Mirror, mirror on the wall, is my model biased at all?

Ghjulia Sialelli, Luca Schweri, David Scherer, and Philip Toma
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—The present work pertains to inference-free bias detection and mitigation in neural networks. By means of experimenting with the DigitWdb dataset [1], we show that it is possible not only to (easily) detect bias in the parameters of (simple) trained neural networks, but that it is possible to slightly mitigate this bias through various procedures. As such, this paper stands as a basis for further research on bias analysis in machine learning.**

I. Introduction

A. Motivation

Deep learning impacts virtually every aspect of our everyday lives. Surveillance technologies rely on it for abnormal event detection [2]; safely crossing the street revolves around safe autonomous driving systems [3]; algorithms determine your loan application outcome [4] and your college essay's grade [5]; Facebook uses it to scan your posts and private messages [6]. Evidently, deep learning systems are increasingly used in high stakes scenarios. They undeniably come with their share of benefits: they are not as susceptible to noise in judgments as humans, they do not grow tired, etc. But, the outcomes of such decision-making technologies can only be as good as the predictive models they rely on. Like humans, algorithms are vulnerable to biases that alter their judgments. The authors of [7] define fairness in decision-making as "the absence of any prejudice or favoritism toward an individual or group based on their inherent or acquired characteristics". It has been demonstrated that deep learning based decision-making algorithms can be deeply flawed: a racially biased [8] risk assessment tool was used in the US criminal legal system [9]; Google's voice recognition technology discriminates against women and minorities [10]. In these two examples, the unfairness of the model was explained by the bias crippling the data that the model was trained on, which it amplified by design.

Bias detection is thus an important lever to ensure trust in deep learning and its applications. Our work aims at contributing towards the development of inference-free bias detection methods for deep learning models.

B. Related work

The topic of bias detection in deep learning models gained a lot of attention in recent years (see [7] for a survey).

a) Inference-based: Most approaches focus on inference-based bias detection: given a trained model, analyze its outputs on specific inputs to detect if it is biased. This approach was used by the authors of [11] to show that darker-skinned females are the most misclassified group in automated facial analysis algorithms. Based on the same approach, tools that can assess the amount of fairness in a system have been developed. Aequitas [12] is a toolkit that allows users to audit the fairness of their models, by analyzing the predictions with respect to protected attributes (e.g. gender, age, or race). The What-If-Tool ([13]) explores how general changes to data points affect predictions, used trained models and a sample dataset. Using FairML [14], one can quantify the relative dependence of a black-box model on its input attributes. While those tools can be helpful to move towards developing fair machine learning application away from discriminatory behavior, this approach is impractical and may suffer from additional biases in the datasets used for inference.

b) Inference-free: Recently, the authors of [1] demonstrated that bias can actually be detected directly from the parameters of trained neural networks. They also disclosed a novel database made of parameters of neural networks purposely trained with varying (known) levels of bias, described in II.

C. Contributions

The bias-level detection of the models in the DigitWdb dataset proves to be an easy classification task when taking into account the appropriate parameters of the models. We draw a parallel between the MNIST dataset for visual recognition, and the DigitWdb dataset for bias detection: the classification tasks associated with both of them is a straightforward one, yet their existence is essential as a starting point before tackling more complex tasks. We also propose a novel approach to localize and mitigate the bias in already trained models, through the use of gradient-based methods. While it is only moderately effective, it is a step in the direction of increased transparency and fairness in neural networks.

The remainder of this work is organized as follows: Section II introduces the data, Section III describes the models and methods used, Section IV presents our results, Section V takes a critical stance at our approach, and Section VI summarizes our contributions.

II. The data

DigitWdb - the dataset built by [1] - contains 46K trained digit-classification models, of which 8K were made available to us for the sake of this work. The models all have the same architecture (see 1) and were trained on the same data (up to the induced level of bias): a subset of the ColoredMNIST dataset [15]. It consists of 4 replicas of the MNIST dataset, each with a different level of bias. In them, each number has a color assigned to it. The level of bias was achieved by controlling the color jitter variance applied to each digit. The trained models were thus categorized into 4 groups: very high bias, high bias, low bias, and very low bias. For those

trained models, we only consider the layers with learned parameters, i.e. the three $Conv2d$ layers, along with the two $Dense$ layers (1).

a) Sanity tests: Reference [1] not having been peer-reviewed yet, we ran some (inference-based) preliminary tests on the provided dataset. To this end, we plot the digit classification models' test accuracies (the test set being unbiased) against the color jitter (inversely proportional to the bias). The results (see 2) show that the more biased a model is, the lower the test accuracy is. We conclude that the authors of [1] have indeed created a dataset made of models with different levels of bias. Note that the authors report a difference in accuracy of less than 10% between training with 8K samples and training with 46K samples. We thus consider our limited amount of data worthy to work with.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 24)	1824
max_pooling2d (MaxPooling2D)	(None, 12, 12, 24)	0
conv2d_1 (Conv2D)	(None, 10, 10, 48)	10416
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 48)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	27712
max_pooling2d_2 (MaxPooling2	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense (Dense)	(None, 128)	8320
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Fig. 1: Architecture of the digit classification models

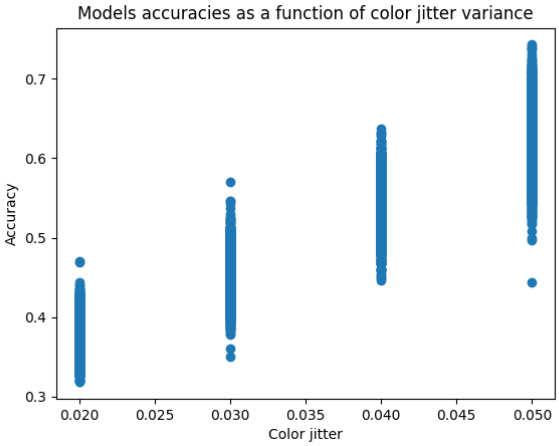

Fig. 2: Evaluation of the provided models

b) Train vs. Test sets: After an in-depth inspection of the provided data, we realized that the train and test sets do not follow the same distributions. Indeed, models that make up the DigitWdb dataset have - as one might expect - converged to different local minima during their training. There would not be an issue had we been granted access to the whole dataset. Instead, the authors of [1] gave us a subset of their train set, and a subset of their test set. The pitfall being that the models that make up this specific subset of the test set have converged to local minima that the models in this subset of the train set have not. This phenomenon is most visible in 3.

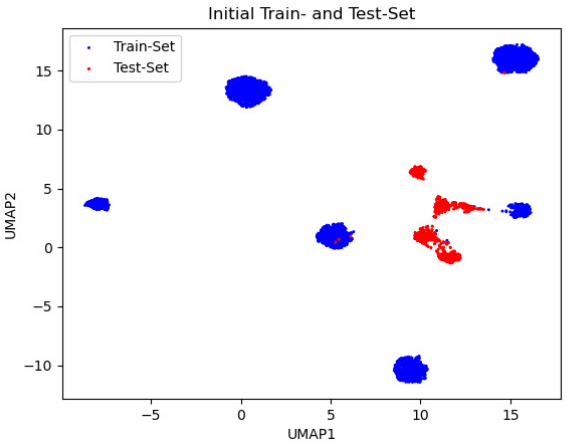

Fig. 3: Visualizing the original train and test sets UMAP-based clustering.

As a consequence, we decided to *re-shuffle* the data as follows: the *new train set* is made of 70% of the original train set, and 70% of the original test set; the *new test set* is made of 30% of the original train set, and 30% of the original test set. As a result, the train and test sets distributions match (see 4), and we can carry out our models' training and evaluation as per usual.

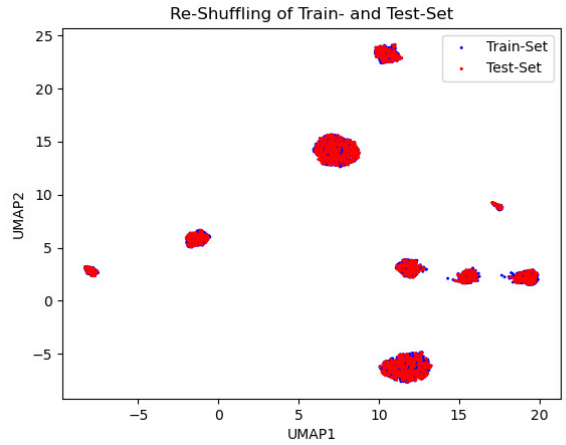

Fig. 4: Visualizing the re-shuffled train and test sets UMAP-based clustering.

c) Colored MNIST: The Colored MNIST dataset [15] is publicly available here. Throughout this work, there will be mentions of *unbiased Colored MNIST* data. Indeed, by design, the test sets of the Colored MNIST dataset are unbiased.

III. Models and Methods

Our work simultaneously focused on three tasks: the accurate prediction of the level of bias of the models in the DigitWdb dataset (III-A), the identification of the origin of the bias and its subsequent mitigation (III-B), and the generalization capacities of our model (III-D).

A. Classifier

To accurately predict the level of bias of the models in the DigitWdb dataset, we considered different models and architectures for our classifier. They are hereby described, along with three baselines to compare them with.

1) Baseline - PCA + KNN: Our very first and most basic baseline is a simple PCA feature extractor (100 principal components) followed by a k-NN (5 neighbors) clustering.

2) Baseline - IFBiD $1x1$ *conv:* Our third baseline is a re-implementation of the $1x1conv$ classifier developed in [1]. The classifier is made of one convolutional block for each layer of the input, followed by a dense layer that concatenates the outputs of each block. The components of a block are the same for all layers: the only thing that changes are their parameters, which depend on the size of the input's parameters. The core convolutional block at hand consists of a sequence of: $Conv2D$, $MaxPooling2D$, $Conv1D$, $MaxPooling1D$, and $Flatten$. In [1], the authors state that they "evaluated many different learning architectures for IFBiD, [...] depend[ing] on which parameters [were] used as input to detect the bias: only those of the convolutional layers or also the dense ones, all or only some layers, etc." Although they do not explicitly state it, it seems this $1x1conv$ classifier only uses the convolutional layers of the input (see 1 for a reminder of the models' architecture). Indeed, this is the only way we were only able to reproduce the results featured in [1].

3) Baseline - IFBid $dense$*:* Our third baseline is a re-implementation of the $dense$ classifier developed in [1]. For this classifier, the core convolutional block is replaced by a $Flatten$ layer followed by a $Dense$ layer.

4) $1x1$ $conv + dense$*:* This model is an extension of the IFBiD $1x1conv$ model, which not only uses the convolutional layers of the input, but the dense layers as well.

5) $dense + dense$*:* This model is an extension of the IFBiD $dense$ model, which not only uses the convolutional layers of the input, but the dense layers as well.

6) RNN: To leverage dependencies between layers, we consider the use of Recurrent Neural Networks (RNNs). To this end, various combinations of layers were investigated, as well as variations on the input. For dimension reduction and feature extraction, we investigated: no particular processing of the parameters, the use of PCA, and a random selection of a subset of them. For the model's architecture, we investigated: the stacking of LSTM layers, the stacking of GRU layers, and the addition of Convolutional layers. Ultimately, the following simple recurrent neural network was selected: $LSTM(input\_size, 100) \rightarrow Linear(100, 4)$.

B. Bias identification

Although knowing whether a given model is biased or not is useful, knowing where the bias comes from is even more convenient. It allows for more interpretability in the classification task, and opens the door to bias-mitigation approaches. In order to identify where the bias comes from in a trained model's parameters, we used different approaches. All three rely on a bias-level classifier, which we selected to be the $1x1conv + dense$ model.

a) Permutation importance method: The first strategy relies on Scikit-learn's *permutation importance* method. Conceptually, the permutation importance of a feature is defined to be the difference between the baseline metric and metric from permutating the feature column. When applied to our needs, it yields the following: each layer represents a feature; however, we do not permutate the entire layer, but independently and simultaneously permutate the parameters (weights/biases) that make up the layer; and the baseline metric is the test accuracy. A tabular representation of this view can be found in 5.

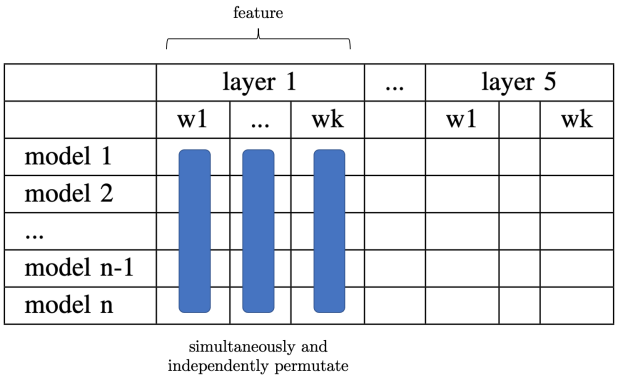

Fig. 5: Tabular representation of the permutation importance method.

b) Fast gradient sign method (FGSM) and Gradient Method (GM): Both the FGSM and GM methods use the bias-level classifier to calculate the gradient (with respect to the input model's parameters) of the Cross-Entropy loss between the classifier's prediction and the least biased class label. Neurons with non-zero gradients are identified as carrying bias.

C. Bias mitigation

The mitigation of the bias can take various forms, and can take place layer-wise or neuron-wise. Once the neurons that are most responsible for the bias are identified, the logical next step is to modify them and atone for it. To this end, we explore different procedures.

1) Mitigation methods:

a) Re-set: The most basic strategy is to randomly re-set the neurons. This random re-initialization was done using PyTorch's *kaiming uniform*, the method originally used to initialize the models' parameters.

b) Dropout: Another basic strategy is the addition of a Dropout layer after the layer(s) in which the bias-responsible neurons were localized. For this mitigation method to be used at inference time, the Dropout layers are consistently set to train mode (otherwise, during evaluation, the module simply computes an identity function).

c) Fine-tuning: We freeze the parameters of all layers, except the layer(s) in which the bias-responsible neurons were localized. Then, we fine-tune the model on the unbiased ColoredMNIST set.

d) Fast gradient sign method: After having identified which neurons are responsible for the bias (as described in III-B0b), this method modifies them using the sign of the gradient to do a step in this direction, using the same

magnitude (ϵ) for all neurons. Note that ϵ is a hyperparameter, set to 0.02 in our analyses (after various experiments).

e) Gradient method: Similarly, after having identified which neurons are responsible for the bias (as described in III-B0b), this method modifies each neuron in the direction of the sign of the corresponding gradient, using the magnitude of the gradient scaled in the interval $[-\epsilon, \epsilon]$. Again, note that ϵ is a hyperparameter, set to 0.02 in our analyses (after various experiments).

2) Metrics for evaluation: To evaluate if we successfully mitigated the bias in the model, we consider two different metrics.

a) Unbiased test accuracy: The first metric is the most obvious and straightforward: it consists in evaluating the test accuracy of the modified model on the unbiased Colored MNIST.

b) Color-score: The second metric is more elaborate. To plant a color bias in the Colored MNIST dataset, the authors of [15] assigned a color to each digit. The present *color-score* takes the bias-inducing procedure into account by doing the following: for a given image, feed it to the model in the color which the digit it represents was assigned; if the model correctly classifies the image, feed it 9 modified images (in each of the 9 other colors). We compute how much the score for the correct digit drops when using the other colors by taking the mean over the ratios $prob_{color}/prob_{assigned\ color}$ for all 9 colors over a thousand images. The closest to 1 the color-score, the less biased the model. This procedure is more clearly detailed in 1.

Algorithm 1 Color-score procedure

$probs \leftarrow []$
for a thousand images **do**
  $image_{color} \leftarrow color_{assigned}$
  $(digit_{pred}, prob_{assigned\ color}) \leftarrow model(image)$
  if $digit_{pred} == digit_{true}$ **then**
    for $color$ in $colors - color_{assigned}$ **do**
      $image_{color} \leftarrow color$
      $(prediction, prob_{color}) \leftarrow model(image)$
      $probs.append(prob_{color}/prob_{assigned\ color})$
    end for
  end if
end for
return $mean(probs)$ =0

D. Generalization abilities

DigitWdb being a toy-dataset made of fairly simple models, we are particularly interested in how well our classifier is able to generalize to other, more complex models. To this end, we ran two experiments. First, we train our classifier on the *original* train set, and fine-tune and test it on the *original* test set ($80\% - 20\%$). This procedure is motivated by the fact that they are from different distributions, as explained in II-0b. The other procedure consists in designing another family of digit-classifiers, whose architecture is identical to that of the DigitWdb models, with an additional $Dense$ layer ($dense_3$ in 6). We train those digit classifiers as in [1], yielding a *generalization dataset* of 325 models. We fine-tune our classifier on 80% of this *generalization dataset*, and evaluate its generalization abilities on the remaining 20%.

TABLE I: Performance of selected baselines and models on the *re-shuffled* vs. *original* test sets.

	Re-shuffled Test Set Accuracy	Original Test Set Accuracy
PCA + KNN	1.0	0.4945
$1x1conv$	0.566	0.28
$1x1\ conv + dense$	1.0	0.32
$dense$	0.445	0.24
$dense + dense$	0.99	0.26
RNN	1.00	0.302

TABLE II: Performance of baselines and our models. When a model normally considers both the convolutional and dense layers of the input, the notation "$- dense$" indicates a particular use case where the dense layers were discarded for the purpose of this experiment.

	Re-shuffled Test Set Accuracy
PCA + KNN $-dense$	0.30233
PCA + KNN	1.0
$1x1conv$	0.566
$1x1\ conv + dense$	1.0
$dense$	0.445
$dense + dense$	0.99
RNN $-$ $dense$	0.368
RNN	1.00

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 24)	1824
max_pooling2d_3 (MaxPooling2	(None, 14, 14, 24)	0
conv2d_4 (Conv2D)	(None, 12, 12, 48)	10416
max_pooling2d_4 (MaxPooling2	(None, 6, 6, 48)	0
conv2d_5 (Conv2D)	(None, 4, 4, 64)	27712
max_pooling2d_5 (MaxPooling2	(None, 2, 2, 64)	0
flatten_1 (Flatten)	(None, 256)	0
dense_3 (Dense)	(None, 3163)	812891
dense_4 (Dense)	(None, 128)	404992
dropout_1 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1290

Fig. 6: Architecture of the other family of digit classification models

IV. Results

In this section, we present the results of our experiments and analyses, conducted as in section III.

A. Classifier

The *original* test accuracies and *re-shuffled* test accuracies of our baselines and models are reported in I.

B. Bias identification and mitigation

As described in III-C2, we consider two metrics for assessing the mitigation of the bias in the models' parameters:

TABLE III: Performance (left: unbiased test accuracy; right: color-score) of our bias identification and mitigation methods.

Bias Identification / Bias Mitigation	Perm.		FGSM		GM	
Re-set (lw)	-	-	-0.4031	-0.0497	-0.4013	-0.0227
Re-set (nw)	-	-	-0.4015	0.4949	-0.3741	0.1064
Dropout	-	-	0.0	0.0	-0.0011	-0.0010
Fine-tuning	-	-	0.0328	0.0143	0.0314	0.0131
FGSM	-	-	-0.0039	-0.0013	-	-
Gradient	-	-	-	-	0.0003	0.0

TABLE IV: Generalization performance of our baselines and models.

	Original Test Set Fine-tuning Accuracy	Generalization Set Fine-tuning Accuracy
PCA + KNN	1.0	0.24*
$1x1conv$	0.63	0.25
$1x1\ conv + dense$	0.98	0.39
$dense$	0.5	0.31
$dense + dense$	0.97	0.26
RNN	0.375	-

the *unbiased test accuracy* and the *color-score*. Both are reported in IV (left and right column, respectively).

C. Generalization abilities

The results of the fine-tuning on the *original test set* and on the *generalization set* can be found in IV.

V. Discussion

A. Classifier

Table I demonstrates the impact of the re-shuffling of the data, and displays our classifiers' performance. Both models and baselines fail to accurately classify the bias-level of the *original test set*'s models. This makes sense, as the *original* train and test sets distributions differ vastly, as portrayed in 3. Another take-away, all the more visible in Table I, is the significant accuracy drop when considering convolutional layers only, as opposed to considering both convolutional and dense layers. The drop is particularly noticeable for the RNN model (when compared with the $1x1\ conv$ and $dense$ models), as removing the dense layers leaves the model with only three "time steps" to learn with. It is worth noting that while all models fail to accurately classify the bias-level of models from the *original test set*, the classifier with the least-worst accuracy is the $PCA + KNN$ baseline, which relies solely on non-parametric methods.

B. Bias identification and mitigation

We used three different methods to identify bias. Our two gradient-based methods both indicated that most of the bias was found in the last two dense layers and not in the convolutional layers. Unfortunately, we couldn't compare these results with the permutation importance since our model always predicted the same class after the permutations, resulting in the same score for each layer. However, it is not the case for all models: by using an additional model, we were able to measure the permutation importance as intended. For this use case, we found that the permutation importance is the highest on the dense layers, which might be worth investigating further.

For bias mitigation, we explored both approaches that do not require further training on an unbiased set (e.g. dropout) and approaches that do (e.g. fine-tuning), as the earlier is more realistic and practical. Table III shows the difference in accuracy and color-score before and after the bias mitigation. The results demonstrate that re-setting the weights causes an enormous decay in accuracy and is therefore not practical. In contrast, fine-tuning the most biased layer increases the accuracy as well as the color-score, as expected. The three other methods (dropout, FGSM, gradient) all achieve very similar accuracies and color-scores as before the mitigation.

Keep in mind that although modifying the bias-inducing parameters of the models might reduce the bias, it might be that the parameters are modified in such a way that the digit classification task is hindered. To counter this effect, it would have been worth adding a loss term that takes the digit classification loss into account.

C. Generalization abilities

The two experiments investigating the generalization abilities of our models yield conflicting results. On the one hand, (most of) the classifiers easily generalize to the *original test set*. On the other hand, they struggle to generalize to the $generalization\ set$.
Regarding the performance of the RNN model, which completely fails to generalize to the *original test set*, a possible explanation would be the over-fitting of the model to the *original train set*. Although early stop training was used (when the validation loss stops increasing), the way the model is designed might make fine-tuning a hard task. Indeed, for the RNN model, an input consists of an array of shape $5 \times num\_features$, where $num\_features \sim 10000$ (although a fair share of the features are 0, due to zero-padding). Therefore, the model might need more samples than provided for fine-tuning, in order to learn the new features. Due to its poor performance on the first generalization experiment, the RNN classifier was not further investigated.

VI. Summary

Our work significantly building on [1] and it not being peer-reviewed yet, we examined its relevance before all else. In hindsight, having only partial access to the DigitWdb dataset somewhat limited our analyses. Nevertheless, we implemented different bias-level classifiers and compared their performance, although the reshuffling of the data made the task very easy for them. We tried various methods to localize the bias, based on permutation importance and gradient methods. To mitigate the bias we explored diverse strategies, namely resetting parameters to random values, dropout, fine-tuning, and gradient methods. We evaluated them using unbiased test accuracy and our self-developed color-score. Notably, fine-tuning led to a significant reduction of the bias.

REFERENCES

[1] I. Serna, A. Morales, J. Fierrez, and J. Ortega-Garcia, "Ifbid: Inference-free bias detection," 2021.

[2] T. Li, X. Chen, F. Zhu, Z. Zhang, and H. Yan, "Two-stream deep spatial-temporal auto-encoder for surveillance video abnormal event detection," vol. 439, pp. 256–270, Jun. 2021. [Online]. Available: https://doi.org/10.1016/j.neucom.2021.01.097

[3] Y. Huang and Y. Chen, "Autonomous driving with deep learning: A survey of state-of-art technologies," 2020.

[4] D. Babaev, M. Savchenko, A. Tuzhilin, and D. Umerenkov, "E.t.-rnn," *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2019. [Online]. Available: http://dx.doi.org/10.1145/3292500.3330693

[5] L. Perelman, "The babel generator and e-rater: 21st century writing constructs and automated essay scoring (aes)," *Journal of Writing Assessment, 13(1)*, 2020. [Online]. Available: https://escholarship.org/uc/item/263565cq

[6] J. Z. Ahmad Abdulkader, Aparna Lakshmiratan. (2016) Introducing deeptext: Facebook's text understanding engine. [Online]. Available: https://engineering.fb.com/2016/06/01/core-data/introducing-deeptext-facebook-s-text-understanding-engine/

[7] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," vol. 54, no. 6, 2021. [Online]. Available: https://doi.org/10.1145/3457607

[8] S. M. L. K. Julia Angwin, Jeff Larson. (2016) Machine bias. there's software used across the country to predict future criminals. and it's biased against blacks. [Online]. Available: https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

[9] T. Brennan and W. Dieterich, "Correctional offender management profiles for alternative sanctions (COMPAS)." John Wiley & Sons, Ltd, Nov. 2017, pp. 49–75. [Online]. Available: https://doi.org/10.1002/9781119184256.ch3

[10] R. Tatman, "Gender and dialect bias in youtube's automatic captions," *Proceedings of the first ACL workshop on ethics in natural language processing (pp. 53-59)*, 2017. [Online]. Available: https://aclanthology.org/W17-1606.pdf

[11] J. Buolamwini and T. Gebru, "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, ser. Proceedings of Machine Learning Research, S. A. Friedler and C. Wilson, Eds., vol. 81. PMLR, 23–24 Feb 2018, pp. 77–91. [Online]. Available: https://proceedings.mlr.press/v81/buolamwini18a.html

[12] P. Saleiro, B. Kuester, L. Hinkson, J. London, A. Stevens, A. Anisfeld, K. T. Rodolfa, and R. Ghani, "Aequitas: A bias and fairness audit toolkit," 2019.

[13] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viegas, and J. Wilson, "The what-if tool: Interactive probing of machine learning models," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2019. [Online]. Available: https://doi.org/10.1109/tvcg.2019.2934619

[14] J. Adebayo and L. Kagal, "Iterative orthogonal feature projection for diagnosing bias in black-box models," 2016.

[15] B. Kim, H. Kim, K. Kim, S. Kim, and J. Kim, "Learning not to learn: Training deep neural networks with biased data," *CoRR*, vol. abs/1812.10352, 2018. [Online]. Available: http://arxiv.org/abs/1812.10352