IMA206 Project

Learning Adversarially Fair and Transferable Representations (LAFTR)

Final Report

Authors:
Mariana DUTRA
Anna JÄRVINEN
Mariana OLM
Felipe VICENTIN

Supervisor:
Loïc LE FOLGOC

June 2025

Contents

1 Introduction

Many fields rely on large datasets do build models that automate decision-making. However, these datasets often suffer from biased data with respect to some sensitive attribute. As an example, a company may decide to automate the selection processes of new employees. Their model would be fed data from the last 50 years of applicants. Due to some bias from the selection committee, people of a certain skin color tended to get a job offer more frequently. Then, the model would learn this pattern and reproduce this behavior to new applicants. Certainly, the skin color should not influence the recruitment of new hires. So, we can say that the skin color is a *sensitive attribute* in this example.

More formally, one can define in a classification problem for $Y \in \{0, 1\}$ a sensitive attribute $A \in \{0, 1\}$ in the dataset $\mathcal{X}$ that should not bias the decision. Surely, a naïve solution is to simply remove A from the dataset, but this often is not possible. In many instances, A can be either easily derived from the other attributes, or A may have a direct impact on the decision, even without biases. We can essentially define "fairness" in a model by *Demographic Parity* (DP) or *Equal Odds* (EO).

The model we are following can be described using the dependency diagram in Figure 1. The parents of each node specify the dependencies. In this case, we see that the input to the model is $P(A)$, $P(Y|A, S)$ and $P(X|A, Y, w)$. Knowing these probability distributions, one can estimate $P(Y|X, A, w, S)$, or, more simply, $P(Y|X, A)$.

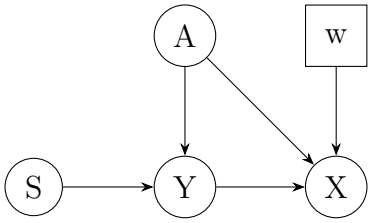

Figure 1: Diagram showing the relationship between A (sensitive attribute), Y (class of data), X (data), w (hidden parameters) and S (whether training or testing scenario).

In a DP setting, we enforce that the rate of a positive outcome ($\hat{Y} = 1$) is the same regardless of A, i.e.,

$$P(\hat{Y} = 1 \mid A = 0) = P(\hat{Y} = 1 \mid A = 1).$$

As previously said, A might have a direct impact on Y even without biases, which formally can be written as $P(Y = 1|A = 0) \neq P(Y = 1|A = 1)$. In this scenario, DP has clear limitations. Thus, the proposed solution (EO) is to enforce the rate of errors to be equal across groups. Said differently,

$$\begin{cases} P(\hat{Y} \neq Y|A = 0, Y = 0) = P(\hat{Y} \neq Y|A = 1, Y = 0); \\ P(\hat{Y} \neq Y|A = 0, Y = 1) = P(\hat{Y} \neq Y|A = 1, Y = 1). \end{cases}$$

The proposed way to deal with these constraints according to the LAFTR paper is to use an adversarial approach [1].

The main idea can be summarized in the following steps, given a dataset $(X, A, Y) \in \mathbb{R}^{n-1} \times \{0, 1\} \times \{0, 1\}$:

1. Train an Encoder $f \colon \mathbb{R}^n \to \mathbb{R}^m$ that will transform the data X to a latent representation $Z \in \mathbb{R}^m$;

2. Train a Classifier $g\colon \mathbb{R}^m \to \{0,1\}$ from the latent representation Z to a prediction $\hat{Y}$;

3. While the classifier and encoder are being trained, also train an adversary h that can predict $\hat{A}$ from the latent representation Z and possibly the label Y.

It is worth noting that a theoretical difference between ensuring DP or EO is the input of the adversary h. In DP, we let the latent representation Z be the only input to h, i.e., $\hat{A} = h(Z)$. When optimizing for EO, we also add the true label Y as an input to the adversary, i.e., $\hat{A} = h(Z, Y)$. The end goal is to have an encoder that transfers all the relevant information to the latent space, without replicating the bias. Then, the adversary will not be able to correctly predict A from it.

To this end, we will construct two adversarial loss functions that guarantee either DP or EO when optimized. Firstly, we define the *sensitive groups* of the dataset $\mathcal{X}$, for $a, y \in \{0,1\}^2$:

$$\mathcal{D}_a = \{(X, A) \in \mathcal{X} \mid A = a\}$$
$$\mathcal{D}_a^y = \{(X, A) \in \mathcal{X} \mid A = a, Y = y\}.$$

Finally, we define the adversarial losses:

$$L_{\text{Adv}}^{\text{DP}}(h) = -1 + \sum_{a\in\{0,1\}} \frac{1}{|\mathcal{D}_a|} \sum_{(X,A)\in\mathcal{D}_a} |h(f(X,A)) - a| \tag{1}$$

$$L_{\text{Adv}}^{\text{EO}}(h) = -2 + \sum_{a\in\{0,1\}} \sum_{y\in\{0,1\}} \frac{1}{|\mathcal{D}_a^y|} \sum_{(X,A)\in\mathcal{D}_a^y} |h(f(X,Y),Y) - a|. \tag{2}$$

In other words, the losses are averaging how many errors were made when predicting $\hat{A}$. The difference between DP and EO is that in DP we average according to A, while in EO we average according to both A and Y.

Now, letting L_C be some classification loss (e.g., Cross Entropy), we can define the following combined loss:

$$L(f, g, h) = L_C(\hat{Y}, Y) - \gamma L_{\text{Adv}}(\hat{A}, A)$$

This combined loss is the one used to optimize the encoder and classifier. The γ parameter is non-negative and controls how the encoder and classifier should punish the adversary. If $\gamma = 0$, then the adversary has no obstacle to learn A from Z and we could say that the latent representation is biased. If $\gamma \to \infty$, then the encoder will hide all information about A in the latent representation that could be learned by the adversary. An overview of the main algorithm can be seen in Algorithm 1.

Algorithm 1: LAFTR train

Input: Dataset $\mathcal{X}$, $\gamma \geq 0$

Output: Encoder f, classifier g and adversary h

for *batch* $(X, A, Y) \subseteq \mathcal{X}$ **do**
  Freeze h
  $f, g \leftarrow \arg\min_{f,g} L(f, g, h)$
  Freeze f and g
  $h \leftarrow \arg\min_h L_{\text{Adv}}(h)$
return f, g, h

1.1 Objective

Our main goal with this project is to both implement the LAFTR model and test it in synthetic data. One of the main shortcomings of its paper is that the dataset used to test the method is biased by default. This means that the number of metrics that can be extracted from it is limited.

The approach proposed in this project, however, can clearly show what the effects of LAFTR are, since we are working with datasets that are at first unbiased. This means that not only we can compare the model with some ground-truth, but we are also able to control how much bias the dataset gets. In particular, we will be using a modified version of MNIST dataset that is binarized.

Secondly, we test the same model in a more general setting. We will use the CIFAR-10 dataset to test whether the proposed method generalizes to a multi-class setting. We also take the opportunity to test non-binary sensitive attributes. This will be detailed in section 2.

1.2 Generalizations

During the development of our project, we decided to generalize the LAFTR model to non-binary data. Specifically, we let $Y \in \{0, \dots, C-1\}$ and $A \in \{0, \dots, K-1\}$. The domains and counter-domains of f, g and h can be easily adapted accordingly. The most complex part to generalize is the adversarial loss.

The original LAFTR implementation does not use Equations 1 and 2 directly as losses. Instead, Madras et. al. used an ℓ_1-norm approach, letting the output of h be $P(A=1)$ through a sigmoid function. It is easy to see that, for all a,

$$|P(A=1) - a| = 1 - P(A=a).$$

We can use this fact to generalize the adversarial losses. If $A \in \{0, \dots, K-1\}$, then the output of h is a vector of probabilities $(P(A=0), \dots, P(A=K))$, computed through a softmax function. Then, we can naturally generalize the loss as

$$L_{\text{Adv}}^{\text{DP}}(h) = -1 + \sum_{a=0}^{K-1} \frac{1}{|\mathcal{D}_a|} \sum_{(X,A)\in\mathcal{D}_a} (1 - [h(Z)]_a) \tag{3}$$

$$L_{\text{Adv}}^{\text{EO}}(h) = -2 + \sum_{a=0}^{K-1} \sum_{y=0}^{C-1} \frac{1}{|\mathcal{D}_a^y|} \sum_{(X,A)\in\mathcal{D}_a^y} (1 - [h(Z)]_a). \tag{4}$$

Thus, the LAFTR model is generalized to arbitrary classifications and non-binary sensitive attributes.

1.3 Project architecture

To structure our code, we decided to take an Object Oriented-based approach. This means that every dataset, model, loss function and even the training was organized into classes. This approach allows us to easily divide the work into modules that operate agnostic to each other. Above all, it naturally avoids confusing code and greatly simplifies the understanding of what each part of the implementation does and its responsibilities.

Figures 2 and 3 show how the project is structured, including base classes and inheritance.

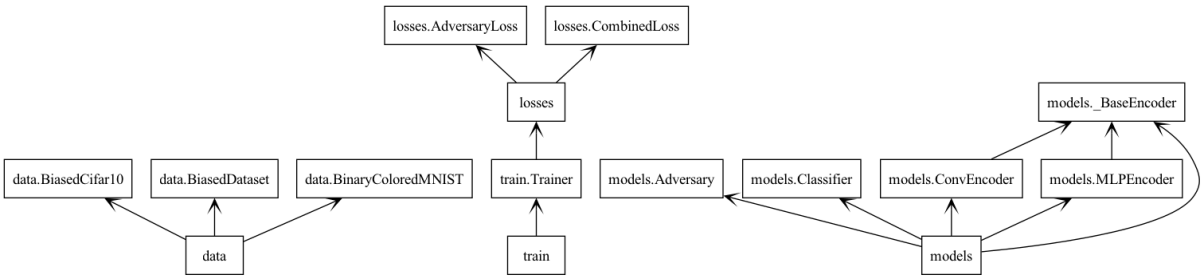

Figure 2: Packages used in the project.

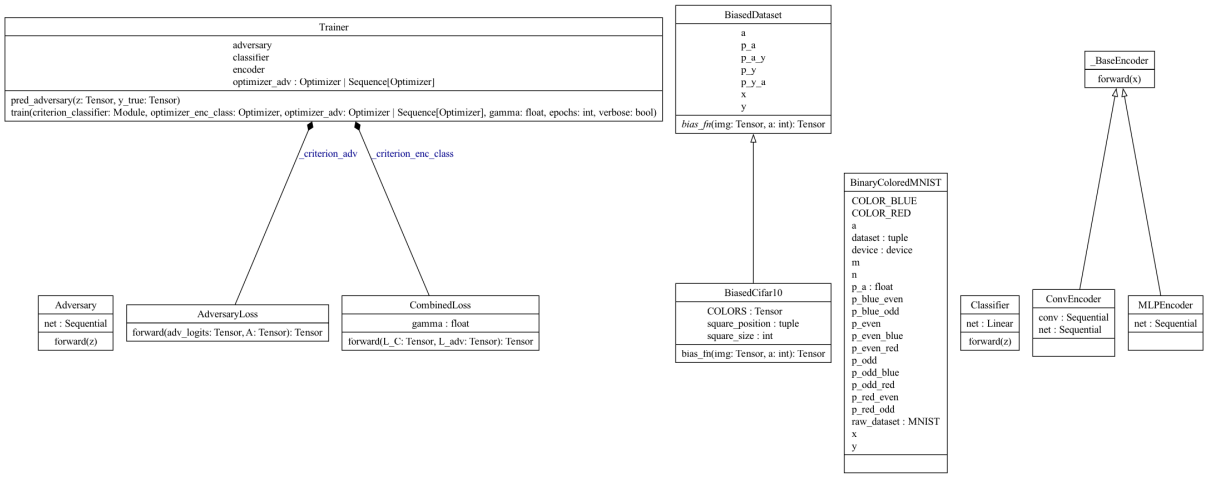

Figure 3: Classes used in the project.

2 Data

To evaluate the effectiveness of the LAFTR framework in reducing unfair biases, we construct a controlled setting in which the bias is intentionally introduced into the dataset. In particular, we inject a spurious correlation between the class label Y and a synthetic sensitive attribute A, which does not exist in the original data. This attribute is statistically linked to the label, yet unrelated to the actual task, mimicking real-world scenarios where sensitive attributes can spuriously influence model predictions. By controlling the strength and structure of this correlation, we can assess how well LAFTR separates meaningful task information from biased or shortcut features.

2.1 Biased Data Construction

We define the bias in the data through a conditional probability matrix $M \in \mathbb{R}^{C \times K}$, where $M_{i,j} = P(Y = i \mid A = j)$. Each column of this matrix represents a probability distribution over the class labels given a fixed attribute value, and must satisfy the condition:

$$\sum_{i=1}^{C} M_{i,j} = 1 \quad \text{for all } j \in \{1, \ldots, K\}.$$

To construct the matrix M, we define a biasing scheme where, for each attribute group $A = j$, the class distribution is a convex combination of a uniform distribution and a deterministic one-hot vector:

$$P(Y = i \mid A = j) = (1 - \beta) \cdot \frac{1}{C} + \beta \cdot \mathbf{1}_{\{i=d_j\}},$$

where $\beta \in [0, 1)$ is a scalar parameter that controls the strength of the spurious correlation, and d_j is the index of the dominant class associated with attribute $A = j$. This formulation allows us to transition smoothly between an unbiased dataset (when $\beta = 0$) and a highly biased one (as β approaches 1), making it possible to carefully adjust the degree of bias introduced during training.

In order to generate biased samples consistent with the defined matrix M, we must assign an attribute value A to each data point based on its ground truth label Y. This requires computing the posterior distribution $P(A \mid Y)$, which is not directly known. We begin by estimating the marginal distribution $P(Y)$ empirically from the dataset, based on the observed class frequencies.

To compute the marginal distribution $P(A)$, we apply the *law of total probability*, which states:

$$P(Y = i) = \sum_{j=1}^{K} M_{i,j} \cdot P(A = j).$$

In matrix notation, this becomes:

$$\mathbf{p}_y = M\mathbf{p}_a,$$

where $\mathbf{p}_y \in \mathbb{R}^C$ is the distribution over class labels and $\mathbf{p}_a \in \mathbb{R}^K$ is the unknown attribute prior.

Our initial approach was to solve this system using the *Moore-Penrose pseudoinverse*:

$$\mathbf{p}_a = M^+\mathbf{p}_y,$$

but this method proved numerically unstable in practice. Instead, we adopt a more stable *least-squares solution*, following the approach recommended in PyTorch's linear algebra utilities, which provides a robust approximation for $\mathbf{p}_a$.

With the estimated $P(A)$, we apply *Bayes' theorem* to compute the posterior distribution:

$$P(A \mid Y) = \frac{P(Y \mid A) \cdot P(A)}{P(Y)},$$

which allows us to sample the attribute A for each data point according to the desired correlation structure.

2.2 Biased Binary Colored MNIST

The original MNIST dataset consists of grayscale images of handwritten digits from 0 to 9. To create a binary classification task, we relabel each image based on the parity of the digit: even digits (0, 2, 4, 6, 8) are assigned label 0, and odd digits (1, 3, 5, 7, 9) are assigned label 1.

To inject a synthetic bias into the data, we define a sensitive attribute $A \in \{0, 1\}$ corresponding to two background colors: red and blue. Each image is modified by coloring its background either red or blue, depending on the value of A. This attribute is sampled using the posterior distribution $P(A \mid Y)$, which is derived using Bayes' theorem as explained in the previous section. By adjusting the parameters $P(Y \mid A)$ and $P(A)$, we control the strength of the correlation between the parity label Y and the color attribute A, simulating a spurious association.

This allows us to create a synthetic biased dataset, such as the one sampled in Figure 4. Images with blue backgrounds are assigned $A = 1$, and those with red backgrounds receive $A = 0$. Importantly, the background color provides no meaningful information for the task of digit parity classification, making it a purely spurious signal.

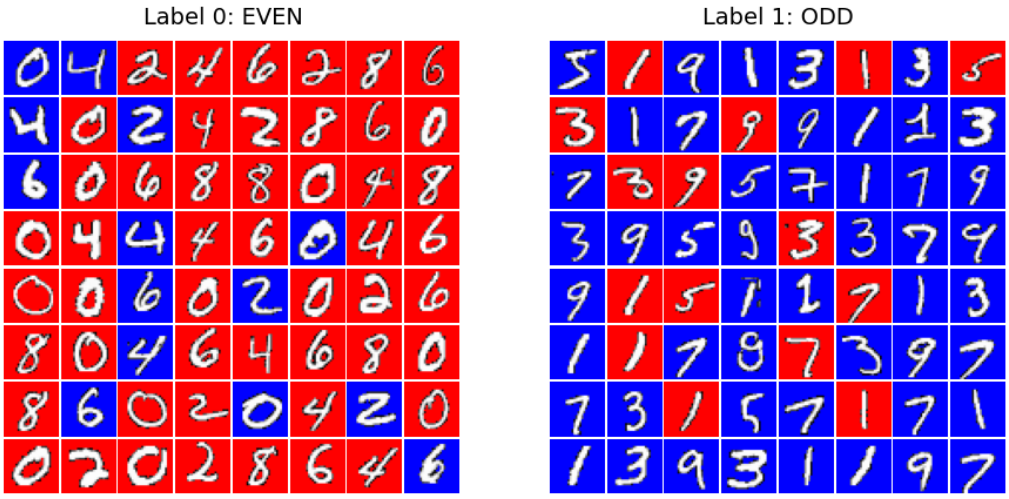

Figure 4: Samples of biased MNIST data by true label. ($\beta = 0.6$)

2.3 Biased CIFAR-10

The original CIFAR-10 dataset contains RGB images from 10 object categories. In this case, we preserve the original multi-class labeling $Y \in \{0, \ldots, 9\}$ and synthetically introduce bias by adding a colored square to each image. The sensitive attribute $A \in \{0, \ldots, 9\}$ corresponds to one of 10 predefined colors, which are injected as visual markers into the image.

For each sample, an attribute value $A = j$ is assigned according to the posterior distribution $P(A \mid Y)$, computed from a chosen conditional distribution $P(Y \mid A)$ and the prior $P(A)$. A visual bias is then applied by adding a square patch of color j to the image, effectively associating each class with a specific color attribute to varying degrees depending on the defined bias strength.

In Figure 4, we can see this visual bias for each color attribute and the corresponding dominant class it is attached to in one instance of Biased CIFAR-10. As with the MNIST case, the injected attribute A is unrelated to the actual classification task and serves as a synthetic spurious feature.

Figure 5: Samples of biased CIFAR-10 data.

3 Methodology

3.1 Adversary Loss

The implementation of the DP loss is straight forward from Equation 3. As explained previously, the adversary in an EO setting also has access to the true classification label Y from each data point. Certainly, one way to implement it is to create a separate adversary model for EO. However, a cleaner and equivalent approach is to reuse the same DP Adversary for the EO scenario. To this end, it is necessary to define a DP adversary for each possible attribute Y, i.e., we have C DP adversaries: $h_1, \ldots, h_C$. So, if we want to predict $\hat{a}$ for (x, y), we can simply use the adversary $h(x, y) = h_y(x)$. It is evident how these two approaches are equivalent. In fact, Madras et. al. decided to implement EO in their paper with many DP adversaries as well. Thus, we greatly simplify our code, while keeping all the theoretical guarantees of an EO adversary.

3.2 Models

The goal of LAFTR is to learn a data representation Z that preserves the predictive features of the input X while removing information about the sensitive attributes A. This is achieved through an adversarial learning approach, where the encoder is optimized to fool an adversary attempting to infer the sensitive attribute.

We do this by following a variant of the generalized model proposed by Madras et. al., which is depicted in Figure 6. As previously stated, the true label Y is used as input of the Adversary h when we optimize for Equalized Odds, which is represented by the dotted arrow in the figure.

Unlike the full LAFTR framework, we do not include a decoder in our architecture because we do not aim to support transferability: our focus is strictly on producing fair representations. We detail each component of this model below.

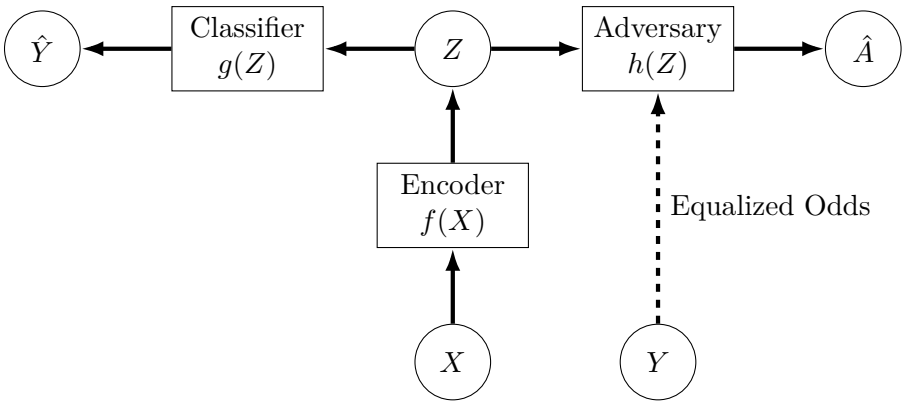

Figure 6: Generalized model for learning adversarially fair representations.

3.2.1 Encoder

We implemented three different encoders - MLPEncoder, ConvEncoderMNIST and ConvEncoderCIFAR - all of which inherit from a shared base class. MLPEncoder is a simple, fully connected network. It takes flattened image pixels as input and passes them through one or more hidden layers to output a latent representation $z \in \mathbb{R}^m$. While the implementation supports an arbitrary number of hidden layers, in practice, we used two.

The ConvEncoderMNIST (Figure 7) consists of two convolutional layers. The first layer has 6 output channels and uses a 5×5 kernel, followed by a LeakyReLU activation and 2×2 max pooling. The second convolutional layer maps from 6 channels to the latent dimension using a 3×3 kernel, followed by another LeakyReLU and an adaptive average pooling layer. Both of these encoders were designed for MNIST dataset.

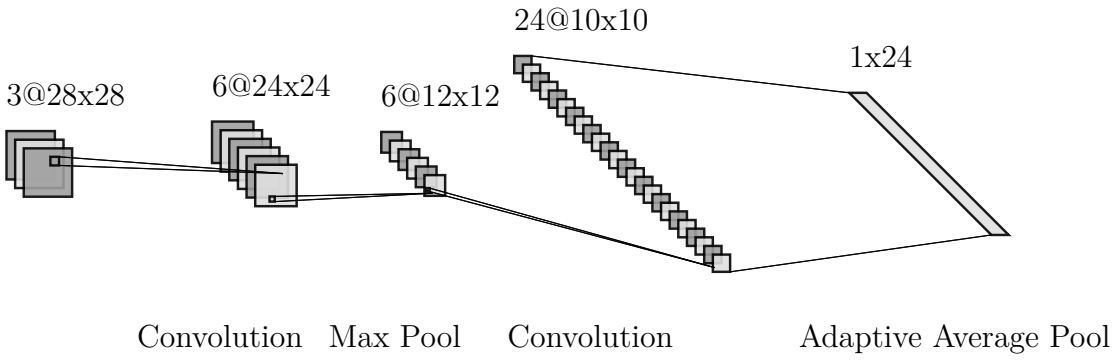

Figure 7: Convolutional architecture of Encoder for MNIST dataset.

The CIFAR-10 dataset is significantly more complex than MNIST, which requires a deeper and more expressive model to capture meaningful features for classification. To address this, we designed ConvEncoderCIFAR, a larger convolutional network designed for higher-resolution inputs. It uses five strided convolutional layers with channel widths [8, 16, 32, 64, *latent_dim*], each followed by ReLU and a batch normalization, ending with Flatten to produce the latent vector.

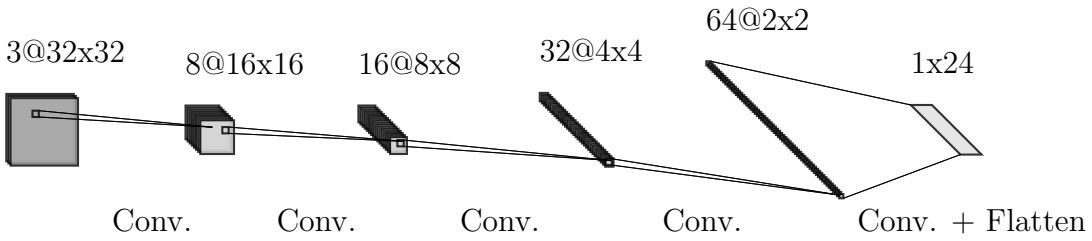

Figure 8: Convolutional architecture of Encoder for CIFAR-10 dataset.

We intentionally made the encoder the most complex component in our architecture, while keeping the classifier and adversary simple. By doing this, our architecture ensures that the encoder must extract a latent representation that is both informative for the prediction task (e.g. digit parity) and invariant to the sensitive attribute (e.g. color).

3.2.2 Classifier and Adversary

Both the classifier and adversary are simple neural networks that operate on the latent representation Z. Classifier is a single linear layer that maps from the latent dimension to C output un-normalised logits aiming to estimate the class label of the input. The adversary consists of a two-layered MLP with ReLU activation. It maps from the latent vector to K output logits and is trained to predict the sensitive attribute A, using the L_{Adv} loss defined in Equation 3 or 4. The choice of adversarial loss function depends on whether DP or EO is selected fairness criterion.

3.3 Training

As specified in Algorithm 1, the training of the model consists of an optimization step in the Encoder and Classifier, followed by the step of the Adversary. This behavior is achieved simply by specifying differing `pytorch` optimizers: one for the f and g, and other (or others) for h.

Training for f and g is simple and will not be explained. We focus in this report on h's training. Algorithm 2 details how the adversary training is performed. In the pseudo-code we abuse notation to let $L^{\text{DP}}_{\text{Adv}}(\hat{A}, A)$, and $L^{\text{EO}}_{\text{Adv}}(\hat{A}, A)$ respectively, be the adversarial loss defined on h. The main point of interest is in the EO case: we might have $|Y_c| = 0$, i.e., no data point from class c in the batch. In this case, we choose not to update the cumulative adversarial loss L_{adv}. This design decision was made because we believe that we should not increase the loss if we are not applying a gradient step in the respective adversary.

Algorithm 2: Train Adversary

Input: Data batch X, sensitive labels A, class labels Y, encoder f, adversary h
Output: Adversarial loss L_{adv}

$Z \leftarrow f(X)$;
$L_{\text{adv}} \leftarrow 0$;
if *Demographic Parity* **then**
    $\hat{A} \leftarrow h(Z)$;
    $L_{\text{adv}} \leftarrow L_{\text{Adv}}^{\text{DP}}(\hat{A}, A)$;
    Back-propagate on h with respect to L_{adv};
else
    for $c \leftarrow 0$ **to** $C - 1$ **do**
        $Y_c \leftarrow \{y_i \in Y \mid y_i = c\}$;
        if $|Y_c| > 0$ **then**
            $Z_c \leftarrow \{z_i \in Z \mid y_i = c\}$;
            $A_c \leftarrow \{a_i \in A \mid y_i = c\}$;
            $\hat{A}_c \leftarrow h_c(Z_c)$;
            $L'_{\text{adv}} \leftarrow L_{\text{Adv}}^{\text{EO}}(\hat{A}_c, A_c)$;
            Back-propagate on h_c with respect to L'_{adv};
            $L_{\text{adv}} \leftarrow L_{\text{adv}} + L'_{\text{adv}}$;
return L_{adv};

4 Results

In this section, we run experiments to evaluate how models trained using the LAFTR framework behave under different test scenarios. Our goal is to understand to what extent the learned representations remain both predictive and fair when exposed to datasets with varying bias configurations.

We begin each experiment by training a baseline model, which consists of a standard encoder and classifier trained with cross-entropy loss, without any fairness constraint. This model is trained on a biased dataset, either MNIST or CIFAR-10, where a synthetic correlation of strength β has been introduced between the labels Y and sensitive attributes A as explained in section 2. We then evaluate the trained baseline on three versions of the test data:

1. A set with the same bias as the training distribution;
2. An unbiased version where A and Y are independent;
3. A version with different bias (which we call "inversed bias"), where the dominant class for each value of A in $P(Y|A)$ is changed.

These experiments reveal how a model trained naively on biased data tends to overfit to spurious correlations and generalizes poorly when the bias structure changes.

We then train models using the LAFTR framework on the same biased training set. For a range of values of the adversarial weight γ, we save the resulting encoder, classifier, and adversary. Each trained encoder-classifier model is then tested on the same three distributions: biased, unbiased, and inversed bias. By doing so, we expect to directly

compare each case against the corresponding baseline and evaluate whether LAFTR achieves its intended goal, and for which values of γ it performs best.

4.1 MNIST $-\ \beta = 0.8$

Figure 9 shows the results on the MNIST dataset with a high bias level of $\beta = 0.8$, using an MLP encoder. In the top row, we observe that the baseline classifier (dashed line) performs very well on the biased test set, achieving around 97% accuracy. However, its performance drops significantly on the unbiased and inversed bias versions, confirming that the model relies heavily on the spurious attribute to make predictions.

As the adversarial weight γ increases in LAFTR, the classifier's accuracy begins to fluctuate and eventually declines. At the same time, we observe a temporary improvement on the unbiased and inversed test sets for intermediate values of γ (e.g., around 10^{-2}), after which performance begins to fall. This indicates that while the adversary helps remove information about the sensitive attribute from the learned representation, it can also suppress task-relevant features when its influence becomes too strong. The trade-off is especially apparent in the inversed bias setting, where moderate values of γ lead to notable gains over the baseline (up to 85% accuracy), before the classifier weakens at higher γ values.

4.2 CIFAR-10 $-\ \beta = 0.8$ and $K = 10$

Figure 10 presents the results on the CIFAR-10 dataset with a high bias level of $\beta = 0.8$ and 10 sensitive attributes ($K = 10$). In the top row, we see that the baseline classifier performs well on the biased test set, achieving around 85% accuracy. However, its performance drops on the unbiased and inversed bias test sets, reaching only around 40%. This confirms that the model has overfit to the spurious correlation between the target label Y and the synthetic attribute A.

Similar to the MNIST results, increasing the adversarial weight γ in LAFTR helps reduce this dependency on A. For intermediate values of γ (around 10^{-1}), the classifier shows improved generalization to the unbiased test set. However, as γ increases further, the adversary begins to suppress task-relevant information, leading to a decline in overall performance. This trade-off is again apparent, the highest improvement over the baseline on the unbiased test set occurs at moderate regularization (43.5%), while performance on the inversed bias set peaks at 34%, matching the baseline but not exceeding it.

4.3 CIFAR-10 $-\ \beta = 0.9999$ and $K = 2$

Figure 11 presents the results for the CIFAR-10 dataset under an extreme bias configuration, with $\beta = 0.9999$ and $K = 10$ sensitive attribute values. In this scenario, the correlation between the target label Y and the synthetic attribute A is nearly deterministic. As expected, the baseline classifier achieves almost perfect accuracy on the biased test set but drops to chance-level performance (around 10%) on both the unbiased and inversed bias sets. This confirms that the model has entirely overfit to the spurious attribute and fails to generalize when the bias distribution changes.

When applying LAFTR, we observe a clear trade-off: as the adversarial weight γ increases, the classifier's reliance on the biased attribute is reduced, which slightly improves performance on the inversed bias set (up to 22.4%). However, no improvement is

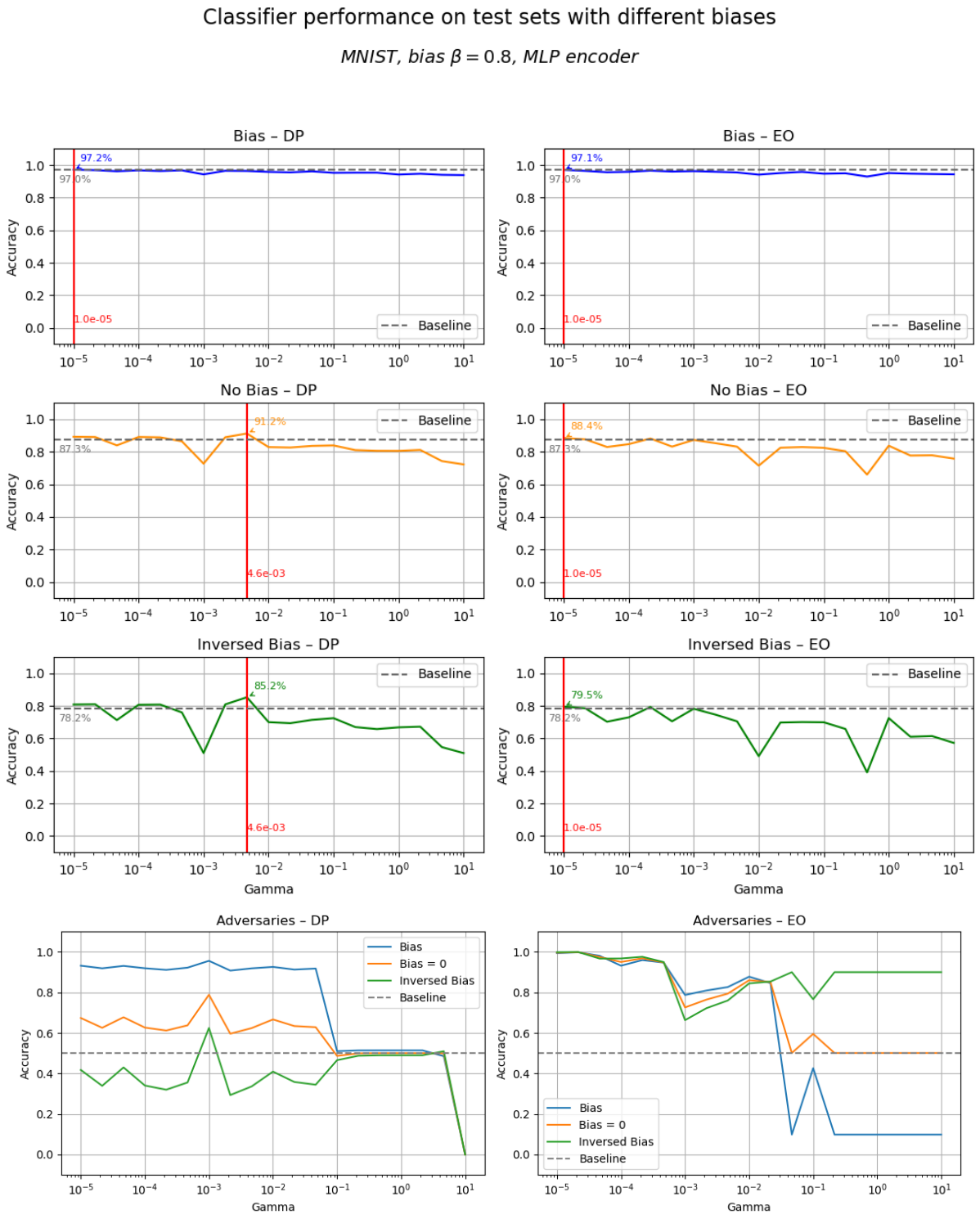

Figure 9: Results on MNIST with $\beta = 0.8$

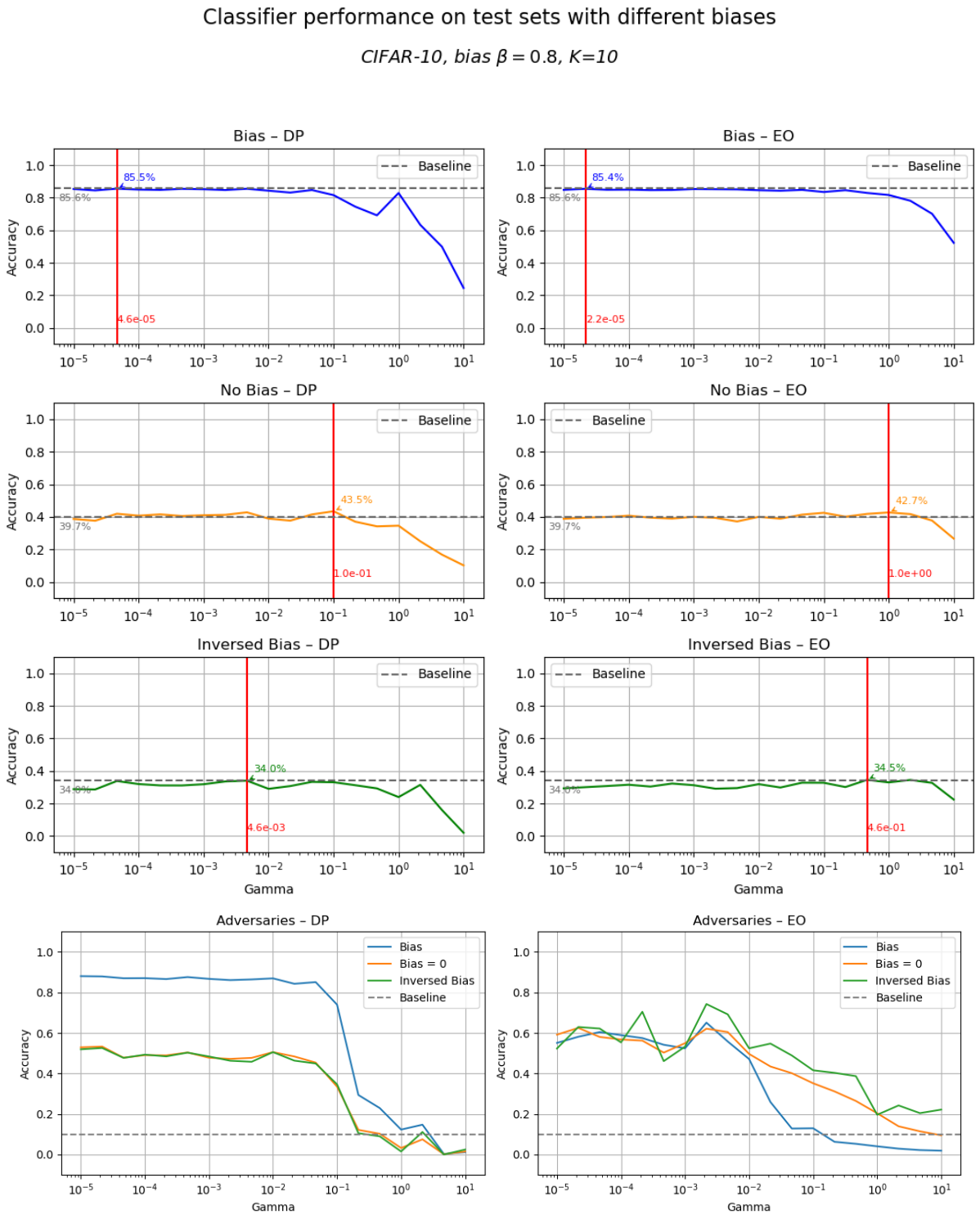

Figure 10: Results on CIFAR-10 with $\beta = 0.8$, $K = 10$

observed on the unbiased test set, which remains at chance level across all values of γ. This is particularly important, as the unbiased setting is the most meaningful evaluation scenario, as it reflects the model's ability to make predictions based on task-relevant features alone. In this extreme case, LAFTR fails to recover a usable representation that generalizes beyond the bias, suggesting that the adversary removes not only spurious information but also critical signal required for classification.

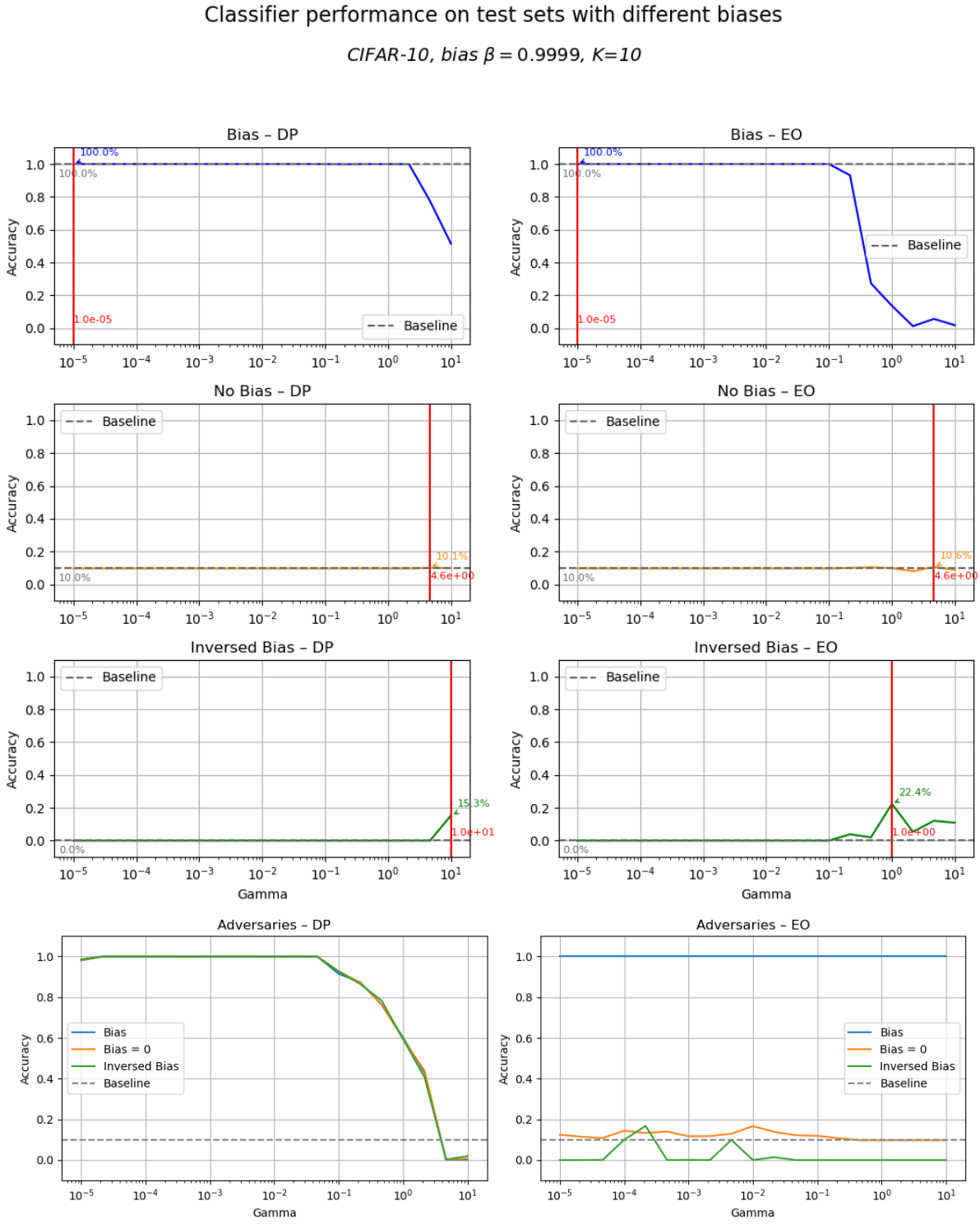

Figure 11: Results on CIFAR-10 with $\beta = 0.9999$, $K = 10$

5 Conclusion

Through the results, we can see a clear trend across experiments: the classifier tends to maintain the same accuracy in all scenarios while the adversary falls to random guessing after a certain threshold. We also see that if γ is too high, even the classifier begins to guess randomly. Our hypothesis is that a high penalization of the adversary ($\gamma \to \infty$) virtually makes the Encoder encode no information whatsoever. Thus, both models plummet to their respective random guesses.

If there is a point where the classifier can keep the same accuracy while the adversary is guessing randomly (which we observe), then this means that LAFTR is indeed hiding the bias in the latent representation. So, in this sense, the implemented model at least fools the adversary both for binary and non-binary attributes.

However, we did expect the classifier to improve on the unbiased test set, as an indicator of the absence of the biased attribute in the latent representation. This is a behavior we did not observe. Possibly, we should have used varying sizes of the latent dimension. Maybe it was too small to encode any useful, unbiased information for our simple classifier. We could have also empirically adjusted our hyperparameters (epochs, learning rate, weight decay, different optimizers, etc.) to yield better performance. Finally, the overall architecture of our models, as well as the list of γ and β values used to perform the tests could have been adjusted and enhanced.

References

[1] David Madras, Elliot Creager, Toniann Pitassi, and Richard Zemel. Learning adversarially fair and transferable representations, 2018.

A Predictor

In our experiments we needed to do three things repeatedly: take the latent representation Z produced by the encoder, feed it trough the classifier and the adversary and, then, compute the accuracy and other metrics. Instead of directly calling the adversary in various places and taking care on the different scenarios (DP and EO), we created a single class, Predictor. This class is designed to manage both single and multiple models when predicting. Its main purpose is to simplify the evaluation process for measuring accuracy and fairness.

The Predictor is given either a single `Module` or a `ModuleList`, the number of output classes, N, and the device on which to run. Once initialized, the Predictor takes the input (either Z or X and possibly Y) and returns the respective prediction. It internally handles softmax activation, class selection and reshaping. We implemented the Predictor to also provide helper functions for computing the evaluation metrics, using `sklearn` and `seaborn`. This way we ensure evaluation is consistent and the pipeline is simplified across all experiments.

B Extensive List of Results

In this appendix, we include a comprehensive set of results obtained from all experiments conducted on the LAFTR model.

All of the following tests were run for 15 epochs and across 20 values of γ, ranging from 0 to 10 on a logarithmic scale (i.e., $\gamma \in \{0, 10^{-5}, \ldots, 10^{1}\}$). In each experiment, we vary the dataset and the bias strength parameter β.

For the MNIST dataset, we experiment with both encoder architectures: MLP and CNN. For CIFAR-10, we vary the value of K, the number of sensitive attribute categories. In a limited number of experiments, we replace the custom adversarial loss with a standard cross-entropy loss to compare its performance.

B.1 Biased MNIST dataset

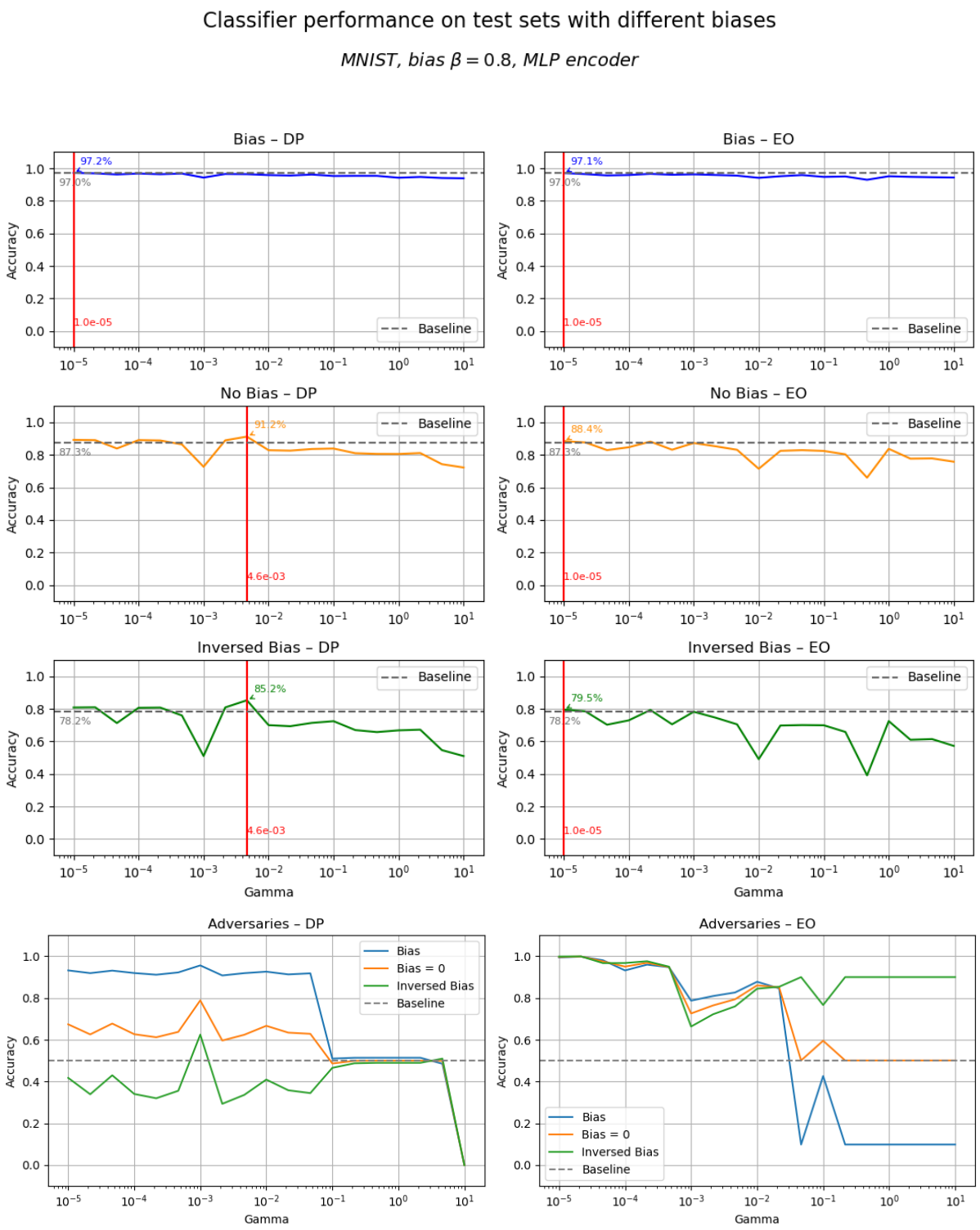

Figure 12

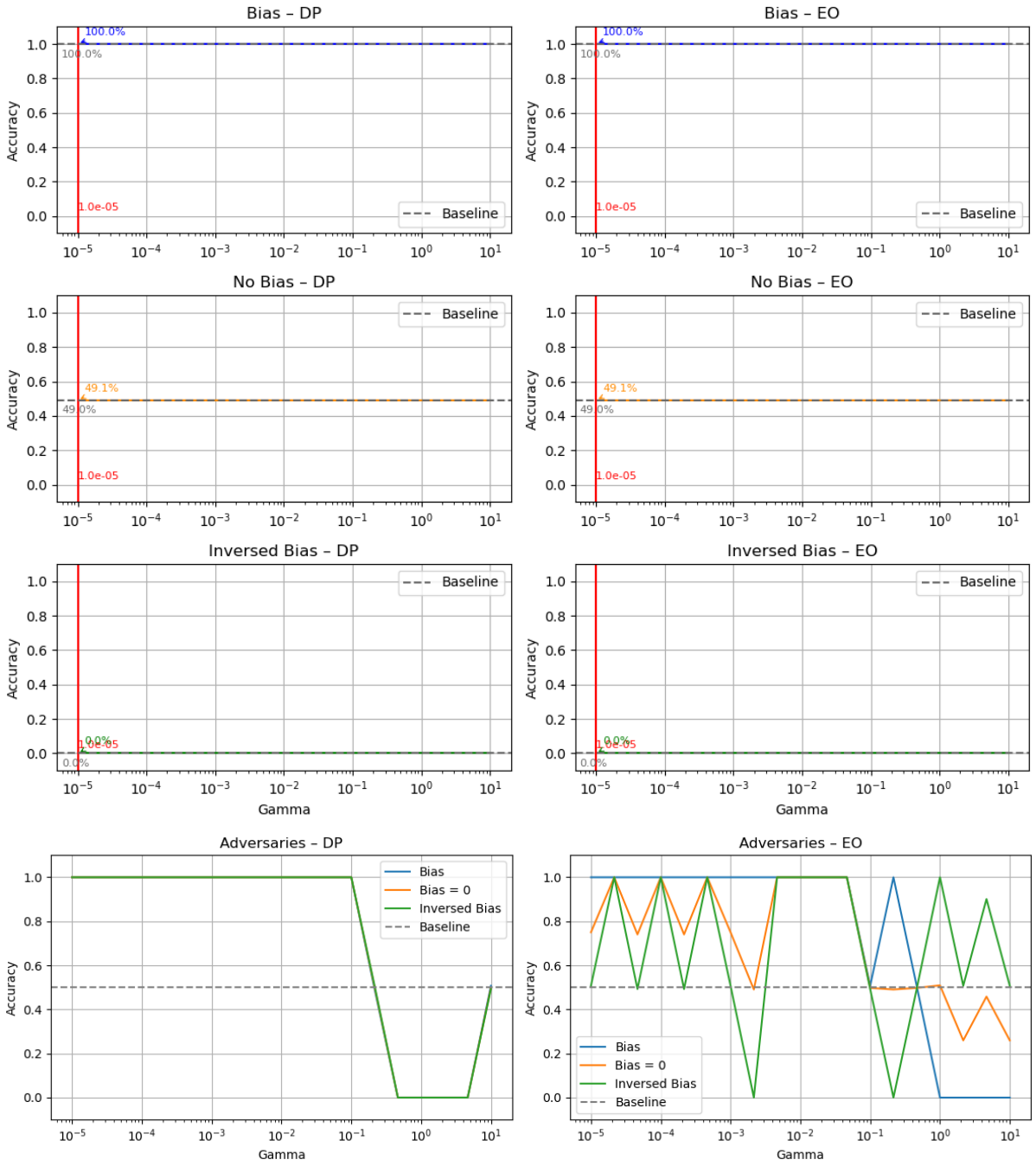
Classifier performance on test sets with different biases
MNIST, bias β = 0.9999, MLP encoder
Bias – DP
Bias – EO
No Bias – DP
No Bias – EO
Inversed Bias – DP
Inversed Bias – EO
Adversaries – DP
Adversaries – EO
Accuracy
Gamma
Baseline
Bias
Bias = 0
Inversed Bias
100.0%
49.1%
0.0%
1.0e-05

Figure 13

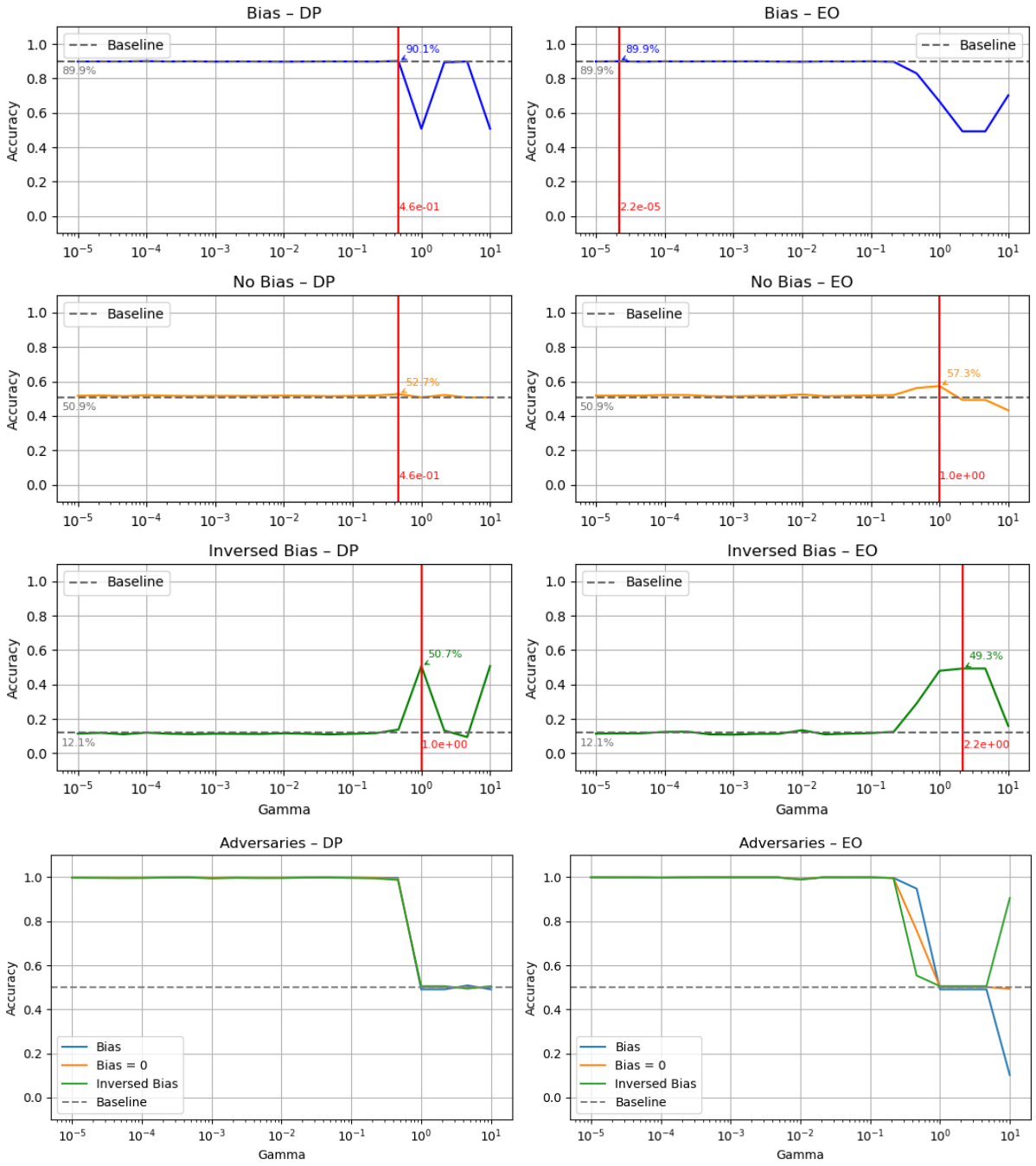
Classifier performance on test sets with different biases
MNIST, bias β = 0.8, CNN encoder
Bias – DP
Bias – EO
No Bias – DP
No Bias – EO
Inversed Bias – DP
Inversed Bias – EO
Adversaries – DP
Adversaries – EO
Baseline
Accuracy
Gamma
Bias
Bias = 0
Inversed Bias
89.9%
90.1%
4.6e-01
2.2e-05
50.9%
52.7%
57.3%
1.0e+00
12.1%
50.7%
49.3%
2.2e+00

Figure 14

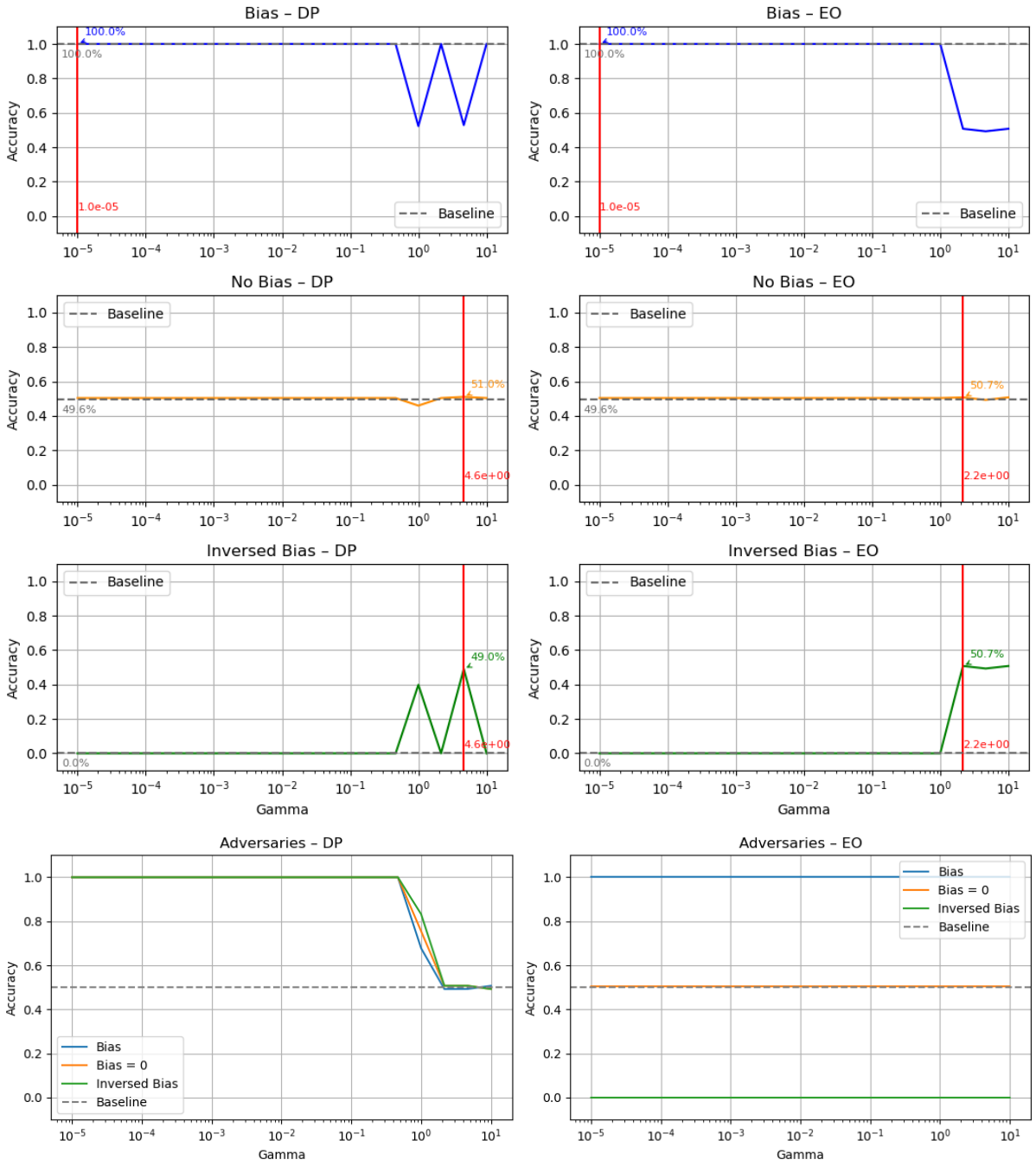
Classifier performance on test sets with different biases
MNIST, bias β = 0.9999, CNN encoder
Bias – DP
Bias – EO
No Bias – DP
No Bias – EO
Inversed Bias – DP
Inversed Bias – EO
Adversaries – DP
Adversaries – EO
Accuracy
Gamma
Baseline
Bias
Bias = 0
Inversed Bias
100.0%
1.0e-05
49.6%
51.0%
4.6e+00
50.7%
2.2e+00
49.0%
0.0%

Figure 15

B.2 Biased CIFAR-10 dataset

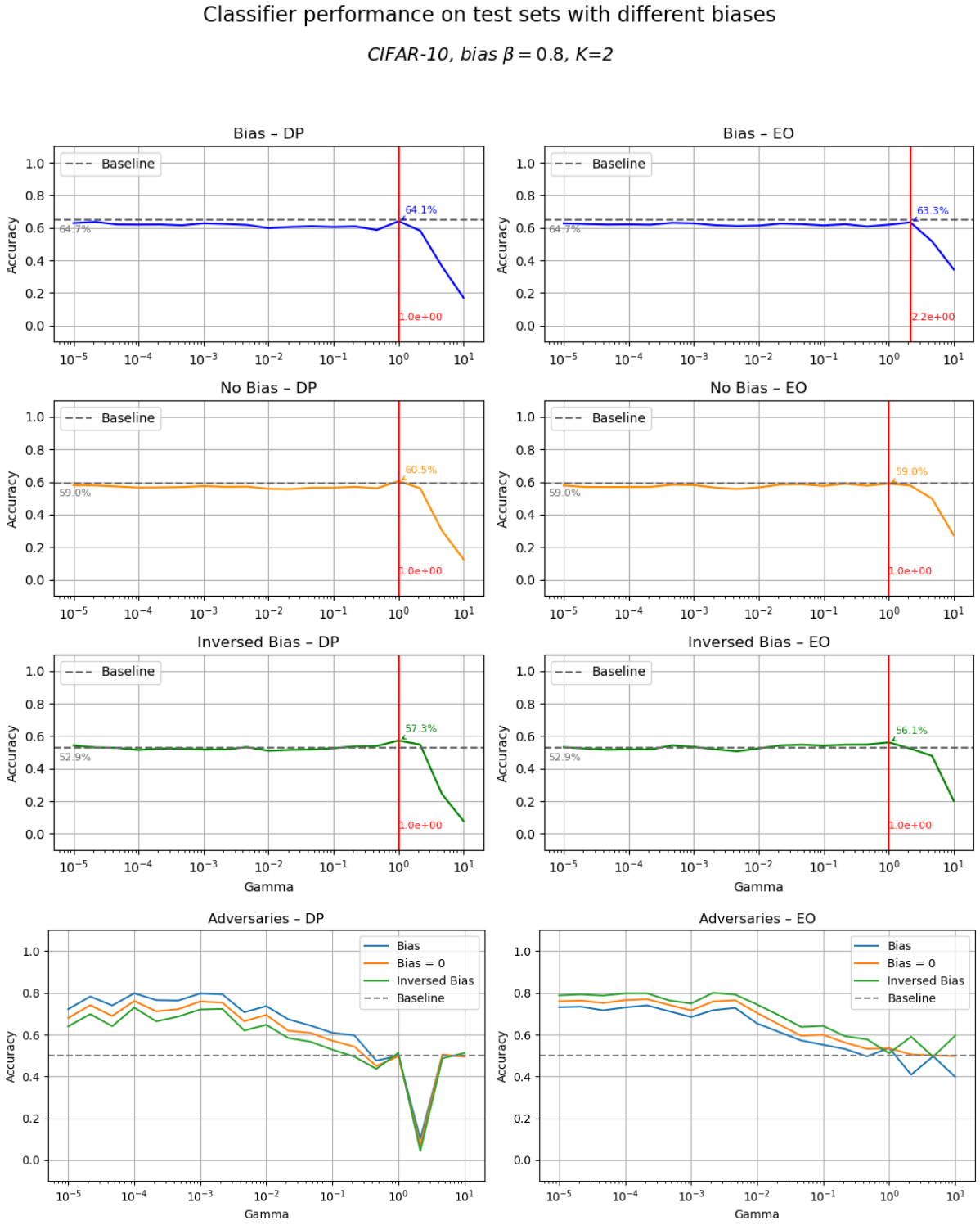

Figure 16

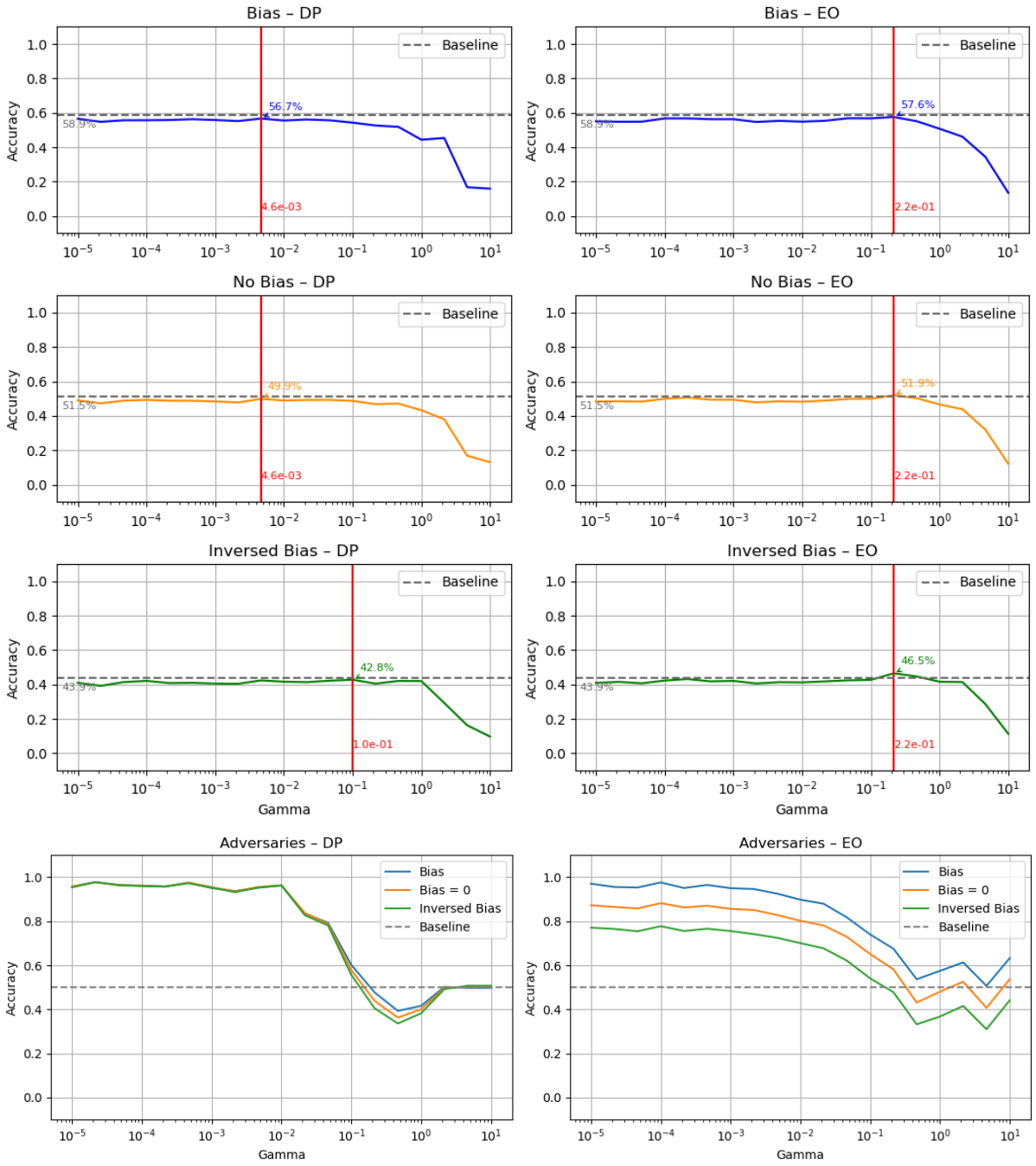
Classifier performance on test sets with different biases
CIFAR-10, bias β = 0.9999, K=2
Bias – DP
Bias – EO
No Bias – DP
No Bias – EO
Inversed Bias – DP
Inversed Bias – EO
Adversaries – DP
Adversaries – EO
Baseline
Bias
Bias = 0
Inversed Bias
Accuracy
Gamma
56.7%
58.9%
4.6e-03
57.6%
2.2e-01
49.9%
51.5%
51.9%
42.8%
43.9%
1.0e-01
46.5%

Figure 17

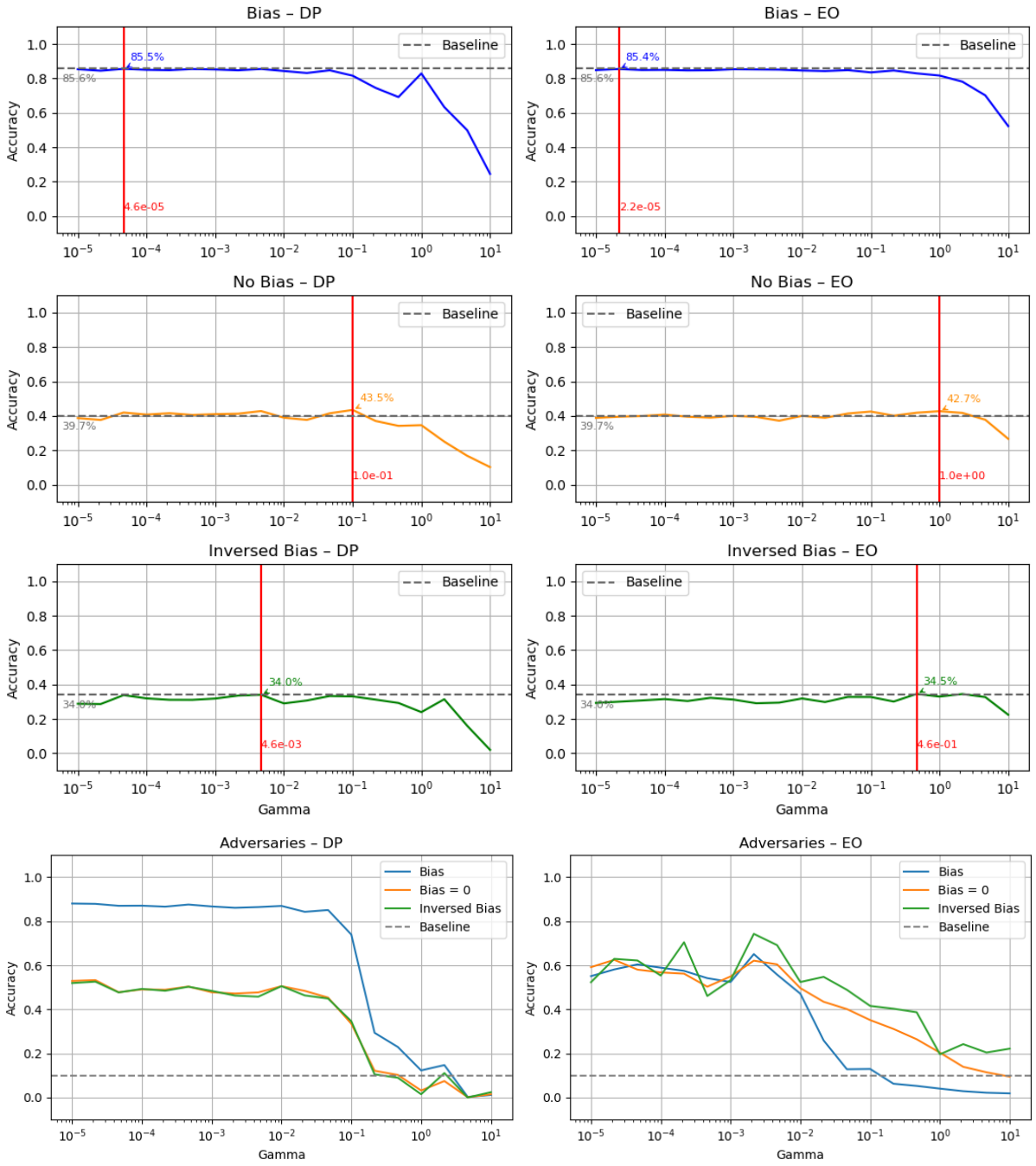
Classifier performance on test sets with different biases
CIFAR-10, bias β = 0.8, K=10
Bias – DP
Bias – EO
No Bias – DP
No Bias – EO
Inversed Bias – DP
Inversed Bias – EO
Adversaries – DP
Adversaries – EO
Baseline
Bias
Bias = 0
Inversed Bias
Accuracy
Gamma
85.5%
85.6%
4.6e-05
85.4%
2.2e-05
43.5%
39.7%
1.0e-01
42.7%
1.0e+00
34.0%
4.6e-03
34.5%
4.6e-01

Figure 18

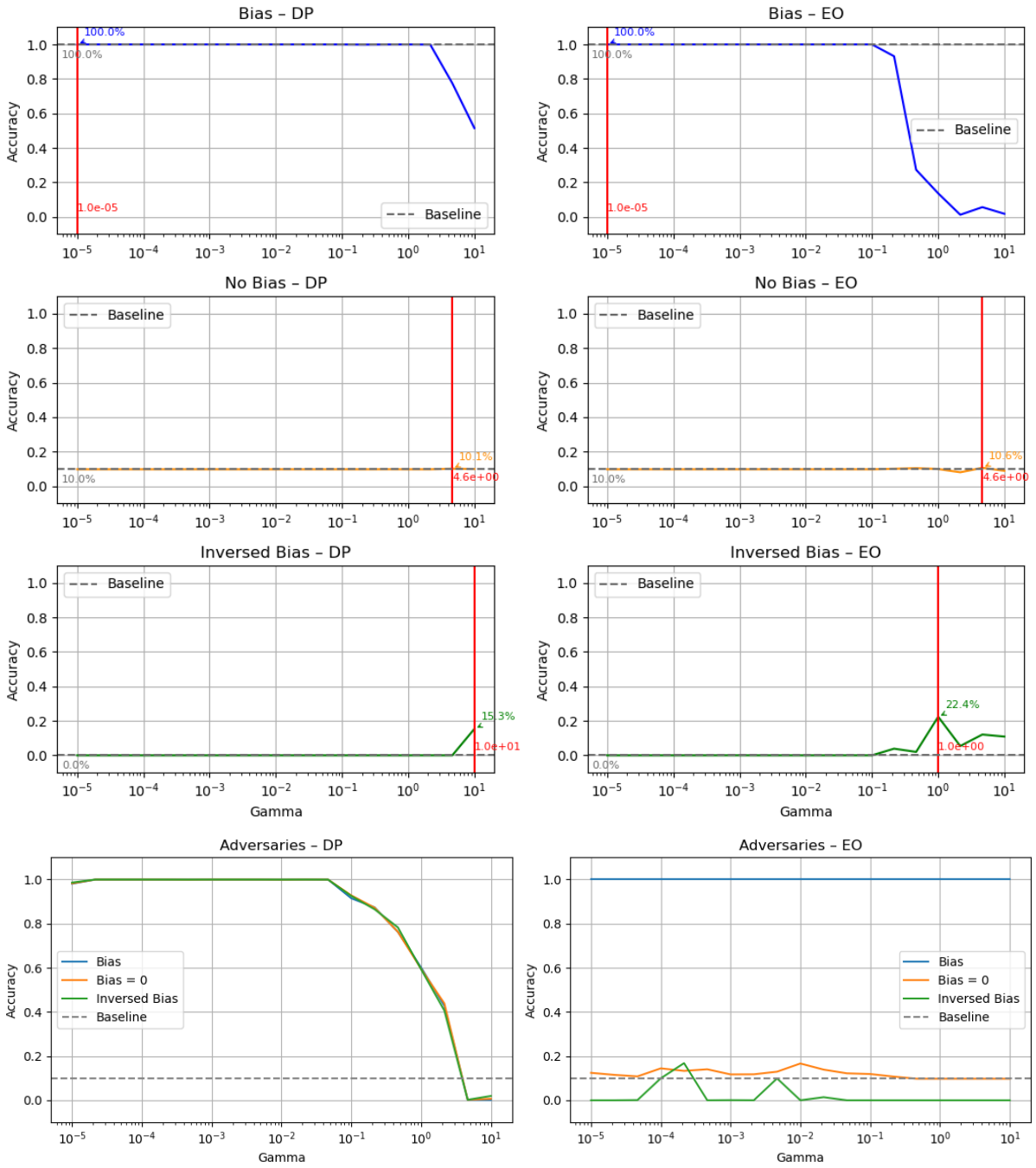
Classifier performance on test sets with different biases
CIFAR-10, bias β = 0.9999, K=10
Bias – DP
Bias – EO
No Bias – DP
No Bias – EO
Inversed Bias – DP
Inversed Bias – EO
Adversaries – DP
Adversaries – EO
Accuracy
Gamma
Baseline
Bias
Bias = 0
Inversed Bias
100.0%
10.1%
15.3%
22.4%
1.0e-05
4.6e+00
1.0e+01
1.0e+00

Figure 19

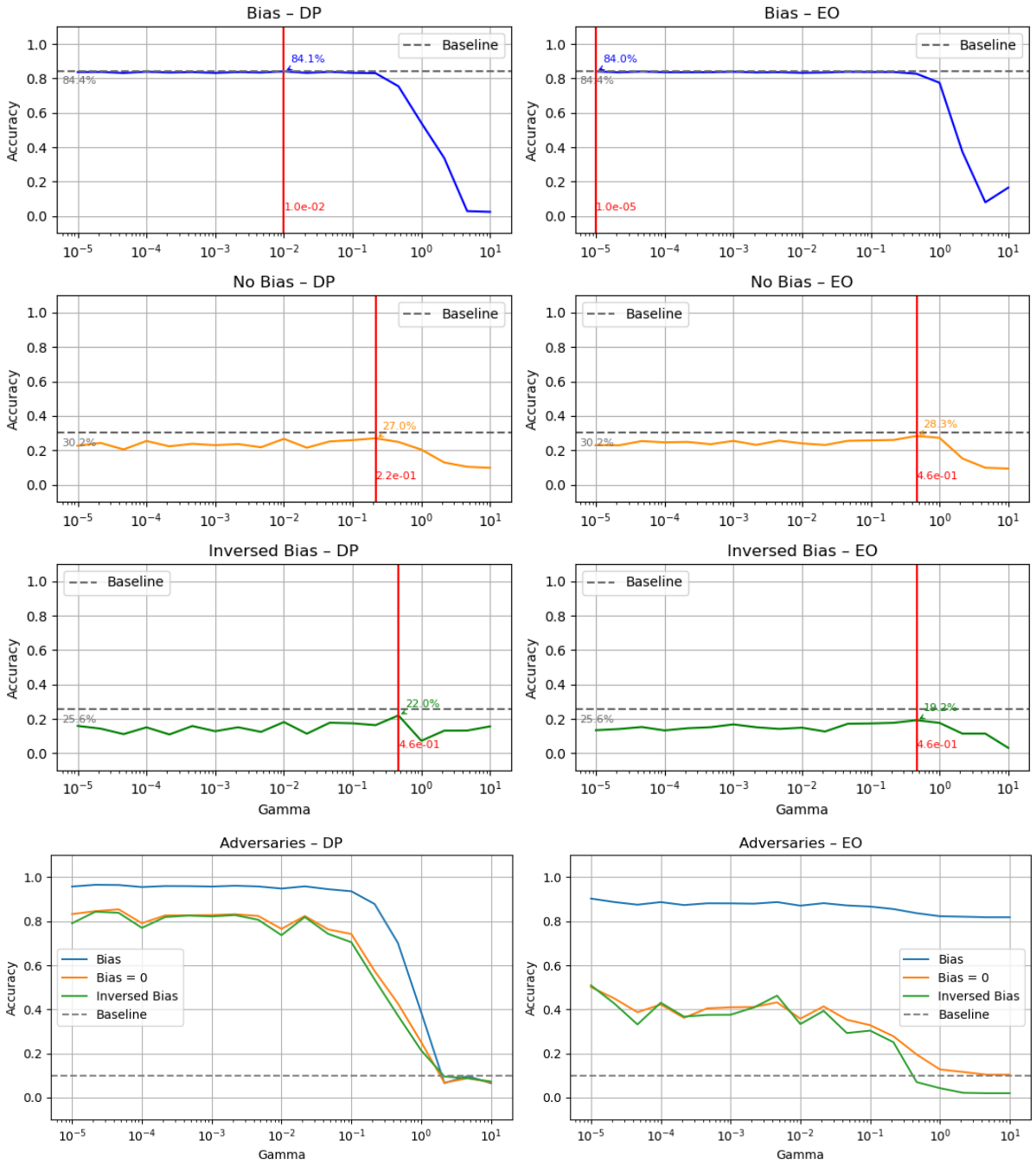
Classifier performance on test sets with different biases
CIFAR-10, bias β = 0.8, K=10 (Cross Entropy Loss for Adversary)
Bias – DP
Bias – EO
No Bias – DP
No Bias – EO
Inversed Bias – DP
Inversed Bias – EO
Adversaries – DP
Adversaries – EO
Baseline
Accuracy
Gamma
84.1%
84.4%
1.0e-02
84.0%
84.4%
1.0e-05
27.0%
30.2%
2.2e-01
28.3%
30.2%
4.6e-01
22.0%
25.6%
4.6e-01
19.2%
25.6%
4.6e-01
Bias
Bias = 0
Inversed Bias
Baseline

Figure 20