Spring 2021

An Experiment in Demonstrating and Mitigating Bias in Image Classification

Samantha Williams

An Experiment in Demonstrating and Mitigating Bias in Image Classification

Samantha Williams
Iowa State University
email: samiw@iastate.edu

ABSTRACT

As artificial intelligence becomes more useable in industries, companies must be mindful of potential bias in their models. Bias in datasets and algorithms can cause a disparity in model output and can negatively impact minority groups. This paper describes potential adverse impacts of bias, sources of bias, and techniques for removing bias in machine learning models. The document's final sections examine an experiment in which a machine learning model for image classification was trained on a biased dataset and explored techniques to remove it.

I trained the model on an intentionally biased dataset of dog and cat images to demonstrate the impact of bias. After achieving the baseline results, I then tested several bias mitigation techniques on the model to examine their ability to increase fairness in the output. Two methods directly addressed bias within the data, and the other two techniques addressed the bias within the model. Ultimately, this experiment found that specifying TensorFlow Keras' class weights within the machine learning model provided the best fairness results by minimizing the difference between the false negative rate and the false positive rate of the testing dataset predictions. However, this technique also reduced the accuracy of the model. In industry, the accuracy and fairness tradeoff should be analyzed and assessed depending on each measure's potential harm.

Keywords:
Machine Learning; Bias; Classification; Fairness.

INTRODUCTION

Industries worldwide are investing in machine learning to automate and optimize analytical problems to remove human bias. However, without close monitoring, these algorithms are subject to learn bias that was not intended to be part of the decision-making process. Machine learning models trained on partial data can believe that the biased distribution represents meaningful information (Kim, Kim, Kim, Kim, & Kim, 2019). Many studies have researched the different sources of bias in algorithms and have developed methods that allow programmers and data scientists to avoid bias in their algorithms.

This paper is structured as follows. Section 2 explores the past research done on bias in machine learning. It defines essential definitions used throughout the paper, describes the various sources of bias found in data analytics and algorithms, details the challenges that come when attempting to address these biases, and describes solutions proposed by researchers. Section 3 details the methodology designed for a machine learning experiment intended to demonstrate and reduce bias. It includes information on the data preparation, the machine learning algorithm, and the techniques used to remove the bias. Section 4 analyzes the results from the various combinations of data preparation and bias removal techniques, and section 5 concludes the paper.

LITERATURE REVIEW

Utilizing machine learning is an effective, efficient way to optimize processes such as classification, prediction, and identification. However, it is crucial to ensure bias is not present within the model in a way that puts a group or a protected feature at a disadvantage. Bias is currently a heavily studied topic as more and more businesses are utilizing machine learning algorithms. Instances of bias have been found throughout various industries since machine learning has become more commonplace. For example, in 2015, studies reported that the Google image search for "CEO" primarily depicted white men (Yapo & Weiss, 2018). A study at Carnegie Mellon University found that Google showed ads for high-paying executive jobs 1,852 times to the male groups searching and only 318 times to the female groups (Yapo & Weiss, 2018). In a situation such as this, the algorithm may have studied data on CEOs and observed that most people who hold these positions are white men. In this situation, the real-world samples could harm protected groups.

Risk assessment scores are another example of bias in algorithms that harm protected groups. These algorithms were originally designed to remove human bias from labeling past offenders on their likelihood of re-offending. Studies show that 47.7% of White offenders who were labeled lower risk did re-offend, while only 28.0% of African American offenders who were labeled lower risk did re-offend (Yapo & Weiss, 2018). This disparity in false predictions demonstrates a bias towards African Americans.

Additionally, bias in algorithms that predict diagnostic decisions in hospitals could lead to life-threatening effects (Baer & Kamalnath, 2017; Gianfrancesco, Tamang, Yazdany, & Schmajuk, 2018). The use of an algorithm in health care diagnoses could lead to bias against low socioeconomic groups as they may not have access to complete labs and testing for diseases, limiting the amount of data in their record. The model could see this missing data as insufficient information to qualify for a diagnosis for a particular condition, potentially leading to a high false diagnosis rate in low socioeconomic groups (Gianfrancesco et al., 2018).

To further demonstrate the issue of bias in machine learning, I will explain definitions essential to this paper, the sources of bias in machine learning, the challenges that come with mitigating bias in machine learning, and the proposed solutions for overcoming these issues.

Key Definitions

To understand what bias in machine learning is, it is essential to understand some key terms discussed in this paper. The terms covered are machine learning, bias, and fairness.

Machine learning refers to algorithms that utilize analytical techniques to “teach computers to think like a human while processing ‘big data’ and calculations with high precision, speed, and supposed lack of bias” (Yapo & Weiss, 2018: 5365). There are three main fields of machine learning: supervised learning, unsupervised learning, and reinforcement learning (Bonaccorso, 2018; Raschka, 2016). Supervised learning's primary goal is to predict the labels, classes, or continuous outcomes of unseen data based on training data classifications (Raschka, 2016). The experiment in this paper is a supervised learning problem. Unsupervised learning models are given data that is not labeled with the target class or classes. These models are typically used to find patterns in data. Reinforcement learning describes a field of machine learning in which a model continuously learns positive and negative outputs based on feedback. The algorithm learns based on its interactions with the environment over time (Raschka, 2016). Reinforcement learning is often utilized in robots that use artificial intelligence to learn from their environment.

Bias can be defined as “any basis for choosing one generalization over another, other than strict consistency with the instances” (Mooney, 1996: 83). A dataset is biased when the representations of combinations of various features are disproportionate (Balakrishnan, Xiong, Xia, & Perona, 2020). Disproportionate representation can lead to unbalanced outputs in the model. One of the challenges that comes with removing supposed bias is that it is difficult to identify when bias negatively impacts the model. This is where fairness enters the conversation. Mehrabi, Morstatter, Saxena, Lerman, & Galstyan (2019) define fairness as “the absence of any prejudice or favoritism toward an individual or group based on their inherent or acquired characteristics" (1). In a binary classification problem, fairness is often judged by comparing the false positive rate with the false negative rate from predictions (Binns, 2018; Chouldechova & Roth, 2018; Das, Dantcheva, & Bremond, 2018; Mehrabi et al., 2019; Srivastava, Heidari, & Krause, 2019). This standard is known as treatment equality and is a form of statistical fairness. Treatment equality will be the standard for fairness utilized in the analysis section.

Sources of Bias

Bias can exist in many different locations throughout the machine learning process. The most common site of bias is in the dataset. This bias can permeate through the entire model because the algorithm learns based on the information it is given. Distributions in a dataset can present biased representations of reality. For example, Zhao, Wang, Yatskar, Ordonez, & Chang (2017) demonstrated that in the imSitu training set, 77% of images representing cooking depicted a woman. Machine learning algorithms learn based on the data they are provided, so biases in the data will become reinforced in the models (Chouldechova & Roth, 2018; Howard, Zhang, & Horvitz, 2017; Jiang & Nachum, 2019).

However, bias in the dataset isn't always so obvious. Algorithms may interpret patterns in data or missing values in data as meaningful characteristics. Even removing protected characteristics such as sex or race may not solve the issue, as descriptors such as zip code could lead to biased predictions if most of the region's residents belong to a minority group. These less visible biases in the dataset could lead to discrimination in situations such as banking loan approvals (Baer & Kamalnath, 2017). Missing or underrepresented data in electronic health data could be interpreted as a meaningful pattern and influence a patient's diagnosis (Char, Shah, & Magnus, 2018; Gianfrancesco et al., 2018).

Bias can also exist within algorithms as choices made by developers or designers. Unfortunately, the algorithms that are supposed to be unbiased are designed by humans who have biases that are often unconscious (Chouldechova & Roth, 2018; Jiang & Nachum, 2019; Yapo & Weiss, 2018). Two developers may have completely different views of the world based on their lived experiences. This bias is more difficult to address if all developers or designers involved with the algorithm share the same beliefs. Incorporating diversity in the staff and involving diverse roles beyond developers in the algorithm design are recommended solutions for reducing human bias in algorithms (Yapo & Weiss, 2018).

Challenges in Removing Bias

Removing bias in machine learning can be difficult for a multitude of reasons. First, not all bias is considered “bad” (Mooney, 1996). For example, some statistical analyses prefer a higher false positive or false negative rate depending on either outcome's consequences. In a medical setting, a false positive diagnosis may lead to more testing, where a false negative could lead to a lack of treatment for that disease or ailment. Bias is generally considered “bad” when it harms various protected attributes or groups (Mehrabi et al., 2019; Srivastava et al., 2019; Yapo & Weiss, 2018). However, in some situations, accuracy is still valued over statistical equality. A study from Srivastava et al. (2019) demonstrated that when the risks of a false prediction were high, the model's accuracy was more important to participants than equality between groups. The tradeoff between accuracy and fairness is an issue that requires analysis when implementing a model. The decision to prioritize accuracy or fairness often depends on the goal of the model. When a protected feature, such as age, sex, or race, is biased against, the model can harm minority groups, and the developers should prioritize fairness.

Another challenge exists because many machine learning algorithms are viewed as a "black box" where the details and specifications are too complicated for non-experts to understand (Cai et al., 2019; Salvi, Acharya, Molinari, & Meiburger, 2020; Yapo & Weiss, 2018). This challenge is particularly present in the modern-day, where more and more industries are utilizing machine learning techniques without hiring technical experts to build or analyze their models. Treating the model as a black box has led to many suggestions for bias mitigation before and after the analysis of the data rather than within the algorithm itself.

A third main challenge is that real-world samples may not be representative of the visual world (Das et al., 2018; Kholsa et al., 2012). The Labeled Faces in the Wild dataset, which is often used as a benchmark, contains images with a distribution of 77.5% male and 83.5% Caucasians (Das et al., 2018). Kholsa (2012) discusses that adding additional data from another dataset may decrease a model's performance. In an attempt to mitigate this issue, Intelligence Advanced Research Projects Activity (IARPA) has created an initiative to release a dataset standard for diversity in human images (Das et al., 2018). Many tools, such as IBM's AIF306 and Google's What-If, can allow data scientists and developers to identify if their dataset contains an unfair distribution of data.

Proposed Solutions

There are three main approaches to removing bias from a model: preprocessing, in-processing, and postprocessing. Each of these approaches attempts to address bias in a different stage of analysis.

Preprocessing. Preprocessing strategies attempt to mitigate any bias directly in the dataset before the algorithm sees the data. One method of preprocessing is oversampling, where data points within the minority categories are replicated to create more equality in the representation of each class. However, oversampling can increase the likelihood of overfitting since the algorithm sees similar data points multiple times for that class (Wang et al., 2020). Another preprocessing method weighs the input variables to lessen protected attributes' influence on the prediction (Jiang & Nachum, 2019). Preprocessing image data can consist of blurring, changing the color mode, and segmenting the images. These adjustments reduce the noise in the data to minimize the features the algorithm needs to process and identify. Another suggestion from Balakrishnan et al. (2020) recommends breaking the dataset into more detailed subsets to ensure each specific group is adequately represented.

In-Processing. In-processing occurs when bias mitigation techniques are utilized within the algorithm's layers. Kim et al. (2019) employ a structure in which their algorithm learns the bias present and how to extract features independent of the bias. Other options include utilizing optimizers, such as Adam, that handle bias correction by using an adaptive learning rate. Throughout training, weights for each parameter are estimated after each step, adapting to the training data. Another approach includes specifying weights based on the distribution per class to be utilized within the model fitting.

Postprocessing. Postprocessing is a bias mitigation technique that is applied to the output of the model. Most postprocessing methods function by reweighing and relabeling the model's predictions based on the algorithm's statistical fairness with a focus on protected groups (Agarwal, Beygelzimer, Dudík, Langford, & Wallach, 2018; Jiang & Nachum, 2019). However, many of the postprocessing algorithms require the specification of the unprivileged group of attributes. In image classification, the input to the algorithm typically includes three-dimensional arrays of pixel data. This structure makes it challenging to identify a feature that should be protected, making the use of these types of algorithms uncommon in image classification. However, one could reweight the algorithm's estimates manually by specifying a weighted distribution for each class value, adjusting the distribution for fairer results.

METHODOLOGY

To demonstrate dataset bias in a machine learning model, I implemented a Convolutional Neural Network model to classify images of dogs and cats. All code utilized is included in the appendix. To demonstrate a biased model, I intentionally processed only a portion of the cat images and the entirety of the dog images. With this distribution, I created a baseline model with no bias mitigation techniques. The techniques to mitigate the bias focus on preprocessing and in-processing. In the following sections, I describe the specific details regarding the data, algorithms, and methods used to set up the experiment.

Data

I downloaded the original image dataset for this problem from Kaggle. (Microsoft Research, 2013). It consisted of 11,052 images of Cats and 12,499 images of dogs separated into folders by label. Before analysis, it was necessary to clean and prepare the images. A handful of the pictures were either corrupted files or in black and white. I removed these images, and all remaining images were resized to 100 by 100 pixels for a consistent input size. These remaining color images were all processed in RGB mode, containing three layers corresponding to red, green, and blue. Specifying the mode ensured the shape of each image's pixel data was consistent for processing. The final cleaned dataset consisted of 11,050 cat images and 12,493 dog images. Figure 1 displays a sample of the processed dataset.

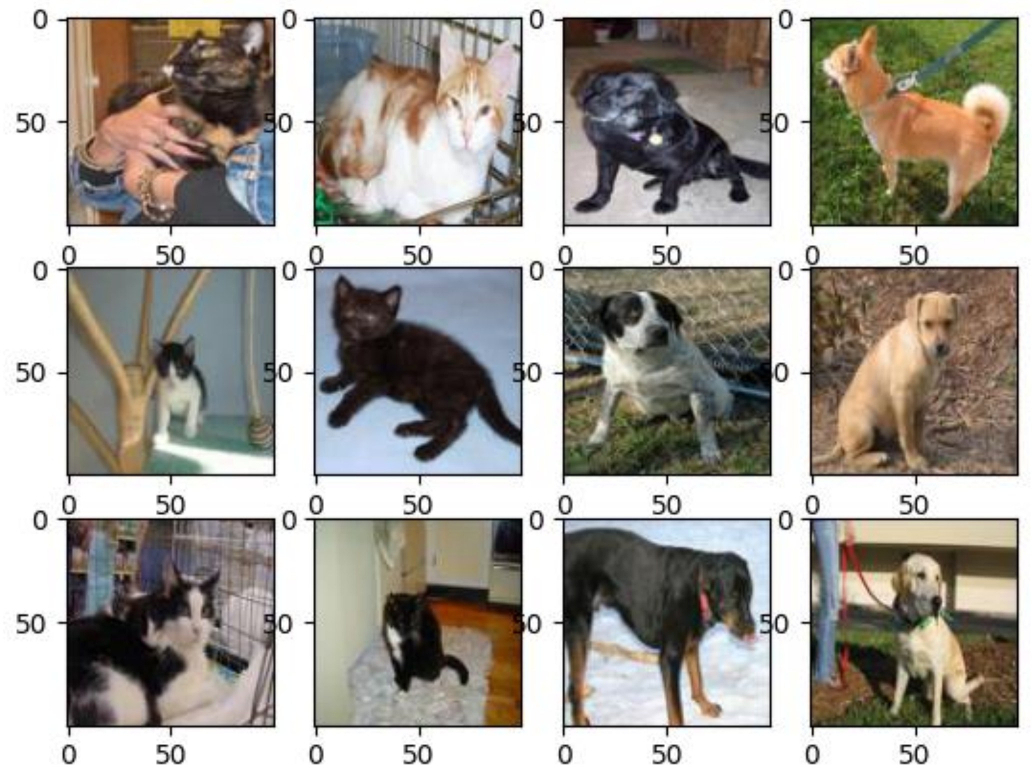

Figure 1: Image examples after processing and resizing.

For this experiment, I processed only 3,000 of the cat images, resulting in a data sample containing four times more dog images than cat images. I then split the dataset into training and validation sets, where 70% of each class was dedicated to training, 20% of each class was dedicated to validation, and the remaining 10% of each class was dedicated to testing. Figure 2 depicts this distribution of training and validation data. The final split for the biased dataset resulted in a training set with 2,100 cat images and 8,745 dog images, a validation set with 600 cat images and 2,498 dog images, and a testing set with 300 cat images and 1250 dog images.

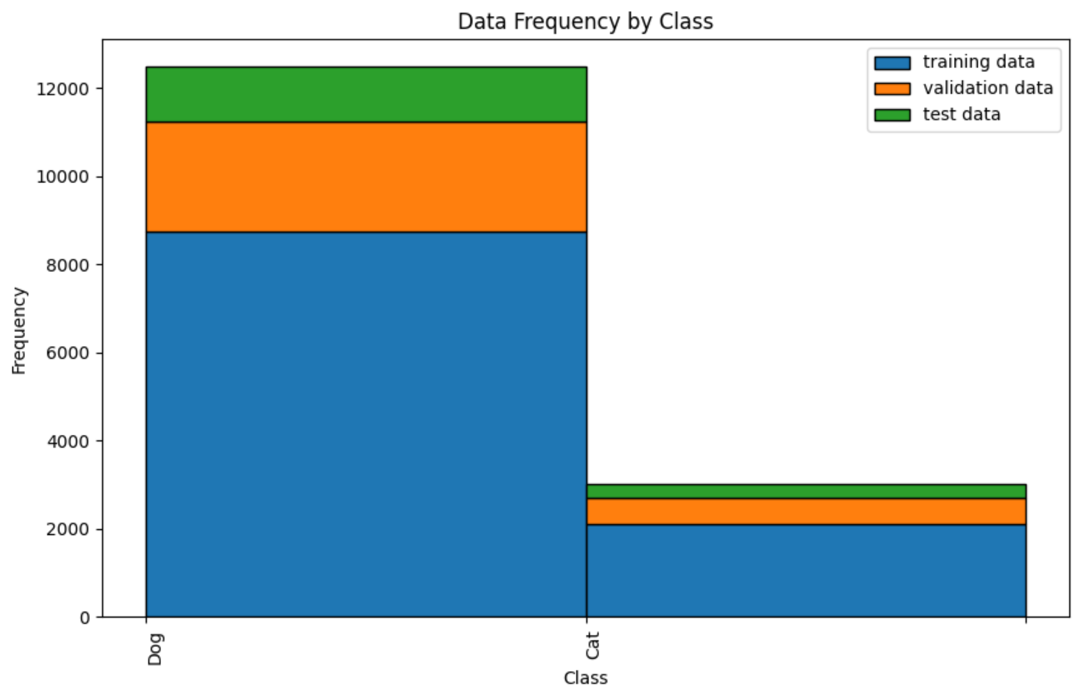

Figure 2: Training, validation, and testing split from the biased dataset.

Algorithm

In this experiment, I built a model to act as the base model for each technique utilized. The algorithm constructed was a TensorFlow Keras Model, based on models from previous studies (Jain; Saha, 2018). TensorFlow is a widely respected and utilized module for machine learning, and as of 2016 is used by over 150 teams at Google (Bonaccorso, 2018). Figure 3 visually depicts the structure of the model built for this analysis.

The first layer of the model is the input layer, which has an input shape (100, 100, 3). The input shape represents the size of the images, 100 by 100 pixels, and the color mode of the images, where 3 denotes RGB images. This input layer is then passed into a convolution layer with 16 filters. This first convolution layer is utilized to capture edges, colors, and other low-level features of the image (Saha, 2018).

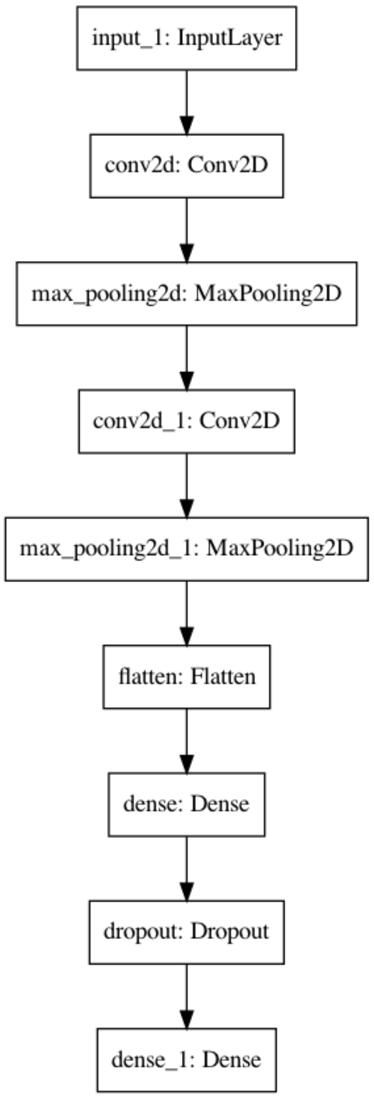

Figure 3: Breakdown by layer of the TensorFlow Keras model utilized.

The convolution layer is then passed into a two-by-two max-pooling layer. The pooling layers reduce the previous layer's output size while also extracting dominant features and suppressing noise in the images (Saha, 2018). These two layers form the first layer of the convolutional neural network. I added a second layer consisting of a convolution layer with 32 filters and a two-by-two max-pooling layer to identify additional low-level details of the images. The output of this second layer is then flattened before being entered into the fully connected layer. This layer is a dense layer with 64 input nodes and utilizes the Rectified Linear Unit (ReLU)

Activation Function. The ReLU activation function returns only positive values inputted and returns 0 for any negative values. This behavior typically enhances the performance of the model by maintaining a distribution between positive values. The connected layer's output is passed into a dropout layer with a dropout rate of 0.5. This layer prevents overfitting by randomly dropping nodes. Finally, the dropout layer outputs are passed into an output layer that utilizes sigmoid activation and a single node. The output of this node is a value between 0 (Cat) and 1 (Dog). The model was compiled using the Adam optimizer with a learning rate of 10^{-6}. The Adam optimizer iteratively updates the weights as training occurs, increasing the performance of the model. Each model was fitted over 30 epochs with a batch size of 32 inputs.

Techniques

In an effort to remove any bias in the model's output, I introduced several preprocessing and in-processing methods to the baseline model described above.

The first technique tested was the preprocessing technique of data augmentation (AUG). This technique consisted of augmenting a copy of all 3,000 cat images to the dataset by rotating and adding noise to the image, creating a slightly different copy of each image. Figure 4 depicts a few examples of this transformation on cat images. After augmentation, there existed 6,000 cat images and 12,493 dog images. These images were then split into training, testing, and validation using the same method as the baseline model, which resulted in 4,200 images of cats and 8,745 images of dogs for training, 1,200 images of cats and 2,498 images of dogs for validation, and 600 images of cats and 1,250 images of dogs for testing.

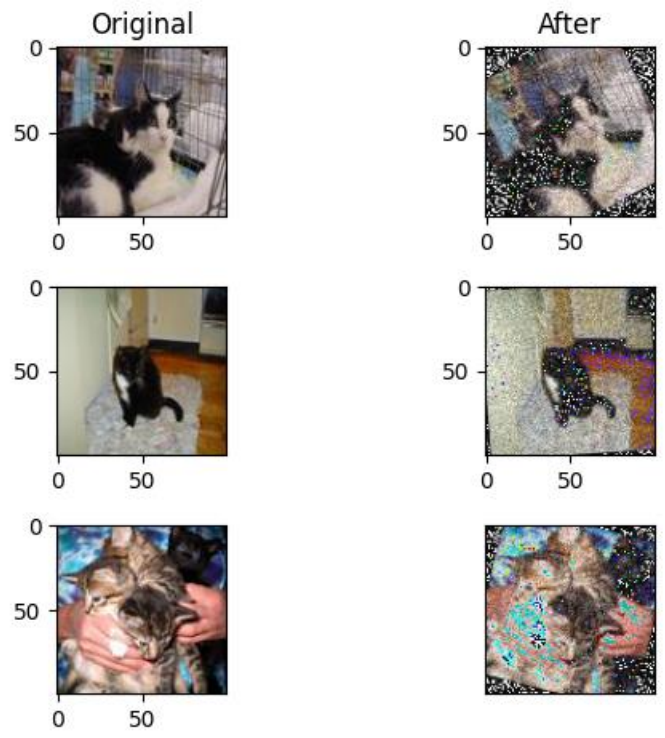

Figure 4: Sample images before and after augmentation.

The second method explored was a preprocessing technique that reduced the number of dog images processed by the model (RED). I reduced the number of dog images to 4,500. This reduction resulted in a training set of 2,100 cat and 3,150 dog images, a validation set of 600 cat and 900 dog images, and a testing set of 300 cat and 450 dog images.

The third method was an in-processing technique that introduced an initial bias measure to the model (INIT). Setting the initial bias allows the model to make a better first guess of values, reducing the model's initial loss. One can calculate a suggested initial bias value using (1). Using this formula, the initial bias value used for this dataset was -1.427.

$$I = \ln(\frac{cats}{dogs}), \qquad (1)$$

where *I* is the initial bias, *cats* is the number of cat samples, and *dogs* is the number of dog samples in the database.

The fourth method utilized was another in-processing technique of introducing class weights within the fitting of the model (WEIGHTS). Adding these weights increases the influence of the inputs of the class with fewer samples. As suggested by TensorFlow's guide to handling imbalanced data (TensorFlow), I calculated the class weight for each class by utilizing (2). The class weight for the dog class was 0.62, and the weight for the cat class was 2.58.

$$W_c = \left(\frac{total}{2c}\right), \qquad (2)$$

where W_c is the weight of class c and *total* is the total number of samples in the dataset.

ANALYSIS

The accuracy, false negative rate, false positive rate, and difference between false positive and false negative rates for each method are displayed in Table 1. The false positive rate (FPR) is calculated using (3).

$$FPR = \frac{FP}{FP+TN}, \qquad (3)$$

where FP is the number of false positives and TN is the number of true negatives predicted. Similarly, the false negative rate (FNR) is found using (4).

$$FNR = \frac{FN}{FN+TP}, \qquad (4)$$

where FN is the number of false negatives and TP is the number of true positives predicted. Table 1 depicts the measures from predicting on the unseen testing dataset. The best score for each criterion is bolded in the table.

By examining the table, one can see that while the WEIGHTS method did decrease the difference between rates, it slightly overweighted the Cat class, resulting in a significant increase in the false negative rate. The class weighting caused a shift in false positives and false negatives in the opposite direction of the baseline model, causing the false negative rate to be higher than the false positive rate. To further explore the class weights technique, I ran an additional model where the class weight of the cat class was calculated by including 2.25 instead of 2 in the denominator of (2) (WEIGHTS2). The goal in slightly decreasing the cat class weight was to achieve a more balanced result where the cat class is more accurately predicted. This model's new class weights were 0.62 for the dog class and 2.07 for the cat class.

	Accuracy	False Negative Rate	False Positive Rate	Difference in Rates
BASELINE	0.803	0.010	0.977	0.967
AUG	**0.828**	**0.006**	0.518	0.512
RED	0.604	0.169	0.737	0.568
INIT	0.803	0.010	0.973	0.963
WEIGHTS	0.517	0.535	**0.263**	0.272
WEIGHTS2	0.610	0.386	0.407	**0.021**

Table 1: Performance measures for bias mitigation techniques on the testing dataset.

Augmenting the images resulted in the highest accuracy and slightly decreased the difference in rates, but it could have caused some overfitting in the model, which would cause the model to perform poorly on new images of cats. Overfitting is a common concern with augmentation and oversampling in data analytics.

I anticipated that reducing the number of dog images would severely decrease the model's performance in terms of accuracy. Still, it could potentially be a solution if bias mitigation were the primary goal of the experiment. Compared to the baseline model, the difference in rates decreased, but the WEIGHTS2 method resulted in better accuracy and a lower difference in rates.

Utilizing the initial bias measure in INIT resulted in measures that were comparable to the baseline model. This method had no significant effect on the model.

The WEIGHTS2 method ultimately resulted in the least biased model with a difference in rates of 0.021. However, this improvement in fairness came at the cost of reduced accuracy. The goal of this experiment was to reduce the bias of the model, but in industry, it is important to examine the business case when setting performance measures for a model.

CONCLUSION

As machine learning becomes more accessible across industries, companies must analyze their models to ensure the algorithms aren't biased against protected groups or features. Bias can manifest within datasets as unequal distributions of classes and within algorithms from choices made by programmers and designers. When bias is present in a dataset, algorithms may interpret the bias as meaningful to the analysis, which leads to biased results.

The process of addressing model's bias can focus on three locations of the model. Preprocessing attempts to address the bias before the algorithm sees the data. In-processing occurs when the bias is handled within the algorithm itself. Postprocessing analyzes the outputs of the algorithm and addresses bias in the results to create a fairer distribution.

Demonstrating an image classification model on an intentionally biased dataset resulted in a baseline model with a severe difference between false positive and false negative rates. Augmenting the data and including class weights within the algorithm were the two most viable options given the results. Specifying the class weights was the most successful method for removing bias, but it did result in a notable decrease in accuracy.

The goal of classifying images of dogs and cats in this experiment demonstrates a simple situation in which bias may not significantly impact the groups involved. In this problem, a developer may prioritize accuracy over bias mitigation since dogs and cats are not considered protected groups. However, in a more complex situation with higher stakes, the model's fairness should be analyzed to ensure safety and equality in predictions. Mitigating bias will continue to be a prominent topic of conversation as machine learning continues to become more accessible across industries.

REFERENCES

Agarwal, A., Beygelzimer, A., Dudík, M., Langford, J., & Wallach, H. 2018. A reductions approach to fair classification. ***International Conference on Machine Learning***: 60-69. PMLR.

Baer, T. & Kamalnath, V. 2017. Controlling machine-learning algorithms and their biases. ***McKinsey Insights***.

Balakrishnan, G., Xiong, Y., Xia, W., & Perona, P. 2020. Towards causal benchmarking of bias in face analysis algorithms. ***Computer Vision – ECCV 2020***: 547-563.

Binns, R. 2018. Fairness in machine learning: lessons from political philosophy. ***Conference on Fairness, Accountability and Transparency***: 149-159. PMLR.

Bonaccorso, G. 2018. ***Machine learning algorithms: Popular algorithms for data science and machine learning*** (2nd ed.). Birmingham, UK: Packt Publishing.

Cai, C.J., Reif, E., Hegde, N., Hipp, J., Kim, B., Smilkov, D., Wattenberg, M., Viegas, F., Corrado, G.S., Stumpe, M.C., & Terry, M., 2019. Human-centered tools for coping with imperfect algorithms during medical decision-making. ***Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems***: 1-14.

Char, D. S., Shah, N. H., & Magnus, D. 2018. Implementing machine learning in health care - Addressing ethical challenges. ***The New England Journal of Medicine***, 378(11): 981-983.

Chouldechova, A., & Roth, A. 2018. The frontiers of fairness in machine learning.

Das, A., Dantcheva, A., & Bremond, F. 2018 Mitigating bias in gender, age, and ethnicity classification: A multi-task convolution neural network approach. ***ECCVW 2018***.

Gianfrancesco, M. A., Tamang, S., Yazdany, J., & Schmajuk, G. 2018. Potential biases in machine learning algorithms using electronic health record data. ***JAMA Intern Med***. 178(11): 1544-1547.

Howard, A., Zhang, C., & Horvitz, E. 2017. Addressing bias in machine learning algorithms: A pilot study on emotion recognition for intelligence systems. ***2017 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)***: 1-7. Austin, TX.

Jain, T. Basics of image classification techniques in machine learning. https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/.

Jiang, H. & Nachum, O. 2019. Identifying and correcting label bias in machine learning.

Kholsa, A., Zhou, T., Malisiewicz, T., Efros, A. A., & Torralba, A. 2012. Undoing the damage of dataset bias. ***Computer Vision – ECCV 2012***: 158-171.

Kim, B., Kim, H., Kim, K., Kim, S., & Kim, J. 2019. Learning not to learn: Training deep neural networks with biased data. ***Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition***: 9012-9020.

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. 2019. A survey on bias and fairness in machine learning.

Microsoft Research. 2013. Dogs vs. Cats. Kaggle. https://www.kaggle.com/c/dogs-vs-cats/data.

Mooney, R. J. 1996. Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning. ***Proceedings of the Conference on Empirical Methods in Natural Language Processing***: 82-91. Philadelphia, PA.

Raschka, S. 2016. ***Python Machine Learning***. Birmingham, UK: Packt Publishing.

Saha, Sumit. A comprehensive guide to convolutional neural networks – The ELI5 way. *Towards Data Science*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. December 15, 2018.

Salvi, M., Acharya, U. R., Molinari, F., & Meiburger, K. M. 2020. The impact of pre-and post-image processing techniques on deep learning frameworks: A comprehensive review for digital pathology image analysis. ***Computers in Biology and Medicine***, 128: 104129-104152.

Srivastava, M., Heidari, H., & Krause, A. 2019. Mathematical notions vs. human perception of fairness: A descriptive approach to fairness for machine learning. ***Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining***: 2459-2468.

TensorFlow. Classification on imbalanced data. *TensorFlow Core*. https://www.tensorflow.org/tutorials/structured_data/imbalanced_data. March 19, 2021.

Wang, Z., Qinami, K., Karakozis, I. C., Genova, K., Nair, P., Hata, K., & Russakovsky, O. 2020. Towards fairness in visual recognition: Effective strategies for bias mitigation. ***Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition***: 8919-8928.

Yapo, A. & Weiss, J. 2018. Ethical implications of bias in machine learning. ***Proceedings of the 51st Hawaii International Conference on System Sciences***: 5365-5372.

Zhao, J., Wang, T., Yatskar, M., Ordonez, V., & Chang, K. W. 2017. Men also like shopping: Reducing gender bias amplification using corpus-level constraints.

APPENDIX

This section contains the code utilized in the experiment in this paper. If this paper is in Word format, each file is inserted as a Word document and can be double clicked to view the entire file. main.py includes functions that specify the calls for each experimental setup. ImageProcessor.py includes the function to process image data from the given directory and the functions that augment the image data. DataProcessor.py contains the functions that prepare the data for modeling and functions to get the weight values for the initial bias and the class weights. Model.py contains the functions to built and fit the Keras model. DataVisualizer.py contains functions that create visuals for the training and testing split as well as results from the model. ResizeImages.py includes code for resizing the images from the dataset to the size 100 by 100 pixels and ensures all images are of type RGB.

main.py

```
import ImageProcessor
import DataProcessor
import DataVisualizer
import Model

RGB = 3
GREY = 1

def biasedBaseline():
    data = ImageProcessor.processImagesBiased('ResizedPetImages')
    train_x, train_y, val_x, val_y, test_x, test_y =
DataProcessor.splitTrainTestByClass(data)
    DataVisualizer.displayGraphsForPreData(train_y, val_y, test_y, "TrainTestBiased")
    train_x, train_y, val_x, val_y, test_x, test_y =
DataProcessor.prepDataForModel(train_x, train_y, val_x, val_y, test_x, test_y)
    mdl = Model.buildModel()
    history = Model.fitModel([train_x, train_y, val_x, val_y, test_x, test_y], mdl)
    DataVisualizer.plotMetrics(history, "BaselineOutput")

def withAugmentation():
    data = ImageProcessor.processImagesBiased('ResizedPetImages')
    data['Cat'] = ImageProcessor.augmentImages(data['Cat'])
    print("Number of cat images after augmenting: " + str(len(data['Cat'])))
    train_x, train_y, val_x, val_y, test_x, test_y =
DataProcessor.splitTrainTestByClass(data)
    DataVisualizer.displayGraphsForPreData(train_y, val_y, test_y,
"TrainTestAugmented")
    train_x, train_y, val_x, val_y, test_x, test_y =
DataProcessor.prepDataForModel(train_x, train_y, val_x, val_y, test_x, test_y)
    model = Model.buildModel()
    history = Model.fitModel([train_x, train_y, val_x, val_y, test_x, test_y], model)
    DataVisualizer.plotMetrics(history, "AugmentationOutput")

def withReducingDogs():
    data = ImageProcessor.processImagesBiased('ResizedPetImages')
    data = ImageProcessor.reduceDogImages(data)
    train_x, train_y, val_x, val_y, test_x, test_y =
DataProcessor.splitTrainTestByClass(data)
    DataVisualizer.displayGraphsForPreData(train_y, val_y, test_y, "TrainTestReduced")
    train_x, train_y, val_x, val_y, test_x, test_y =
DataProcessor.prepDataForModel(train_x, train_y, val_x, val_y, test_x, test_y)
    model = Model.buildModel()
    history = Model.fitModel([train_x, train_y, val_x, val_y, test_x, test_y], model)
    DataVisualizer.plotMetrics(history, "ReducedOutput")

def withBiasMeasure():
    data = ImageProcessor.processImagesBiased('ResizedPetImages')
```

ImageProcessor.py

```
import PIL
from PIL import Image
import os
import numpy as np
import imgaug as ia
import imgaug.augmenters as iaa
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2

#helper function that applies augmentation on an image
def augment(image):
    #rotate image
    rotate = iaa.Affine(rotate=(-50, 30))
    image2 = rotate.augment_image(image)
    #add noise to image
    noise = iaa.AdditiveGaussianNoise(10,20)
    image2 = noise.augment_image(image2)
    return image2

#This function reads in the images from the designated folder and returns a dictionary
of the image data separated by class.
def processImagesBiased(image_dir):
    print("Processing images from " + image_dir + "...")
    data = {}
    cats = 1
    for directory in os.listdir(image_dir):
            for filename in os.listdir(image_dir + '/' + directory):
                if 'Cat' in directory:
                    if cats > 3000:
                        break
                    else:
                        cats += 1
                path = image_dir + '/' + directory + '/' + filename
                image = Image.open(path)
                classifier = directory
                if 'RGB' not in image.mode:
                    print("Image not RGB: " + path)
                    continue
                image = np.array(image)
                image = image.astype('float32')
                image /= 255
                if classifier not in data:
                    data[classifier] = [image]
                else:
                    data[classifier].append(image)
    print("Number of dog photos: " + str(len(data["Dog"])))
```

DataProcessor.py

```
import random
import csv
import tensorflow as tf
import numpy as np
import sklearn

class_dict = {}

def getInitialBiasMeasure(data):
    num_cats = len(data['Cat'])
    num_dogs = len(data['Dog'])

    #initial bias calculation
    bias = np.log([num_cats/num_dogs])
    return bias

def getClassWeights(data):
    cats = len(data['Cat'])
    dogs = len(data['Dog'])
    total = dogs + cats

    #class weight values
    weight0 = (1/cats)*(total)/2
    weight1 = (1/dogs)*(total)/2
    class_weight = {0: weight0, 1: weight1}
    print("Weight for class Cat: " + str(weight0))
    print("Weight for class Dog: " + str(weight1))
    return class_weight

#shuffle the data points while maintaining the input/target pairs
def shuffleData(x, y):
    x, y = sklearn.utils.shuffle(x, y, random_state=7)
    return x, y

#get the numerical class dictionary for the given classes
def getClassDict(arr):
    global class_dict
    if not class_dict:
        categories = np.unique(arr)
        class_dict = {v : k for v, k in zip(categories,
list(range(0,len(categories))))}
        print(class_dict)

#enumerate the array of class data as its numerical representation
def enumerateClassArray(arr):
    getClassDict(arr)
    data = []
```

Model.py

```
from numpy.random import seed
import tensorflow.random
from tensorflow.keras.layers import Dense, Flatten, Dropout, Input, Conv2D,
MaxPooling2D, BatchNormalization
from tensorflow.keras import Model, initializers
from tensorflow.data import Dataset
from tensorflow.keras import metrics
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from tensorflow.keras.utils import plot_model
import numpy as np
import sys
import time

seed(7)
tensorflow.random.set_seed(7)
img_size = 100

METRICS = [
      metrics.TruePositives(name='tp'),
      metrics.FalsePositives(name='fp'),
      metrics.TrueNegatives(name='tn'),
      metrics.FalseNegatives(name='fn'),
      metrics.BinaryAccuracy(name='accuracy'),
      metrics.Precision(name='precision'),
      metrics.Recall(name='recall'),
      metrics.AUC(name='auc'),
]

def buildModel(metrics=METRICS, bias=None):
    if bias is not None:
        bias = initializers.Constant(bias)

    img_input = Input(shape=(img_size, img_size, 3))
    #first convolution
    lay = Conv2D(16, 3, activation='relu')(img_input)
    lay = MaxPooling2D(2)(lay)
    #second convolution
    lay = Conv2D(32, 3, activation='relu')(lay)
    lay = MaxPooling2D(2)(lay)
    #flatten
    lay = Flatten()(lay)
    #connected layer
    lay = Dense(64, activation='relu', kernel_initializer='random_normal')(lay)
    #dropout layer
    lay = Dropout(0.5)(lay)
```

DataVisualizer.py

```
import matplotlib.pyplot as plt
import numpy as np

#displays the ditribution of training and testing data by class
def displayGraphsForPreData(train_y, val_y, test_y, outputFilename):
    np.warnings.filterwarnings('ignore', category=np.VisibleDeprecationWarning)
    print("Visualizing the pre-data")
    plt.figure(figsize=(10,6))
    data = [train_y, val_y, test_y]
    num_bins = len(np.unique(train_y))
    labels = ['training data', 'validation data', 'test data']
    n, bins, patches = plt.hist(data, bins=num_bins, edgecolor='black', linewidth=1,
stacked=True,label=labels)
    plt.title("Data Frequency by Class")
    plt.xlabel('Class')
    plt.ylabel('Frequency')
    plt.legend(loc='upper right')
    plt.xticks(ticks=bins, labels=np.unique(train_y).tolist(), rotation=90,
horizontalalignment='left')
    plt.savefig(outputFilename)

def plotMetrics(history, outputFilename):
    metrics = ['loss', 'auc', 'precision', 'recall']
    for n, metric in enumerate(metrics):
        name = metric.replace("_"," ").capitalize()
        plt.subplot(2,2,n+1)
        plt.plot(history.epoch, history.history[metric], label='Train')
        plt.plot(history.epoch, history.history['val_'+metric], linestyle="--",
label='Val')
        plt.xlabel('Epoch')
        plt.ylabel(name)
        if metric == 'loss':
            plt.ylim([0, plt.ylim()[1]])
        else:
            plt.ylim([0,1])
        plt.legend()
    plt.tight_layout()
    plt.savefig(outputFilename)
```

<u>ResizeImages.py</u>

```
import PIL
from PIL import Image
import os
import sys

img_size = 100
count = 0
p = 'PetImages'
for directory in os.listdir(p):
    if '.DS_Store' in directory:
        continue
    for filename in os.listdir(p + '/' + directory):
        path = p + '/' + directory + '/' + filename
        img = Image.open(path)
        copy = img.resize((img_size, img_size))
        if copy.mode in ('RGBA', 'P', 'RGB'):
            copy = copy.convert('RGB')
        if not os.path.exists('ResizedPetImages/' + directory + '/'):
            os.mkdir('ResizedPetImages/' + directory + '/')
        path = 'ResizedPetImages/' + directory + '/' + filename
        copy.save(path, 'JPEG')
        print(path)
```