Efficient Bias Mitigation Without Privileged Information: Appendix

A Ethical Considerations

Our work improves a DNN's fairness through a pragmatic and effective pipeline. We hope this may lead to TAB being included as a standard debiasing good practice in real-world workloads. Nevertheless, we recognise there are potential negative societal aspects to consider. First, TAB trains its model twice, doubling the carbon footprint of training [2]. This, however, can be amortised in practice by TAB's lack of model selection. Second, the ability of TAB to identify a large portion of bias-conflicting samples implies that bad actors may use this information to further increase a model's bias (e.g., by removing these training samples). Nevertheless, we argue that this ill-intended use of our method is not unique to our approach, but rather an inherent but unlikely negative aspect of all BM methods.

B Summary of Previous Bias Mitigation Methods

To complement our related work discussion in Section 2, in this section we include a summary of previous work that is closely aligned with our proposed method. In Table B.1, we show several bias mitigation methods together with (1) their level of group supervision (whether they need group labels in the training set, in the validation set, or in both), and (2) the set of key hyperparameters that require fine-tuning each method. We note, however, that most of these approaches have more underlying hyperparameters than those shown as they may still require fine-tuning of hyperparameters specific to the target model (e.g., the underlying model's weight decay, learning rate, batch size, etc). Nevertheless, as these are hyperparameters that are not unique to any BM method but rather a necessity of the model chosen to be debiased, in Table B.1 we show only the BM-specific hyperparameters. By contrasting methods this way, we see how TAB is unique within fully unsupervised BM approaches by not requiring any extra hyperparameters, avoiding a potentially costly model selection and enabling easy adoption in modern training pipelines.

C Price of Unawareness and Mean Model Selection

In this section, we formally define the metrics discussed in Section 4 and describe how we can compute such metrics in practice.

Price of Unawareness (PoU) Borrowing inspiration from similar concepts in game theory [17, 24], we capture the cost "paid" by a BM pipeline

Table B.1: Bias mitigation methods together with their group annotation requirements. To highlight the practical considerations involved when using any of these approaches, we include the number and identity of key sensitive hyperparameters for each approach. However, we note that these are hyperparameters required **on top of the standard ERM hyperparameters needed to train a model** (e.g., "learning rate", "batch size", etc.). † These papers were published around the time of our submission and, therefore, constitute concurrent works.

Method	Train Groups	Val Groups	# of Hyperparameters - {Hypers}
Adversarial Debiasing [36]	✓	✓	2 - $\{A, \alpha\}$
G-DRO [29]	✓	✓	4 - $\{C, \lambda, \eta_q, \eta_\theta\}$
PGI [1]	✓	✓	1 - {invariance penalty}
DFR [15]	✓	✓	2 - {retrain epochs, # retrains}
Multiaccuracy [13]	✗	✓	2 - $\{\mathcal{A}, \alpha\}$
CVaR DRO [19]	✗	✓	1 - $\{\alpha\}$
LfF [23]	✗	✓	1 (q)
JTT [21]	✗	✓	2 - $\{T, \lambda_{\text{up}}\}$
Spectral Decoupling [28]	✗	✓	2 - $\{\lambda, \gamma\}$
EIIL [4]	✗	✓	1 or more - {underlying learner dependent}
CIM [33]	✗	✓	1 - $\{\alpha\}$
OccamNet [31]	✗	✓	2 - $\{\tau_{\text{acc},0}, \lambda_{\text{CS}}\}$
SELF [18]	✗	✓	2 or more - $\{n, g\}$
GEORGE [32]	✗	✗	4 or more - $\{f_\theta$, proj. comps, F, $s_{\min}\}$
MaskTune [3]	✗	✗	2 or more - $\{\tau, \mathcal{G}\}$
DebiAN [20]	✗	✗	1 or more - {discoverer D}
AGRO [25]	✗	✗	6 or more - $\{T_1, T_2, m, \alpha, \lambda, \phi_h\}$
CB Last-layer retraining† [18]	✗	✗	1 - {retrain epochs}
uLA† [34]	✗	✗	4 - $\{T_{\text{ssl}}, T_{\text{stop}}, \eta, \tau\}$
TAB (ours)	✗	✗	0 - $\emptyset$

when lacking proper group labels during model selection through its *price of unawareness*. Formally, given a BM pipeline $\mathcal{B}_\gamma$ with hyperparameters $\gamma \in \Gamma$, let $\hat{\theta}(\mathcal{B}_\gamma, \mathcal{P}_{\text{train}})$ be the parameters $\hat{\theta} \in \Theta$ learnt by running $\mathcal{B}_\gamma$ on a training set formed by N i.i.d. samples from $\mathcal{P}_{\text{train}}$. In this setup, we define the PoU of $\mathcal{B}$ as follows:

$$\text{PoU}(\mathcal{B}, \mathcal{P}_{\text{train}}, \mathcal{P}_{\text{test}}, \mathcal{P}_{\text{val}}) := \frac{\max_{\gamma' \in \Gamma} \text{WGA}\big(f_{\hat{\theta}(\mathcal{B}_{\gamma'}, \mathcal{P}_{\text{train}})}, \mathcal{P}_{\text{test}}\big)}{\text{WGA}\big(f_{\hat{\theta}(\mathcal{B}_{\gamma_{\text{val-acc}}}, \mathcal{P}_{\text{train}})}, \mathcal{P}_{\text{test}}\big)} \tag{C.1}$$

where $\mathcal{P}_{\text{train}}$, $\mathcal{P}_{\text{test}}$, and $\mathcal{P}_{\text{val}}$ are the training, testing, and validation distributions (assumed to be the same but written separately for clarity), and $\gamma_{\text{val-acc}}$ are the hyperparameters chosen when performing model selection based on the validation average accuracy (for notational clarity, we do not explicitly write their dependency on $\mathcal{B}$, $\mathcal{P}_{\text{train}}$, and $\mathcal{P}_{\text{val}}$). That is:

$$\gamma_{\text{val-acc}} := \arg\max_{\gamma \in \Gamma} \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{P}_{\text{val}}(\mathbf{x}, y)} \big[\mathbb{1}\big(f_{\hat{\theta}(\mathcal{B}_\gamma, \mathcal{P}_{\text{train}})}(\mathbf{x}) = y\big)\big] \tag{C.2}$$

In practice, we estimate the PoU through a discrete grid search over $\Gamma_{\text{cands}} \subseteq \Gamma$ while ensuring that samples in the training, testing, and validation sets are entirely disjoint. If during this grid search we find that the test WGA of $f_{\hat{\theta}(\mathcal{B}_{\gamma_{\text{val-acc}}}, \mathcal{P})}$

is 0 for some hyperparameters γ, then we discard these hyperparameters as otherwise the PoU may become infinite and non-informative. Hence, we estimate this value based on hyperparameterisations with non-zero validation WGA only.

We note that, if the validation distribution $\mathcal{P}_{\text{val}}$ is the same as the testing distribution $\mathcal{P}_{\text{test}}$, as it is commonly the case for in-distribution domains, then for large enough validation sets (i.e., $N >> 1$), the PoU is equivalent to computing the ratio between the WGA of the model selected using *validation WGA* and the WGA of the model selected using validation mean accuracy. Nevertheless, because access to a large validation set is rare, we defined the numerator of equation C.1 in terms of the test distribution to provide a realistic upper bound for a method's achievable WGA. This enables us to compute a PoU that (1) is stable even in extremely group-imbalanced domains where minority groups may be missing in a small validation set, and (2) is guaranteed to be bounded below by 1, enabling easy interpretation of the PoU as discussed in Section 4.

Mean Model Selection WGA (MMS) In contrast to the PoU, which aims to capture the worst-case cost of lacking validation group labels during BM, the Mean Model Selection WGA (MMS) is designed to capture the expected behaviour of a BM pipeline across multiple hyperparameters. In other words, we are interested in measuring the expected WGA across all possible hyperparameters. This serves as a proxy measurement of what one expects to get from a BM approach when deploying it across a task without an extensive model selection. The MMS is therefore formally defined as follows:

$$\text{MMS}(\mathcal{B}, \mathcal{P}_{\text{train}}, \mathcal{P}_{\text{test}}) := \mathbb{E}_{\gamma' \sim \text{Unif}(\mathit{\Gamma})}\Big[\text{WGA}\big(f_{\hat{\theta}(\mathcal{B}_{\gamma'}, \mathcal{P}_{\text{train}})}, \mathcal{P}_{\text{test}}\big)\Big] \tag{C.3}$$

where, as in the case for our fomarlisation of the PoU, $\mathcal{B}_\gamma$ is a BM pipeline taking hyperaparameters $\gamma \in \mathit{\Gamma}$.

In practice, we follow a similar approach as done with the PoU and estimate the MMS using a grid search over a set of potential hyperparameters $\mathit{\Gamma}_{\text{cands}} \subseteq \mathit{\Gamma}$. More specifically, we compute this estimate using hyperparameters in $\mathit{\Gamma}_{\text{cands}}$ to compute an empirical Monte Carlo estimate of the MMS:

$$\text{MMS}(\mathcal{B}, \mathcal{P}_{\text{train}}, \mathcal{P}_{\text{test}}) \approx \frac{1}{|\mathit{\Gamma}_{\text{cands}}|} \sum_{\gamma' \in \mathit{\Gamma}_{\text{cands}}} \text{WGA}\big(f_{\hat{\theta}(\mathcal{B}_{\gamma'}, \mathcal{P}_{\text{train}})}, \mathcal{P}_{\text{test}}\big) \tag{C.4}$$

D Targeted Augmentations for Bias Mitigation

In Section 5 we provide a detailed motivation and description of TAB, our proposed unsupervised bias mitigation pipeline. Here, we complement that discussion by providing pseudocode for TAB (Algorithm D.1). This format highlights two key things: first, as discussed in Section 5, TAB takes no extra hyperparameters on top of those of its underlying learning algorithm $\mathcal{L}$. Second, TAB's implementation is extremely simple and can be easily added to existing training pipelines. We believe that both of these key properties make TAB a likely candidate for adoption in practical real-world scenarios.

Algorithm D.1 TAB

```
**Input:** Learning algorithm $\mathcal{L}$, target neural network $f_\theta$, and training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}^{(i)}, y^{(i)}) \mid \mathbf{x}^{(i)} \in \mathbb{R}^n, y^{(i)} \in \{1, \cdots, L\}\}_{i=1}^N$
**Output:** Robust model $f_{\hat{\theta}_{\text{TAB}}}$
$f_{\hat{\theta}_{\text{base}}}, \{\mathbf{h}^{(i)} \in \mathbb{R}^T\}_{i=1}^N \leftarrow \mathcal{L}(f_\theta, \mathcal{D}_{\text{train}})$    ▷ Train identifier and build loss histories $\mathbf{h}^{(i)}$
**for** $l \in \{1, \cdots, L\}$ **do**
    $\mathcal{H}_l \leftarrow \{\mathbf{h}^{(i)} \mid y^{(i)} = l\}$    ▷ Get loss histories of all samples with label $l$
    $\mathcal{D}_l^+, \mathcal{D}_l^- \leftarrow$ k-means$(\mathcal{H}_l, k = 2)$    ▷ Split all samples into two clusters
    **if** $|\mathcal{D}_l^+| < |\mathcal{D}_l^-|$ **then**
        $\mathcal{D}_l^+, \mathcal{D}_l^- \leftarrow \mathcal{D}_l^-, \mathcal{D}_l^+$    ▷ $\mathcal{D}_l^-$ will be the minority cluster
    **end if**
    $\mathcal{D}'_l \leftarrow \emptyset$    ▷ Construct this class' augmented **multiset**
    **while** $|\mathcal{D}'_l| < |\mathcal{D}_l^+| - |\mathcal{D}_l^-|$ **do**
        $j \leftarrow$ UniformSample$(\mathcal{D}_l^-)$    ▷ Randomly select a sample index from $\mathcal{D}_l^-$
        $\mathcal{D}'_l \leftarrow \mathcal{D}'_l \cup \{(\mathbf{x}^{(j)}, y^{(j)})\}$    ▷ Add the $j$-th training sample to $\mathcal{D}'_l$
    **end while**
**end for**
$\mathcal{D}'_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \left(\bigcup_{l=1}^L \mathcal{D}'_l\right)$    ▷ Construct our augmented training **multiset**
$f_{\hat{\theta}_{\text{TAB}}} \leftarrow \mathcal{L}(f_\theta, \mathcal{D}'_{\text{train}})$    ▷ Learn robust model from scratch
**Return:** $f_{\hat{\theta}_{\text{TAB}}}$
```

E Dataset Details

In this section, we provide a detailed description of the datasets we use for our evaluation in Section 6. Specifically, we show the main characteristics of each task in Table E.2 and describe each of the specific tasks in more detail below.

Table E.2: Key characteristics of all datasets used in this paper. All tasks in our evaluation are classification tasks with L categorical labels. We show the total number of groups k for each task as well as its *worst-group size*, defined as the ratio between the number of samples in the smallest group and the number of samples in the entire dataset. Both `BAR` and `CUB` have not known group labels. Nevertheless, for `BAR` there is a distribution shift between the training and the test set where actions are displayed on different backgrounds than those used in the training set.

Dataset	# of Samples (N)	# of Features (n)	# of Classes (L)	# of Groups (k)	Worst-Group Size (%)
`Even-Odd` ($p = 99\%$)	48,000	(3, 28, 28)	2	4	0.41%
`cMNIST` ($p = 98\%$)	48,000	(3, 28, 28)	10	100	0.01%
`Waterbirds`	4,795	(3, 224, 224)	2	4	1.17%
`CelebA` (subsampled)	24,416	(3, 224, 224)	2	4	0.83%
`BAR`	1,552	(3, 224, 224)	6	12	Unknown
`CUB`	4,796	(3, 224, 224)	200	Unknown	Unknown

Even-Odd The `Even-Odd` task is a synthetic binary ($L = 2$) visual classification task where the strength of existing spurious correlations can be easily controlled. Taking inspiration from the Colour-MNIST [23] dataset, we construct our task by colouring handwritten digits from the MNIST [6] dataset with two colours. Specifically, each sample $\mathbf{x}^{(i)}$ of the original MNIST task is transformed from a 28×28 grayscale image to a $3 \times 28 \times 28$ normalised RGB image (i.e., $\mathbf{x}^{(i)} \in [0,1]^{3\times 28 \times 28}$) by colouring the digit with either "red" or "green" hues proportional to the grayscale pixel intensities. We define a binary task where the label determines whether an image is "odd" ($y = 0$) or "even" ($y = 1$) and introduce a spurious correlation of strength $p \in [50\%, 100\%]$ by selecting $p\%$ of all "odd" samples and colouring them "red" while colouring the rest of $1-p$ samples using "green". Similarly, we introduce an equivalent correlation with strength p between "even" and "green" samples. This yields a task with 4 groups, one for each pair $(\text{label}, \text{colour})$, of which the smallest subgroup has an expected size of $N \times \frac{1}{2} \times \frac{100-p}{100}$.

The training, testing, and validation sets in Even-Odd are constructed using the procedure above on the standard train, test, and validation sets in the MNIST task. We note that to enable accurate evaluation, we use a balanced test set that has an equal representation on all $(\text{class}, \text{colour})$ combinations. This allows us to accurately estimate the WGA, as otherwise the smallest group in the test set is too small for getting a good estimate of the model's accuracy on members of that group. Nevertheless, when computing test mean accuracy on this group-balanced test dataset, we weight each sample based on the representation of their respective $(\text{class}, \text{colour})$ group in the training distribution. That way, the mean average accuracy in the test set behaves as if it is from the same distribution as the training set.

cMNIST Similar to Even-Odd, the cMINST task is a Colour MNIST-based classification task with a predefined spurious correlation strength. Each sample $\mathbf{x}^{(i)}$ of the original MNIST dataset is transformed from a 28×28 grayscale image to a $3\times 28\times 28$ normalised RGB image (i.e., $\mathbf{x}^{(i)} \in [0,1]^{3\times 28\times 28}$) by colouring the digit with one of 10 colours (the colours are randomly-generated RGB colours fixed beforehand). In this dataset, we define a multilabel task ($L = 10$) where the label determines the digit's identity ($y \in \{0, 1, \cdots, 9\}$. As in Even-Odd, we introduce a spurious correlation of strength $p \in [50\%, 100\%]$ by selecting $p\%$ of all samples with any given class label $l \in \{0, 1, \cdots, 9\}$ and colouring them the l-th colour while colouring the rest of $1 - p$ samples of that class with a colour randomly selected from the remaining $L - 1$ colours. This results in a task with L^2 groups, one for each pair $(\text{label}, \text{colour})$ of which the smallest subgroup has an expected size of $N \times \frac{1}{10} \times \frac{100-p}{100}$. We generate the training, testing, and validation sets as in 'Even-Odd", constructing a group-balanced test set for stability in evaluation and using a weighted accuracy to compute the mean accuracy.

Waterbirds The Waterbirds [29] task is a binary bird classification task ($L = 2$) where each sample is formed by an image of a bird (selected from the CUB dataset [35]) on top of a either a "land" background or a "water" background (selected from the Places dataset [38]). Birds in this task are split between "waterbirds" (birds that are either seabirds or waterfowl) and "landbirds" (rest of birds). There is a spurious correlation introduced between the type of bird (i.e., the downstream label y) and the background selected from the Places dataset.

Specifically, images from the "ocean" and "natural lake" categories in Places are spuriously correlated with "waterbirds" (i.e., $y = 0$), while backgrounds from the "bamboo" or "broadleaf" forest categories are spuriously correlated with "landbirds" ($y = 1$). This results in a total of 4 subgroups $\{$("waterbird", "water background"), ("waterbird", "land background"), ("landbird", "water background"), ("landbird", "land background")$\}$ of which ("waterbird", "land background") and ("landbird", "water background") are minority bias-conflicting subgroups.

We use the same training/validation/testing splits from Sagawa et al. [29] however during validation and testing, as in the MNIST-based tasks, we use a weighted accuracy for the mean accuracy where every sample is weighted based on the training distribution of its (label, background) subgroup in the training set. This is because, in the standard `Waterbird` splits, the training set is highly group-imbalanced while the validation and test sets are both group-balanced. Therefore, using the validation set as is may result in accidental group information being leaked during model selection (an unrealistic assumption as the validation set is usually sampled from the same distribution as the training set). All images are resized to be 3×224 RGB images (in floating point representation) and are normalised using the mean and standard deviations of the ImageNet [5] dataset. Finally, during training we perform random croppings and horizontal flips.

CelebA We construct a real-world human-centric task based on the CelebA face recognition dataset [22], a large collection of celebrities' face images collected from the internet annotated with an identity label and a set of 40 binary attributes. We follow the approach by Sagawa et al. [29] and construct a binary task ($L = 2$) out of this dataset by using the "Blonde Hair" attribute as the downstream label. We use this setup as there is a strong spurious correlation between the annotated perceived gender of each image and the annotated colour of their hair. In this setup, samples annotated as "not blonde" and "male" are significantly underrepresented (worst bias-conflicting group), leading to models exploiting perceived gender to determine the hair colour. More generally, the bias-conflicting samples correspond to ("blonde", "male") and ("not blond" and "female") samples, with the ("blonde", "male") group being significantly smaller than the ("not blonde", "female") group. Although the original splits of the CelebA dataset contain $162,770$ training samples and $19,867$ test samples, for our experiments we use 15% of the training set to enable a tractable and exhaustive model selection across all methods with hyperparameters as discussed above (this is particularly important for costly methods such as JTT). This results in a training set with approximately $24,000$ samples. Furthermore, we also use 15% of the validation set for consistency and resize all samples to $3 \times 224 \times 224$ images while applying random horizontal flips during training. All images are turned into their floating-point representation and normalised using ImageNet [5]'s means and standard deviations as normally done for this task.

BAR Next, we evaluate all models on the Biased Action Recognition (`BAR`) task, a real-world action detection dataset. This dataset contains samples of humans performing one of six actions ("Climbing", "Fishing", "Racing", "Throwing", "Vaulting"), and we are tasked with predicting each action from the image ($L = 6$). Each action in this dataset is spuriously correlated with a specific scene: "Climbing" actions are most commonly done on "Rock Walls", "Diving" actions are most commonly done "Underwater", "Fishing" actions are most commonly done on the "Water Surface", "Racing" actions are most commonly done on a "Paved Track", "Throwing" actions are most commonly done on a "Playing Field", and "Vaulting" actions are most commonly shown in front of "Sky" backgrounds. This dataset does not have group annotations, making it impossible to accurately compute the WGA during training, testing, or validation. Nevertheless, the test set is constructed so that actions are more commonly shown with scenes that are not aligned with the action's spurious background found in the training set. Therefore, the mean accuracy and worst-class accuracy in the test set serve as a good proxy for WGA for this task, given the distribution shift between the train and test sets.

We use the original train and test splits. However, we randomly select 20% of the training set as a validation set. All images are resized to be 3×224 RGB images (in floating point representation) and are normalised using the mean and standard deviations of the ImageNet [5] dataset. Finally, during training we perform random scalings and horizontal flips.

CUB Finally, we use the `CUB` dataset as a real-world task without known spurious correlations. Samples in this task correspond to RGB images of birds and are annotated with their bird type identity ($L = 200$) as the downstream label. We process all images as in [16] by normalising and randomly flipping and cropping each image during training (normalisation is done using ImageNet's mean and standard deviation). This results in a dataset with approximately $6,000$ images with shape $3\times299\times299$ that is split into a train, validation, and test subsets using the same splits as Koh et al. [16] (training set has approximately $4,800$ images).

F Experimental Details

In this section, we describe our experimental setup across all tasks and baselines for our experiments described in Section 6. We begin by describing the underlying architecture and optimisation setup used across baselines. Then, we proceed to describe each method's hyperparameter selection process.

F.1 Architecture and Optimisation Setups

For all MNIST-based synthetic tasks (i.e., `Even-Odd` and `cMNIST`), we train as the underlying model of all baselines a six-layered Convolutional Neural Network (CNN). Specifically, we construct a CNN formed by six 2D Convolutional layers with 3×3 filters, "same" padding, and $\{16, 16, 32, 32, 64, 64\}$ output feature maps. A ReLU non-linear activation follows each of these layers and a single linear layer is used to map the output of the last convolution to the number of output classes (no activation is used after this layer as we work with logit outputs).

We learn this model's parameters via an Adam optimiser [14] with its default configuration (learning rate $\eta = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$). Moreover, given the smaller model sizes for these tasks, we use a relatively large batch size of $2,048$ samples to better utilise our hardware. Notice that in these synthetic tasks, we do not perform a hyperparameter search on the optimiser's learning rate as we observed good performance with the default optimiser configuration and a significant drop in ERM validation accuracy when this learning rate was decreased.

In contrast, for our real-world tasks we use a Resnet-18 [11] architecture whose weights for all layers but the output linear layer are initialised to those from a Resnet-18 model pretrained on ImageNet [5] (using Pytorch Vision's default loaded weights [26]). To reduce memory consumption and parallelise model selection, we use a vanilla SGD optimiser for training these models with a learning rate η (a hyperparameter selected from $\{10^{-3}, 10^{-4}\}$ as described in further detail below) and a momentum of 0.9. We aim to maximise our hardware utilisation by using a batch size that is as large as possible to fit the model's weights, activations, and gradients during training in the memory of a single of our GPUs. Moreover, to simplify our implementation of TAB within the Pytorch ecosystem, we look for batch sizes that are divisors of the training set (as this enables us to very easily store a model's loss history during training). We note, however, that this is not in any way a hard requirement for TAB but rather something we exploited to easily adapt it to a commonly used framework such as Pytorch. This process resulted in using the following batch sizes for our real-world datasets: $B = 137$ for `Waterbirds`, $B = 436$ for `CelebA`, $B = 194$ for `BAR`, and $B = 128$ for `CUB`.

For all models and tasks, we train models for a total of maximum T epochs with $T = 100$ for the synthetic tasks, $T = 300$ for `Waterbirds`, $T = 150$ for `CelebA`, $T = 200$ for `BAR`, and $T = 300$ for `CUB`. We stop training before this limit using early stopping based on the validation accuracy. In this setup, we interrupt training if a model's validation accuracy does not increase by more than 0.001 ("*stopping delta*") in the last 5 epochs ("*patience*"). Finally, to help find better optima across all tasks and models, we reduce the learning rate by $\times 0.1$ when the training loss plateaus for 10 epochs.

F.2 Model Selection

To conduct a fair evaluation of all baselines, we perform an extensive hyperparameter search for all methods. Here, we discuss which hyperparameters we search over for all methods, as well as the hyperparameters selected for each task after averaging the results over three distinct runs. This hyperparameter selection is done by performing a grid search over all combinations of hyperparameter candidates and selecting the model with the *highest validation accuracy*. The only exception for this is G-DRO, for which we perform model selection based on validation WGA given that we assume group labels are provided during training and we use G-DRO as an upper bound for unsupervised BM. Finally, to further reduce the computational cost of our model selection, the learning rate η and weight decay λ_{ℓ_2} hyperparameters used for all non-ERM and non-G-DRO methods are set to those selected for their ERM equivalent (i.e., we first train the ERM baseline, find the best learning rate η and weight decay λ_{ℓ_2} and fix those when performing model selection and training of all other baselines). Below we describe the hyperparameters we searched over for each baseline.

ERM - $\{\eta, \lambda_{\ell_2}\}$ When training ERM models, we select their learning rates from $\eta \in \{10^{-3}, 10^{-4}\}$ and their weight decay factors from $\lambda_{\ell_2} \in \{0, 0.0001, 0.01, 1\}$ (with 0.00001 added for the MNIST-based datasets as they are smaller and allow for larger searches). This results in a total of $2 \times 4 \times 3 = 24$ models trained per task to get the mean validation accuracy of each setup across three different random seeds. When selecting models based on validation accuracy, we got the following configuration for all tasks: $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0)$ for `Even-Odd`, $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0.0001)$ for `cMNIST`, $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0)$ for `BAR`, $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0.01)$ for `Waterbirds` and $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0.0001)$ for `CelebA`. For `CUB`, we observed that $\eta \in \{10^{-3}, 10^{-4}\}$ performed poorly, so we added $\eta = 10^{-2}$ as a candidate to this set. This resulted in us selecting $(\eta, \lambda_{\ell_2}) = (10^{-2}, 0.0001)$ for `CUB`.

G-DRO [12, 29] - $\{\eta, \lambda_{\ell_2}\}$ As in ERM, for G-DRO we carefully select the learning rate and the weight decay as both of these hyperparameters have been shown to be highly important for good worst-group performance [29]. Therefore, we select hyperparameters by searching over candidate learning rates $\eta \in \{10^{-3}, 10^{-4}\}$ and candidate weight decays $\lambda_{\ell_2} \in \{0, 0.0001, 0.01, 1\}$. Moreover, we use the stable online G-DRO algorithm by Sawaga et al. [29], with an exponential decay factor of $\gamma = 0.01$, as this approach has been shown to be more stable in practice, particularly when minority groups are small. Our hyperparameter grid search yields the following hyperparameters across the different group-annotated tasks: $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0.0001)$ for `Even-Odd`, $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0)$ for `cMNIST`, $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0.1)$ for `Waterbirds`, and $(\eta, \lambda_{\ell_2}) = (10^{-3}, 0.01)$ for `CelebA`.

LfF [23] - $\{q\}$ LfF uses a generalised cross-entropy loss [37] to dynamically learn a biased model for weighting the loss of an "unbiased" model. This loss depends on a hyperparameter $q \in (0, 1]$ that controls the level of bias amplification of the biased model (the loss becomes a vanilla cross-entropy loss as $q \to 0$). Therefore, across all tasks, we attempted values $q \in \{0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95\}$ for LfF. This yields the following hyperparameters across our different tasks: $q = 0.05$ for `Even-Odd`, `cMNIST`, `Waterbirds`, `CelebA`, and `CUB`, and $q = 0.1$ for `BAR`. We note that, as opposed to the hyperparameters selected when using best validation accuracy, the optimal configurations for LfF (i.e., the hyperparameters with the highest test loss) are more diverse: $q = 0.1$ for `Even-Odd` and CUB, $q = 0.75$ for `cMNIST` and `Waterbirds`, $q = 0.5$ for `CelebA`, and $q = 0.95$ for `BAR`. These results, therefore, show how determining the right value for q can be very hard in practice.

JTT [21] - $\{T, \lambda_{\mathbf{up}}\}$ As discussed in Section 2, JTT has two hyperparameters: T, the number of epochs one trains the identifier model for, and λ_{up}, how much we should upweight each mispredicted sample by the identifier model. Following the same official implementation by Liu et al. [21], we perform the upweighting in JTT by constructing a new dataset in which each mispredicted training sample is included λ_{up} times. This enables significantly better results and much more stable training. However, it comes with the added cost of significant training times as this new dataset could be as large as $\lambda_{\text{up}} N$ in the worst case. Following similar values used by the original authors of this work, we perform a grid search over $T \in \{1, 5, 10, 25, 50\}$ and $\lambda_{\text{up}} \in \{10, 25, 50, 100, \text{“ratio”}\}$, where “ratio” indicates a dynamic computation of λ_{up} whereas we set this value to the ratio between the size of the set of samples correctly predicted by the identifier model and the size of the set of samples mispredicted by the identifier model. The only exception for this is in CUB, where we only search over $T \in \{10, 25, 50\}$ and $\lambda_{\text{up}} \in \{10, 25\}$ as otherwise training times got intractable. This process yielded the following selected hyperparameters across all tasks: $(T, \lambda_{\text{up}}) = (50, 10)$ for `Even-Odd`, $(T, \lambda_{\text{up}}) = (50, 25)$ for `cMNIST`, $(T, \lambda_{\text{up}}) = (10, 25)$ for `Waterbirds`, $(T, \lambda_{\text{up}}) = (10, 1)$ for `CelebA`, $(T, \lambda_{\text{up}}) = (50, 25)$ for `BAR`, and $(T, \lambda_{\text{up}}) = (25, 25)$ for `CUB`.

MaskTune [3] - $\{\tau\}$ MaskTune operates by (1) first training a model via ERM, (2) then generating a new dataset by masking each sample $\mathbf{x}^{(i)}$ based on the saliency map [7] of that sample by the ERM model, and (3) finally fine-tuning the ERM model for a single epoch using a small learning rate on the newly constructed dataset. The intuition of this approach is that by masking areas of the image that a model is attending to, one may get rid of easily exploitable spurious correlations and force the model to learn to make a prediction using alternative features (i.e., by learning to generalise). Assuming that (1) we always fine-tune for a single epoch (as recommended by the authors), (2) we use X-GradCam [10] as the underlying saliency method (as used in the original paper), and (3) we set the fine-tuning learning rate to be the end learning rate of the original ERM model (as suggested by the authors), the only hyperparameter in MaskTune we control is $\tau \in \mathbb{R}$, the threshold controlling which pixels to mask in the saliency map of each sample. Following the values for τ used in MaskTune's original paper, we try values of τ in $\tau \in \{\mu, \mu + 0.5\sigma^2, \mu + \sigma^2, \cdots, \mu + 2.5\sigma^2, \mu + 3\sigma^2\}$ where μ is the mean of the saliency map pixels of a specific sample and σ^2 is the standard deviation (i.e., the actual threshold is a function of the sample's saliency map). This process results in us selecting the following hyperparameters across our tasks: $\tau = \mu + 3\sigma^2$ for `Even-Odd`, $\tau = \mu + 2.5\sigma^2$ for `cMNIST`, $\tau = \mu$ for `Waterbirds`, $\tau = \mu + 2.5\sigma^2$ for `CelebA`, $\tau = \mu + 1.5\sigma^2$ for `BAR`, and $\tau = \mu + 3\sigma^2$ for `CUB`.

TAB - $\emptyset$ As discussed in Section 5, our method TAB has no hyperparameters and, therefore, no need for fine-tuning. Therefore, we deployed TAB without changing any of the hyperparameters used for the equivalent ERM model with the exception that when training TAB's identifier model to collect training loss histories, we perform early *stopping based on the training accuracy* rather than the validation accuracy. We do this so that training of the identifier stops once the model has perfectly fitted the training set, at which point additional losses in the history come with diminishing returns.

F.3 Software and Hardware Used

Software For this work, we constructed our codebase based on the MIT-licensed open-source repository by Espinosa Zarlenga et al. [8,9], which provided a useful starting point for easy experiment tracking and deployment. Our implementation of TAB and all underlying DNNs is built on top of PyTorch 1.12 [26], a commonly used deep learning library with a BSD license. For G-DRO, we adapted the official implementation of this method by Sawaga et al. [29] by updating the code so that it could run within our infrastructure while maintaining all key pieces as in their MIT-licensed public repository. For LfF, we adapted the authors' official implementation into our own infrastructure, keeping the setup and generalised cross-entropy computation intact. For JTT, we reimplemented their training algorithm based on their official implementation. Finally, for MaskTune, we ported the authors' official implementation into our own infrastructure.

All figures with the exception of Figure 1 were generated using Matplotlib 3.5, a BSD-licensed Python plotting library. Figure 1 was instead generated using draw.io, an open-sourced drawing software distributed under an Apache 2.0 license. All of the code, configs, and scripts needed to recreate our results, will be made public through an open-source repository upon publication of this paper.

Resources We ran all of our experiments on two compute clusters. The first cluster consisted of a machine with four Nvidia Titan Xp GPUs and 40 Intel(R) Xeon(R) E5-2630 v4 CPUs. The second cluster consisted of a machine with a single Nvidia Quadro RTX 8000 GPU and eight Intel(R) Xeon(R) Gold 5218 CPUs. We estimate that all experiments, including initial explorations and all the model selections we ran, required between 450 and 500 GPU hours.

G Details for Motivation Experiments

In this section we provide details on the constructions of the figures used in Section 4, where we describe the importance of validation group annotations for model selection in BM pipelines. Figures 2, 3, and 4 were all generated by training a *single* Resenet-34 model for 50 epochs. This model is trained on our `Waterbirds` task for all figures and on the `CelebA` task for Figure 4. We follow the same approach as described in Appendix F to train this model in each respective task with the difference that, for the models shown in Figures 2 and 4, we do not perform early stopping (as we want to emphasise how biases in the validation set can in fact lead to an unwanted early stopping). Notice that in Figure 2 we use colours to indicate points in a curve that are higher (green) or lower (red) than the the metric at the beginning of the marked grey zone. Finally, in Figure 4 we split samples between bias-conflicting and bias-aligned by identifying minority groups lacking the spurious correlation as bias-conflicting. In `Waterbirds` we use ("waterbirds on land backgrounds" and "landbirds on water backgrounds") as our bias-conflicting groups, as they are both similarly underrepresented and lacking the spurious correlation, while in `CelebA` we use "not blonde male faces" as the bias-conflicting slice shown (as they are severely underrepresented and lacking the exploitable gender-based spurious correlation).

H Random Transformations to Augmented Samples

In this section, we explore how domain-specific knowledge can be used to improve TAB's performance even further through the use of augmentation transformations. For this, when we construct TAB's augmented set $\mathcal{D}'_l$, we apply an stochastic transformation function $\eta : \mathbb{R}^n \rightarrow \mathbb{R}^n$ to each sample (i.e., $\mathcal{D}'_l = \{(\eta(\mathbf{x}'^{(i)}), y'^{(i)}) \mid (\mathbf{x}'^{(i)}, y'^{(i)}) \in \mathcal{D}_l^-\}_{i=1}^z$). We explore this in our MNIST-based tasks as we know handwritten digits should be invariant to small perturbations in their angles and locations. Specifically, we let η be a transformation pipeline that applies small random rotations ($\theta \sim \text{Unif}([-\pi/6, \pi/6])$), translations ($\Delta x, \Delta y \sim \text{Unif}([0, l/10])$), and Gaussian blurring (using a 5x5 filter).

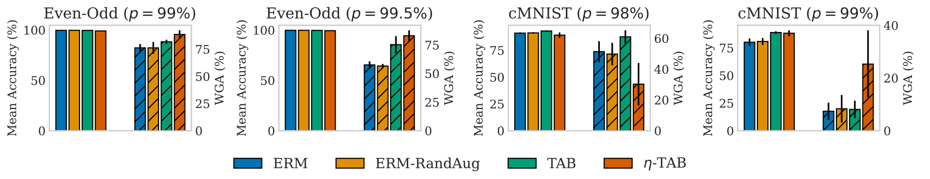

Fig. H.1: Mean accuracy (left group of solid bars) and WGA (right group of dashed bars) of TAB with and without random transformations of upsampled examples in `Even-Odd` and `cMNIST` as we vary p for both datasets. As baselines, we show the performance of an equivalent ERM model trained both with and without the same random transformations for its training set.

Our results, shown in Fig. H.1, show that in highly imbalanced domains (e.g., $p = 0.99$ in `cMNIST` and $p \in \{0.99, 0.995\}$ in `Even-Odd`), TAB's WGA can be significantly improved by adding domain-specific transformations to their augmented samples without a clear sacrifice in mean accuracy. Specifically, TAB's version including the transformations (*η-TAB*) achieves up to a 17% absolute improvement in WGA over the standard TAB pipeline for `cMNIST` when $p = 0.99$. Moreover, we see that these same benefits cannot be harvested by simply randomly applying the same transformations to the original training set during training. This is shown by the similar performance between the original ERM model and *ERM-RandAug*, an ERM model trained on a dataset where samples are randomly augmented using the same transformations as for η-TAB during training.

In contrast to the boost obtained by applying target transformations, we notice that such transformation may result in worse WGA when the worst-group imbalance is not extreme (e.g., $p = 0.98$ for `cMNIST`). Hence, although some of our preliminary results here suggest that these transformations can be useful in extremely imbalanced setups, more work is needed to understand (1) how to correctly design these transformations when domain knowledge is available, (2) how to determine whether such transformations can be properly applied without damaging a model's worst-group performance, and (3) what are some of the repercussions on TAB's PoU of introducing new hyperparameters coming from such transformation pipelines. We believe these to be promising and important directions for future work.

I Effect of Batching Clustering for TAB

In this section, we explore the effect of using mini-batched k-means [30] for clustering history losses in TAB's pipeline. Exploring how mini-batching can be used as a part of TAB can be insightful to understanding how our method may be able to scale to datasets where k-means may become a bottleneck (i.e., datasets with a size in the order of millions or billions of samples). With this goal in mind, in Figure I.2 we show the effect of using batches for k-means in our synthetic datasets. For these results, we use Scikit-learn's official MiniBatched-k-Means [27] implementation with its default settings except for the "reassignment ratio", which we decrease to 0.00001 to enable small clusters to be discovered (as we operate under the assumption that one of the clusters will be significantly smaller).

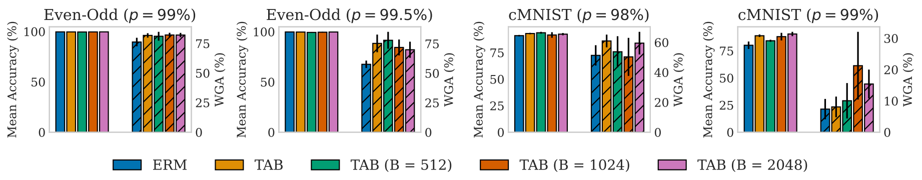

Fig. I.2: Mean accuracy (left group of solid bars) and WGA (right group of dashed bars) of TAB when using mini-batched k-means clustering for its dataset partition step. We show our results in `Even-Odd` and `cMNIST` while we vary p to simulate extremely biased setups. As baselines, we show the performance of equivalent ERM and TAB models.

Our results show that mini-batching k-means enables TAB to still learn less biased models than standard ERM models. This suggests that our method can still be easily deployed in practice to learn models that are less biased than their ERM equivalents in extremely large datasets without the need for any model selection. However, we also notice that batching comes with a significant variance that can, in some instances, lead to bad clusters being discovered. This results in a loss on some of the benefits we were able to obtain from TAB's original full dataset clustering. We hypothesise that this is due to the fact that, in these extremely group-imbalanced datasets, it is likely that no spurious-conflicting samples may be found in a given batch when learning clusters using mini-batched

k-means. Therefore, future work may focus on understanding how to fully recover TAB's worst-group performance when using only batches or subsets of the training set when discovering bias-conflicting and bias-aligned subgroups.

References

1. Ahmed, F., Bengio, Y., Van Seijen, H., Courville, A.: Systematic generalisation with group invariant predictions. In: International Conference on Learning Representations (2020)
2. Anthony, L.F.W., Kanding, B., Selvan, R.: Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. arXiv preprint arXiv:2007.03051 (2020)
3. Asgari, S., Khani, A., Khani, F., Gholami, A., Tran, L., Mahdavi Amiri, A., Hamarneh, G.: Masktune: Mitigating spurious correlations by forcing to explore. Advances in Neural Information Processing Systems **35**, 23284–23296 (2022)
4. Creager, E., Jacobsen, J.H., Zemel, R.: Environment inference for invariant learning. In: International Conference on Machine Learning. pp. 2189–2200. PMLR (2021)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
6. Deng, L.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)
7. Erhan, D., Bengio, Y., Courville, A., Vincent, P.: Visualizing higher-layer features of a deep network. University of Montreal **1341**(3), 1 (2009)
8. Espinosa Zarlenga, M., Collins, K.M., Dvijotham, K., Weller, A., Shams, Z., Jamnik, M.: Learning to receive help: Intervention-aware concept embedding models. Advances in Neural Information Processing Systems (2023)
9. Espinosa Zarlenga, M., Pietro, B., Gabriele, C., Giuseppe, M., Giannini, F., Diligenti, M., Zohreh, S., Frederic, P., Melacci, S., Adrian, W., et al.: Concept embedding models: Beyond the accuracy-explainability trade-off. In: Advances in Neural Information Processing Systems, vol. 35, pp. 21400–21413. Curran Associates, Inc. (2022)
10. Fu, R., Hu, Q., Dong, X., Guo, Y., Gao, Y., Li, B.: Axiom-based grad-cam: Towards accurate visualization and explanation of cnns. arXiv preprint arXiv:2008.02312 (2020)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
12. Hu, W., Niu, G., Sato, I., Sugiyama, M.: Does distributionally robust supervised learning give robust classifiers? In: International Conference on Machine Learning. pp. 2029–2037. PMLR (2018)
13. Kim, M.P., Ghorbani, A., Zou, J.: Multiaccuracy: Black-box post-processing for fairness in classification. In: Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society. pp. 247–254 (2019)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
15. Kirichenko, P., Izmailov, P., Wilson, A.G.: Last layer re-training is sufficient for robustness to spurious correlations. ICLR (2023)

16. Koh, P.W., Nguyen, T., Tang, Y.S., Mussmann, S., Pierson, E., Kim, B., Liang, P.: Concept bottleneck models. In: International Conference on Machine Learning. pp. 5338–5348. PMLR (2020)
17. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Annual symposium on theoretical aspects of computer science. pp. 404–413. Springer (1999)
18. LaBonte, T., Muthukumar, V., Kumar, A.: Towards last-layer retraining for group robustness with fewer annotations. NeurIPS (2023)
19. Levy, D., Carmon, Y., Duchi, J.C., Sidford, A.: Large-scale methods for distributionally robust optimization. Advances in Neural Information Processing Systems **33**, 8847–8860 (2020)
20. Li, Z., Hoogs, A., Xu, C.: Discover and mitigate unknown biases with debiasing alternate networks. In: European Conference on Computer Vision. pp. 270–288. Springer (2022)
21. Liu, E.Z., Haghgoo, B., Chen, A.S., Raghunathan, A., Koh, P.W., Sagawa, S., Liang, P., Finn, C.: Just train twice: Improving group robustness without training group information. In: International Conference on Machine Learning. pp. 6781–6792. PMLR (2021)
22. Liu, Z., Luo, P., Wang, X., Tang, X.: Large-scale celebfaces attributes (celeba) dataset. Retrieved August **15**(2018), 11 (2018)
23. Nam, J., Cha, H., Ahn, S., Lee, J., Shin, J.: Learning from failure: De-biasing classifier from biased classifier. Advances in Neural Information Processing Systems **33**, 20673–20684 (2020)
24. Papadimitriou, C.: Algorithms, games, and the internet. In: Proceedings of the thirty-third annual ACM symposium on Theory of computing. pp. 749–753 (2001)
25. Paranjape, B., Dasigi, P., Srikumar, V., Zettlemoyer, L., Hajishirzi, H.: AGRO: Adversarial Discovery of Error-prone groups for Robust Optimization. arXiv preprint arXiv:2212.00921 (2022)
26. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703 (2019)
27. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
28. Pezeshki, M., Kaba, O., Bengio, Y., Courville, A.C., Precup, D., Lajoie, G.: Gradient starvation: A learning proclivity in neural networks. Advances in Neural Information Processing Systems **34**, 1256–1272 (2021)
29. Sagawa, S., Koh, P.W., Hashimoto, T.B., Liang, P.: Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. arXiv preprint arXiv:1911.08731 (2019)
30. Sculley, D.: Web-scale k-means clustering. In: Proceedings of the 19th international conference on World wide web. pp. 1177–1178 (2010)
31. Shrestha, R., Kafle, K., Kanan, C.: Occamnets: Mitigating dataset bias by favoring simpler hypotheses. In: European Conference on Computer Vision. pp. 702–721. Springer (2022)
32. Sohoni, N., Dunnmon, J., Angus, G., Gu, A., Ré, C.: No subclass left behind: Fine-grained robustness in coarse-grained classification problems. Advances in Neural Information Processing Systems **33**, 19339–19352 (2020)
33. Taghanaki, S.A., Choi, K., Khasahmadi, A.H., Goyal, A.: Robust representation learning via perceptual similarity metrics. In: International Conference on Machine Learning. pp. 10043–10053. PMLR (2021)

34. Tsirigotis, C., Monteiro, J., Rodriguez, P., Vazquez, D., Courville, A.: Group robust classification without any group information. NeurIPS (2023)
35. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The caltech-ucsd birds-200-2011 dataset. Tech. Rep. CNS-TR-2011-001, California Institute of Technology (2011)
36. Zhang, B.H., Lemoine, B., Mitchell, M.: Mitigating unwanted biases with adversarial learning. In: Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society. pp. 335–340 (2018)
37. Zhang, Z., Sabuncu, M.: Generalized cross entropy loss for training deep neural networks with noisy labels. Advances in neural information processing systems **31** (2018)
38. Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., Torralba, A.: Places: A 10 million image database for scene recognition. IEEE transactions on pattern analysis and machine intelligence **40**(6), 1452–1464 (2017)