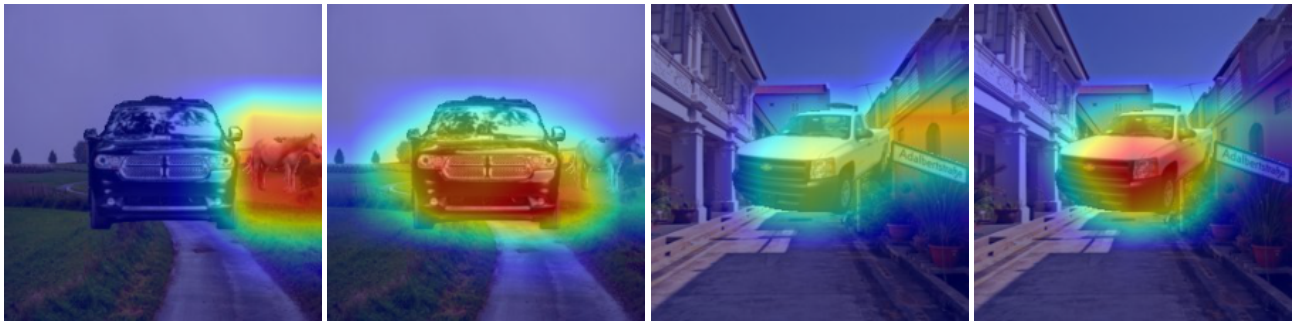
6. Complexity Analysis

Regarding the computational complexity of BAdd, we provide a detailed analysis comparing BAdd to the baseline (vanilla) model for a typical example involving a ResNet18 network, input images of size $3\times224\times224$, and two classes for the biased attribute (similar to CelebA or UTKFace experiments).

Regarding the FLOPs of the trainable components, the baseline model's computational cost is 1.818 GFLOPs. The BAdd's addition of bias-capturing features into the main model's penultimate layer increases the computational complexity by 2.82e-7% (i.e., +512 FLOPs), while the fine-tuning process, which involves updating only the final classification layer, incurs an additional computational cost of 5.63e-7% (i.e., +1.024 FLOPs). This phase remains efficient as only a small number of parameters are updated.

Regarding the inference FLOPs, if projection is employed for extracting bias features, the computational cost is only 1024 FLOPs. On the other hand, the computational cost of the bias-capturing model depends on its architecture, which in this case matches the main model. However, it is important to note that this model acts as a feature extractor, so its corresponding features need to be computed only once.

Moreover, the inference time of BAdd is equivalent to the baseline, as the bias-capturing component is not involved during inference. Similarly, the number of trainable parameters in BAdd is the same as in the baseline model, as the additional components related to the bias features are not trainable. The main potential overhead arises if bias-labels are unavailable or if there is no pretrained bias-capturing model. In such cases, training a bias-capturing model from scratch would add to the overall computational effort. In summary, BAdd introduces minimal overheads during training (e.g., feature addition and fine-tuning the classification layer), while the inference complexity and number of trainable parameters remain equivalent to the baseline.



(a) Method: Vanilla, Target: Country, Background: Country, CoObj: Country

(b) Method: BAdd, Target: Country, Background: Country, CoObj: Country

(c) Method: Vanilla, Target: Country, Background: Urban, CoObj: Urban

(d) Method: Vanilla, Target: Country, Background: Urban, CoObj: Urban

(e) Method: Vanilla, Target: Urban, Background: Urban, CoObj: Urban

(f) Method: Vanilla, Target: Urban, Background: Urban, CoObj: Urban

(g) Method: Vanilla, Target: Urban, Background: Country, CoObj: Country

(h) Method: BAdd, Target: Urban, Background: Country, CoObj: Country

Figure 6. Vanilla vs BAdd: GradCam activations on bias-aligned and bias-conflicting samples of UrbanCars dataset.