Towards Effective Tools for Debugging Machine Learning Models

by

Julius Adebayo

B.Sc., Meche, Brigham Young University (2012)
S.M., TPP, Massachusetts Institute of Technology (2016)
S.M., EECS, Massachusetts Institute of Technology (2016)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author .....
Department of Electrical Engineering and Computer Science
August 26, 2022

Certified by .....
Hal Abelson
Class of 1922 Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards Effective Tools for Debugging Machine Learning Models

by

Julius Adebayo

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis addresses the challenge of detecting and fixing the errors of a machine learning (ML) model—model debugging.

Current ML models, especially overparametrized deep neural networks (DNNs) trained on crowd-sourced data, easily latch onto spurious signals, underperform for small subgroups, and can be derailed by errors in training labels. Consequently, the ability to detect and fix a model's mistakes prior to deployment is crucial.

Explainable machine learning approaches, particularly post hoc explanations, have emerged as the defacto ML model debugging tools. A plethora of approaches currently exist, yet it is unclear whether these approaches are effective.

In the first part of this thesis, we introduce a framework to categorize model bugs that can arise as part of the standard supervised learning pipeline. Equipped with the categorization, we assess whether several post hoc model explanation approaches are effective for detecting and fixing the categories of bugs proposed in the framework. We show that current approaches struggle to detect a model's reliance on spurious signals, are unable to identify training inputs with wrong labels, and provide no direct avenue for fixing model errors. In addition, we demonstrate that practitioners struggle to use these tools to debug ML models in practice.

With the limitations of current approaches established, in the second part of the thesis, we present new tools for model debugging. First, we introduce an approach termed model guiding, which uses an audit set—a small dataset that has been carefully annotated by a task expert—to update a pre-trained ML model's parameters. We formulate the update as a bilevel optimization problem that requires the updated model to match the expert's predictions and feature annotations on the audit set. Model guiding can be used to identify and correct mislabelled examples. Similarly, we show that the approach can also remove a model's reliance on spurious training signals.

The second debugging tool we introduce uses the influence function of an estimator to help identify training points whose labels have a high effect on an ML model's disparity metric such as group calibration.

Taken together, this thesis makes advances towards better debugging tools for machine learning models.

Thesis Supervisor: Hal Abelson
Title: Class of 1922 Professor of Electrical Engineering and Computer Science

Acknowledgments

This thesis is a culmination of work that began during my initial time, at MIT, in the TPP program. I took Danny and Hal's privacy legislation course with students from Georgetown law school. Through that experience, I became interested in methods for interpreting black-box ML models. I have Hal and Danny to thank for that experience, and the opportunity to come back to MIT for the PhD. They thought research questions relating to the reliability of machine learning models was important before it became fashionable. I am indebted to them. Similarly, I am immensely grateful to the rest my thesis committee: Prof. Martin Wattenberg, and Prof. Marzyeh Ghassemi for feedback that improved this work.

Several of the papers that constitute this thesis came out of collaboration with Micheal Muelly and Been Kim; I thank them for being great sounding board for research. I would also like to thank Moritz Hardt for mentorship from afar, and helping to think through one of the first papers that started the line of work that this thesis sprung from. Along those lines, I had several fruitful discussions with David Bau, Prof. Himabindu Lakkaraju, and Prof. Sameer Singh on research directions, which ultimately made me a better researcher.

I would like to thank the other IPRI students that I overlapped with during my time at MIT: Jess Van Brummelen, Jonathan Frankle, Mike Specter, Ben Yuan, Leilani Gilpin, Cecilia Testart Pacheco, Sam DeLaughter, Vaik Mugunthan, Grace Abuhamad, Rebecca Spiewak, and Kevin Paeth. All these folks contributed to the great IPRI atmosphere. I'd also like to thank IPRI staff, Taylor Reynolds and Mel Robinson, for always finding a way for me to get cloud credits for running my ML experiments. I'd also like to thank the EECS administrative staff, especially, Janet Fischer, for all the help as I navigated the program.

I made several friends, too many to name, through MIT BGSA, EECS Black Tea, CSAIL student groups, TPP, and fellow Nigerian students. All of these groups were crucial to my successfully navigating the PhD. In no particular order, I'd like to thank Mobolaji Akinpelu, Kesiena Owho-Ovuakporie, Angi Acocella, Mureji Fatunde, Chibueze Amanchukwu, Biodun Olaoye, Marianne, Daniel Alabi, Caris Moses, Noah Jones, Lelia Hampton, Dayo Aderibole, Jide Ezike, Sami Khan, Tiziana Smith, Lily Mwalenga, Serdar OzcanFahad Alhasoun, Garett Schroath, Brock Laney, Eric Mibuari, and several more that I am forgetting to mention here.

I am grateful to my undergraduate research mentor: Sean Warnick for getting me started on the research path, and always willing to check-in with me. Through Sean's group, I met

Phil Pare and Anurag Rai who have remained friends over the years, and are always willing to give me both personal and research advice.

The Open Philanthropy fellowship funded me throughout my PhD. I am tremendously grateful to them for their generous and no-strings-attached funding.

I am grateful to the Johnsons, in Houston, who have become my second family away from home. With the completion of this thesis, I am glad to report that I am no longer in school.

Of course, this journey would not have been possible without my family: Mom, Bukky, Ola, and my brother-in-law Sewedo. I thank them for their support throughout the PhD journey. The decision to do the PhD was a tough one; after going back-and-forth over it, I made the call to pursue it after a long phone call with my Dad. Unfortunately, he passed away during my first semester of graduate school. I am grateful to him for the tremendous amount of sacrifices he made to allow my siblings and I to pursue our dreams.

Contents

1 Introduction **17**

1.1 Model Debugging: A Preview . . . 21

1.2 Part I: Empirical Assessment of Current Methods . . . 26

1.3 Part II: New Tools for model debugging . . . 29

1.4 Thesis Bibliography . . . 31

2 Model Debugging **33**

2.1 Overview . . . 33

2.2 Model Debugging as Model Understanding . . . 33

2.3 A Categorization of Model Bugs in Supervised Learning . . . 36

2.3.1 Prediction Phase Model Bugs . . . 39

2.4 Related Work . . . 42

3 Post hoc Explanation Methods **43**

3.1 Overview . . . 43

3.2 Feature Attribution Methods . . . 43

3.3 Concept Activation Methods . . . 46

3.4 Training Point Ranking . . . 47

3.5 How are the approaches used in practice? . . . 52

4 Challenges with Model Debugging Using Post hoc Explanation Methods **53**

4.1 Overview & Summary of Results . . . 54

4.2 Challenges with Debugging Reliance on Spurious Signals . . . 55

4.2.1 Experimental Design for Spurious Signal Detection . . . 56

4.2.2 Results for Feature Attributions . . . 59

4.2.3 Results for Concept Activation Methods . . . 61
4.2.4 Results for Training Point Ranking . . . 63
4.3 Challenges with Debugging Noisy Training Labels . . . 63
4.4 Challenges with Debugging Parameter Contamination . . . 64
4.5 Challenges with Debugging Out-of-Distribution Inputs . . . 66
4.6 Challenges with Current Tools in Practice: User Study . . . 67
4.7 Related Work . . . 74
4.8 Epilogue . . . 77

5 Model Guiding **79**
5.1 Overview . . . 79
5.2 Model Debugging with Model Guiding . . . 80
5.2.1 Model Guiding Pre-Processing Step: Audit Set Annotation & Model Carving . . . 81
5.2.2 The Bilevel Update: Formulation . . . 85
5.2.3 Solving a Bilevel Optimization problem . . . 86
5.2.4 The Model Guiding Scheme . . . 89
5.3 Model Guiding for Spurious Correlation & Noisy Training Label Bugs . . . 90
5.4 Related Work . . . 93

6 Debugging Fairness Violations with Influence Functions **97**
6.1 Overview . . . 97
6.2 Setup & Background . . . 99
6.3 Empirical Assessment of Label Sensitivity . . . 100
6.3.1 Test-time Label Sensitivity . . . 101
6.3.2 Training-time Label Sensitivity . . . 102
6.4 Identifying 'Fairness Violations' with Influence Functions . . . 103
6.5 Empirical Assessment . . . 105
6.6 Related Work . . . 107

7 Conclusion **111**

A Chapter 4 Appendix: Challenges **129**
A.1 Experimental Details for Spurious Correlation . . . 129

A.2 Experimental Details Mislabelled Examples, Weight-Reinitialization, & Out-of-Distribution Robustness . . . 133
A.2.1 Mislabelled Examples . . . 134
A.2.2 Weight Re-initialization . . . 134
A.3 Additional Details on Mislabelled Examples . . . 135
A.4 Additional Details on User Study . . . 137

B Chapter 6 Appendix: Influence Functions **145**
B.1 Appendix: Datasets, Models, & Experimental Details . . . 145
B.1.1 Datasets . . . 145
B.1.2 Models . . . 147
B.1.3 Additional Discussion on ECE . . . 147

List of Figures

1-1 **Example of a model relying on spurious image tag**. A model trained to classify a hand radiograph into different age groups is relying on the hospital tag, in the image, to identify the age of the Early/Mid Puberty patients. . . . 18

1-2 *Example of a feature attribution/saliency map 'explaining' the prediction of DNN model, trained for animal classification, for two different inputs.* 19

1-3 A summary table showing three various model bugs and two feature attribution methods. We that the two feature attribution approaches are only effective for detecting known spurious signals. 27

1-4 Three different kinds of feature attributions for a DNN model trained to perform animal classification. On the same input, the attribution for a setting where the input was trained with the correct label, and another where the input was trained without the correct label is shown. 28

1-5 **Model guiding schematic**. 29

3-1 The Figure shows feature attributions for two inputs for a CNN model trained to distinguish between birds and dogs. 44

3-2 **A**: Here we show 5 different inputs, one for each bone age category, and the corresponding Gradient feature attribution map. We refer to the Appendix for an equivalent visualization for other feature attribution methods considered. We test three additional feature attribution methods: SmoothGrad, Integrated Gradients, and Guided BackProp. **B&C**:Concept Importance Methods for a Normal (non-spurious) model. Here we show the TCAV score for all clinical concepts as well as the spurious concepts for a normal model that was confirmed to not rely on the spurious signals. **D**:Top ranked influential examples for a test pre-puberty input on a normal model. 47

4-1 The various spurious signals considered: A hospital tag, vertical stripes, and background blur. 56

4-2 **Detecting Spurious *Tag*.** Here we show in A) Feature attributions for 5 different inputs across the four feature attribution methods with a normal model but with spurious Tag inputs; B) Feature attributions on the same 5 inputs as in (A), but **without** spurious Tag inputs with a model that has learned a spurious alignment between Pre-Puberty and Tag; C) Feature attributions on the same 5 inputs as in (A), but **with** the spurious Tag inputs with a model that has learned a spurious alignment between Pre-Puberty and Tag. 60

4-3 **Detecting Spurious *Stripe*.** Here we show in A) Feature attributions for 5 different inputs across the four feature attribution methods with a normal model but with spurious Stripes inputs; B) Feature attributions on the same 5 inputs as in (A), but **without** the spurious Stripe with a model that has learned a spurious alignment between Pre-Puberty and Stripe; C) Feature attributions on the same 5 inputs as in (A), but **with** the spurious Stripe with a model that has learned a spurious alignment between Pre-Puberty and Stripe. 60

4-4 **Detecting Spurious *Blur*.** The blur images are analogous to the Tag and Stripe settings. 61

4-5 **Concept Results for Normal, Tag, Stripe and Blur Models.** TCAV scores for a model reliant on the spurious Tag and a model reliant on Blur. . 62

4-6 Feature Attributions for noisy label detection. 64

4-7 **Evolution of several model attributions for successive weights re-initialization of a VGG-16 model trained on ImageNet.** Qualitative results (left) and quantitative results (right). The last column in qualitative results corresponds to a network with completely re-initialized weights. 65

4-8 **Fashion MNIST OOD on several models.** The first row shows feature attributions on a model trained on Fashion MNIST. In the subsequent rows, we show feature attributions for the same input on an MNIST model, BVD-CNN model trained on birds-vs-dogs, and lastly, a pre-trained VGG-16 model on ImageNet. 66

4-9 Overview of the interface as it was shown to the users. 68

4-10 Description of the different attribution signs as it was explained to users. . . . 69

4-11 Age, Employment, and Education break-down among participants. 71

4-12 Gender, and Ethnicity break-down among participants. 72

4-13 ML Experience and Attribution Familiarity break-down among participants. . 72

4-14 **Left: Participant Responses from User Study.** We show a Box plot of participants responses across the three different attribution methods: *Gradient, SmoothGrad, and Integrated Gradients*, and 7 model conditions. On the vertical axis, we have the user reported likelihood of recommendation that a model condition be sold to external customers, from 1 : *Definitely Not* to 5 : *Definitely*. We see that across all attribution types, the participants tend to recommend that a normally trained model be sold to external customers. **Right: Motivation for Recommendation.** We show an overview of the breakdown in motivation according to both attribution and model manipulation type. On the Y axis we have the Likert recommendation by the participant. Higher means the participants are more likely to recommend this action. On the X-axis we have the motivation for selecting the different Likert recommendations. The values on the bars represent the average recommendation for each motivation. We now provide the legend for the motivation: 1: On some or all of the images, the dog breed was wrong; 2: The explanation did not highlight the part of the image that I expected it to focus on; 3: The explanation highlighted the parts of the image that I expected it to focus on; 4: The dog breeds were correct; & 5: Other, please specify. 74

5-1 Model debugging as a dialog with an auditor. 81

6-1 **Test-time Label Flipping Results across 6 datasets**. For each dataset, we plot the percent change in calibration error versus the corresponding percentage change in label error. Here, we plot the minority (smallest) group as well as the majority (largest) group. These two groups represent two ends of the spectrum for the impact of label error. We observe that across all datasets, the minority group incurs higher percentage change in group calibration compared to the majority group. 102

6-2 **Training-time Label Flipping Results across 6 datasets**. For each dataset, we plot the percent change in calibration error versus the corresponding percentage change in label error for the training set. Here, we plot the minority (smallest) group as well as the majority (largest) groups by size. Similar to the test-time setting, we observe that across all datasets, the minority group incurs higher percentage change in group calibration compared to the majority group. However, we observe a larger magnitude change for the minority groups. 103

6-3 **Empirical Results for Training Point Ranking Across 6 datasets**. For the top 50 most influential examples, we show the proportion of samples whose labels were flipped in the training data. Confidence intervals is computed for N=5 different models. 106

7-1 **Hypothetical model debugging workflow.** 112

A-1 **Feature Attributions for Mislabeled examples.** 135

A-2 **Feature Attributions for Mislabeled examples.** 135

A-3 **Feature Attributions for Mislabeled examples.** 136

A-4 **Saliency Maps for a Normal Model: Gradient.** 137

A-5 **Saliency Maps for a Normal Model: SmoothGrad.** 138

A-6 **Saliency Maps for a Normal Model: Integrated Gradients.** 139

A-7 **Saliency Maps for a Top layer Random Model: Gradient.** 140

A-8 **Saliency Maps for a Top layer Random Model Model: SmoothGrad.** 140

A-9 **Saliency Maps for a Top layer Random Model Model: Integrated Gradients.** 141

A-10 **Saliency Maps for a Spurious Model: Gradient.** 141

A-11 **Saliency Maps for a Spurious Model: SmoothGrad.** 142

A-12 **Saliency Maps for a Spurious Model: Integrated Gradients.** 142

A-13 **Saliency Maps for Out-of-Distribution Examples: Gradient.** 143

List of Tables

2.1 **Table of different bugs.** We show different examples of bugs under the different contamination classes: data, model and test-time. We also formalize these bugs for the traditional supervised learning setting. 39

4.1 Performance metrics for each attribution method across tasks for the Tag Setting. Below each metric in the Table is another row (SEM) that indicates the standard error of the mean for each value. 61

4.2 Tag Metrics. 63

4.3 Training Point Ranking 63

4.4 **Test-time Explanation Similarity Metrics.** We observe visual similarity but no ranking similarity. We show each metric along with the standard error of the mean calculated for 190 examples. FMNIST → MNIST model means a comparison of FMNIST attributions for an FMNIST model with FMNIST attributions derived from *an MNIST model*. We present both SSIM and Rank correlation metrics. 67

5.1 Performance (Area under the receiver operating curve AUROC) of model guiding at detecting noisy training labels compared to baselines. 92

5.2 Performance (Area under the receiver operating curve AUROC) of model guiding at suggesting the correct fix for the noisy training labels compared to baselines. 92

5.3 Performance (Area under the receiver operating curve AUROC) of model guiding at suggesting at detecting examples for which a model is relying on a spurious signal. 93

5.4 The worst-group test set performance of a the post-processed model guiding approach compared to other baselines. 93

6.1 Dataset characteristics. n is the training set size and d is the number of features. 100

Chapter 1

Introduction

In this thesis, we describe contributions that chart a path towards effective tools for detecting and fixing the mistakes of a machine learning (ML) model—*model debugging*. The march towards incorporating or outright replacing human-based decision-making systems with ML models across facets of human life is accelerating. It is now routine to train models, with potentially billions of parameters, for tasks in protein folding [Jumper et al., 2021, Rives et al., 2021], and healthcare [McBee et al., 2018, Wiens and Shenoy, 2018]. ML models have been used as components of automated decision systems to assess an individual's credit worthiness [Bacham and Zhao, 2017], predict recidivism and suitability for bail [Kleinberg et al., 2018, Rigano, 2019, Završnik, 2021], and to target humanitarian assistance [Aiken et al., 2021]. The hope, in using ML models across these applications, is that it will lead to efficiency gains, and a reduction in human-induced errors.

Current ML models are unreliable. Despite impressive performance on benchmarks, state-of-the-art (SOTA) ML models, particularly overparametrized deep neural networks (DNNs) trained on crowd-sourced datasets, are mostly inscrutable and prone to mistakes. If an ML model makes a mistake regarding a defendant's bail decision, an applicant's loan request, or a patient's diagnosis, the ramifications of such a mistake could be severe for the individual in question. While these examples might seem hypothetical, evidence continues to accrue showing that ML models are prone to errors. ML models trained via classic empirical risk minimization (ERM), often latch onto 'spurious'—a notion we will make more precise later—signals in the training data [Nagarajan et al., 2020, Sagawa et al., 2019, 2020]. For example, Badgeley et al. [2019] showed that an Inception-V3 DNN model trained to detect

hip fracture, from radiographs, was relying on the type of scanner used to take the radiograph as the key signal for its output. ML models trained on clinical notes that have even been striped of markers that might signal a patient's race still show disparate performance across racial groups [Adam et al., 2022]. DNN models often base their output, solely, on object backgrounds [Xiao et al., 2020], the texture of an image [Geirhos et al., 2018], and skin tone [Stock and Cisse, 2018]—all signals that are unrelated to the task for which the model was trained. Beyond latching onto spurious signals, ML models tend to underperform for minority groups—in size—[Buolamwini and Gebru, 2018], and are easily derailed by errors in the training labels [Northcutt et al., 2021].

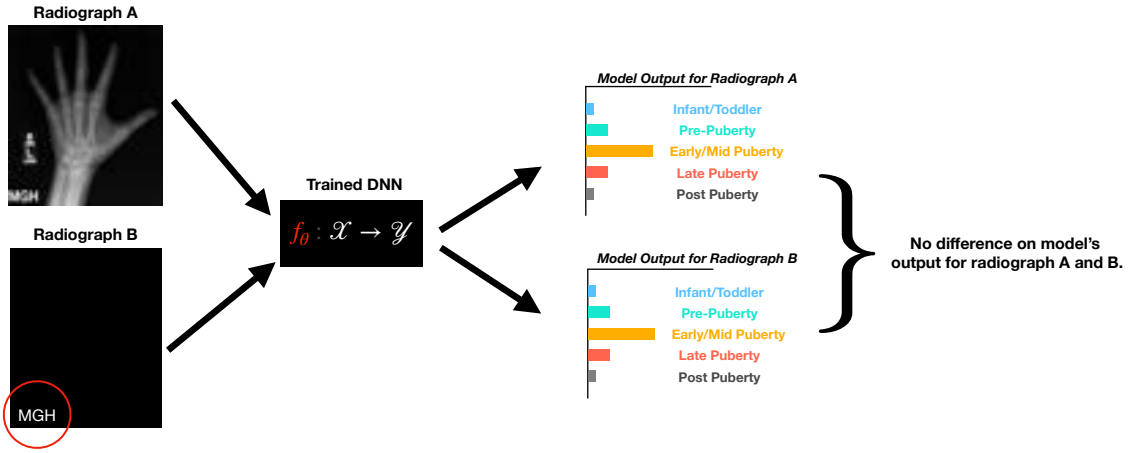

Figure 1-1: **Example of a model relying on spurious image tag**. A model trained to classify a hand radiograph into different age groups is relying on the hospital tag, in the image, to identify the age of the Early/Mid Puberty patients.

Model debugging is the process of detecting and fixing the mistakes of a machine learning model. Developing tools that make model debugging easier is important for both ML practitioners and regulators that have to audit ML models. For example, the Equal Credit Opportunity Act (ECOA) [ECOA, 1974], a key consumer protection statute in the United States (USA), prohibits 'discrimination' on the basis of *protected attributes* like race, sex, religion, and disability in decisions about credit. Similarly, the recent draft of the Artificial Intelligence Act [Commission, 2022] and General Data Protection Regulation (GDPR) [Commission, 2018] that took effect in 2018 contain language that mandates the need for explanation tools for ML models. While enforcement details regarding the AI Act, the GDPR's 'right to explanation', and ECOA's protected act prohibition, as it pertains to ML models, remain vague; it seems clear that there is need for model debugging tools that

can help ML practitioners meet compliance requirements, and help regulators better vet ML models.

Post hoc explanations have become the defacto tools for model debugging. In response to the pressing need for model debugging tools, *post hoc model explanation methods*, have emerged as defacto tools for performing model debugging. Post hoc model explanation methods are approaches that produce artifacts, to be shown to an end-user, from an already trained model, as a way to 'explain' the model's 'reasoning'. These approaches give insight into the associations that a model has learned from data. For example, a common post hoc model explanation is a *feature attribution*, which is a method that apportions the output of a model across all the input dimensions for a single example. Consequently, a feature attribution can help indicate which dimensions of an input are most 'relevant' to the output decision of the model being 'explained'. Inspecting feature attributions has become a standard approach for debugging an ML model.

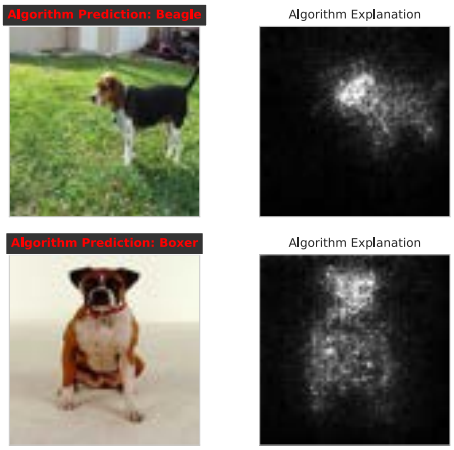

Figure 1-2: *Example of a feature attribution/saliency map 'explaining' the prediction of DNN model, trained for animal classification, for two different inputs.*

A plethora of post hoc explanation approaches exist but it is unclear if they are effective. While *post hoc model explanation methods* have been shown to be effective for detecting model mistakes in specific settings, their status as the defacto model debugging tool remains unjustified. In a now celebrated example, Ribeiro et al. [2016] used LIME, a feature attribution method[1], to show that a DNN model was relying on the presence of snow in an image to detect huskies in an animal classification task. Following this demonstration, a flurry of work introducing new post hoc explanation methods followed; a review of this line of work identified about 46 post hoc explanation approaches for the image modality alone [Gilpin et al., 2018, Samek et al., 2021]. Despite such valuable methodological contribution, little guidance exists for the practitioner about which method to pick for a model debugging

[1]Strictly speaking, a feature attribution method apportions the output across the dimensions of the input, so LIME is a feature 'importance' method. However, we'll abuse terminology in this thesis and refer to all methods that assign a relevance score to each dimension of the input as a feature attribution methods.

task at hand. For example, should a method that is effective for detecting which training samples are mislabelled also be effective used for detecting whether an ML model is relying on spurious signals? A successful answer to these questions should provide concrete guidelines for practitioners looking to use model debugging tools effectively [Wattenberg et al., 2016].

Post hoc explanation methods are unreliable. Even more concerning, the current state of the literature on post hoc model explanations provides conflicting evidence on their effectiveness. Despite initial evidence that explanations might be useful for detecting that a model is reliant on spurious signals [Lapuschkin et al., 2019, Rieger et al., 2020], a different line of work directly counters this evidence. Chu et al. [2020] found that end-users were unable to effectively use feature attributions to detect that a model was relying on a spurious signal in an age prediction task. Similarly, Zimmermann et al. [2021] showed that feature visualizations [Olah et al., 2017] are not more effective than dataset examples at improving a human's understanding of the features that highly activate a DNN's intermediate neuron. In a different setting, Sixt et al. [2022] found, in a user study, that both concept-based and counterfactual explanations were not more effective than simply inspecting input-output examples for identifying model biases. Increasing evidence demonstrates that current post hoc explanation approaches might be ineffective in practice [Alqaraawi et al., 2020, Balagopalan et al., 2022, Bolukbasi et al., 2021, Chen et al., 2021, Ghassemi et al., 2021, Poursabzi-Sangdeh et al., 2018].

Thesis Goal. This thesis takes steps to resolve conflicting evidence on the effectiveness of post hoc explanation methods. Specifically, the goal of this thesis is:

to identify and develop tools that are effective for debugging machine learning models.

Key Thesis Questions. We address the thesis goal by tackling the following two questions:

1. First, given a categorization of model bugs, can we identify which classes of post hoc model explanations are effective for which classes for model bugs? A successful answer to this line of inquiry will provide guidance, to both a practitioner and a regulator, about when, and for what task, a post hoc explanation method is effective.

2. Second, can we develop model debugging approaches that overcome limitations of current approaches? Specifically, can we develop model debugging approaches that

can simultaneously address several debugging tasks while also incorporating human expertise?

Key Thesis Contributions. In addressing the thesis questions, we make the following contributions:

1. **Part I: Empirical Assessment of Current Methods.** First, we provide a way to categorize the kind of bugs that can emerge as part of the standard supervised learning pipeline, and then conduct a comprehensive assessment of current model debugging approaches—typically post hoc model explanation methods—to identify which current tools are effective for which model bugs. We show that current feature attribution approaches struggle to detect a model's reliance on spurious signals, are unable to identify training inputs with wrong labels, and provide no direct avenue for fixing model errors.

2. **Part II: New tools for model debugging.** In the second part of the thesis, we introduce two new approaches that directly address the limitations of current methods.

Overview of the rest of the introduction. We now discuss the structure of the rest of this introductory chapter. First, we give the reader a preview of the bug categorization and the general model debugging principle that grounds this thesis. Second, we overview results of the empirical assessment of current methods, and then discuss the two new methods for model debugging that we introduce in this thesis. We close the introduction with a discussion of the research papers that constitute the thesis, and how they map to each chapter in the rest of the document.

1.1 Model Debugging: A Preview

What is a *model bug*? In this thesis, we consider model bugs to be 'contamination' in the learning, data, and/or prediction pipeline that causes a model to produce incorrect predictions. We restrict our focus to the standard supervised learning paradigm where the goal is to minimize an empirical risk. Specifically, we assume that we are given input-label pairs, $\{(x_i, y_i)\}_{i=1}^n$, where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, and seek to learn a model—a function— $f_\theta : \mathcal{X} \to \mathcal{Y}$, that *generalizes*—performs well on new inputs not seen during training. The model, f_θ, is

then used to provide predictions on test examples, $x_{\text{test}} \in \mathcal{X}$, as $y_{\text{test}} = f_\theta(x_{\text{test}})$. We say a model bug has occurred when a model's output differs, *significantly*, from the 'true' label. We make the notion more precise below.

Model Bug: Given a tolerance parameter, $\tau \geq 0$, a sample, x_k, whose true label is y_k, a *model bug* occurs when: $\|f_\theta(x_k) - y_k\|_p \geq \tau$.

In the model bug definition, the condition that triggers a bug is the relationship: $\|f_\theta(x_k) - y_k\|_p \geq \tau$; however, the cause can be any component of a ML pipeline. The tolerance, τ, and the choice of the norm, $\|.\|_p$, are user provided parameters that allow the user to specify how much wiggle room can tolerated before a bug is triggered. For example, consider a house price prediction task where the 'true' price of a house is 100k USD, and a model predicts a price that is within 10k USD of this true price. If the model designer is comfortable with a 10k USD wiggle room, then there is no need to trigger a model bug for this prediction. However, if the model's prediction, on a 100k USD house is 50k USD, then one might trigger a bug for this scenario. In Chapter 2, we provide variations of the definition of a model bug for generic functions of a model's parameters and a set of inputs.

A systematic synthesis of model debugging is missing. As ML models increasingly replace traditional software and hand-coded rules, previous approaches for debugging traditional software have proved to be largely ineffective. A codified knowledge and guidance for a practitioner looking to effectively debug a machine learning model is currently missing. As it stands, there exist rules of thumb, and adhoc retrospectives about specific model bugs [Sculley et al., 2015, Zinkevich Martin, 2020]; however, a systematic synthesis of model debugging is missing.

To develop tools that can help better debug ML models, it is important to understand the space of bugs that can derail an ML model. Each type of model bug can then inform tool development, and an overall approach to address the bug. As it currently stands, the landscape of possible model bugs is wide, and mostly untamed. Researchers and practitioners often approach each model bug in a specialized manner. A categorization of model bugs can lead to systematization, whereby, a class of model bugs can be fixed with a particular set of strategies or tools. Towards such an end, in this thesis, we take a step towards a model bug categorization for the supervised learning setting.

A Categorization of Model Bugs for Supervised Learning. The dominant learning paradigm in supervised learning is empirical risk minimization (ERM). Given a model class

(e.g. deep neural networks (DNNs)), ERM involves minimizing a loss function, evaluated on a distribution that places a point mass on each training sample, in order to obtain a single model, from the pre-specified model class, that should perform well on unseen examples. Equation 1.1 shows the components of ERM. There are often two distinct phases in the ML life cycle: 1) the first phase where ERM is performed and a model is obtained, 2) a second phase where the already trained model is now used to forecast on new inputs. Based on these two phases, we partition model bugs into two: the **training/learning** phase model bugs, and the **inference/prediction** phase model bugs.

$$\text{Learning: } \underbrace{\arg\min_{\theta}}_{\textbf{Model}} \sum_{i=1}^{n} \ell(\overbrace{(x_{\text{train}}, y_{\text{train}})}^{\textbf{Training Data}}; \theta). \tag{1.1}$$

$$\text{Prediction: } y_{\text{test}} = f_{\theta}(\overbrace{x_{\text{test}}}^{\textbf{Test-data}}) \tag{1.2}$$

In this thesis we focus primarily on bugs that manifest themselves at prediction/inference time. Significant focus has gone into developing strategies for ensuring that ML models can be easily optimized. Note that even though we focus on model bugs that occur at inference time, the cause of the bug can be due to components derived from training such as the training set, labels, or other choices made prior to the learning phase. Concretely, we focus on two kinds of bugs: 1) a spurious correlation bug, and 2) a noisy training label bug.

Spurious Correlation. At a high level, a spurious training signal is a feature that, based on a domain expert, is unrelated to the task at hand. For example, the presence of a hospital tag in a radiograph is unrelated to any diagnosis that is to be made from the radiograph. Machine learning models, especially DNNs trained on crowd-sourced data, very easily latch on to unrelated signals in the training set, that happen to correlate with the label of interest, to solve a prediction task. For example, Leino and Fredrikson [2020] showed that a model learned to associate "Tony Blair" with all images that had a pink background because there was a single image of Tony Blair with a pink background in the training set. Contemporary empirical evidence indicates that DNNs can learn a spurious association from as little as 3 training examples [Singla and Feizi, 2021a, Yang and Chaudhuri, 2022]. Perhaps

even more concerning, evidence suggests that DNNs solve prediction tasks by first learning simple concepts (or correlations) in the data before potentially memorizing more difficult examples [Feldman, 2020, Pezeshki et al., 2021]. The simplicity bias [Shah et al., 2020] of DNN learning behavior suggests that when a simple, easy-to-learn, but spurious, signal exists in a dataset, the DNN **will** take advantage of it.

Why do spurious signals lead to model bugs? The link between a spurious signal and its downstream effect of inducing a model bug might be obvious; however, we discuss it first, since it will inform our proposed definition. Consider a scenario where the training data for an object classification task has a spurious signal like a watermark. Specifically, let's say all the images of horses were obtained from a platform that includes a company watermark on each image from the platform. A DNN model trained on this dataset learns the shortcut that any image with a watermark is a horse. On the training data of this task, there is a high correlation between the presence of a watermark and the label horse. However, for images of horses collected from a different website, where a watermark is not present on the horse images, the model that as learned to associate image watermark to horses fails. The watermark is an 'unreliable' signal for the presence of horses.

What is a spurious signal? A spurious signal is a feature/concept whose correlation with the label/output changes, *widely*, from one domain to another [Arjovsky et al., 2019]. In Chapter 2, we make the definition more precise. The key insight that the definition relies on is the notion that comparing a signal's strength of association with the label across different domains will reveal unstable correlations. Returning to the Pneumonia classification from chest x-ray example, consider data from two different hospitals. We are interested in determining whether the presence of a hospital tag on a radiograph is a spurious signal for the Pneumonia classification task. Let's assume that we gather data from a first hospital and find that the correlation between the image tag concept/feature and the Pneumonia label is 0.8. However, for data from a second hospital we find that the correlation is instead 0.0, because none of the x-ray images from the second hospital contain image tags. Consequently, the strength of the association between the image tag and the Pneumonia label varies widely from one hospital to another; hence, reliance on the tag concept will not enable high performance across hospitals.

The proposed definition is not a panacea; however, this definition is amenable operationalization. As we'll see in Chapter 5, the proposed definition immediately suggests an

approach for identifying and correcting a model's reliance on spurious signals: among the correlations that a model is relying on, find the 'unstable' ones, and reduce the model's dependence on such correlations.

With the definition of a spurious correlation now established, a spurious correlation bug is simply a model bug for which the cause of the bug is a spurious correlation signal.

Spurious Correlation Bug: A spurious correlation bug is a model bug that is induced by a spurious signal.

Knowledge of the Spurious Signal. In practice, when a model is being tested, it is not often clear whether the model is relying on a spurious signal as primary basis for its output. In addition, even if a model is suspected to be relying on a spurious signal, the exact form that the signal takes is usually unclear. Case in point, Ilanchezian et al. [2021] found that DNNs were able to predict a patient's gender, with high accuracy, from retinal fundus images. Similarly, Gichoya et al. [2022] found that DNNs were also able to accurately predict a patient's race from the chest x-ray. In both these cases, the specific mechanism and features upon which the model relies for accurate gender and race prediction remains unclear. To highlight the challenge that arise with detecting a model's reliance on spurious signals in practice, we draw a distinction between two settings: when the spurious signal is known and when it isn't.

Separating settings where the spurious signal is known from those where it isn't is crucial. Post hoc explanation methods are often assessed assuming that the spurious signal is known. However, in practice, it is often the case that the potential scope of spurious signals is vast, i.e. image tag, low frequency signals etc, so a search over the space of potential spurious signals often needed. We will find that when the spurious signal is unknown, a large class of post hoc explanation methods are ineffective.

Noisy Training labels. The second type of model bug that we consider is due to errors in the training labels. A random sampling of a thousand samples from the Google Emotions [Demszky et al., 2020] dataset had about 30 percent errors in the labels of the training samples examined [Chen, 2022]. Similarly, an analysis of the MS COCO dataset found that up to 37 percent of the total annotations had errors, which amounted to about 273,834 errors found [Murdoch, 2022]. Evidence continues to accrue that the presence of label error in a crowd-source dataset is the norm.

In Chapter 6, we find that errors in the training labels often have disproportionate impact

on the minority groups in the dataset. In addition, these errors often impact downstream disparity metrics like group calibration, false positive rate, false negative rate, and others that are usually monitored as part of a 'fairness' audit.

Errors in training labels for a particular group in the training data limit the ability of the model to learn the right associations for that group, and consequently induce the trained model to make more errors on new inputs that belong to that group.

Noisy Training Label Bug: A noisy training label bug is a model bug that is induced due to the presence of a wrong labels in the training set.

Errors due to noisy training labels manifest themselves as regular model bugs whose cause is due to a noisy data collection process. Consequently, identifying training points whose labels might be wrong, and correcting these labels will stem noisy training label bugs.

The two key types of bugs we consider in this work are the noisy training label and the spurious correlation bug. However, the proposed framework is able to capture model bugs beyond these two. In Chapter 2, we further delve into the framework and present instantiations that cover other types of model bugs.

1.2 Part I: Empirical Assessment of Current Methods

In the first part of the thesis, we assess whether current post hoc explanation methods can detect model bugs. We first address detection, since if a method cannot detect a bug, it holds no hope for suggesting a fix.

Classes of Post hoc explanations considered. We consider three types of post hoc model explanations: feature attribution, concept activation methods, and training point ranking. These approaches are different in scope, and the kind of information they convey to the end-user. Feature attributions assign a relevance score for each dimension of an input towards an output. Concept activation [Bau et al., 2017, Kim et al., 2018] approaches measure the dependence of a DNN's prediction on user-defined features. Training point ranking approaches rank all the training samples in order of importance/influence on the loss (or prediction) of a test example.

Experimental Design & New Suite of Metrics. We provide an end-to-end experimental design for assessing the effectiveness of an explanation method. Towards this end, we use carefully craft semi-synthetic datasets that make it easy to manipulate the signal

that a model trained on the dataset will rely on. Consequently, we can ascertain what the ground-truth explanation for a model on a particular input is ahead of time. Using these datasets, we construct two kinds of models: *normal* & *defective* models. A normal model is one that does not contain a bug, while a defective model has a specific bug that has been manually inserted using the semi-synthetic dataset. We then define various scores to ascertain that the models are indeed defective, such as a model spurious score. If an explanation method's output cannot be used to distinguish between a *normal* model and a *defective* model, then that explanation method cannot be effective for detecting the bug. We concretize the proposed comparison with a series of metrics, and then perform a comprehensive assessment of methods belonging to the three aforementioned explanation categories.

Spurious Correlation. When the spurious signal is known, we find that the feature attribution methods tested, and the concept activation importance approach are able to detect visible spurious signals like an image tag and distinctive stripped patterns. However, these approaches are ineffective for detecting a non-visible spurious signal like background blur. Perhaps even more concerning, we find that feature attribution methods are susceptible to erroneously indicating dependence on a spurious signal when the model does not rely on the signal as basis for its output decision. Similarly, training point ranking methods are effective for detecting spurious signals when the the signal is known a priori, and if the test input used by the practitioner also contains the spurious signal. Ultimately, a comprehensive assessment of post hoc explanation methods point to their inability to effectively help an end-user detect that a model is relying on a spurious signal.

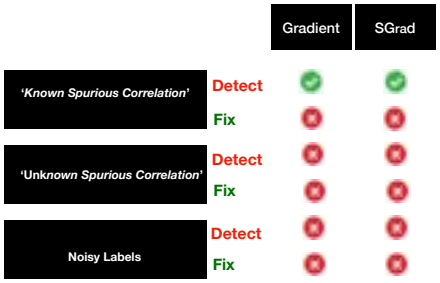

Figure 1-3: A summary table showing three various model bugs and two feature attribution methods. We that the two feature attribution approaches are only effective for detecting known spurious signals.

Noisy Training Labels. Based on the proposed experimental design, testing whether an explanation method might be effective, in the hands of an end-user, for detecting a mislabelled training sample is simple. First, train two copies of a model, one in which the

training sample has the correct label, and a second where the training sample has the wrong label. For the same sample, now compare the explanation for the two model settings. If the explanations are similar, then the explanation method in question is unlikely to be effective for detecting that an input is mislabelled. Indeed, we find that feature attribution methods output explanations that are similar under the normal and mislabelled model setting, which indicates that these approaches are not effective for fixing label error in the training set (See Figure 1-4). On the other hand, training point ranking approaches do indeed rank mislabelled examples highly, so that they can be more carefully examined.

User Study. Beyond tests in simulation settings, it is also important to measure how effective a model debugging tool will be in the hands of an end-user. Often, simulation tests provide ideal settings that might not match real-world usage. Towards such an end, we conduct user studies with individuals familiar with machine learning and provide them with a few methods to use for debugging a specific task. In a first study, we find that users rely, primarily, on the model predictions to ascertain that a model is defective, even in the presence of post hoc model explanations. In a second blinded study, we specifically focus on end-users ability to detect whether a model is relying on a spurious signal. We find that when participants are not provided with prior knowledge of the spurious signal, none of the methods tested are effective. More concerning, even when the participants had prior knowledge of the spurious signal, we find evidence that only the concept activation approach, for visible spurious signals, is effective.

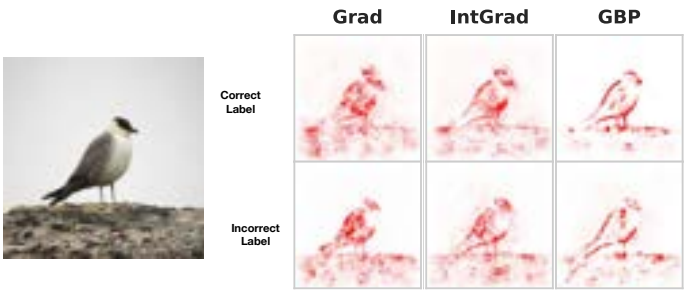

Figure 1-4: Three different kinds of feature attributions for a DNN model trained to perform animal classification. On the same input, the attribution for a setting where the input was trained with the correct label, and another where the input was trained without the correct label is shown.

1.3 Part II: New Tools for model debugging

In the second part of the thesis, we introduce two new tools for model debugging. The empirical assessment of current approaches produced somber results: current approaches struggle to detect a model's reliance on spurious training signals. While training point ranking approaches are able to identify mislabelled training points, it remains unclear how to actually fix the issue. Inspired by these challenges, in the second part of the thesis, we introduce new tools that directly address these concerns.

Model Guiding. The first approach we introduced is termed *model guiding*. The key goal of model guiding is to enable interaction between a task expert—an individual like a radiologist with past experience on the task—and a trained model. The goal of this interaction is to incorporate feedback both on the training labels and the model's 'explanation', from the task expert, into the model. The feedback from the task expert serves as a prior that can further help constrain the model and align it with the task expert. Model guiding requires a small, carefully annotated, auxiliary dataset—the audit set—that consists of samples and labels similar to the training set, but drawn from a different domain. On this dataset, the task expert carefully checks the sample labels, and provides feature importance annotations

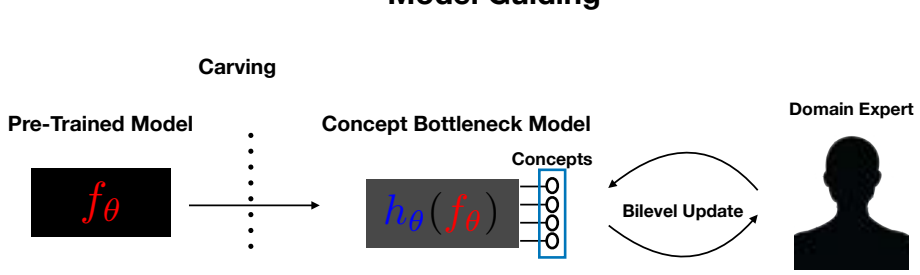

Figure 1-5: **Model guiding schematic**.

1. update the parameters of an already trained model;

2. obtain a **new set of labels** on the entire training dataset that allow the model with updated parameters to match the domain expert's provided labels on the audit dataset; and,

3. obtain a **new set of feature importance estimates** on the training dataset that allow the model with updated parameters to also match the feature annotation provide by the task expert on the audit set.

Model guiding formulates the update to model parameters and labels as a bilevel optimization problem. In a bilevel optimization problem, one of the constraints of the main objective also requires solving an optimization problem. Solving the bilevel optimization problem then results in an updated model as well as new labels for the entire training set. To detect mislabelled examples, the new training labels are compared to the old ones and those that differ substantially are likely mislabelled. Critically, the new labels are also the proposed fix when a bug is identified. Similarly, to detect inputs for which a model is relying on a spurious signal, we simply compare the feature importance estimates of the initial model to those obtained from the updated model. Given a pre-specified tolerance parameter, the inputs for which the updated feature estimates differ, by more than the tolerance parameter, from the feature estimates for the initial model are those that the initial model was relying on spurious signal.

Applications of Model Guiding. We compare model guiding's performance, on a mislabelled example detection task, to other noisy label detection approaches, across several datasets. We find that the approach matches the performance of state-of-the-art approaches across all the tasks tested. A second key capability of model guiding is detecting and correcting reliance on spurious signals in the training set. We compare model guiding's detection capabilities to recent state-of-the-art methods that train a simple classifier, ERM based, to detect under performing groups. Across the datasets considered, model guiding significantly outperforms these approaches. In fixing a model's reliance on spurious signal, we compare to group-distributionally robust [Sagawa et al., 2019] approaches that rely on a priori knowledge of what the spurious signal is. Even without knowledge of the spurious signal, model guiding outperforms these approaches for learning models that are robust to spurious signals in the training set.

Correcting 'Fairness' Violations with Influence Functions. The second debugging approach we propose is to use influence functions [Koh and Liang, 2017] to help identify inputs that are 'causing' a model to have disparate performance across subgroups of the data. Specifically, we modify the influence functions approach to help prioritize mislabelled inputs that have a high effect on a model's group-based disparity metrics like equal odds [Hardt et al., 2016], group calibration [Pleiss et al., 2017], and false positive rate [Barocas et al., 2019, Garg et al., 2020].

As an example of a use-case for this approach: consider an ML engineer who has trained

a model, and found that the model is less well-calibrated for group A in the dataset compared to group B. This engineer might ask the question, which of the training points can I relabel to improve the calibration of my model for group A? More generally, if we seek to better reduce disparate performance for an ML model, can we devise a prioritization scheme to identify the most critical examples to relabel?

Training point ranking provides a tractable way to estimate the effect of perturbing a training point's label on a model's group disparity metrics. We empirically assess the proposed approach on a variety of datasets and find a 10-40% improvement, compared to alternative approaches that focus solely on model's loss, in identifying training inputs that improve a model's disparity metric.

Summary. The two approaches that we present here: 1) model guiding, and 2) modified influence functions directly address the challenge of detecting and fixing model bugs caused by spurious training signals, and mislabelled training labels.

1.4 Thesis Bibliography

The work in this thesis is derived from a series of research papers that the dissertation writer co-authored. In this section, we provide the reader with a brief overview of each chapter along with the research papers from which the chapter is derived.

In Chapter 2 we discuss the model debugging categorization in greater detail, and make the case that model understanding is equivalent to model debugging. An initial version of the discussion appeared the experiments and backgrounds section of the paper:

- ***Debugging Tests for Model Explanations***, **Julius Adebayo**, Michael Muelly, Ilaria Liccardi, Been Kim, **Advances in Neural Information Processing Systems (NeuRIPs)**, 2020.

In Chapter 3 we describe the three kinds of post hoc explanation methods that we assess in this work. This material is not new, and is mostly introduced to familiarize the reader with the state-of-the-art on post hoc explanation methods. The content of the chapter is derived from:

- ***Explaining ML Predictions: State-of-the-art, Challenges, Opportunities***, Prof. Himabindu Lakkaraju, **Julius Adebayo**, and Prof. Sameer Singh, **NeurIPS 2020 Tutorial**, 2020.

In Chapter 4, we describe the results of the empirical assessment of various post hoc explanation methods. The material is derived from:

- ***Assessing the trustworthiness of saliency maps for localizing abnormalities in medical imaging***, Nishanth Arun, Nathan Gaw, Praveer Singh, Ken Chang, Mehak Aggarwal, Bryan Chen, Katharina Hoebel, Sharut Gupta, Mishka Gidwani, **Julius Adebayo**, Matthew Li, Jayashree Kalpathy-Cramer, **Radiology: Artificial Intelligence**, 3 (6): 2021.

- ***Debugging Tests for Model Explanations***, **Julius Adebayo**, Michael Muelly, Ilaria Liccardi, Been Kim, **Advances in Neural Information Processing Systems (NeuRIPs)**, 2020..

- ***Post hoc explanations may be ineffective for detecting unknown spurious signals***, **Julius Adebayo**, Michael Muelly, Hal Abelson, Been Kim, **International Conference on Learning Representations (ICLR),** 2022.

In Chapter 5 we present the model guiding formulation along with demonstration of the approach's effectiveness for detecting noisy training labels and spurious signals. The discussion is based on:

- ***Guiding Models with Expert Annotations***, **Julius Adebayo**, and Hal Abelson, Presented at XAI Debugging Workshop, Neurips 2021, & Princeton Symposium on Interpretability in Machine Learning; Conference version to be submitted.

In Chapter 6 we discuss the second model debuging tool for identifying mislabelled training points that have the most effect on a downstream disparity metric of interest. The discussion is based on:

- ***From Label Quality to Model Fairness: Quantifying the sensitivity of model disparity metrics to label errors***; **Julius Adebayo**, Melissa Hall, Bowen Yu, & Bobbie Chern. Submitted & Under Review.

In Chapter 7 we conclude the thesis.

Chapter 2

Model Debugging

2.1 Overview

In this section, we motivate why model debugging operationalizes model understanding. We then move to discuss various definitions of model bugs. We categorize model bugs into two: those that occur during the learning phase, and those that occur after a model has been trained—the prediction phase. In this work, we focus primarily on the later. With a general overview of model bugs established, we then discuss the two key model bugs that we focus on in this thesis: spurious correlation bugs, and noisy training label bugs. We end the chapter with a discussion of related work.

2.2 Model Debugging as Model Understanding

An oft-lamented issue in research on explainable machine learning is the claim that there is no clear definition of explainability. Doshi-Velez et al. [2017] take a stab with the claim that, to interpret means "to present in understandable terms to a human." Another definition operationalizes ML explainability as the ability to better simulate the output of an ML model on new examples [Lipton, 2018]. Regardless of a definition, the goal of explainability is to improve an end-user's—human interacting with a model—understanding of a model. In this thesis, we take the view that debugging and understanding are intertwined.

What does it mean to understand a system? In a now celebrated paper titled "Could a Neuroscientist Understand a Microprocessor", Jonas and Kording [2017] advanced the thesis that an individual understands a system if the person can:

repair a broken implementation of the system.

Stated differently, if one can both detect that a system is broken, diagnose the cause of the malfunction, and then repair the system, then one understands the system.

In their paper, Jonas and Kording [2017] applied state-of-the-art analysis techniques, in neuroscience, to input-output data from a well understood system: a micro-processor. Their hypothesis was that analysis tools in neuroscience were being used to make claims about the functioning of various components of the brain using input-output data. However, the insights obtained from such analyses were often difficult to assess because the true function of the different parts of the brain is not known a priori, and the interventional experiments needed to verify the results of an analysis are typically not obvious. To circumvent the challenge, Jonas and Kording [2017] turn to the tactic of applying an analyses tool to a well-understood toy setting. If an analysis tool can recover the behavior and functioning of a well-understood system, then it might hold hope for recovering the behavior of an unknown complex system.

Why does 'repairing' a system imply 'understanding' the system. When a complex system fails, it is either because:

1. a component of the system has failed or
2. an interconnection between components of the system failed.

This immediately suggests a way to fix/repair a broken system:

1. examine each component of the system to ensure that they are working as expected, or
2. examine the interconnection between components to ensure that these are still functioning.

The requirements above necessitate:

1. knowing the constituent components of a complex system as well as the normal functioning behavior, and
2. knowing how the components of a complex system are connected.

Consequently, to debug a system, the operator needs to know the proper functioning behavior of each system component and how these components combine to form the larger

system. We argue that this level of knowledge constitutes understanding the system. Because of this simple underlying reasoning, we claim that debugging operationalizes understanding.

The claim that the ability to debug a system is equivalent to understanding the system is not new. Jonas and Kording [2017]'s definition was inspired by an earlier paper by Lazebnik [2002] titled, "Can a biologist fix a radio?", where the author argued for an engineering approach to conducting research in molecular biology. In computer science and software engineering, the task of fixing a broken implementation of a software system is known as debugging. Much has been written on how to go about debugging a software system, and the critical link between understanding the software system and the ability required to debug the system. Inspired by Jonas and Kording [2017], and the critical importance that debugging plays in traditional software, in this thesis, we make debugging the center piece of our focus.

An objection? A potential objection to the claim that model understanding is equivalent to model debugging might be that there are strategies that can be used to repair a system but that donot require understanding. For example, turning a device off and on can sometimes return the system to normal behavior—a fix that does not require understanding the system. First, we contend that the on/off strategy is not reliable, and will often fail to stem breakdowns in the future. Second, understanding comes in different forms. Famously, Marr [1982] posited that understanding of a system can be achieved at various levels: 1) the algorithmic, 2) the computational, and 3) the implementational. Even at each level, understanding lies on a spectrum. Overall, we see model debugging as a way to more concretely verify that an individual understands a model.

In this thesis, we focus on debugging machine learning models after they have been trained. Translating the challenge of debugging to the machine learning setting results in different categories of bugs—a notion we will make more precise shortly. A first category of bug are those due to **software implementation errors**. These kind of bugs are the standard type of bugs encountered in traditional software engineering like: type mismatch errors, out-of-memory errors, performance errors, or even errors that stem from incorrect implementation of an algorithm. This thesis does not focus on standard software implementation bugs since it is more well-studied, and there exists tools to better address them [Naik and Tripathy, 2008].

A second kind of bug are learning phase model bugs. These are the kind of bugs that 'prevent' a model from being trained, i.e., the training loss is not successfully minimized.

As an example, a learning phase bug would prevent a simple LeNet model from getting above 70 percent training accuracy on the MNIST digit recognition dataset. Despite their importance, this thesis does not focus on these kind of model bugs. Bugs that occur in the learning phase [Schneider et al., 2021, Tobin et al., 2019] essentially deter any concrete model from being learned in the first place, so a significant amount of research in optimization, model architecture development, and other research areas have directly contributed to stemming learning phase bugs. For example, innovations like the ReLU non-linearity [Nair and Hinton, 2010], residual connections [He et al., 2016], BatchNorm [Ioffe and Szegedy, 2015], LayerNorm [Ba et al., 2016], and the Adam optimizer [Kingma and Ba, 2014] were all discovered with the goal of successfully sidestepping learning phase model bugs. To a large extent, the success of the deep learning paradigm can be credited to the tremendous amount of effort that went into discovering new insights that make DNNs more easily trainable, i.e., sidestep learning phase model bugs.

In this thesis we focus primarily on a third kind of model bug: prediction/inference model bugs—a notion we will make more concrete in the next section. We do this because these kind of model bugs are less well-understood, and no clear recipe currently exists to address them. Increasingly, tools that claim to explain an already trained model are being used for model debugging with mixed success [Chen et al., 2021]. In the next section, we discuss our proposed categorization of model bugs in greater detail.

2.3 A Categorization of Model Bugs in Supervised Learning

A model bug is a 'contamination' in the learning, and/or prediction pipeline that causes a model to produce incorrect predictions. We will concretize this definition shortly; however, we note that, in this thesis, we focus primarily on the supervised learning paradigm. Specifically, given a model class (e.g. deep neural networks (DNNs)), empirical risk minimization (ERM) involves minimizing a loss function, evaluated on a distribution that places a point mass on each training sample, in order to obtain a single model, from the pre-specified model class, that should perform well on unseen examples.

ERM Notation. In standard supervised learning, we are usually given input-label pairs, $\{(x_i, y_i)\}_{i=1}^n$, where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, and seek to learn a model—a function— $f_\theta : \mathcal{X} \to \mathcal{Y}$. The model, f_θ, is then used to provide predictions on test examples, $x_{\text{test}} \in \mathcal{X}$, as $y_{\text{test}} =$

$f_\theta(x_{\text{test}})$.

ERM & The two phases of Supervised ML. The dominant learning paradigm in supervised learning is empirical risk minimization (ERM). Equation 2.1 shows the components of ERM. In general, the supervised ML process can be partitioned into: 1) the learning (or training) phase, and 2) the prediction phase.

The learning phase corresponds to solving the ERM problem to obtain a trained model. Once the training objective, loss, has been successfully minimized, we obtain a model, f_θ that can then be used to predict labels for new instances—the prediction phase.

$$\text{Learning Phase: } \underbrace{\arg\min_{\theta}}_{\textbf{Model}} \frac{1}{n}\sum_{i=1}^{n} \ell(\overbrace{(x_{\text{train}}, y_{\text{train}})}^{\textbf{Training Data}}; \theta). \tag{2.1}$$

$$\text{Prediction Phase: } y_{\text{test}} = f_\theta(\overbrace{x_{\text{test}}}^{\textbf{Test-data}}) \tag{2.2}$$

We classify model bugs into two: learning phase model bugs, and prediction phase model bugs based on which stage of the process the bug occurs in. Before we go into further detail about model bugs, and give a more concrete definition, we first walk through the supervised ML pipeline to categorize its constituent components as well as which phase these components affect.

Components of a typical supervised ML Pipeline. To successfully detect and fix mistakes that occur in a complex system, the core constituent parts of the system and their associated function need to be known—even if at a high level. This is because when an error occurs, the cause of the error will be attributed to one or more of these constituent parts. Turning to the ML setting, we also need to take stock of the constituent parts of the pipeline. Using the DNN training process as an example, the key constituent parts of the standard supervised learning process are:

- (Learning & Prediction Phase) **The model & its initialization**: this component includes the type of function class selected, e.g. DNN, linear model, decision tree etc.

For a model class like DNN, it further includes components like dropout, choice of activation function, batchnorm, and other details like whether the architecture is a transformer or a convolutional neural network (CNN). In general, all the choices that go into specifying the model and its initialization fall under this component;

- (Learning Phase) **Optimizer**: this broadly includes the choice of the optimization algorithm, the learning rate schedule, and the hyper-parameters used by the optimizer;
- (Learning & Prediction Phase) **Data**: training data, validation data, test data, as well as the data augmentation and normalization strategies; and
- (Learning Phase) **Loss function & regularizer**: this is the ERM objective that is minimized. Common choices include the cross-entropy loss, squared loss, and hinge loss.

With the components of a supervised ML pipeline now denominated, we will now define a model bug.

A more concrete definition of a Model Bug. We say a model bug has occurred when a model's output, on a sample, differs from the 'true' label of that sample by more than a user-defined tolerance.

Model Bug: Given a tolerance parameter, $\tau \geq 0$, a sample, x_k, whose true label is y_k, a *model bug* occurs when: $\|f_\theta(x_k) - y_k\|_p \geq \tau$.

In the model bug definition, the condition that triggers a bug (or error) is the relationship: $\|f_\theta(x_k) - y_k\|_p \geq \tau$. The tolerance, τ, and the choice of the norm, $\|.\|_p$, are user provided parameters. The choice of the tolerance and norm allows the user to specify how much wiggle room can tolerated before a bug is triggered.

On the basis of these distinct phases of supervised ML, we can classify bugs into:

1. **Learning Phase Model Bugs**: these are model bugs that occur during the learning phase. Overall, these bugs usually manifest themselves by increasing the difficulty of learning an ML model that has low error.
2. **Prediction/Inference Phase Model Bugs**: These are bugs that manifest themselves after a model has been trained.

Variations on the definition of a Model Bug. We defined a model bug in terms of a single example, and the difference between the model output and its prediction. However, the definition can also be extended to sets of examples and other functions of a model's parameters. Let, S_m, be a set of m inputs written in matrix form as $X_m \in \mathbb{R}^{m \times d}$. The matrix, X_m, has each sample in the set S_m. Let the vector of true labels be $\mathbf{y_m} \in \mathbb{R}^m$. A version of the model bug definition for a the set of inputs, S_m is then:

Model Bug (Set Definition) : Given a tolerance parameter, $\tau \geq 0$, a matrix, X_m, whose true label vector is $\mathbf{y_m}$, a *model bug* occurs when:
$\|f_\theta(X_m) - \mathbf{y_m}\|_p \geq \tau$.

The proposed definition of a model bug is flexible, and can be used to even capture notions of "fairness" or bias. For example, if the set S_m were to be the a selection of houses that belong to a zipcode that has a high concentration of a particular ethnic group, then the definition above corresponds to a fairness violation of the model.

2.3.1 Prediction Phase Model Bugs

In this subsection, we discuss the two primary prediction phase bugs that we focus on in this thesis: spurious correlation bug and the noisy training label bug.

Bug Class	Specific Examples	Formalization
Prediction Phase	Spurious Correlation	$\arg\min_{\theta} L(X_{\text{spurious artifact}}, Y_{\text{train}}; \theta)$
Prediction Phase	Labelling Errors	$\arg\min_{\theta} L(X_{\text{train}}, Y_{\text{wrong label}}; \theta)$
Prediction Phase	Initialized Weights	$f_{\theta_{\text{init}}}(x_{\text{test}})$
Prediction Phase	Frozen Layers	$\arg\min_{\theta_{\text{frozen}}} L(X_{\text{train}}, Y_{\text{wrong label}}; \theta)$
Prediction Phase	Out of Distribution (OOD)	$f_\theta(x_{\text{OOD}})$
Prediction Phase	Pre-Processing Mismatch (PM)	$f_\theta(x_{\text{PM}})$
Prediction Phase	Test Adversarial Examples (Adv)	$f_\theta(x_{\text{Adv}})$

Table 2.1: **Table of different bugs.** We show different examples of bugs under the different contamination classes: data, model and test-time. We also formalize these bugs for the traditional supervised learning setting.

What is a spurious signal? A spurious signal is a feature/concept whose correlation with the label/output changes, *widely*, from one domain to another [Arjovsky et al., 2019]. To make this intuition more concrete, we specify some notation.

Notation. Let's assume that we are given data from two domains: $\mathcal{D}_1$: $\{(x_{i,d_1}, y_{i,d_1})\}_{i=1}^n$, and $\mathcal{D}_2$: $\{(x_{i,d_2}, y_{i,d_2})\}_{i=1}^m$, consisting of n, and m samples respectively. As usual, the goal here is to learn a classifier, f_θ, that maps from inputs, $x_i \in \mathbb{R}^d$, to a labels, $y_i \in \mathbb{R}$. We assume that this classifier can be written as: $f_\theta(x_i) = h(g(x_i))$, where the function, $g : x_i \to z_i$, is an encoder that maps an input to a latent vector, $z_i \in \mathbb{R}^k$. The function, h, is then a simple classifier on top of the latent code, and maps the latent code to a label. To ground the setup, we can think of the task as detecting the presence of pneumonia in a chest radiograph. The first domain, $\mathcal{D}_1$, represents data from hospital 1, while the second domain, $\mathcal{D}_2$, represents data from hospital 2. Each component of the latent code, z_i, represents a higher level concept. For example, the first dimension can correspond to a concept that indicates whether a hospital tag is present in the radiograph. We'll term the matrix, $\mathbf{Z}^1 \in \mathbb{R}^{n \times k}$, to be the matrix of the latent code for all inputs in the first domain $\mathcal{D}_1$, while $\mathbf{Z}^2 \in \mathbb{R}^{m \times k}$ will be the matrix of the latent code for all inputs in the second domain. Each column of a latent code matrix, for a domain, is a feature/concept vector. We can also index each column to refer to a particular concept vector, so z_j^2 is the jth column from the matrix $\mathbf{Z}^2$. With critical notation now out of the way, we can now define a spurious signal.

Spurious Signal: Given a tolerance parameter, $\tau \geq 0$, classifier $f_1 : x_{i,d_1} \to y_{i,d_1}$, trained on data from the first domain, $\mathcal{D}_1$, another classifier $f_2 : x_{i,d_2} \to y_{i,d_2}$ trained on data from the second domain, $\mathcal{D}_2$, so that $\mathbf{Z}^1 \in \mathbb{R}^{m \times k}$ is the latent code for inputs in the first domain, and $\mathbf{Z}^2 \in \mathbb{R}^{n \times k}$ is the latent code for inputs in the second domain, a concept z_j is spurious if:

$$\|\text{corr}\left(z_j^1, \mathbf{y}_{d_1}\right) - \text{corr}\left(z_j^2, \mathbf{y}_{d_2}\right)\|_p \geq \tau.$$

In the definition above, $\mathbf{y}_{d_1}$ and $\mathbf{y}_{d_2}$ are the vectors of the labels from $\mathcal{D}_1$, and $\mathcal{D}_2$ respectfully. The function, corr, refers to correlation, so corr $\left(Z_j^1, \mathbf{y}_{d_1}\right)$ is the correlation coefficient between the jth latent concept for the classifier, f_1, learnt on domain 1 and its labels, $\mathbf{y}_{d_1}$.

The spurious signal definition compares the strength of a signal's correlation with the label in two domains. If a concept/features's association with the label is stable across domains then it will enable high performance; however, an unstable correlation indicates that a model will underperform in a domain where the strength of that feature/concept's correlation with the label is not high.

The proposed definition of a spurious signal relies on the strength of associations across

domains. There are other alternative definitions that one could pose. For example, another definition of a spurious concept would be a concept that is not 'causally' related to the outcome. However, such a definition is difficult to operationalize for multiple reasons. First, predictive models trained via ERM typically don't capture any notion of causality or the underlying data generating mechanism, so it is unclear why such models should be expected to capture causal concepts. If the underlying causal generative model of the task at hand is known, then a causal definition might be feasible. However, we focus on settings where such causal knowledge is unknown. Another definition of a spurious signal might require that the high level concept linearly separate the input data. The proposed definition is a strictly weaker version of that requirement. Ultimately, the proposed definition is not a panacea; however, we have chosen this definition in this thesis because it is amenable operationalization.

Spurious Correlation Bug: A spurious correlation bug is a model bug that is induced by a spurious signal.

With the definition of a spurious correlation now established, a spurious correlation bug is simply a model bug for which the cause of the bug is a spurious correlation signal.

Knowledge of the Spurious Signal. In practice, when a model is being tested, it is not often clear whether the model is relying on a spurious signal as primary basis for its output. In addition, even if a model is suspected to be relying on spurious signal, the exact form that the signal takes is usually unclear.

1. **Known Spurious Signal**: corresponds to the setting where the potential spurious signal is known ahead of time by the individual seeking to assess the reliability of a trained model.

2. **Unknown Spurious Signal**: corresponds to the setting where the, specific, potential spurious bug is not know at test-time to the individual assessing the reliability of the trained model.

Noisy Training labels. The second key type of model bug that we consider is due to errors in the training labels. Errors in training labels for a particular group in the training data limit the ability of the model to learn the right associations for that group, and consequently

induces the trained model to make more errors on new inputs that belong to the group with high label error.

Noisy Training Label Bug: A noisy training label bug is a model bug that is induced due to the presence of a wrong labels in the training set.

Errors due to noisy training labels manifest themselves as regular model bugs whose cause is due to a noisy data collection process. Consequently, identifying training points whose labels might be wrong, and correcting these labels will stem noisy training label bugs.

2.4 Related Work

The goal of debugging machine learning models has been a long-standing one in the machine learning community [Kulesza et al., 2015, Sculley et al., 2015, Tobin et al., 2019], so it'll be impossible to completely cover all relevant literature. For example, one could argue that traditional regression diagnostics [Fox, 2019] like the Cook's distance [Cook and Weisberg, 1982] are aimed at assessing model reliability. However, in this section, we focus on those that are directly relevant to the discussion in this Chapter.

The model bug classification discussed in this chapter takes its naming from Schneider et al. [2021]'s work, where they introduce a tool, cockpit, for identifying and fixing bugs in the learning phase. Different from their discussion, we focus on the prediction phase. In addition, other work have also introduced a taxonomy of ways in which a model or an AI agent can be derailed [Amodei et al., 2016]. In practice, the niche of machine learning engineering focused on testing/auditing an ML model is now known as ML monitoring [Bacelar, 2021]. As at the writing of this dissertation, the ML monitoring field is still in its infancy, so model debugging guidelines are still emerging.

More recently, Cadamuro et al. [2016] and Bhadra and Hein [2015] tackled the challenge of debugging least squares models. Similarly, Lourenço et al. [2019] studied the problem of detecting mislabelled examples. Augenstein et al. [2019] studied the problem of performing model debugging in a federated learning setting, and gave a comprehensive list of model bugs that are often encountered in practice.

Chapter 3

Post hoc Explanation Methods

3.1 Overview

In the first part of this thesis we evaluate three kinds of post hoc model explanation methods to see if they are effective for model debugging. In this chapter, we provide a discussion of each kind of explanation considered. First we discuss feature attribution methods in Section 3.2, and then move to concept activation approaches. The third kind of method we consider is an approach that ranks the training points by influence on the loss on a test example: influence functions. We present a detailed but non-rigorous derivation of the influence functions approach since we'll modify the approach in Chapter 6 to help identify mislabelled inputs that cause a fairness violation. To further ground the reader, we provide a brief overview of how each of these methods is actually used in practice.

3.2 Feature Attribution Methods

In this section, we will provide an overview of the feature attribution methods that we consider in this work. First, we set the stage with some notation. We consider classification problems, i.e, settings where the task is to learn a mapping, $f_\theta : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} \in \mathbb{R}^d$ is the feature space, and $\mathcal{Y} \in \mathbb{R}^c$ is the output space. Here, c is the number of output classes. We assume that we are given a training set: $\{(x_i, y_i)\}_{i=1}^n$ of n examples. Here, we term the tuple (x_i, y_i), z_i, which means the n training points can be written as: $\{z_i\}_{i=1}^n$. Throughout this section, we will only consider learning via empirical risk minimization (ERM), which corresponds to: $\hat{\theta} := \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_i^n \ell(z_i, \theta)$. Note that we write $f_{\theta,i}$ to indicate the ith

dimension of the output. We will also assume that $f_{\theta,i}$ is the output of the model without the softmax normalization.

Feature Attributions: In this thesis, we define a feature attribution method to be a method that takes as input: a trained model, a single input, and the model's output on that instance to produce a **scalar relevance score** for each dimension of input. The scalar relevance score can then be interpreted as the importance of that input dimension towards the model's output decision on the specific input.

Figure 2: Alternative

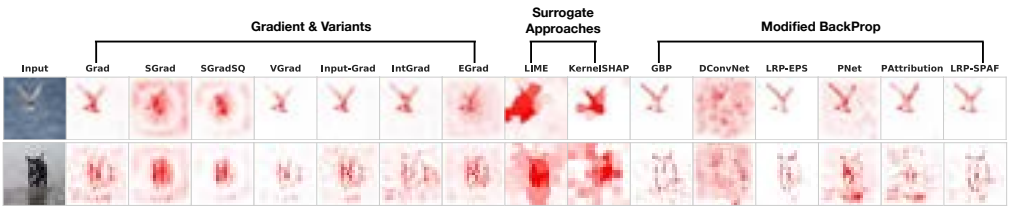

Figure 3-1: The Figure shows feature attributions for two inputs for a CNN model trained to distinguish between birds and dogs.

A Definition: A feature attribution functional, $E : f_\theta \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, maps the input, $x_i \in \mathbb{R}^d$, the model, f_θ, output, $f_{\theta,i}(x_i)$, to an attribution map, $M_{x_i} \in \mathbb{R}^d$. Note that $f_{\theta,i}(x_i)$ is the ith dimension of the model's output vector on the input x_i.

The landscape of model attribution methods is vast, so we focus on a subset here. We will first begin with methods that are based on the derivative an output scalar with respect to the input.

The **Gradient (Grad)** [Baehrens et al., 2010, Simonyan et al., 2014] feature attribution map is obtained as $M_{\text{grad}}(x_i) = |\nabla_{x_i} f_{\theta,i}(x_i)|$. This is the derivative of the ith dimension of the model's output with respect to the input. The output of this approach is a vector that is the same size as the input. The Gradient map is a key primitive of local post hoc explanation methods, and it is often a precursor in other approaches.

SmoothGrad (SGrad) [Smilkov et al., 2017], is computed as $M_{\text{sg}}(x_i) = \frac{1}{N}\sum_{i=1}^{N} \nabla_{x_i} f_{\theta,i}(x_i + n_i)$ where n_i is sampled according to a random Gaussian noise.

SmoothGrad Squared (SGradSQ) [Hooker et al., 2019] is the element-wise square of SmoothGrad: $M_{\text{SGradSQ}}(x_i) = M_{\text{sg}}(x_i) \odot M_{\text{sg}}(x_i)$.

VarGrad (VGrad) [Adebayo et al., 2018a] is the variance analogue of SmoothGrad: $M_{\text{VGrad}}(x_i) = \mathbb{V}[M_{\text{grad}}(x_i + n_i)]$, where $\mathbb{V}$ is the variance operator, and n_i is sampled according to a random Gaussian noise.

Input-Grad [Shrikumar et al., 2016] is the element-wise product of the gradient and input $|\nabla_{x_i} f_{\theta,i}(x_i)| \odot x_i$. Several other methods have been shown to be equivalent to this product. For example, for a DNN with all ReLU activations, LRP-Z, a variant of layer-wise relevance propagation that will discuss shortly, in the modified back-propagation section, is equivalent to Input-Grad [Kindermans et al., 2016].

Integrated Gradients (IntGrad) [Sundararajan et al., 2017] sums gradients along an interpolation path from a "baseline"/null, $\bar{x}$, to original inputx_i. The formulation is: $M_{\text{IntGrad}}(x_i) = (x_i - \bar{x}) \times \int_0^1 \frac{\partial f_{\theta,i}(\bar{x}+\alpha(x_i-\bar{x}))}{\partial x_i} d\alpha$.

Expected Gradients (EGrad) [Erion et al., 2019] computes IntGrad but with a baseline input that is an expectation over the training set:
x_i: $M_{\text{EGrad}}(x_i) = \int_{x'} \left((x_i - \bar{x}) \times \int_0^1 \frac{\partial f_{\theta,i}(\bar{x}+\alpha(x_i-\bar{x}))}{\partial x_i} d\alpha \right) p_D(x') dx'$.

Feature attribution methods that modified/post-process back-propagation. Next we discuss gradient-based feature attribution methods that modify the back-propagation procedure in various ways. Recall that several of the approaches described above can be obtained via simple gradient calls in an automatic differentiation package. The next set of approaches we discuss modify that routine. As we will see in the empirical evaluation, even though the modification is slight, it often leads to dramatic differences in the output visual quality.

Deconvnet [Zeiler and Fergus, 2014] & **Guided Backpropagation (GBP)** [Springenberg et al., 2014] both modify the backpropagation process at ReLu units in DNNs. Let, $a = \max(0, b)$, then for a backward pass, $\frac{\partial l}{\partial s} = 1_{s>0} \frac{\partial l}{\partial b}$, where l is a function of s. For Deconvnet, we have that $\frac{\partial l}{\partial s} = 1_{\frac{\partial l}{\partial s}>0} \frac{\partial l}{\partial b}$, and for GBP, we have $\frac{\partial l}{\partial s} = 1_{s>0} 1_{\frac{\partial l}{\partial s}>0} \frac{\partial l}{\partial b}$.

PatternNet & Pattern Attribution, [Kindermans et al., 2018]. PatternNet and Pattern Attribution first estimate a 'signal' vector from the input, x_i; then, the attribution

(in the case of Pattern Attribution) corresponds to the covariance between the estimated signal vector, and the output, $f_{\theta,i}(x_i)$, propagated all the way to the input.

DeepTaylor [Montavon et al., 2017]. DeepTaylor describes a family of methods that iteratively compute local Taylor approximations for each output unit in a DNN. These approximations produce unit attribution that are then propagated and redistributed all the way to the input.

Layer-wise Relevance Propagation (LRP) & Variants, [Alber et al., 2018, Bach et al., 2015]. LRP attribution methods iteratively estimate the relevance of each unit of a DNN starting from the penultimate layer all the way to the input in a message-passing manner. We consider 4 variants of the LRP method that correspond to different rules for propagating unit relevance.

We will now consider methods that approximate a model around a single input with linear models in order to explain the model's output on that input.

LIME [Ribeiro et al., 2016] locally approximates $f_{\theta,i}$ around x_i with a simple function, g. LIME corresponds to: $\arg\min_{g \in G} L(f, g, \text{pert}(x_i)) + \Omega(g)$, where $\text{pert}(x_i)$ are local perturbations of the input x_i, and $\Omega(g)$ is a regularizer.

SHAP [Lundberg and Lee, 2017] Similar to LIME, SHAP provides an approximation to the Shapley value [Shapley, 1988] to explain a model's output. The shapley value provides a way to allocate 'value' amongst a group of players in a cooperative game setting. In the model explanation interpretation, the dimensions of the input are the players in the game, and the relevance score of each dimension of the input is the payoff for that player. The SHAP approach provides a tractable approximation to computing the shapley value. Under certain regularity conditions, the relevance scores produced by SHAP and LIME are equivalent.

We now end the discussion on feature attribution approaches.

3.3 Concept Activation Methods

In this section, we switch to concept activation approaches. These are approaches that measure the sensitivity of a model output (or unit in the model) to a user defined concept [Bau et al., 2017, Kim et al., 2018]. In the previous section, we considered the influence of dimensions of the input on the output. However, one might interested in understanding whether the model has learned higher level concepts that cannot be simply derived from the input. We center

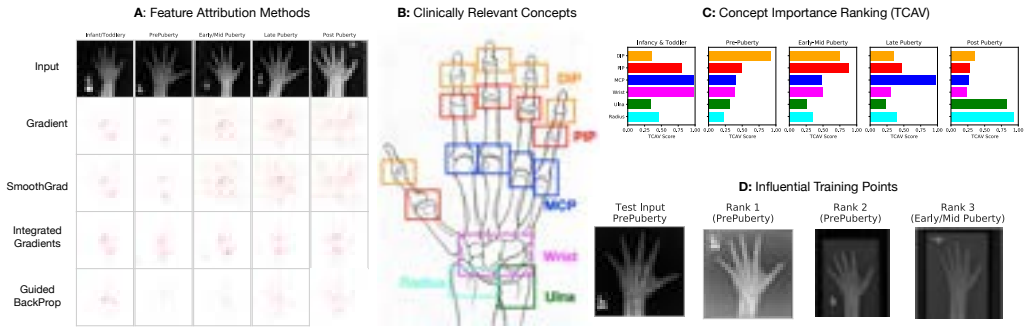

Figure 3-2: **A**: Here we show 5 different inputs, one for each bone age category, and the corresponding Gradient feature attribution map. We refer to the Appendix for an equivalent visualization for other feature attribution methods considered. We test three additional feature attribution methods: SmoothGrad, Integrated Gradients, and Guided BackProp. **B&C**:Concept Importance Methods for a Normal (non-spurious) model. Here we show the TCAV score for all clinical concepts as well as the spurious concepts for a normal model that was confirmed to not rely on the spurious signals. **D**:Top ranked influential examples for a test pre-puberty input on a normal model.

our discussion primarily on the TCAV approach of Kim et al. [2018].

Concept activation approaches quantify the sensitivity of a DNN's output score to user provided inputs that represent a concept of interest. For example, one might want to know whether a model trained to do object and animal recognition relies on the presence of stripes to detect whether there is a Zebra in the image.

Given hidden representations, h_l, from a particular layer of a DNN for for images belonging to concept class C. We can derive the sensitivity score as: $\nabla h_{l,k}(f_l(x)).\theta_c^l$. The previous expression indicates the sensitivity of the class score (logit) for class k to inputs indicating concept, C, given hidden representations from layer l from the DNN f. The concept vector, θ_c^l, typically corresponds to a the weights of a linear classifier trained to separate the images for a particular concept class from or images.

3.4 Training Point Ranking

The final kind of interpretation that we consider is training point ranking via influence functions. In the case of training point ranking via influence functions, we rank the training samples, in terms of 'influence', on the loss of a test example. Specifically, if we up-weighted a training point and retrained the model, then by how much would the loss on a given test example change? Koh and Liang [2017] analytically derive the analytically formulas

for computing this quantity. We provide a detailed, but non-rigourous derivation of these formulas here. In Chapter 6, we will modify these formulas to identify fairness violations.

Updated Notation We consider prediction problems, i.e, settings where the task is to learn a mapping, $\theta : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} \in \mathbb{R}^d$ is the feature space, and $\mathcal{Y} \in \{0, 1\}$ is the output space. We term the tuple, (x_i, y_i), z_i, which means the n training points can be written as: $\{z_i\}_{i=1}^n$. Throughout this section, we will only consider learning via empirical risk minimization (ERM), which corresponds to: $\hat{\theta} := \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_i^n \ell(z_i, \theta)$. Similar to Koh and Liang [2017], we will assume that the ERM objective is twice-differentiable and strictly convex in the parameters. We focus on binary classification tasks, however, our analysis can be easily generalized.

Upweighting a training point Let $\hat{\theta}_{-z_i}$ be the ERM solution when a model is trained on all data points except the training point z_i. The influence, $\mathcal{I}_{\text{up,params}}$, of datapoint, z_i, is then defined as the change: $\hat{\theta}_{-z_i} - \hat{\theta}$.

We can define the empirical risk as:

$$R(\theta) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \ell(z_i, \theta).$$

Since we assumed that the ERM solution is twice-differentiable and strictly convex in the parameters, we can specify the hessian as:

$$H_{\hat{\theta}} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta^2 \ell(z_i, \theta) = \nabla^2 R(\theta)$$

To start, let:

$$\hat{\theta}_{\epsilon,\ z_i} = \arg\min_{\theta \in \Theta} R(\theta) + \epsilon \ell(z_i, \theta).$$

Then we can define the parameter change to be: $\Delta_\epsilon = \hat{\theta}_{\epsilon,\ z_i} - \hat{\theta}$. Since $\hat{\theta}$ does not depend on ϵ, we have that:

$$\frac{d\hat{\theta}_{\epsilon,\ z_i}}{d\epsilon} = \frac{d\Delta_\epsilon}{d\epsilon}.$$

We know that $\hat{\theta}_{\epsilon,\ z_i}$ is a solution, i.e., minimizer, so we will form the first-order optimality conditions and form a Taylor expansion of that expression.

To do this we have that:

$$\nabla_\theta(R(\theta) + \epsilon\ell(z_i, \theta)) = 0,$$
$$\nabla R(\theta) + \epsilon\nabla\ell(z_i, \theta) = 0.$$

The Taylor expansion of the above expression is then:

$$[\nabla(R(\hat{\theta}) + \epsilon\nabla\ell(z_i, \theta)] + [\nabla^2(R(\hat{\theta}) + \epsilon\nabla^2\ell(z_i, \theta)\Delta_\epsilon] \approx 0.$$

Above, we only keep the first two terms of the Taylor Expansion.

We will now solve for Δ_ϵ in the above equation and then differentiate by ϵ to obtain the final expression.

Solving for Δ_ϵ, results in:

$$\Delta_\epsilon \approx -[\nabla^2(R(\hat{\theta}) + \epsilon\nabla^2\ell(z_i, \theta)]^{-1}[\nabla(R(\hat{\theta}) + \epsilon\nabla\ell(z_i, \theta)].$$

If we look at the expression for Δ_ϵ, we find that $\nabla(R(\hat{\theta})$ is zero, since we know that $\hat{\theta}$ minimizes the empirical risk R. In looking at the term $[\nabla^2(R(\hat{\theta}) + \epsilon\nabla^2\ell(z_i, \theta)]$, we find that it is equivalent to $[H_{\hat{\theta}} + \epsilon\nabla^2\ell(z_i, \theta)]$. We can further make the assumption that the contribution of the $\epsilon\nabla^2\ell(z_i, \theta)$ term is small, so we have $[H_{\hat{\theta}} + \epsilon\nabla^2\ell(z_i, \theta)] \approx H_{\hat{\theta}}$.

With the above assumptions and substitutions, we arrive at the new expression for Δ, which is now:

$$\Delta_\epsilon \approx -H_{\hat{\theta}}^{-1}\nabla\ell(z_i, \theta)\epsilon.$$

Let's differentiate the above expression by ϵ, and we have that:

$$\frac{d\Delta_\epsilon}{d\epsilon} = -H_{\hat{\theta}}^{-1}\nabla\ell(z_i, \theta).$$

Now recall that:

$$\frac{d\hat{\theta}_{\epsilon,\ z_i}}{d\epsilon} = \frac{d\Delta_\epsilon}{d\epsilon},$$

which means:

$$\frac{d\hat{\theta}_{\epsilon,\ z_i}}{d\epsilon} = \frac{d\Delta_\epsilon}{d\epsilon} = -H_{\hat{\theta}}^{-1}\nabla\ell(z_i, \theta).$$

As previously discussed, $\mathcal{I}_{\text{up,params}}$ is defined to be the influence of estimate for training point z_i, so:

$$\mathcal{I}_{\text{up,params}} \stackrel{\text{def}}{=} \frac{d\hat{\theta}_{\epsilon,\ z_i}}{d\epsilon} = -H_{\hat{\theta}}^{-1}\nabla\ell(z_i, \theta).$$

As we have seen, we can obtain the influence of a training point on a the loss of a test simply by applying the chain rule to the loss-derivative quantity of interest. We will now extend this notion of influence.

Given a closed-form approximation to $\mathcal{I}_{\text{up,params}}$, we can estimate the influence of a training point on functions of the parameters. For example, Koh and Liang [2017] show, using a chain-rule argument, that the influence, $\mathcal{I}_{\text{up,loss}}(z_i, z_t)$, of a training point, z_i, on the test loss for a test example has the following closed-form expression:

$$\begin{aligned}
\mathcal{I}_{\text{up,loss}}(z_i, z_t) &\stackrel{\text{def}}{=} \left.\frac{d\ell(z_t, \hat{\theta}_{\epsilon,\ z_i})}{d\epsilon}\right|_{\epsilon=0} \\
&= -\nabla_\theta\ell(z_t, \hat{\theta})^\top \left.\frac{d\hat{\theta}_{\epsilon,\ z_i}}{d\epsilon}\right|_{\epsilon=0}, \\
&= -\nabla_\theta\ell(z_t, \hat{\theta})^\top H_{\hat{\theta}}^{-1}\nabla_\theta\ell(z_i, \hat{\theta}). \qquad (3.1)
\end{aligned}$$

As we have seen, we obtain the influence of a training point on a the loss of a test simply by applying the chain rule to the loss-derivative quantity of interest. We will now extend this notion of influence.

Perturbing a training point A second notion of influence that Koh and Liang [2017] study is how perturbing a training point leads to changes in the model parameters. Specifically, given a training input, z_i, that is a tuple (x_i, y_i), then how would the perturbation, $z_i \to z_{i,\delta}$, which is defined as $(x_i, y_i) \to (x_i + \delta, y_i)$, change the model's predictions? The key issue here is how an infinitesimal change in the input example changes the model parameters and

predictions. Let $\hat{\theta}_{\epsilon,z_{i,\delta},-z_j}$ be the ERM solution to the following minimization problem:

$$\arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} \ell(z_i,\theta) + \epsilon\ell(z_{j,\delta},\theta) - \epsilon\ell(z_j,\theta).$$

As shown, $\hat{\theta}_{\epsilon,z_{i,\delta},-z_j}$ is the ERM solution obtained when an infinitesimal mass, ϵ, is shifted from z_j i.e. (x_j, y_j) to $z_{j,\delta}$, i.e. $(x_j+\delta, y_j)$. Similarly, Koh and Liang [2017] show that the closed-form approximation for such training point perturbation on the parameters is:

$$\begin{aligned}\mathcal{I}_{\text{up,params}}(z_{j,\delta}) - \mathcal{I}_{\text{up,params}}(z_j) &= \left.\frac{d\hat{\theta}_{\epsilon,z_{i,\delta},-z_j}}{d\epsilon}\right|_{\epsilon=0}, \\ &\approx -H_{\hat{\theta}}^{-1}[\nabla_x\nabla_\theta\ell(z_j,\hat{\theta})]\delta. \end{aligned} \tag{3.2}$$

With a closed-form expression for the influence of perturbing a training point, we can also obtain similar forms for functions of the parameters. Consequently, to obtain the influence of perturbing a training point on the model's prediction (i.e. a model with parameters $\hat{\theta}$), we differentiate with respect to δ, and apply the chain rule to obtain:

$$\begin{aligned}\mathcal{I}_{\text{pert,loss}}(z_j,z_t) &\stackrel{\text{def}}{=} \left.\nabla_\delta \ell(z_t,\hat{\theta}_{z_{j,\delta},-z_j})\right|_{\delta=0} \\ &\approx -\nabla_\theta\ell(z_t,\hat{\theta}_{z_{j,\delta},-z_j})^\top H_{\hat{\theta}}^{-1}\nabla_x\nabla_\theta\ell(z_j,\hat{\theta}). \end{aligned} \tag{3.3}$$

As we see in these previous equations, we now have a closed-form expression for the influence of a training point on the test-loss of a new example. On inspecting the influence expressions, we find that they typically involve taking the inverse of a the loss hessian. However, if a model has millions of parameters, it might not be even possible to form the hessian, let alone invert it. Hessian-vector products are typically used to speed up the computation in practice. Other approaches for ranking training inputs that side-step the need for an inverse hessian include: Representer points [Yeh et al., 2018], and TracIn [Pruthi et al., 2020]. We will now discuss how the approach is used in practice.

3.5 How are the approaches used in practice?

In this section, we discuss how each class of approach introduced in the previous section is used in practice.

Feature Attributions To debug a model, a practitioner inspects, one sample at a time, the attribution of a collection of inputs. In inspecting these attributions, the practitioner can then form an hypothesis about the behavior of the model on certain inputs. Using their prior knowledge of the task, the practitioner can then identify whether the model is relying on features that it should not or if something else is amiss.

Concept Activation For each concept of interest, the practitioner collects input examples that indicate the concept. For example, to test a model's dependence on a hospital tag, we collect images that all have hospital tags in them. Given this collection of examples, the TCAV approach can then be used to calculate a score, TCAV score, that corresponds to the sensitivity of a particular output class to the hospital tag concept.

Training Point Ranking These class of approaches ranks all training points by influence on the test loss of an input. Qualitatively, if a training point has a high influence on the test loss of an input, it means that if the model were re-trained without that training point, the test loss of that test point would change significantly. Intuitively, the most highly ranked training points for a given test input should also be points for which the model relies on semantically similar features.

Chapter 4

Challenges with Model Debugging Using Post hoc Explanation Methods

In this chapter, we comprehensively assess whether current post hoc explanation approaches are effective for model debugging. Specifically, we evaluate each explanation type against a series of model bugs. We ask: can the explanation even detect whether a model bug has occurred? If the method does not pass this filter, it holds not hope for even helping to fix the model bug.

Our findings paint a dim picture—current post hoc approaches are mostly ineffective for detecting model bugs. In the specific settings where they are effective, they are only able to detective the model bug, but not suggest a possible fix.

First, we ask whether post hoc explanations are able to detect that a model is relying on spurious signals. Second we consider the noisy training label bug. Third, we induce parameter contamination in trained models and ask whether feature attributions can distinguish a normal model from one whose parameters have been contaminated. Fourth, we assess whether explanations can tip an auditor off that a input, at test-time, is out-of-distribution for the model. In all the previous settings, we devise an experimental design along with quantitative metrics to help simulate a model debugging process. However, effectiveness in simulation does not always translate to the real-world. Consequently, we conduct a large-scale study to assess whether feature attributions are effective, in the hands of end-users, for the aforementioned model bugs.

Before we dive into the assessment, in the next section, we provide an overview of the

results for each assessment type.

4.1 Overview & Summary of Results

Spurious Correlation. In Section 4.2, we assess whether three kinds of explanations—feature attributions, concept activation methods, and training point ranking approaches—can help detect whether a model is relying on a spurious signal. We will consider two principal settings for the spurious signal as follows:

1. **Known Spurious Signal**. In this setting, the spurious signal is known ahead of time by the individual seeking to assess the reliability of a trained model. Consequently, the goal will be to use a post hoc explanation method to test whether the model being debugged is reliant on the spurious signal of interest.

2. **Unknown Spurious Signal**: In this setting, the spurious signal is not known to the individual assessing a model's reliability, which mimics what is typically encountered in the real-world.

Based on how current post hoc explanation methods are used, detecting that a model relies on an unknown signal would require the end-user to either: 1) hypothesize what a set of possible spurious signal is and test for the model's explicit dependence on each hypothesized signal, or 2) inspect explanations of the model being debugged under 'normal' conditions—a notion more concrete later in the chapter. Based on two options available when the spurious signal is not known, it might be immediately obvious that post hoc explanations are unlikely to be effective. In the first case, the model auditor has to hope that 1) if the model is relying on a spurious signal, that spurious signal is in the hypothesized set of signals, or 2) that the post hoc explanation method being used is not susceptible to false positives, i.e., indicating that a model is relying on spurious signal when it does not, and lastly, 3) explanations of a normal input for a defective model should indicate that the model is anomalous. As we will see, the unknown spurious signal setting proves to be challenging for current methods.

We find that concept methods can indicate a model's reliance on Tag and Stripe signals when known. However, the approach struggles to detect Blur signal even when known. As is the case with feature attributions, when a spurious signal is not explicitly tested for,

our significance tests indicate that reliance on the signal cannot be detected from other non-spurious concepts.

Noisy Training Labels. In Section 4.3 we demonstrate that attributions of mislabelled examples are visually similar to attributions for these same examples but derived from a model with correct input labels. This result suggests that feature attributions are unlikely to be effective for detecting mislabelled training examples. On the other hand, recent evidence has demonstrated that training point ranking based on influence functions is effective at detecting inputs that are mislabelled [Koh and Liang, 2017].

Out-of-distribution bug (Section 4.5). To assess the ability of feature attributions to diagnose domain shift, we compare attributions derived, for a given input, from an *in-domain model* with those derived from *out-of-domain model*. For example, we compare the attribution for an MNIST digit, derived from a model trained on MNIST, to an attribution for the same digit, but derived from a model trained on Fashion MNIST, ImageNet, and a birds-vs-dogs model. We find high visual similarity for these attributions, which indicates that feature attributions are unlikely to be effective for detecting that an input is out-of-distribution for a model being debugged.

Parameter Contamination: Weight Re-initialization (Section 4.4). Next, we evaluate bugs related to model parameters. Specifically, we consider the setting where the weights of a model are accidentally re-initialized prior to prediction [Adebayo et al., 2018b]. We find that modified back-propagation methods like Guided Back-Propapagtion (GBP), DConvNet, and certain variants of the layer relevance propagation (LRP), including Pattern Net(PNet) and Pattern Attribution (PAttribution) are invariant to higher layer weights of DNNs for the ImageNet and MNIST tasks.

User Study. In section 4.6, we discuss a user study conducted to assess whether practitioners can use current post hoc explanation tools to detect model bugs. In this study, we consider only feature attribution methods for 6 model bugs. The primary takeaway is that users rely on a model's predictions, instead of the explanations, to detect a model bug.

4.2 Challenges with Debugging Reliance on Spurious Signals

In this subsection, we present our experimental design, a suite of metrics for assessing a post hoc method's effectiveness, and the class of models considered. We then consider each

explanation type, in turn, and present results on their effectiveness for detecting a model's reliance on spurious signals.

4.2.1 Experimental Design for Spurious Signal Detection

We consider 3 (2 visible and 1 non-visible) kinds of spurious signals (See Figure 4-1):

1. a localized image tag;
2. a distinctive stripped pattern; and
3. Gaussian blur applied to the image background.

The signals are all spatially localized, so we can easily obtain ground-truth expected explanations.

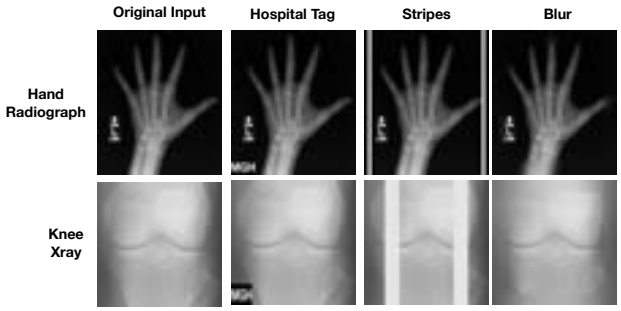

Figure 4-1: The various spurious signals considered: A hospital tag, vertical stripes, and background blur.

To induce reliance on spurious signals, we train models on "contaminated" versions of the training set. Given input-label pairs, $\{(x^i, y^i)\}_i^n$, where $x^i \in \mathcal{X}$ and $y^i \in \mathcal{Y}$, we can learn a classifier, f_θ, via empirical risk minimization (ERM) that corresponds to minimizing a loss function, ℓ: $\arg\min_\theta \sum_{i=1}^n \ell(x^i, y^i; \theta)$. To contaminate the training set, we apply a spurious contamination function (SCF) to the training set; SCF : $\mathcal{X} \times \mathcal{Y} \times \mathcal{C} \to \mathcal{S}$, where $\mathcal{C}$ is the spurious signal set and $\mathcal{S}$ is the transformed set. An example of an SCF is a function that pastes an hospital tag onto the bone age radiographs of all pre-puberty individuals in the dataset. To derive models reliant on a spurious signal, $c_i \in \mathcal{C}$, we simply learn a new classifier via ERM on the modified dataset as follows: $\arg\min_\theta \sum_{i=1}^n \ell\big(\text{SCF}(x^i, y^i, c_i)\big)$ to obtain θ_{spu}. Contemporary evidence suggests that this approach produces models that easily latch onto the spurious signal [Nagarajan et al., 2020].

We focus on the classification setting, and restrict spurious signals to encode, only, for a single class—the *spurious aligned class*. We measure a model's reliance on the spurious signal via a score.

Definition 4.2.1. (Spurious Score). Given a spurious signal, c_i, the index of its spurious aligned class, $j \in [k]$, a model, $\theta_{\text{spu}} : \mathbb{R}^d \to \mathbb{R}^k$, where $\arg\max(\theta_{\text{spu}})$ indicates the classifier's predicted class, we define the spurious score as:

$$\text{SC}_{c_i,j}(\theta_{\text{spu}}) \coloneqq \mathbb{P}_{\{x^i | \theta_{\text{spu}}(x^i) \; != \; j\}}[\arg\max(\theta_{\text{spu}}(\text{SCF}(x^i, y^i, c_i))) = j].$$

Given an input that does not contain the spurious signal, and for which the model's prediction is not the spurious aligned class, the model's spurious score is the probability that the model assigns the input to the spurious aligned class if the spurious signal is added to the input.

Model Conditions. We focus our analysis on two model conditions:

1. a 'normal model', f_{norm}, for which we can rule out dependence on any of the spurious signals tested across all classes on the basis of the spurious score, and

2. a 'spurious model', f_{spu}, for which one of the spurious signals encodes for a particular output class.

We empirically estimate the spurious score and term models that have a score above 0.85 for any of the pre-defined signals 'spurious models'. We term a model 'normal' if the spurious score is below 0.1 across all classes and the 3 pre-defined spurious signals.

Spurious Signal Detection Reliability Measures. Equipped with spurious (f_{spu}) and normal (f_{norm}) models, we are now able to quantitatively assess the motivating question of this work. We do this by comparing explanations derived from spurious models, f_{spu}, to those derived from normal models (f_{norm}). We can partition the kinds of inputs used for deriving explanations into two: 1) *spurious inputs* (x_{spu})—inputs that include the spurious signal and 2) *normal inputs* (x_{norm})—inputs do not not contain the spurious signal. Comparing the explanations produced by these two classes of inputs for normal and spurious models, we derive reliability performance measures.

- **Known Spurious Signal Detection Measure (K-SSD)** - measures the similarity of explanations derived from *spurious models on spurious inputs* to the ground truth

explanation. The ground truth explanation is one that only assigns relevance to the spurious signal as explanation of the output of a spurious model on a spurious input. K-SSD measures method reliability when the spurious signal is known. Given a similarity metric, S_d, then K-SSD corresponds to: $S_d\big(\mathrm{E}_{f_{\mathrm{spu}}}(x_{\mathrm{spu}}), x_{\mathrm{gt}}\big)$; where $\mathrm{E}_{f_{\mathrm{spu}}}(x_{\mathrm{spu}})$ are explanations derived from the spurious model for spurious inputs, and x_{gt} is the ground truth explanation. The similarity function, S_d, depends on the type of explanation considered—we will make our choice of this function concrete shortly.

- **Cause-for-Concern Measure (CCM)** - measures the similarity of explanations derived from *spurious models for normal inputs* to explanations derived from *normal models for normal inputs*: $S_d\big(\mathrm{E}_{f_{\mathrm{spu}}}(x_{\mathrm{norm}}), \mathrm{E}_{f_{\mathrm{norm}}}(x_{\mathrm{norm}})\big)$. This measure simulates the setting where a practitioner does not know the spurious signal, and can only inspect explanations for inputs without the signal. If this measure is high, then it is unlikely that such a method alert a practitioner that a spurious model exhibits defects.

- **False Alarm Measure (FAM)** - measures the similarity of explanations derived from *normal models for spurious inputs* to explanations derived from *spurious models for spurious inputs*: $S_d\big(\mathrm{E}_{f_{\mathrm{norm}}}(x_{\mathrm{spur}}), \mathrm{E}_{f_{\mathrm{spu}}}(x_{\mathrm{spu}})\big)$. We also introduce a variant of this measure, FAM-GT, which measures the similarity of a explanations derived from *normal models for spurious inputs* to the ground truth explanation of a spurious model for that spurious input. If this measure is high, then that approach is more likely to signal to a practitioner that a model is relying on spurious signal when the model does not.

Metrics for Feature Attribution. For feature attribution methods, we use the Structural Similarity Index (SSIM). SSIM measures the visual similarity between two images. Concretely, given a set of normal inputs, we obtain a corresponding spurious set of these inputs by applying the spurious contamination function, SCF to these inputs. Consequently, we can then compute the K-SSD, CCM, and FAM metrics given these two sets of inputs using the SSIM metric.

Metrics for Concept Activation. We measure comparison between two concept rankings using a Kolmogorov-Smirnoff (KS) test comparing two distributions where the null hypothesis is that the two distributions are identical; we set significance level to be 0.05.

Metrics for Training Point Ranking. Recently, Hanawa et al. [2020] introduced the

Identical class metric' (ICM), which is the fraction of the top training inputs, for a given test example, that belong to the same class as the true class of the test example in question. Here we also use the KS test to compare the ICM distributions for two different models and set the significance level to be 0.05.

Taken together, these measures provide a comprehensive overview of an explanation method's performance for detecting spurious signals.

4.2.2 Results for Feature Attributions

In this section, we present results on whether feature attributions are effective for detecting unknown spurious correlation.

Setup. We consider 3 kinds of spurious signals which we term:

1. *Tag* for the 'MGH' hospital tag added to the pre-puberty class,
2. *Stripe* for the paired vertical stripped signal, and
3. *Blur* for the background blur.

Given these signals, we then compute the three performance measures of interest: K-SSD, CCM, FAM, and FAM-GT. K-SSD indicates a method's reliability when the spurious signal is known, CCM when the signal is not known and the practitioner only has access to inputs that don't encode the spurious signal. Lastly, FAM and FAM-GT indicates the susceptibility of a method to false positives. An oft-used heuristic based on prior work [Adebayo et al., 2020] for interpreting SSIM scores is that SSIM scores $0.2-0.4$ indicate weak visual similarity, $0.5-0.7$ indicate moderate similarity, and > 0.75 high similarity. This is because two random images typically have SSIM much less than 0.1. For example, we empirically estimate the similarity of two random (229×229) Gaussian images to be less than 0.00023. Even for natural images, we still find the SSIM values to be below 0.005, which substantiates the previous heuristic.

Results. We show performance measures for all the feature attribution methods tested for the Tag and Blur settings in Table 4.1. For the tag setting, the attribution methods are indeed able to attribute to the visible spurious signal and the K-SSD measure indicates this finding with mean scores typically above 0.65 for the bone age setting. Contrary to previous findings, we find that GBP outperforms other approaches for known spurious signals.

Alternatively, GBP is more suceptible to false positives based on the FAM and FAM-GT score. Across all methods, we find that these methods also seem to attribute to the spurious signal (FAM-GT > 0.4) even when the signal is not being relied on by the model. We observe similar findings for the strip setting as well across all tasks. The CCM measure further indicates that these methods do not indicate presence of spurious signals when the signal is unknown for both the Tag and Stripe signals. This finding, however, reverses for the non-visible blur spurious signal. Across all measures, we find that all methods struggle to reliably indicate that spurious models are reliant on the blur signal.

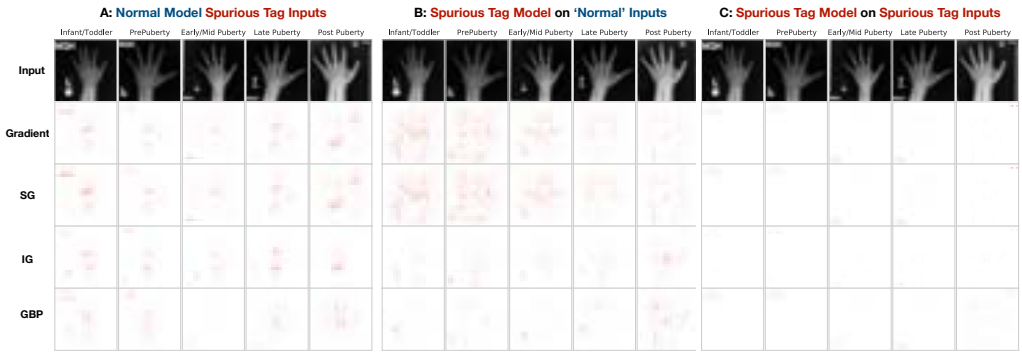

Figure 4-2: **Detecting Spurious *Tag*.** Here we show in A) Feature attributions for 5 different inputs across the four feature attribution methods with a normal model but with spurious Tag inputs; B) Feature attributions on the same 5 inputs as in (A), but **without** spurious Tag inputs with a model that has learned a spurious alignment between Pre-Puberty and Tag; C) Feature attributions on the same 5 inputs as in (A), but **with** the spurious Tag inputs with a model that has learned a spurious alignment between Pre-Puberty and Tag.

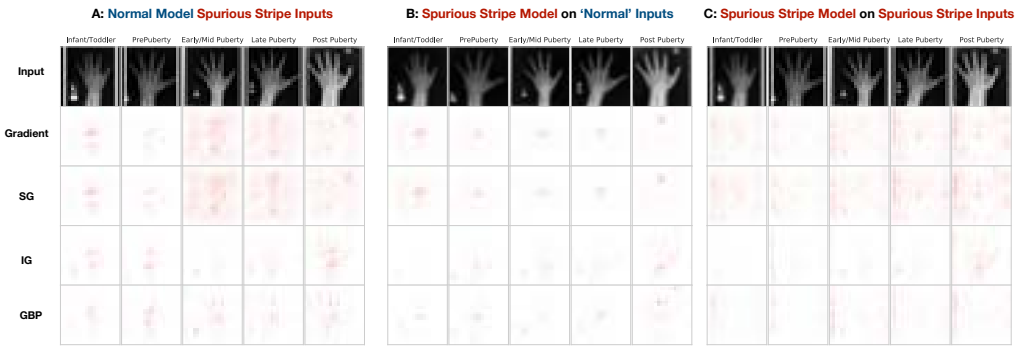

Figure 4-3: **Detecting Spurious *Stripe*.** Here we show in A) Feature attributions for 5 different inputs across the four feature attribution methods with a normal model but with spurious Stripes inputs; B) Feature attributions on the same 5 inputs as in (A), but **without** the spurious Stripe with a model that has learned a spurious alignment between Pre-Puberty and Stripe; C) Feature attributions on the same 5 inputs as in (A), but **with** the spurious Stripe with a model that has learned a spurious alignment between Pre-Puberty and Stripe.

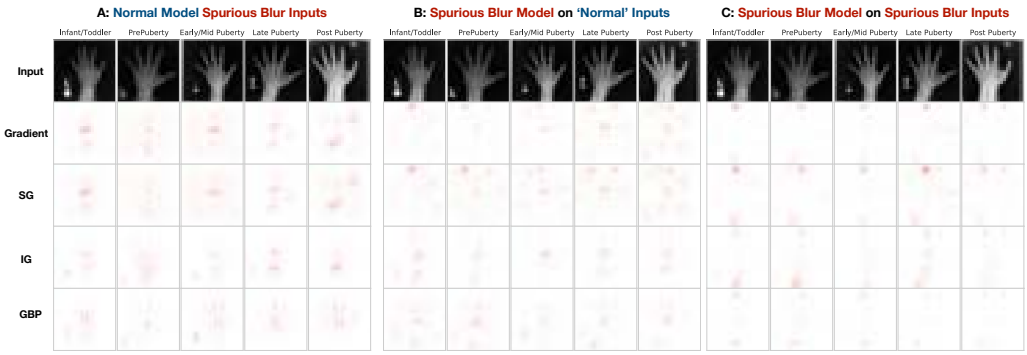

Figure 4-4: **Detecting Spurious *Blur*.** The blur images are analogous to the Tag and Stripe settings.

Table 4.1: Performance metrics for each attribution method across tasks for the Tag Setting. Below each metric in the Table is another row (SEM) that indicates the standard error of the mean for each value.

	Bone Age				Knee				Dog Breeds			
Method	Grad	SG	IG	GBP	Grad	SG	IG	GBP	Grad	SG	IG	GBP
K-SSD	0.65	0.66	0.67	0.81	0.51	0.49	0.47	0.76	0.71	0.76	0.79	0.88
K-SSD (SEM)	0.0097	0.013	0.019	0.006	0.012	0.017	0.019	0.023	0.01	0.011	0.014	0.01
CCM	0.37	0.39	0.35	0.75	0.32	0.33	0.35	0.66	0.42	0.41	0.39	0.64
CCM (SEM)	0.0031	0.002	0.015	0.029	0.027	0.023	0.029	0.014	0.013	0.016	0.012	0.015
FAM	0.51	0.55	0.53	0.68	0.46	0.47	0.45	0.69	0.59	0.64	0.68	0.73
FAM (SEM)	0.0029	0.0019	0.018	0.024	0.023	0.024	0.019	0.016	0.015	0.011	0.022	0.035
FAM-GT	0.56	0.53	0.46	0.61	0.42	0.48	0.41	0.63	0.76	0.73	0.77	0.81
FAM-GT (SEM)	0.017	0.035	0.0253	0.028	0.016	0.019	0.0045	0.006	0.011	0.033	0.024	0.0053

Additionally, we also find that the FAM scores are typically higher than the CCM scores across the tasks. This finding indicates that the feature attribution methods tested are more susceptible to false positives than they are to indicate to a practitioner that a model is defective in the absence of the spurious signal, a finding that casts doubt on the utility of such approaches in practice.

4.2.3 Results for Concept Activation Methods

Overview & Setup. In this setting, we compute the 3 performance metrics of interest: K-SSD, CCM, and FAM. To measure comparison between two concept rankings, we use a Kolmogorov-Smirnoff (KS) test comparing two distributions where the null hypothesis is that the two distributions are identical; we set significance level to be 0.05.

Result. In Figure 4-5, we show TCAV scores for each bone age class for both the spurious tag and the blur models. In Table 4.2, we show results of the KS-Test across metrics for the Tag Setting. Here, a ✗ means we reject the null, while an ✓means we are unable to do so at

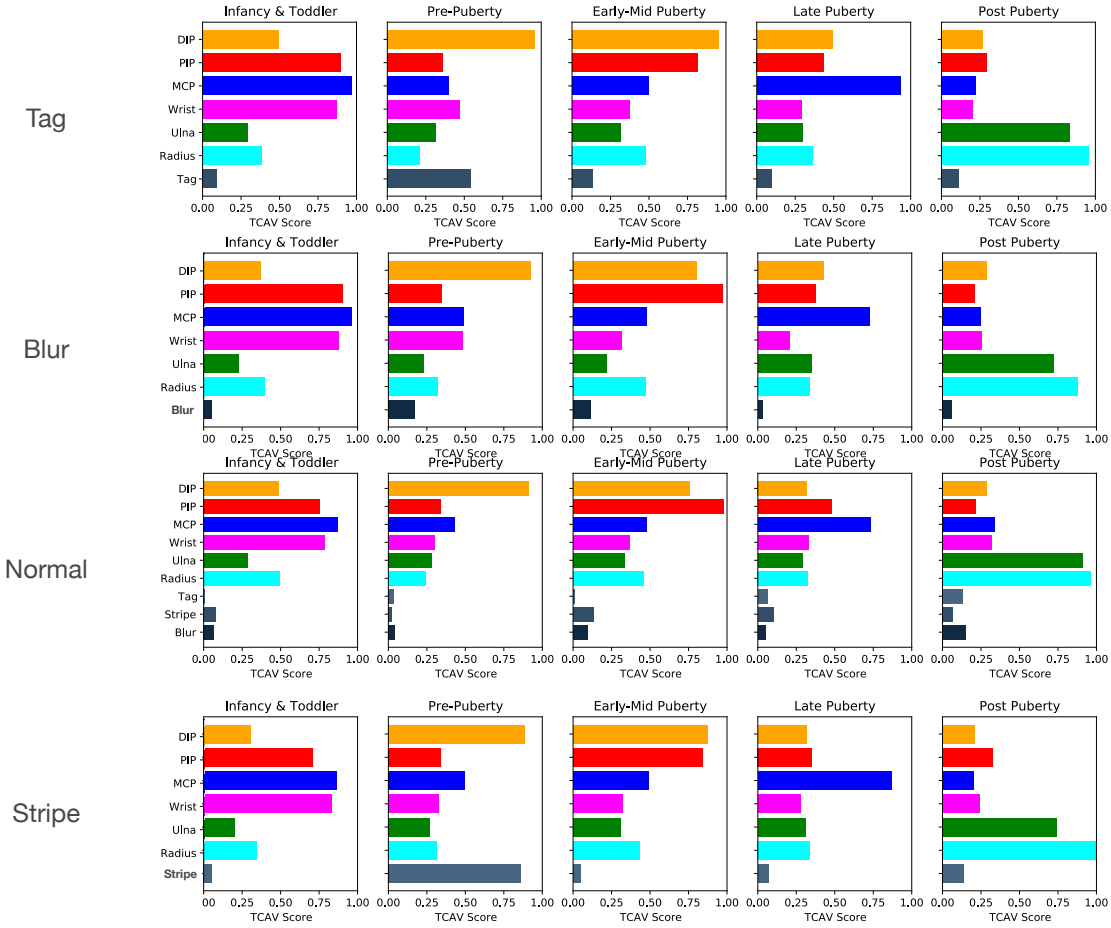

Figure 4-5: **Concept Results for Normal, Tag, Stripe and Blur Models.** TCAV scores for a model reliant on the spurious Tag and a model reliant on Blur.

the pre-specified significance level. For the K-SSD metric, the KS-test rejects the null for the Tag and Stripe signals. However, we find that the opposite is the case for the blur signal. This suggests that concept rankings can help detect reliance on the visible signals but not non-visible signals when the spurious signals are unknown. However, for the CCM metric, we are unable to reject the hypothesis that the distribution of concept rankings are not similar for all spurious signals. This finding suggests that when the spurious signal is unknown, and we only compare the distributions of known (non-spurious) concepts for a normal model and a spurious model, there is high similarity. Lastly, a difference in means test as well as a KS-Test for the FAM measure indicates that the normal models do not rely on the spurious signals as well. Overall, this finding suggests that TCAV is less susceptible to false positives.

4.2.4 Results for Training Point Ranking

Overview & Setup. We now describe our empirical findings for the training point ranking via influence functions approach. Here we present results for the case where the spurious signal is aligned with the Pre-Puberty class. **Results.** The main take away is that given a known spurious signal, the fraction of top ranked training spurious signal inputs increases with a spurious model across all of the spurious signals. While this might seem encouraging, we note that such increase actually indicates that the ICM metric might provide illusory confidence in a spurious model. Ultimately, a critical requirement here is knowing what the spurious signal ahead of time and to be able to select the right inputs to inspect.

Table 4.2: Tag Metrics.

Metric	Result
K-SSD	✗
CCM	✓
FAM	✓

4.3 Challenges with Debugging Noisy Training Labels

In this section, we study whether feature attribution approaches are effective for detecting whether a training point is mislabelled.

Feature Attributions In the setup, we compare attributions for a training input derived from: 1) a model where this training input had the correct label, and 2) the same model settings but trained with this input mislabeled. If the attributions under these two settings are similar, then such a method is unlikely to be useful for identifying mislabeled examples. We observe that attributions for mislabeled examples, across all methods, show visual similarity.

Table 4.3: Training Point Ranking

Metric	Result
K-SSD	✗
CCM	✓
FAM	✓

General Data and Model Setup. We consider a birds-vs-dogs binary classification task. We use dog breeds from the Cats-v-Dogs dataset [Parkhi et al., 2012a] and Bird species from the Caltech-UCSD dataset [Wah et al., 2011]. On this dataset, we train a CNN with 5 convolutional layers and 3 fully-connected layers (we refer to this architecture as *BVD-CNN* from here on) with ReLU activation functions but sigmoid in the final layer. The model

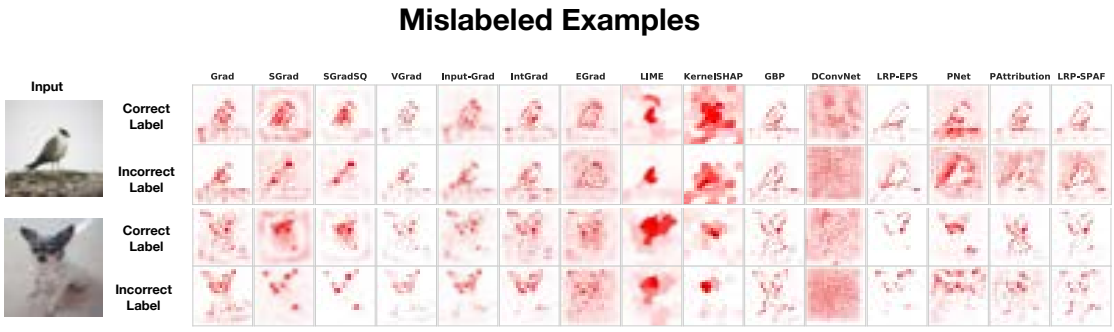

Figure 4-6: Feature Attributions for noisy label detection.

Results. We find that attributions from mislabelled examples for a defective model are visually similar to attributions for these same examples but derived from a model with correct input labels. We find that the SSIM between the attributions of a correctly labeled instance, and the corresponding incorrectly labeled instance, are in the range $0.73 - 0.99$ for all methods tested. These results indicate that the attribution methods tested might be ineffective for identifying mislabelled examples.

4.4 Challenges with Debugging Parameter Contamination

In this section, we consider model bugs that result from a parameter contamination. Specifically, we consider the setting where the weights of a model are accidentally re-initialized prior to prediction. Prior to this work, Adebayo et al. [2018b] considered the same task. They considered 8 features attribution methods, and tested whether these methods are sensitive to parameter weights. They found that Guided Backprop Springenberg et al. [2014] is incentive to re-initialization of the higher layer parameters for ImageNet, and MNIST tasks.

As discussed in Chapter 2, parameter re-intialization of an already trained model is type of contamination that can result in a model bug, particularly, if done unintentionally. Here, we extend the sensitivity test to 17 different feature attribution methods. We find that modified back-propagation methods like Guided Back-Propapagtion (GBP), DConvNet, and certain variants of the layer relevance propagation (LRP), including Pattern Net(PNet) and Pattern Attribution (PAttribution) are invariant to higher layer parameters for ImageNet

and MNIST. In the Chapter epilogue, we further comment on the generality of these findings.

Bug Implementation. We instantiate this bug on a pre-trained VGG-16 model on Imagenet Russakovsky et al. [2015]. Similar to Adebayo et al. [2018b], we re-initialize the weights of the model starting at the top layer, successively, all the way to the first layer. We then compare attributions from these (partially) re-initialized models to the attributions

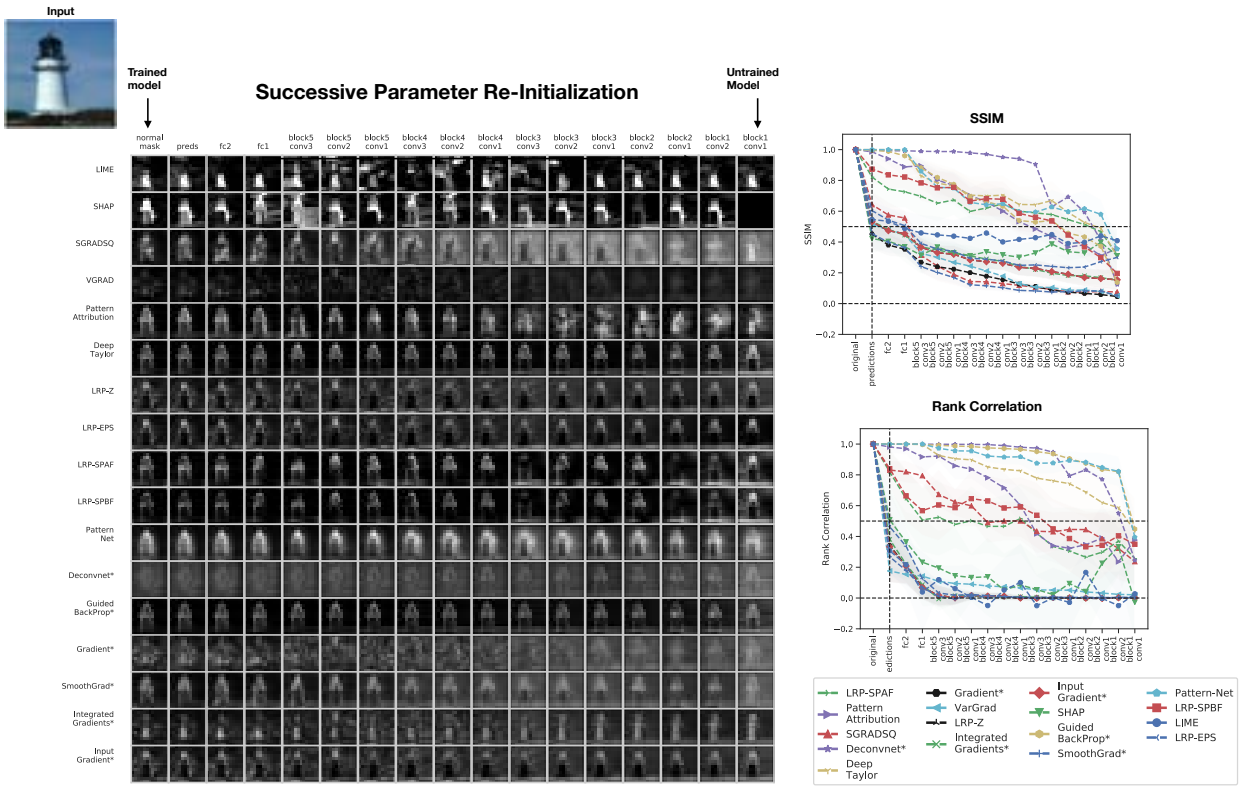

Figure 4-7: **Evolution of several model attributions for successive weights re-initialization of a VGG-16 model trained on ImageNet.** Qualitative results (left) and quantitative results (right). The last column in qualitative results corresponds to a network with completely re-initialized weights.

Results: modified back-propagation methods are invariant to higher layer model parameters of a DNN on ImageNet and MNIST tasks. As seen in Figure 4-7, the class of modified back-propagation methods, including Guided BackProp, Deconvnet, DeepTaylor, PatternNet, Pattern Attribution, and LRP-SPAF are visually and quantitatively invariant to higher layer parameters of the VGG-16 model. This finding corroborates prior results for Guided Backprop and Deconvnet [Adebayo et al., 2018b, Mahendran and Vedaldi,

2016, Nie et al., 2018]. These results also support the recent findings of Sixt et al. [2019], who prove that these modified back-propagation approaches produce attributions that converge to a rank-1 matrix.

4.5 Challenges with Debugging Out-of-Distribution Inputs

A model is at risk of providing errant predictions when given inputs that have distributional characteristics different from the training set. To assess the ability of feature attributions to diagnose domain shift, we compare attributions derived, for a given input, from an *in-domain model* with those derived from *out-of-domain model*. For example, we compare the attribution for an MNIST digit, derived from a model trained on MNIST, to an attribution for the same digit, but derived from a model trained on Fashion MNIST, ImageNet, and a birds-vs-dogs model. We find visual similarity for certain settings: for example, feature attributions for a Fashion MNIST input derived from a VGG-16 model trained on ImageNet are visually similar to attributions for the same input on a model trained on Fashion MNIST. However, the quantitative ranking of the input dimensions are widely different.

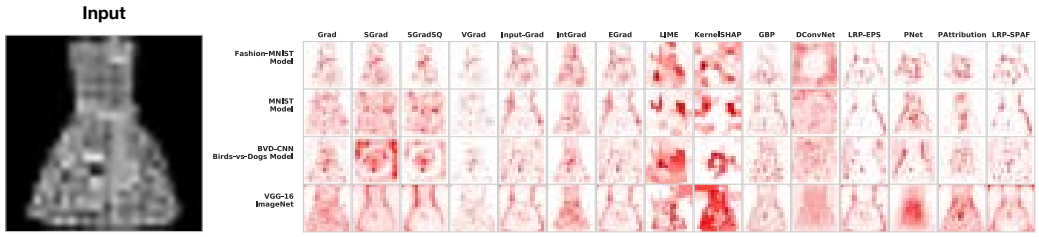

Figure 4-8: **Fashion MNIST OOD on several models.** The first row shows feature attributions on a model trained on Fashion MNIST. In the subsequent rows, we show feature attributions for the same input on an MNIST model, BVD-CNN model trained on birds-vs-dogs, and lastly, a pre-trained VGG-16 model on ImageNet.

Bug Implementation. We consider 4 dataset-model pairs: a BVD-CNN trained on MNIST, Fashion MNIST, the Birds-vs-dogs data, and lastly a VGG-16 model trained on ImageNet. We present results on Fashion MNIST. Concretely, we compare 1) feature attributions of Fashion MNIST examples derived from a model trained on Fashion MNIST, and 2) feature attributions of Fashion MNIST examples for models trained on MNIST, the birds-vs-dogs dataset, and ImageNet.

Results. As shown in Figure 4-8, we observe visual similarity between in-domain Fashion MNIST attributions, and attributions for these samples on other models. As seen in Table 4.4,

Metric	Grad	SGrad	SGradSQ	VGrad	Input-Grad	IntGrad	EGrad	LIME	KernelSHAP	GBP	DConvNet	LRP-EPS	PNet	PAttribution	LRP-SPAF
SSIM (FMNIST → MNIST Model)	0.7	0.54	0.49	0.92	0.71	0.69	0.71	0.46	0.41	0.81	0.5	0.77	0.58	0.77	0.66
SEM	0.0093	0.012	0.016	0.0047	0.01	0.015	0.01	0.02	0.024	0.014	0.01	0.02	0.026	0.009	0.03
RK (FMNIST → MNIST Model)	0.0013	8.8e-4	0.37	0.37	0.0021	-0.003	0.002	-0.01	0.034	0.51	0.027	0.011	-0.14	0.0082	0.12
SEM	0.0016	0.0032	0.026	0.029	0.002	0.002	0.002	0.04	0.028	0.014	6e-4	0.0034	0.027	0.0026	0.023
SSIM (FMNIST → BVD-CNN)	0.7	0.5	0.55	0.93	0.72	0.7	0.72	0.72	0.72	0.82	0.63	0.79	0.53	0.36	0.66
SEM	0.0083	0.011	0.013	0.0045	0.009	0.013	0.009	0.009	0.01	0.009	0.014	0.019	0.03	0.025	0.035
RK (FMNIST → BVD-CNN)	0.0012	0.0078	0.43	0.25	0.0002	0.002	0.00025	0.18	0.067	0.078	-0.05	-0.013	0.25	-0.0095	0.044
SEM	8.5e-4	0.0017	0.009	0.011	0.0007	0.001	0.0007	0.04	0.034	0.008	0.0011	0.0027	0.045	0.0023	0.02
SSIM (FMNIST → VGG-16 ImageNet)	0.57	0.46	0.5	0.87	0.64	0.67	0.64	0.5	0.38	0.8	0.36	0.64	0.66	0.12	0.2
SEM	0.012	0.011	0.015	0.0056	0.01	0.015	0.011	0.015	0.03	0.009	0.01	0.02	0.018	0.0049	0.024
RK (FMNIST → VGG-16 ImageNet)	-0.0023	-0.0098	-0.0097	0.028	-0.0025	-0.0017	-0.0025	0.005	-0.045	0.25	0.03	0.0045	0.32	0.066	0.14
SEM	0.0017	0.0025	0.02	0.018	0.0023	0.0016	0.002	0.033	0.024	0.004	0.0035	0.0018	0.034	0.0053	0.019

Table 4.4: **Test-time Explanation Similarity Metrics.** We observe visual similarity but no ranking similarity. We show each metric along with the standard error of the mean calculated for 190 examples. FMNIST → MNIST model means a comparison of FMNIST attributions for an FMNIST model with FMNIST attributions derived from *an MNIST model*. We present both SSIM and Rank correlation metrics.

we observe visual similarity, particularly for the VGG-16 model on ImageNet, but essentially no correlation in feature ranking.

4.6 Challenges with Current Tools in Practice: User Study

In this section, we discuss a user study conducted to assess whether practitioners can use current post hoc explanation tools to detect model bugs. In this study, we consider only feature attribution methods for the following model bugs 6 model bugs. The primary takeaway is that users rely on a model's predictions, instead of the explanations, to detect a model bug.

General Setup. We consider a 10-way dog breed classification task, which allows us to leverage prior common knowledge about dogs. Concretely, we assess three model attribution methods across 7 different model conditions (1 control and 6 model bugs). We recruited participants, 55 in total (but 54 analyzed; dropped a random participant due to balancing requirement), a majority with self-assessed prior experience of machine learning, and asked them to take on the role of a quality assurance tester at a machine learning startup. Ultimately, each participant was asked to provide their recommendation on whether a model was fit for sale to external customers based on the model outputs, along with their attributions.

Datasets. We created a modified combination of the Stanford dogs dataset [Khosla et al., 2011] and the Oxford Cats and Dogs datasets [Parkhi et al., 2012b] as our primary data source. We restrict to a 10-class classification task consisting of the following breeds: *Beagle, Boxer, Chihuahua, Newfoundland, Saint Bernard, Pugs, Pomeranian, Great Pyrenees, Yorkshire*

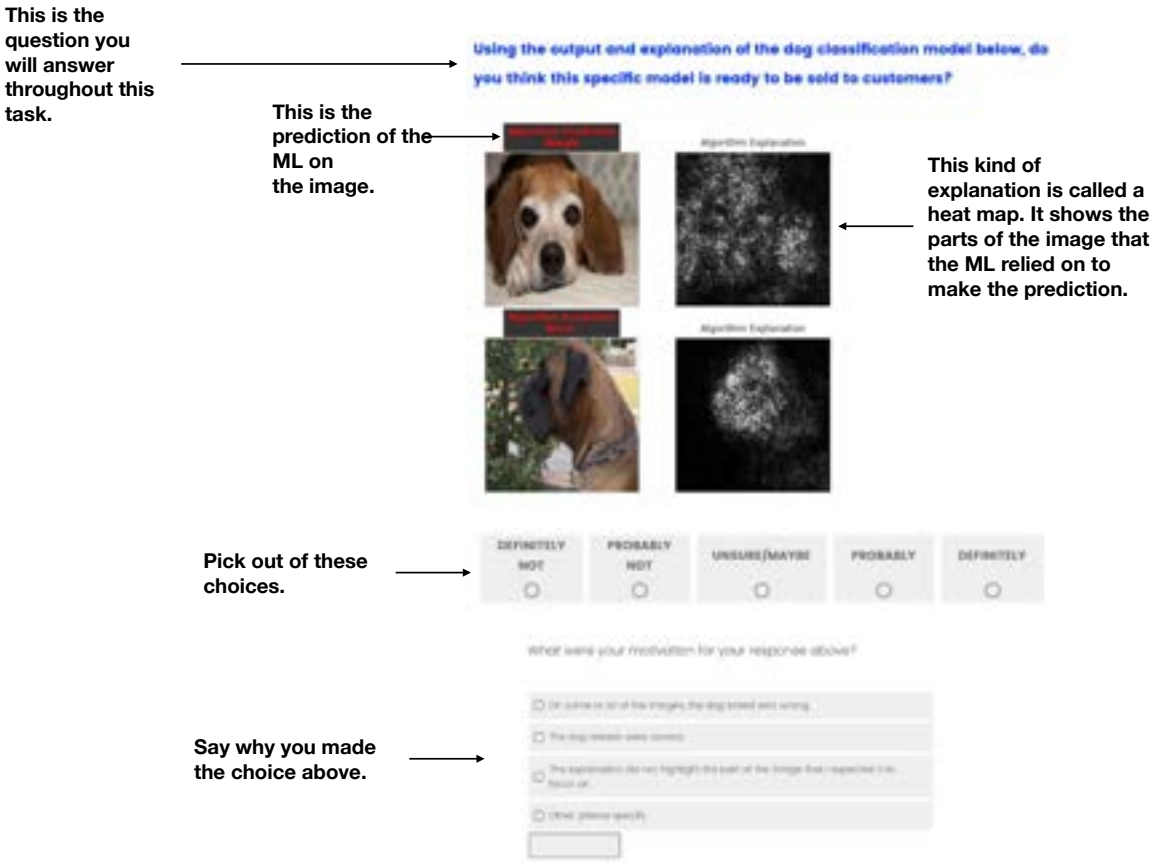

Figure 4-9: Overview of the interface as it was shown to the users.

Terrier, Wheaten Terrier. In total, each class of dogs consists of 400 images in total making 4000 images. We had the following train-validation-test split: (350, 25, 25) images. Each split was created in an IID manner. The Oxford portion of the Dogs datasets includes a segmentation map that we used to manipulate the background of the images. The Stanford dogs dataset includes bounding box coordinates that we used to crop the images for the spurious versions of the data sets that we created. We consider the following model bugs:

- Normal Model: In this case, we have the standard dataset without alteration. This is the control.
- (Model parameter bug) Top-Layer: Here we make no changes to the data set, but re-intialize the DNN model's top layer.
- (Model parameter bug) Half-Way: Here we re-intialize the top half of a trained model's parameters.

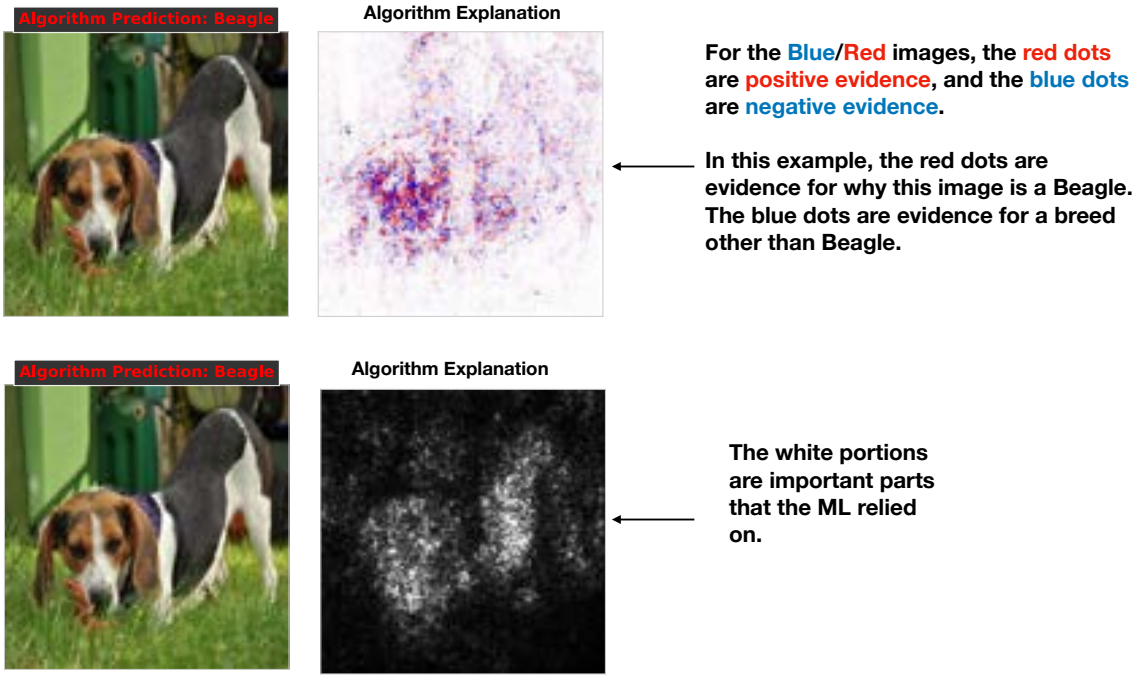

Figure 4-10: Description of the different attribution signs as it was explained to users.

- (Noisy training labels bug) Random Labels: Here we created a version of the dataset where all the training labels were randomized. This is an extreme version of the noisy training label setting.

- (Spurious correlation bug) Partial-Spurious Correlation: Here we created a version of the dataset where 75 percent of the training images had their backgrounds replaced. Here a particular class is associated with a single background type. The remaining 25 percent of the training set have regular backgrounds. We provide example images of the background in Figures 30-34.

- (Spurious correlation bug) Spurious: Here we replace the background on all training points with the background that was pre-associated with that specific class. Note, here that we also test on the spurious images as well.

- (OOD bug) Out-of Distribution. Here we apply a normal model on breeds of dogs that were not seen during training. As expected, the model outputs the wrong predictions on these inputs.

Dog Species	Associated Spurious Background
Beagle	Canyon
Boxer	Empyt Room
Chihuahua	Blue Sky
Newfoundland	Sand Dunes
Saint Bernard	Water Fall
Pugs	High Way
Pomeranian	Track
Great Pyrenees	Snow
Yorkshire Terrier	Bamboo
Wheaten Terrier	Wheat Field

Models. Across all model conditions, we fine-tune a ResNet-50 model for the 10-way classification task. More specifically, we do not freeze any of the lower layer parameters, but update them. In each case, except the Random Labels model, we train for 10 Epochs over the training set. In the case of the random labels model, we train for 20 Epochs. In the table below, we indicate the model performance on the test-set. In the case of the spurious and partially-spurious model, their performance is reported on a test set containing similarly spurious artifacts.

Model Condition	Top-1 Test Accuracy
Normal Model	91.0
Top-Layer Random	9.8
Half-Way Random	9.2
Random Labels	11.8
Partial-Spurious Correlation	89.0
Spurious Correlation	99.1

Study Design. As originally noted, the task at hand is that of classifying images of Dogs into different breeds. We manually selected 10 breeds of dogs based on authors' familiarity. All model conditions were trained to perform a 10-way classification task. As part of the recruitment, participants were directed to an online survey platform (Qualtrics survey). Once the task was clicked, they were presented a consent form that outlined the aim and

motivation of the study. We then provided a quick guide on the breeds of dogs the model was trained on, the set-up, and study interface.

Participants were asked to take on the role of a quality assurance tester at a machine leaning start-up that sells animal classification models. The goal of the study was then for them to assess the model, using both the model labels and the attributions provided. For each condition, each participant was shown images of dogs along with model labels and attributions for a specific model condition. The participant was then asked: **using the output and explanation of the dog classification model below, do you think this specific model is ready to be sold to customers?** Participants were asked to then respond on a 5-point Likert scale with options ranging from Definitely Not to Definitely. A second sub-question also asked participants to provide the motivation for their choice. Taken together, each participant was asked 21 unique questions and 1 repeat as an attention check. We include images of the example and interface below.

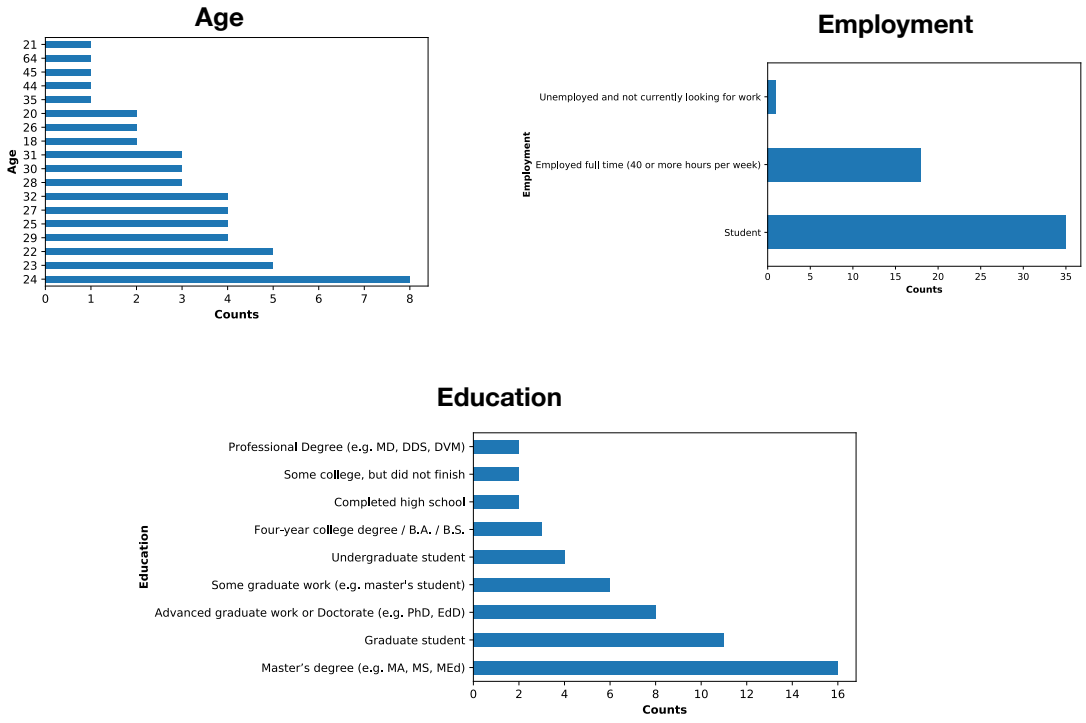

Figure 4-11: Age, Employment, and Education break-down among participants.

Taken together, this design translates to 3-non-overlapping datasets, with 21 distinctive items. In order to mitigate learning effects, participants could only see a certain type of image using one of the model attribution method and manipulation technique. For these

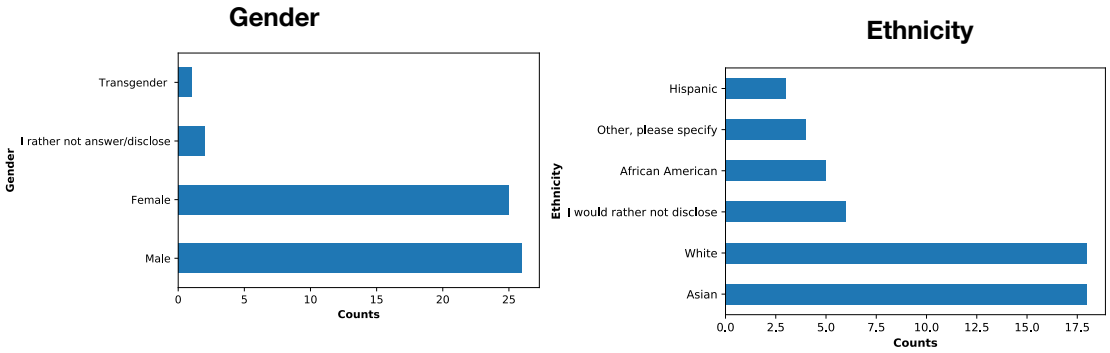

Figure 4-12: Gender, and Ethnicity break-down among participants.

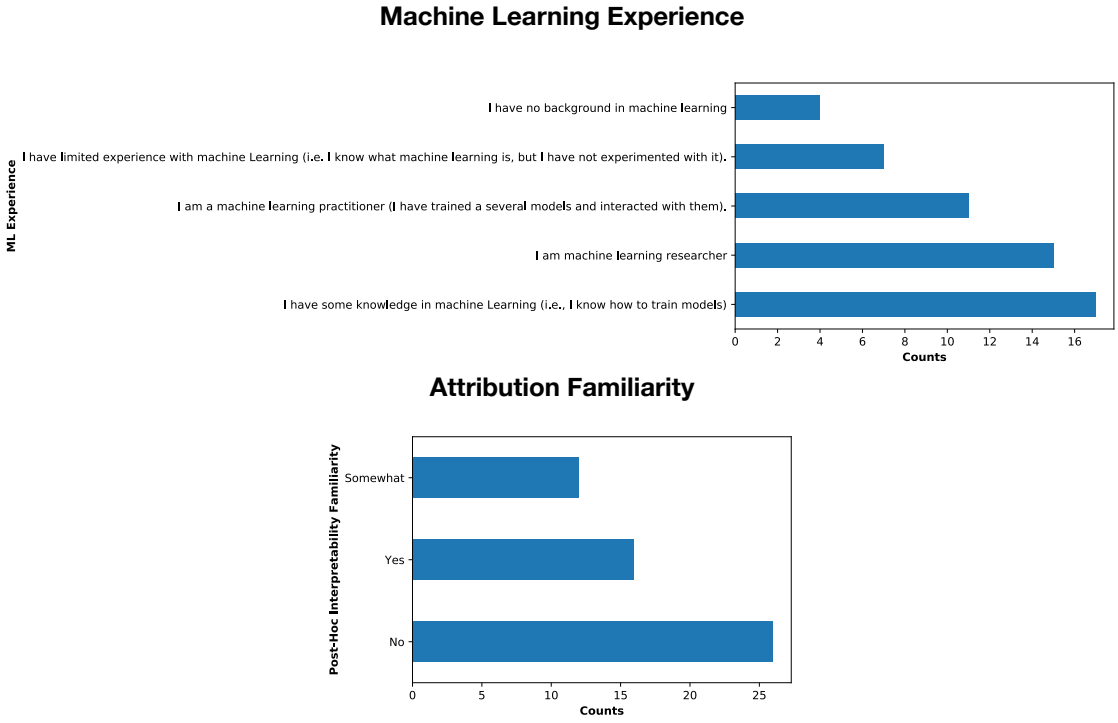

Figure 4-13: ML Experience and Attribution Familiarity break-down among participants.

reasons, we used a between-group design by dividing participants into three groups. For each of the 21 datasets, one group saw Gradient method, another saw Integrated Gradients, and the last saw SmoothGrad with the same manipulation technique applied. We ensured that each dataset was seen only once by one group, and each group saw exactly 3 model attribution methods in each of the 7 model conditions.

Participants and Demographic Information. Here we report on demographic information for the participant group that we examine. The age ranges from 18 to 64. The modal age is 24. A majority of the participants are students and staff at a North-American institution. Of the 54 participants considered, 26 are female. Combined, the top two groups of the participants are White and Asian by Ethnicity. We asked participants to self-assess and report their level of expertise in machine learning. In addition, we asked a simple test question on the effect of parameter regularization. More then 80% of the participants reported past experience with machine learning.

Statistical Analysis. We perform a series of statistical tests to assess significance. The bedrock of our analysis centered on the Oneway Anova along with an All Pairs Tukey-Kramer test. Given the study design, we perform a one-way Anova for each manipulation condition since these are separate conditions. Recall that the goal of our study is to understand which model attributions are effective for identifying specific bugs. Consequently, we perform a one-way ANOVA for each manipulation and 3 model attribution pair. In total, we perform 7 one-way ANOVA means comparision. In each case, the degrees of freedom is 2, with a total sample count of 54. We also estimate the lower and upper 95 percent confidence interval for the Likert means of each model attribution method.

Summary of Findings. As observed in Figure 4-14, we find that across all model attribution types, users were willing to recommend that a normal model be sold to external customers. In the case of models with random parameters or one that was trained on random labels, across all model attribution types, users were able to successfully decline that the model be sold to external customers. From Figures 4-14, we see that participants are mostly relying on the labels to make this assessment. This suggests that for a task that prior knowledge confers domain expertise, the label alone can provide enough signal that overwhelms the need to inspect the model attributions. For a model trained on completely spurious data, there doesn't seem to be a clear definitive take away from the study, as user recommendations vary widely in this setting. Similarly for the test-time contamination tests. Taken together, our results suggest that, in the presence of model labels, users are able to reliably use model attributions to decline to recommend models trained on random labels, and those with model contamination defects.

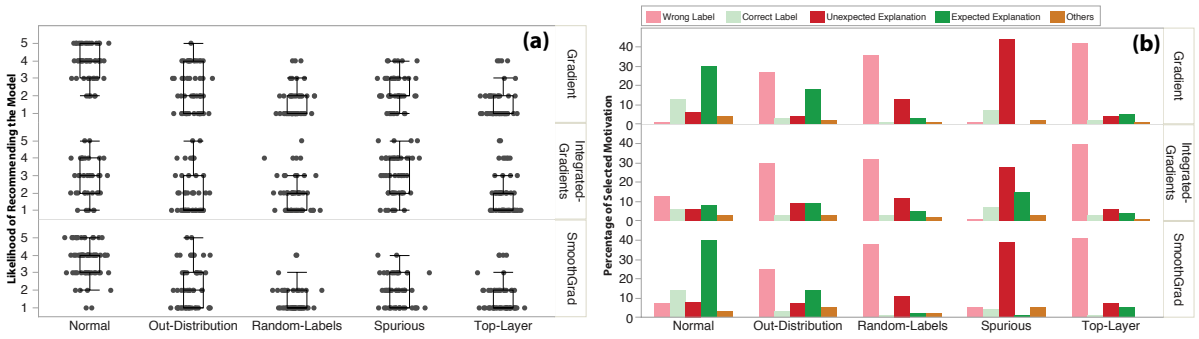

Figure 4-14: **Left: Participant Responses from User Study.** We show a Box plot of participants responses across the three different attribution methods: *Gradient, SmoothGrad, and Integrated Gradients*, and 7 model conditions. On the vertical axis, we have the user reported likelihood of recommendation that a model condition be sold to external customers, from 1 : *Definitely Not* to 5 : *Definitely*. We see that across all attribution types, the participants tend to recommend that a normally trained model be sold to external customers. **Right: Motivation for Recommendation.** We show an overview of the breakdown in motivation according to both attribution and model manipulation type. On the Y axis we have the Likert recommendation by the participant. Higher means the participants are more likely to recommend this action. On the X-axis we have the motivation for selecting the different Likert recommendations. The values on the bars represent the average recommendation for each motivation. We now provide the legend for the motivation: 1: On some or all of the images, the dog breed was wrong; 2: The explanation did not highlight the part of the image that I expected it to focus on; 3: The explanation highlighted the parts of the image that I expected it to focus on; 4: The dog breeds were correct; & 5: Other, please specify.

4.7 Related Work

Han et al. [2020] demonstrate that training point ranking via influence functions is able to identify the dependence of an NLP model on dataset artifacts. In addition, they show correspondence between the insights observed with the input-gradient feature attribution and the training point ranking. Along similar lines, Guo et al. [2020] present fast approximations for computing the training point ranking for a test point. In addition, they show how to identify and correct model errors in a natural language task.

Post hoc explanations, more generally, have been shown to be able to identify a model's reliance on spurious training signals [DeGrave et al., 2020, Lapuschkin et al., 2019, Meng et al., 2018, Ribeiro et al., 2016, Ross et al., 2017]. Recent work by Rieger et al. [2020] showed that regularizing model attributions during training can help lead to models that avoid spurious correlation and enable improved debugging by experts. Similarly, Erion et al. [2019] show that regularizing the expected gradient attribution during training confers similar

benefits. Koh and Liang [2017] used influence functions to identify domain shift. Kim et al. [2018] also perform a user study to understand if attribution methods can be used for catch spurious correlation.

However, similar methods have also been shown to struggle in the hands of end-users for diagnosing model errors [Alqaraawi et al., 2020]. This contradiction reflects the challenge that we explore in this work. Often, post hoc explanations have been shown to be effective for identifying spurious signals that were suspected or known a priori; however, these methods seem to struggle when confronted with the task of identifying an unexpected spurious signal.

Increasingly, insights into why overparametrized DNNs rely on spurious training set signals is starting to be theoretically and empirically analyzed [Khani and Liang, 2020, Nagarajan et al., 2020, Sagawa et al., 2019, 2020], yet it is still unclear how to reliably detect that a model is relying on such signals prior to model deployment.

Assessing whether a post hoc explanation approach is faithful to the underlying model being explained has been addressed in recent works, yet this challenge remains elusive [Hooker et al., 2019, Tomsett et al., 2020]. Generally, the class of approaches that modify back-propagation with positive aggregation have been shown to be invariant to the higher layer parameters of a DNN [Adebayo et al., 2018b, Mahendran and Vedaldi, 2016, Nie et al., 2018, Sixt et al., 2019]. In an intriguing demonstration, Srinivas and Fleuret [2021] show that the input-gradient, a key feature attribution primitive, might not capture discriminative signals about input sensitivity. Instead they show that input-gradient likely captures the ability of the model to be able to generate class-conditional inputs.

User studies are typically the classic approach for evaluating the effective of an explanation [Doshi-Velez and Kim, 2017]. Poursabzi-Sangdeh et al. [2018] manipulate the features of a predictive model trained to predict housing prices to assess how well end-users can identify model mistakes. Their results indicate that users found it challenging to debug these linear models with the model coefficients. Recent work by Chu et al. [2020] and Shen and Huang [2020] has shown similar results in the DNN setting as well. Alqaraawi et al. [2020] find that the LRP explanation method improves participant understanding of model behavior for an image classification task, but provides limited utility to end-users when predicting the model's output on new inputs.

Yeh et al. [2020] and Ghorbani et al. [2019b] both present approaches to automatically discover concepts and quantify a model's dependence on these concepts. These approaches

are promising directions for addressing the challenge we identify in this work. Ghorbani et al. [2019b]'s approach, ACE, segments input images and clusters to discover inherent clusters. We present some analysis on this approach in the appendix. Critically, this approach would identify spurious signals that the underlying segmentation algorithm can discover such the image tag, but not the blur or other visually imperceptible features. Koh et al. [2020] and Chen et al. [2020] present approaches that learn DNN models whose features inherently map onto concepts of interest. In this work, we assume the model is given, focusing on post hoc explanations for models that are not inherently interpretable. Recent work at the intersection of causal inference and explanations might also open up avenues to help reveal unexpected confounding, some of which could be unknown spurious signals. Along this line, [Bahadori and Heckerman, 2021] present an instrumental variable approach for debiasing concept based explanations that might be confounded. [Kazhdan et al., 2020] present CME, an approach identify the important concepts that can help improve a model's performance.

We rely on Guo et al. [2020] for fast approximations for computing the training point ranking for a test point. In addition, they show how to identify and correct model errors in a natural language task.However, [Basu et al., 2020] show influence functions for DNNs are fragile and perhaps inaccurate for deeper networks.

Other recent work has cast doubt on the utility of trying to explain 'traditionally' trained deep network models. For example, [Srinivas and Fleuret, 2021] show that the input-gradient might not reflect the discriminative capabilities of a DNN, but instead encode for an implicit density model. More recently, Shah et al. [2021] show that the input and loss gradients of traditionally trained models *do not* indicate the important features that a DNN model relies on for its output—a phenomenon they term feature leakage. Further they show that adversarially trained models do not exhibit feature leakage. Taken together these results might explain some of the counter intuitive findings that we observe even when the spurious signal is known since we only consider non-adversarially trained models in this work. Along a different direction, Post hoc explanations have been shown to be fragile and very easily manipulated [Anders et al., 2020, Dombrowski et al., 2019, Ghorbani et al., 2019a, Heo et al., 2019, Lakkaraju and Bastani, 2020, Slack et al., 2020]. Our work tackles a difference concern: whether they are suitable for detecting unexpected spurious training set signals.

4.8 Epilogue

This chapter is focused on empirical assessing post hoc explanation methods in order to better understand the settings under which they are effective for detecting model bugs [Adebayo et al., 2020, 2022b, Arun et al., 2021]. One argument against the line of work pursued in this chapter is that the approaches that we tested were not explicitly designed to address model debugging. In a lot of the papers that introduced these methods, the goal was to introduce a general diagnostic tool to better understand model decisions. We agree with this critique. However, one of the primary motivations for this line of work is that post hoc explanation methods, in particular, feature attribution approaches in the image setting, are starting to be used as defacto model debugging tools. However, it is not clear whether the insights produced by these tools can enable model debugging. The goal of the analysis, in this chapter, is not to unfairly criticize post hoc explanations, but to subject them to rigorous tests, and to better understand when conclusions derived from these approaches can be 'trusted'.

Limitations. It is important to point out some caveats to our results. First, as with an empirical work, our insights might not generalize to other tasks, data modalities, or settings that differ from the one we considered. To demonstrate this limitation, we will consider the one feature attribution method: Guided Backprop.

For DNNs trained on ImageNet and MNIST, when the higher level layers of the network is re-initialized, the Guided Backprop feature attribution does not change. At first glance, this behavior suggests that Guided Backprop should be jettisoned as a feature attribution method. However, in the spurious signal detection setting, when the signal is known, Guided Backprop is one of the best performing methods. Indeed, feature attribution methods that modify the backpropagation process with a positive relu, i.e., zero out the negative entries of the intermediate matrix vector products, behave this way. In fact, Sixt et al. [2019] proved a theorem demonstrating this phenomenon. Despite such daming evidence of ineffectiveness, Yona and Greenfeld [2021] recently demonstrated that Guided Backprop can be made sensitive to the weights of trained model, on MNIST and ImageNet, for certain tasks. This means that the effectiveness of an attribution method might be dependent on the task, model architecture, and other elements in the ML pipeline.

To further drive home the point that an attribution method's effectiveness might be

dependent on the task and details of model training, Shah et al. [2021] show that the input of traditionally trained models *do not* highlight discriminative features, while those of adversarially trained model do. This might mean that while the explanation methods we tested here might not be effective for standard models, they are for adversarially robust models!

As of the writing of this thesis, the dust on how to effectively assess post hoc explanation methods has not yet settled.

Chapter 5

Model Guiding

5.1 Overview

In this section, we present the essence of model guiding. We start with a discussion of why this kind of approach is needed, and then discuss each component of the model guiding formulation.

What is missing in our current tools? In the previous chapter, we empirically assessed different explanation approaches, and ultimately showed that current methods struggle with detecting spurious signals and mislabelled training examples. Even when current methods are able to detect the presence of a model bug, they usually provide no avenue to fix the bug.

Model guiding directly aims to address these challenges. First, it aims to enable interaction between a task expert that wants to audit a machine learning model and the trained model. Using the model's output and feature importance estimates as it communication medium, model guiding uses a well-curated audit dataset to update the parameters of an already trained model. A second added benefit of model guiding is that it serves as a way for a task expert to *inject more domain knowledge into the model*.

Lending Example. Throughout this section, we will use a lending task as way to demonstrate the method. In the lending task, the goal is to decide the best credit limit for an individual given the individual's socio-demographic variables. We assume the input to the DNN is a collection of all available information from their credit report, pre-processed into text format, and fed to a DNN with appropriate architecture. The model is then trained

end-to-end via stochastic gradient descent. The task described is a regression one, and secondly it uses a DNN as its base model. For pedagogical reasons, we present the model guiding formulation in the regression setting. With slight modification, the framework can also be extended to classification tasks; indeed, most of the empirical results we present in this work are classification tasks. We use a DNN as the base pre-trained model since it corresponds to the most difficult to address interpretability setting. As we'll see, simpler model classes like linear regression fiteasily within the proposed framework.

5.2 Model Debugging with Model Guiding

We now state the high-level model guiding scheme:

Model Guiding Requirements: Given an already trained model, a concept library provided by a task expert, model guiding uses a small, well-annotated, audit set that includes:

- carefully checked labels for each sample in the audit set,
- a feature annotation vector that for each sample,
- a vector, for each instance, that indicates how important each feature/concept, in the global concept dictionary, is;
- a score (between [0-1]), for each sample, that denotes how confident the domain expert is in their annotation for that sample;

to update the parameters of the already trained model, and propose a new set of training labels for all training samples, so that:

1. **(Requirement 1)** the updated model's prediction, on the audit samples, matches the labels provided by the domain expert; and
2. **(Requirement 2)** the updated model's feature importance scores, on the audit samples, matches the feature importance scores provided by the domain expert.

We will work up to the entire model guiding scheme. Here, we start with the general requirements that the proposed scheme needs to satisfy.

At a high level, model guiding uses an audit set to post-process an already trained model,

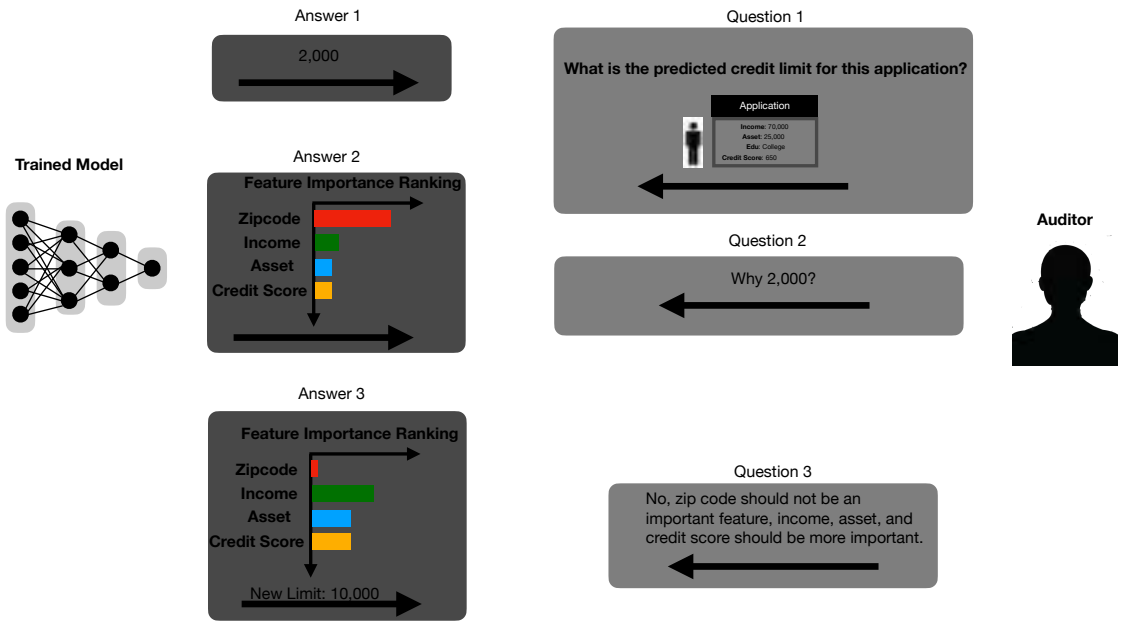

Figure 5-1: Model debugging as a dialog with an auditor.

so that the model obtained after the post-processing has behavior similar to what a task expert has annotated on a audit set. The key goal of model guiding is to use the audit set as a form of 'unit' test for the pre-trained model. On this audit set, the task expert has helped to provide annotations on labels and features that are important for each example. Now, model guiding then formulates an optimization problem that uses the audit set to both: 1) detect an already trained model's mistakes, and 2) fix them. In the rest of this chapter, we take steps towards making the aforementioned requirements more concrete.

With the high level scheme established, we will now discuss details such as the concept library, and the audit set.

5.2.1 Model Guiding Pre-Processing Step: Audit Set Annotation & Model Carving

As stated so far, we have the high-level goal for model guiding. However, before we move to discussing the mathematical formulation of the proposed objective, we provide an overview of the pre-cursors that need to be in-place to apply model guiding.

Inputs: The primary inputs that model guiding requires are:

1. the training set (features and labels), $\{(x_i, y_i)\}_{i=1}^n$ that was used to train the model that we want to post-process;

2. an already trained model, $f_\theta : x_i \to y_i$, that maps from the inputs to the output. Here we will assume that $x \in \mathbb{R}^d$, and $y_i \in \mathbb{R}$. We will sometimes represent the model simply as θ, the vector of parameters. Later in the chapter, we will discuss how to extend the setup to classification. We will assume that the model, f_θ, consists of an encoder $g : x_i \to z_i$, that maps the input to a latent code, and $h : z_i \to y_i$, a linear classifier that maps the latent code into the output label. Consequently, $y_i = f(x_i) = h(g(x_i))$;

3. a set of m audit samples, $\{\tilde{x}_i\}_{i=1}^m$, which we term the audit set. The number of examples in the audit set, m, should be much smaller than the training set size. In our tasks, m, ranged from 25 to 150.

We need the original training set because we will identify inputs with mislabelled examples in the set. The m audit samples will be used to post-process the pre-trained model.

Concept Library. A concept library is simply a list/dictionary of the important features that an expert has deemed relevant for a task at hand. In the case of the lending example, an expert might surmise that an individual's income, asset, credit score, and level education are features that ought to be important for the task of credit limit estimation. If these concepts are chosen, then the concept library then consists of the four features listed above. We let d be the size of the concept library.

Why do we need a concept library? As we previously noted, the goal of model guiding is to enable 'communication' between a model and a domain expert. However, for this communication to occur, both the model and the domain expert need a common code of expression. The concept library serves as such a code. The pre-trained model will be transformed such that its intermediate representations can be expressed as a weighted combination of the concepts in the expert pre-specified library.

Audit Data Annotation. Once the concept library has been specified by the task expert, we will now move to annotate the m audit examples. As previously mentioned, the audit set, $\{\tilde{x}_i\}_{i=1}^m$, will serve as a external unit test for the model, and a way for the domain expert to further inject prior knowledge into the ML model. As with all unit tests, we will

need to know the 'correct answers'. These answers will be in the form of labels and feature importance scores, for each feature in the correct library, that a task expert expects for a reliable model. Concretely, the task expert will need to carefully annotate the audit set so that for each sample in the audit set, we have:

1. A **Label**, $\tilde{y}_i$, that indicates what the expected output an ideal ML model should be on this input. In the lending example, this is the 'true' expected credit limit for a credit report in the audit set.

2. A **feature/concept annotation vector**, $\tilde{a}_i$, where the domain expert gives the value of each feature/concept, in the concept library, for that sample. For example, in the lending scenario previously considered, given an individual, x_i's credit report, a possible feature vector is: {"income": 50k USD, "asset": 100k USD, "credit score": 650, "Education": College}. As expected, the above representation can be easily transformed into a feature vector.

3. A **feature/concept importance vector**, $\tilde{s}_i$, that indicates how important each concept is for the specific instance in question. In this work, feature importance takes on 3 possible values: $\{-1, 0, 1\}$. A '-1' indicates that the feature is negatively relevant to the output, a '0' indicates no relevance, while '$+1$' indicates that the feature is positively relevant. Building on the lending example above, an example feature importance vector is: {"income": $+1$, "asset": $+1$, "credit score": $+1$, "Education": 0}, which says that the specific individual's income, asset, and credit are all positively relevant towards the credit limit, but education has no relevance. Even though we ask the domain expert to provide instance level feature importance, it could be the case that the task is amenable to global feature importance, in which case the domain expert only provides a single feature importance vector for all samples.

4. A **confidence score**, $c_i \in [0, 1]$, that indicates the task expert's level of confidence regarding their annotation.

To recap, we have now defined a concept library, and also annotated the audit set. At the end of this process, the audit set, which previously consisted of only of sample features, $\{\tilde{x}_i\}_{i=1}^m \xrightarrow{\text{annotation}} \{\tilde{x}_i, \tilde{y}_i, a_i, s_i, c_i\}_{i=1}^m$ has now been annotated with labels, $\tilde{y}_i$, concept/feature values, , a_i, feature importance, s_i, and also a sample confidence score.

Model Carving: Transforming a black-box DNN model into an interpretable one. The final preliminary step involves converting the black-box DNN model into a concept bottleneck model (CBM) [Koh et al., 2020]. The goal of this step is to convert the DNN model, whose representations might be in a basis that the task expert does not understand, into one that matches the concept library provided by the task expert.

A CBM is a DNN model that requires one of its layers to map directly to human-interpretable concepts. Revisiting the lending example, the CBM version of the original DNN will first map the raw inputs to the a concept vector matching income, asset, credit score, and educational level. The final output is then obtained as a linear model mapping from the concepts to the label. The goal of the CBM is to ensure that the coefficients of the higher level concepts can then be interpreted as feature importance for the model. To train a CBM model, each training input must be annotated with a higher level concept; however, we don't have these feature annotations for the entire training set. Instead, we will convert an already trained DNN model into a CBM one post hoc using the technique of Yuksekgonul et al. [2022].

We now give a overview of the approach for converting a DNN into a post hoc CBM model. Instead of annotating the entire training set, the post hoc CBM requires a small, auxiliary dataset that has been annotated with higher level concepts as a way to convert the black-box DNN into a CBM.

Let's define a concept library to be $I = \{i_1, i_2, \ldots, i_d\}$ with d concepts. Using the lending example, i_1 is the income concept, and i_2 is the asset concept etc. The steps for converting a black-box model into a post-hoc CBM are as follows:

1. **Collect Data**: For each concept, i, in the concept library, if the concept is a binary variable then collect positive and negative examples for the concept. If the concept is scalar, then simply collect the scalar annotations for all examples available in the audit set.

2. **Train Model**: For binary or categorical concepts, train a linear classifier to distinguish between positive and negative examples. The features of the classifier are derived from an embedding layer in the black-box DNN. For concepts with scalar output, like income, train a linear regression model to predict the output scalar. Let, f_i, be the function that maps from an input to the label(whether the concept is present or not)

for an input. Since f_i is a linear model, it can be represented as, $\beta_i^\top \tilde{x}_i$. The concept activation vector for concept, i, is then β_i. We let $B \in \mathbb{R}^{d \times l}$, where each row of B is a concept β_i.

3. **Constrain the original model with new basis**: Given the matrix, B, and an input in the training set, x_i, we can project all these inputs onto the subspace spanned by the concept matrix. This gives us each input in terms of the interpretable concept dictionary.

4. **Learn the parameters of the new basis**: The final step of the conversion process is to learn a linear model from the concept basis to the output, for all the training points. We term the parameters of this final model: θ.

With the above steps, we are now able to transform a black-box DNN model into a CBM that is a composition of an encoder, and a simple linear classifier from the expert-defined concept space to the labels. Revisiting our lending example, the original DNN model took in credit reports and directly mapped them to a loan amount. Now the new CBM model takes in credit reports, maps them to the concept space, i.e. income, asset etc, then a linear model maps the concepts to the final label.

Summary We have now concluded the setup needed for model guiding. We have annotated the audit dataset, and also converted a pre-trained DNN into a CBM. With these two precursors in place, we are now ready to formulate the model guiding objective.

5.2.2 The Bilevel Update: Formulation

We now discuss the core component of model guiding, which is the bilevel optimization objective that helps update 1) the training labels, and 2) the model parameters using the well annotated audit dataset.

Based on the model guiding requirements, the bilevel optimization problem can be stated as follows:

$$\text{Upper Level Objective}: \min_{\delta \in \mathbb{R}^n; \theta} \overbrace{\gamma \frac{\|\delta\|_1}{n}}^{\substack{\text{Perturbation to} \\ \text{Training label.}}} + \overbrace{\frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i + \delta_i; \theta)}^{\substack{\text{New labels should be consistent} \\ \text{with original training data.}}} + \overbrace{\frac{1}{m}\sum_{i=1}^{m} c_i \ell(\tilde{x}_i, \tilde{y}_i; \theta)}^{\substack{\text{Match expert's} \\ \text{labels on} \\ \text{audit set.}}} + \|\overbrace{\theta \odot \tilde{x}_i - \tilde{a} \odot s_i}^{\substack{\text{Match expert's} \\ \text{feature importance} \\ \text{annotations} \\ \text{on the audit set.}}}\|_1;$$

$$\underset{\text{Level Objective}}{\text{Lower}} \text{ s.t. } \theta = \arg\min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i + \delta_i; \theta).$$

Upper Level Objective We now consider each term in the upper level objective, and explain the intuition behind the term's inclusion.

- $\gamma \frac{\|\delta\|_1}{n}$: $\delta \in \mathbb{R}^n$ represents a small perturbation that is added to a training input's label. Given an instance: x_i, whose original label is y_i, after solving the bilevel optimization problem above, we obtain the new label as: $y_{i,\text{new}} = y_i + \delta_i$. The term, $\gamma \frac{\|\delta\|_1}{n}$, is the average perturbation across all training samples weighted by a hyper-parameter γ. We seek to learn a perturbation that is as small as possible.

- $\frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i + \delta_i; \theta)$: this term ensures that the perturbed labels, $y_{i,\text{new}} = y_i + \delta_i$, still help minimize the loss on the training set, ultimately resulting in a well-performing model on the original training data.

- $\frac{1}{m} \sum_{i=1}^{m} c_i \ell(\tilde{x}_i, \tilde{y}_i; \theta)$: This term minimizes the model's loss on the audit set. The tuple, $(\tilde{x}_i, \tilde{y}_i)$, is a sample from the annotated audit set. This term ensures that the parameters of the updated model yield predictions, on samples in the audit set, that match the labels provided by the task expert.

- $\|\theta \odot \tilde{x}_i - \tilde{a} \odot s_i\|_1$: This term ensures that the parameters of the updated model yield concept annotations, on the sample in the audit set, that match the concept annotations provided by the task expert. Here, we note that the notation, $\odot$ refers to the Hadamard (element-wise) product.

Lower level Objective The lower level objective seeks to obtain a set of parameters that minimize the model's loss on the perturbed labels.

We have now transformed the requirements of the model guiding scheme into an optimization problem.

5.2.3 Solving a Bilevel Optimization problem

The formulation proposed in the previous section is an example of a bilevel optimization problem [Colson et al., 2007, Franceschi et al., 2018, Maclaurin et al., 2015]. This is because

one of the constraints of the objective requires solving an optimization problem. Here we provide a high level discussion of our approach to solving the bilevel optimization problem using tools from the literature. To keep the discussion contained, we will mostly discuss our approach at a high level, and refer the interested reader to additional sources.

Automatic differentiation tools have made it easy to compute gradients needed for optimization algorithms that minimize a model's loss. However, these tools typically do not differentiate through a solver, and it is unclear how such tools can be adapted to solving bilevel problems. However, as we will discuss later, recent literature on bilevel optimization has resulted in tools that can similarly differentiate through a solver, akin to how automatic differentiation tools are able to differentiate a model's loss. We will return to this later.

Bilevel optimization is now an increasingly useful tool in machine learning for formulating hyper-parameter optimization. There is an emerging toolbox of approaches for solving these problem. We provide an overview of these approaches here. This discussion is derived form the recent work of Blondel et al. [2021] on an automatic bilevel solver, Kolter et al. [2021]'s Neurips 2021 tutorial on Deep Implicit Layers, and Grosse [2022]'s course notes on bilevel optimization.

In our discussion, we will consider solving the abstracted bilevel problem specified as follows: :

$$\begin{aligned} &\underset{\text{Level Objective}}{\text{Upper}}: \quad \lambda^* \in \arg\min_{\lambda} \mathcal{J}_{\text{upper}}(\lambda, \theta^*) \\ \text{s.t.} \quad &\underset{\text{Level Objective}}{\text{Lower}}: \quad \theta^* \in \arg\min_{\theta} \mathcal{J}_{\text{lower}}(\lambda, \theta). \end{aligned}$$

The bilevel formulation for model guiding is an instantiation of the above formulation.

Karush-Kuhn-Tucker (KKT) Based Solution. If the lower level problem satisfies certain regularity conditions (i.e. unconstrained and strongly-convex), then a general strategy for solving bilevel optimization problems is to replace the lower level problem with its KKT condition. One can then relate the problem's solution to the inputs and manually derive the gradients of the entire relation, which can then be used as part of a gradient descent scheme. The work of Zhang et al. [2018] that we build upon follows such an approach. However, that approach can be tedious, and involves manual derivations for every new problem instance.

Gradient Descent on the Hyper-Gradient The approach we take is to perform gradient-descent on the hyper-gradient. This will allow us to take advantage of automatic differentiation tools. In the generic bilevel formulation presented, we seek to solve for λ via gradient descent, so we need the total gradient of $\mathcal{J}$ with respect to the optimization parameter λ. Using the chain rule, this can be expressed as:

$$\frac{d}{d\lambda}[\mathcal{J}(\lambda, r(\lambda))] = \frac{\partial \mathcal{J}}{\partial \lambda}(\lambda, r(\lambda)) + \frac{\partial r}{\partial \lambda}(\lambda)^\top \frac{\partial \mathcal{J}}{\partial \theta}(\lambda, r(\lambda)).$$

Based on previous work [Koh and Liang, 2017], we can usually set the first term to zero [Grosse, 2022]. The term, $\frac{\partial r}{\partial \lambda}(\lambda)$ is known as the response Jacobian, and it measures how the optimal solution changes due to infinitesimal perturbations to λ. Following Koh and Liang [2017], we can obtain the formula for the response gradient as:

$$\frac{\partial r}{\partial \lambda}(\lambda) = -\left(\frac{\partial^2 \mathcal{J}}{\partial^2 \theta}(\lambda, r(\lambda))\right)^{-1} \frac{\partial^2 \mathcal{J}}{\partial \lambda \partial \theta}(\lambda, r(\lambda)).$$

The details of the derivation is analogous to the derivation provided in the influence functions section of Chapter 3, so we skip the details. Note that, $\frac{\partial^2 \mathcal{J}}{\partial^2 \theta}(\lambda, r(\lambda))$ is the hessian matrix: H. Now, we have a closed-form formula for the hyper-gradient.

Computing the hyper-gradient The hyper-gradient is what we need to optimize the bilevel formulation via gradient descent. There are broadly two ways for estimating the hyper-gradient: 1) solving the linear system of equations above with a method like conjugate gradients or 2) unrolling the inner optimization and then backproping through it. In this work, we go with the first approach and solve the linear systems of equations. The first approach is increasingly standard practice in solving bilevel optimization problems.

Automatic Implicit Differentiation. In exciting recent work, Blondel et al. [2021] provide a way to perform automatic implicit differentiation. Their proposed approach subsumes the previous discussion. In their formulation, the user provides python expressions of the bilevel formulation, as well as the optimality conditions. Blondel et al. [2021]'s framework can the automatically differentiate the optimization problem. Their approach provides a

clean interface that side-steps some of the challenges discussed above.

Returning to model guiding As it stands, we now know how to solve the bilevel optimization problem that we formulated. Consequently, we now have all the ingredients necessary to perform the entire model guiding scheme.

5.2.4 The Model Guiding Scheme

We now discuss the end-to-end model guiding scheme as presented in Algorithm 1.

The Setup & Inputs Recall that model guiding needs the following as inputs: 1) the pre-trained black-box model θ_{init}, 2) a training set: $D_t = \{(x_i, y_i)\}_{i=1}^n$, and 3) the annotated audit set $D_a = \{\tilde{x}_i, \tilde{y}_i, \tilde{a}_i, \tilde{s}_i, c_i\}_{i=1}^m$. As previously noted, we will assume that the model θ_{init} is a DNN. The first step, which we term the carving step, is to convert the model, θ_{init}, into a concept bottleneck model (CBM) as discussed in Section 5.2.1. Recall, that the CBM uses representations of the model θ_{init} to learn a sparse linear model in the task expert's provided concept's basis. Once a model has been converted into a CBM, for any training input, x_i, we can obtain its concept representation, z_i, simply by projecting an embedding of the input onto the concept basis learned in carving step.

Bilevel Update With the inputs setup, the next step is the bilevel update. This step involves solving the bilevel optimization problem from section 5.2.3. In the problem, the optimization variables are: 1) the perturbation to the training labels, and 2) an updated model parameters that is both accurate on the perturbed labels and matches the domain expert's annotations on the audit set. Performing the bilevel update returns a perturbation for the labels as well as a new set of parameters.

Identifying Noisy Label & Spurious Correlation Bugs Based on user provided tolerance parameters, we can then select the inputs that are mislabelled and for which the model was relying on spurious signals. The proposed fix is the perturbation added to the old label. Similarly for spurious signals, we compare the concept attribution of the original model to the concept attribution of the updated model. The samples for which the change in attribution is greater than a pre-specified tolerance factor are the inputs for which the model relies on a spurious signal. These inputs are selected and passed to domain expert to inspect.

Algorithm 1 Model Guiding

Input : 1) θ_{init}, a black-box model or a concept bottleneck model,
2) the training set, $D_t = \{(x_i, y_i)\}_{i=1}^{n}$, and
3) the annotated audit set: $D_a = \{\tilde{x}_i, \tilde{y}_i, \tilde{a}_i, \tilde{s}_i, c_i\}_{i=1}^{m}$.

Hyperparameter : 1) τ_{nl}, noisy label tolerance, and
2) τ_{spu}, spurious correlation tolerance,
3) $b \ll n$, the inspection budget, which is the maximum number of items the domain expert can inspect per round.

if θ_{init} is not a concept bottleneck model **then**
  $\theta_{\text{init}} \xrightarrow{\text{carving}} \theta_{\text{cbm}}$; `/* θ_cbm is a DNN plus a linear model in concept space. */`

Initialize: δ_i, γ_i, $F_{\text{nl}} = \{\}$, $F_{\text{spu}} = \{\}$, and $\theta_i = \theta_{\text{cbm}}$

δ_{i+1}, θ_{i+1}= BilevelUpdate(θ_i, δ_i)

$F_{\text{nl}} = \{i \mid \delta_{i+1} > \tau_{\text{nl}}\}$; `/* indices of labels that violate tolerance. */`

$F_{\text{spu}} = \{i \mid \|\theta_{\text{cbm}} \odot z_i - \theta_{\text{i+1}} \odot z_i\|_1 > \tau_{\text{spu}}\}$; `/* samples that contain spurious signals, where` z_i `is a training label's concept vector. */`

if $|F_{\text{nl}}| > b$ **then**
  F_{nl} = SortFilter(F_{nl}, b) ; `/* Sort the set` F_{nl} `by magnitude of the violation and returns top b. */`

if $|F_{\text{spu}}| > b$ **then**
  F_{spu} = SortFilter(F_{spu}, b) ; `/* Sort the set` F_{spu} `by magnitude of the violation and returns top b. */`

Send F_{spu} F_{nl} to domain expert for inspection.

Modification for Classification Tasks. The formulation we presented here is targeted to regression tasks, where the output is a scalar. The proposed procedure can also be applied to classification tasks as well. The output label is no longer a scalar, but a vector, so δ_i will be represented by a vector on the probability simplex. The newly proposed label will be δ_i and the distance between the old and new label can be measured as $1 - \delta_{i,y_i}$, i.e., how much the old label changed. The final change to the bilevel program is to require that δ_i sum to 1.

We have now concluded an overview of the model guiding scheme.

5.3 Model Guiding for Spurious Correlation & Noisy Training Label Bugs

In this section, we will now discuss the empirical performance of model guiding on various datasets.

Datasets

- Water Birds Dataset([Sagawa et al., 2019]): The waterbirds dataset aims to distinguish

between land birds and water birds. The dataset is generated by combining images of birds with landscapes as background. The goal is to classify birds based on whether they are a land bird or a water bird. The spurious signal in this case is the background of the image. The dataset comes equipped with concepts that capture various parts of the bird.

- Dog Breeds Dataset: We created a modified combination of the Stanford dogs dataset [Khosla et al., 2011] and the Oxford Cats and Dogs datasets [Parkhi et al., 2012b] as our primary data source. We restrict to a 10-class classification task consisting of the following breeds: *Beagle, Boxer, Chihuahua, Newfoundland, Saint Bernard, Pugs, Pomeranian, Great Pyrenees, Yorkshire Terrier, Wheaten Terrier.* The spurious signals we consider here are an image tag and a background blur.

- Credit Dataset: The task is to predict the credit limit for each customer given aggregate demographic and credit information for each customer. We take Gender (Male & Female in the dataset) to be the sensitive attribute of interest, and focus only on a subset of high earners (income $>$ 6000 in local currency), and consider a subset of the features: age, asset, and income. We get two groups: 11,480 males and 2,937 females.

- Bone Age Dataset: This dataset consists of Hand [Halabi et al., 2019] radiographs [Chen et al., 2019]. We consider the high stakes task of predicting the bone age category from a radiograph to one of five classes based on age: Infancy/Toddler, Pre-Puberty, Early/MiD Puberty, Late Puberty, and Post Puberty. This task is one that is routinely performed by radiologists and as been previously studied with a variety of DNN. The dataset we use is derived from the Pediatric Bone Age Machine learning challenge conducted by the radiological society of North America in 2017 [Halabi et al., 2019]. The dataset consists of 12282 training, 1425 validation, and 200 test samples. We resize all images to (299 by 299) grayscale images for model training. We note here that the training, validation, and test set splits correspond to similar splits used for the competition, so we retain this split. The spurious signals we consider here are an image tag, and a background blur.

Model Across all model conditions, we fine-tuned a ResNet-50 model.

Dataset	Model Guiding	Influence Functions	DUTI
Dog Breeds	0.77	0.73	0.80
Water Birds	0.90	0.78	0.92
Bone Age	0.85	0.84	0.86

Table 5.1: Performance (Area under the receiver operating curve AUROC) of model guiding at detecting noisy training labels compared to baselines.

Dataset	Model Guiding	Influence Functions	DUTI
Dog Breeds	0.67	0.60	0.69
Water Birds	0.85	0.72	0.83
Bone Age	0.73	0.61	0.85

Table 5.2: Performance (Area under the receiver operating curve AUROC) of model guiding at suggesting the correct fix for the noisy training labels compared to baselines.

Detecting & Fixing Noisy Training Labels The first task we evaluate model guiding against is the ability to detect mislabelled examples in the training set. For each dataset-model combination we switch the labels of 20 percent of the training set and then train the resnet-50 model on this modified dataset. We compare model guiding to the following baselines: Influence Functions([Koh and Liang, 2017]), and DUTI([Zhang et al., 2018]).

Table 5.1 shows the AUROC of each method computed for all 20 percent training points. We see that model guiding outperforms the influence functions approach, and only slightly underperforms the DUTI method. Model guiding's formulation is based on DUTI, so the close performance of both of these approaches is expected. We then turn to the task of suggesting a fix for the mislabeled example. Specifically, we ask whether the suggested label fixes are correct. Indeed, we find that model guiding and DUTI perform similarly, and both of these approaches outperform the influence functions approach. The primary takeaway from these results are that model guiding matches, DUTI, the current state-of-the-art approach for correcting label error.

Detecting & Fixing Spurious Correlation We now turn to the task of detecting that a model is relying on a spurious signal. A common strategy for identifying inputs for which a model is relying on a spurious signal is to train a simple model (linear or MLP based model) on the dataset and then use the points that the model makes an error on as the points for which a model is relying on a spurious signal [Creager et al., 2021]. We term this approach Errors ERM. This approach is often used as part of a separate strategy for obtaining a model that is robust to spurious correlation. We decouple these two steps and evaluate them

Dataset	Model Guiding	Errors ERM
Dog Breeds	0.89	0.61
Water Birds	0.95	0.71
Bone Age	0.96	0.77
Credit Data	0.84	0.56

Table 5.3: Performance (Area under the receiver operating curve AUROC) of model guiding at suggesting at detecting examples for which a model is relying on a spurious signal.

Dataset	Model Guiding	GDRO	SSA	ERM
Dog Breeds	85.5	63.7	65.3	61.1
Water Birds	93.1	89.2	87.1	72.6
Bone Age	73.0	41.3	48.9	21.6

Table 5.4: The worst-group test set performance of a the post-processed model guiding approach compared to other baselines.

separately.

In Table 5.3, we show the performance of model guiding compared to Errors ERM. We find that model guiding outperforms Errors ERM across the datasets considered. In addition, model guiding also updates the parameters of the pre-trained model in order to stem reliance on spurious signals. We compare the worst-group accuracy of the updated model to the performance of state-of-the-art approaches like group distributionally robust optimization (GDRO)([Sagawa et al., 2019]), and spread spurious attribute (SSA)([Nam et al., 2022]), and a baseline ERM model. Across the datasets considered, model guiding outperforms these approaches.

5.4 Related Work

Why is model guiding effective. In essence, the audit dataset is crucial to the performance of model guiding. Clearly, poor feature annotations and labels on the audit set will further harm the performance of an updated model. Imagine an audit dataset that is a subset of the exact training dataset, then in this case, the model will not updated. Consequently, it is critical that the audit set contain new examples, and that the corresponding feature importance annotations are also accurate.

The model guiding formulation that we present here ([Adebayo and Abelson, 2022]) was inspired by and builds directly on the DUTI algorithm of Zhang et al. [2018]. Intriguingly, DUTI formulates model debugging as a bilevel optimization algorithm, and uses an audit

set to post process a model in order to identify mislabelled training points. Model guiding departs from this work in critical ways: first, we augment the formulation to also identify samples for which the model is relying on a spurious signal, a capability DUTI does not enable. Second, DUTI is tailored to kernel logistic regression Here, we generalize the formulation and and scaled it to DNNs. Our proposal to match feature annotations has also been previously used to learn models that align with a task expert's prior knowledge [Rieger et al., 2020, Ross et al., 2017].

In general, the model guiding formulation proposed here is an instantiation of *machine teaching* [Zhu et al., 2018], where instead of learning from data, the goal is to 'teach' a model. Instead of 'teaching' a model from scratch, we 'guide' an already trained model. DUTI is part of recent iteration of debugging approaches that seek to update an already trained model to either augment it with model debugging capabilities.

Model debugging has become an important problem. A promising direction that is similar in principle to model guiding is model editing [Bau et al., 2020, Santurkar et al., 2021]. In model editing, the goal is to localize undesirable behaviors and then use a rank-1 update to 'erase' the model's dependence on the undesirable behavior. Model guiding is similar to that approach, in principle. The goal is to identify undesirable spurious signals that the model is relying on, and then use the audit dataset to correct such dependence. The model editing approach has been scaled to large language models with intriguing results [Meng et al., 2022, Mitchell et al., 2021, 2022]. Ribeiro and Lundberg [2022] introduced the AdaTest framework that debugs a large language model using tests generated by another large language model in a human-in-the-loop manner.

Increasingly, an emerging niche termed data centric AI has also placed model debugging on a central footing [Polyzotis and Zaharia, 2021]. Recent work in this area has focused on *slice discovery*—the challenge of finding data groups where the model has low performance [Eyuboglu et al., 2022].

A key design choice we made is to convert a black-box DNN model into an interpretable one. We use the approach of Yuksekgonul et al. [2022] to convert DNNs into CBMs. This design choice aligns with increasing calls to jettison black-box DNNs for interpretable-by-design approaches [Rudin, 2019]. Towards such end, Ilyas et al. [2022] develop datamodels, a framework that describes classes of models that map subsets of training inputs to an output. Similar to CBMs, Wong et al. [2021] train sparse linear models on top of a DNN's

representations and show that the resulting model can be analyzed to more easily explain a misclassification, and also identify spurious correlation. Singla and Feizi [2021b] introduced a modified ImageNet dataset that uses a human-in-the-loop routine to detect a model's reliance on spurious training signals.

As the discussion in this Chapter suggest, the scope of work targeted at debugging DNNs is still involving; here, we have limited our scope to work that is directly related to model guiding.

Chapter 6

Debugging Fairness Violations with Influence Functions

6.1 Overview

Label error (noise)— mistakes associated with the label assigned to a data point — in both training and test data is a pervasive problem in machine learning [Northcutt et al., 2021, Shepardson, 2019]. Label error often arises as part of the data annotation pipeline and can be caused by annotators' lack of familiarity with the task, non-representative sampling, and feature noise in samples for certain data groups [Goel and Faltings, 2019, Hsueh et al., 2009]. In this work, we focus on mislabelling that occurs either with the training or test data—the two key sources of data in an ML pipeline. The broader literature uses the term label noise—of which label error is a subcategory. Contemporary work has led to approaches for obtaining accurate models in spite of label errors, such as instance weighting [Jiang and Nachum, 2020, Ren et al., 2018], robust loss functions [Ghosh et al., 2017, Ma et al., 2020], or trusted data [Hendrycks et al., 2018].

However, little is known about the effect of label noise on a model's group-based disparity metrics like equal odds [Hardt et al., 2016], group calibration [Pleiss et al., 2017], and false positive rate [Barocas et al., 2019, Garg et al., 2020]. It is now common practice to conduct 'fairness' audits (see: [Bakalar et al., 2021, Buolamwini and Gebru, 2018, Raji and Buolamwini, 2019]) of a model's predictions to identify differential performance across data subgroups. Label error in the test data used to conduct an audit leads to unreliable group-based disparity

metrics. Similarly, label error in the training data, especially if systematically more prevalent in certain data subgroups, can lead to models that associate such erroneous labels to these groups. The reliability of a fairness audit rests on the assumption that labels are *accurate*; yet, the sensitivity of a model's disparity metrics to label error is still poorly understood.

Here we address the aforementioned challenge via the following questions:

1. **Research Question 1**: What is the sensitivity of a model's disparity metric to label errors in training and test data? Does the effect of label error vary based on group size?

2. **Research Question 2**: How can a practitioner identify training points whose labels have the most 'influence' on a model's group disparity metric for a test dataset?

Contributions & Summary of Findings. Towards these questions, we make two broad contributions: First, we conduct comprehensive empirical sensitivity tests across several datasets and model classes to quantify the impact of labels errors—for test and training data—on a series of disparity metrics. Second, we propose a method, based on the influence function, to identify training points that have a high effect on a model's test disparity metric.

Empirical Sensitivity Tests. We assess the sensitivity of model disparity metrics to label errors with a label flipping experiment. First, we iteratively flip the labels of samples in the test set, for a fixed model, and then measure the corresponding change in the model disparity metric compared to an unflipped test set. Second, we fix the test set for the fairness audit but flip the labels of a proportion of the training samples. We then measure the change in the model disparity metrics for a model trained on the data with flipped labels. We perform these tests across datasets and model combinations.

Training Point Influence on Disparity Metric. We propose an approach, based on a modification to the influence of a training example on a test example's loss(Koh and Liang [2017]), to identify training points whose labels have undue effects on any disparity metric of interest on the test set. We empirically assess the proposed approach on a variety of datasets and find a 10-40% improvement, compared to alternative approaches like RDIA (Kong et al. [2021]) that focus solely on model's loss, in identifying training inputs that improve a model's disparity metric.

6.2 Setup & Background

In this section, we discuss notation, and set the stage for our contributions by discussing the disparity metrics that we focus on. We also provide an overview of the datasets and models used.

Overview of Notation. We consider prediction problems, i.e, settings where the task is to learn a mapping, $\theta : \mathcal{X} \times \mathcal{A} \to \mathcal{Y}$, where $\mathcal{X} \in \mathbb{R}^d$ is the feature space, $\mathcal{Y} \in \{0, 1\}$ is the output space, and $\mathcal{A}$ is a group identifier that partitions the population into disjoint sets e.g. race, gender. We term the tuple, (x_i, a_i, y_i), z_i, which means the n training points can be written as: $\{z_i\}_{i=1}^n$. Throughout this work, we will only consider learning via empirical risk minimization (ERM), which corresponds to: $\hat{\theta} := \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_i^n \ell(z_i, \theta)$. We assume that the ERM objective is twice-differentiable and strictly convex in the parameters.

A few of the metrics we select require binning the probabilistic output of the classifier. Consequently, we can group predictions into M interval bins (each size $1/M$) and calculate the accuracy of each bin. Let B_m be the set of indices of samples whose prediction confidence falls into the interval $I_m = [\frac{m-1}{M}, \frac{m}{M}]$. Here the accuracy of B_m is then: $\text{acc}(\text{B}_\text{m}) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}(\hat{y}_i = y_i)$, where y_i is the true label for sample x_i derived from a model θ with prediction $\hat{y}_i = \theta(x_i)$. Similarly, we can define the average confidence within a bin B_m as: $\text{conf}(\text{B}_\text{m}) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i$.

Disparity Metrics. We define a group disparity metric to be a function, $\mathcal{GD}$, that gives a performance score given a model's probabilistic predictions (θ outputs the probability of belonging to the positive class) and 'ground-truth' labels. We consider the following metrics:

1. **Calibration**: defined as $\mathbb{P}\left(\hat{y} = y | \hat{p} = p\right), \forall p \in [0, 1]$. In this work, we measure calibration with two different metrics: 1) Expected Calibration Error (ECE) [Naeini et al., 2015, Pleiss et al., 2017], and 2) the Brier Score [Rufibach, 2010] (BS). ECE approximates a notion of miscalibration, so it is defined as: $\mathcal{GD}_{\text{ece}} = \sum_{m=1}^M \frac{|B_m|}{n} |\text{conf}(\text{B}_\text{m}) - \text{acc}(\text{B}_\text{m})|$, while the Brier Score is the mean squared error between the labels and a models' predicted probabilities: $\mathcal{GD}_{\text{bs}} = \sum_{i=1}^n (\hat{y}_i - y_i)^2$. We report results for the ECE in the main text, and defer Brier Score results to the appendix.

2. **(*Generalized*) False Positive Rate (FPR)**: is $\mathcal{GD}_{\text{fpr}}(\theta) = \mathbb{E}[\theta(x_i) \mid y_i = 0]$ (see [Guo et al., 2017]),

Dataset	Classes	n	d	Group	Source
CivilComments	2	1,820,000	768	Sex	Koh and Liang [2017]
ACSIncome	2	195,665	10	Sex, Race	Ding et al. [2021]
ACSEmployment	2	378,817	16	Sex, Race	Ding et al. [2021]
ACSPublic Coverage	2	138,554	19	Sex, Race	Ding et al. [2021]
Credit Dataset	2	405,032	6	Sex	De Montjoye et al. [2015]

Table 6.1: Dataset characteristics. n is the training set size and d is the number of features.

3. **(*Generalized*) False Negative Rate (FNR)**: is $\mathcal{GD}_{\text{fnr}}(\theta) = \mathbb{E}[(1 - \theta(x_i)) \mid y_i = 1]$,

4. **Error Rate (ER)**: is the $\mathcal{GD}_{\text{er}}(\theta) = 1 - \text{acc}(\theta)$.

We consider these metrics separately for each group, a_i, in the data as opposed to relative differences. We do this because we are primarily interested in the behavior of these metrics as more label error is induced either in the test or training data.

Datasets. We consider datasets across different modalities: 3 tabular, and a text dataset. A description of these datasets is provided in Table 6.1. Each dataset contains annotations with a group label for both training and test data, so we are able to manipulate these labels for our empirical sensitivity tests. We assume that the provided labels are the ground-truth—a strong assumption that nevertheless does not impact the interpretation of our findings.

Model. We consider three kinds of model classes in this work: 1) a logistic regression model, 2) a Gradient-boosted Tree (GBT) classifier for the tabular datasets, and 3) a ResNet-18 model for the text data. These three classes of models allow us to also test whether the sensitivity of a model's disparity metric depends on the amount of overparametrization and the model class (i.e. Tree-based models vs Neural Networks). We refer readers to the Appendix for details of model training and hyper-parameters.

6.3 Empirical Assessment of Label Sensitivity

In this section, we perform empirical sensitivity tests to quantify the impact of label error on test group disparity metrics. We report here results on a logistic regression model for subgroup calibration error (ECE), and provide a high-level summary of other results.

We conduct tests for two different stages of the ML pipeline: 1) Test-time (test dataset) and 2) Training-time (training data). The test and training set are the two data sources; hence, the settings where label error could affect the ML model pipeline. We use as our

primary experimental tool: label flipping, i.e., we flip the labels of a percentage of the samples uniformly at random in either the test or training set and then measure the concomitant change in the model calibration.

We assume that each dataset's labels are the ground truth and that flipping the label results in label error for the samples whose labels have been overturned. Recent literature has termed this setting synthetic noise, i.e., the label flipping simulates noise that might not be representative of real-world noise in labels [Arpit et al., 2017, Jiang et al., 2020, Zhang et al., 2021]. For example, it could be the case that label error is more likely for certain kinds of data. In real-world datasets, there is often uncertainty in the training labels themselves that might not be distributed uniformly across samples.

6.3.1 Test-time Label Sensitivity

Overview & Experimental Setup. The goal of the Test-time empirical tests is to measure the impact of label error on the group calibration error of a fixed model. Consider the setting where a model has been trained, and a fairness assessment is to be conducted on the model. What impact does label error, in the test set used to conduct the audit, have on the calibration error on the test data? The test-time empirical tests answer this question. Given a fixed model, we iteratively flip a percentage of the labels, uniformly at random, ranging from zero to 30 percent in the test data. We then estimate the model's calibration using the modified dataset. Critically, we keep the model fixed while performing these tests across each dataset.

Results. In Figure 6-1, we show results of the label flipping experiments across 6 datasets. On the horizontal axis, we have the percentage of labels flipped in the test dataset, while on the vertical axis, we have the percentage change in the model's calibration. For each dataset, we compute model calibration for two demographic groups in the dataset, the majority and the minority—in size–groups. We do this since these two groups constitute the two ends of the spectrum in the dataset. As shown, we observe a more distinctive effect for the minority group across all datasets. This is to be expected since flipping even a small number samples in the minority group can have a dramatic effect on test and training accuracy within this group. For both groups, we observe a super-linear effect on the calibration error. For example, for the Income prediction task on the Adult dataset, a 10 percent label error induces a 20 percent change in the model's test calibration error. These results suggest that test-time

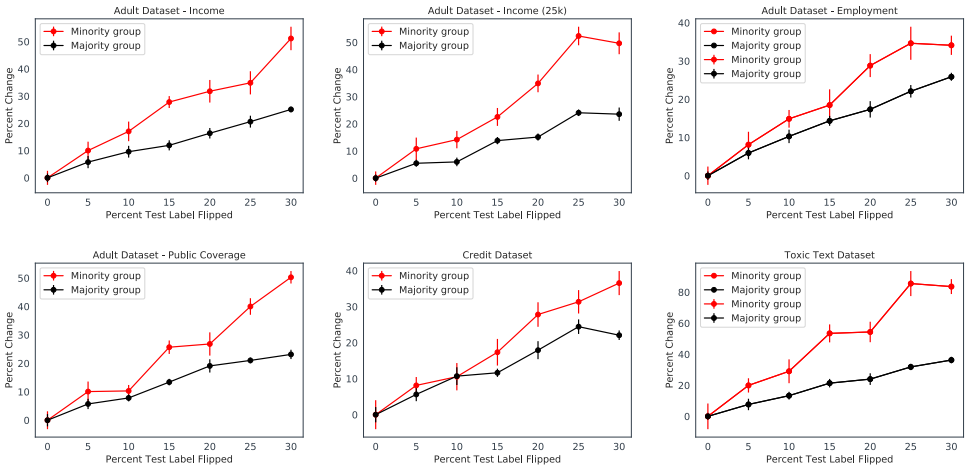

Figure 6-1: **Test-time Label Flipping Results across 6 datasets**. For each dataset, we plot the percent change in calibration error versus the corresponding percentage change in label error. Here, we plot the minority (smallest) group as well as the majority (largest) group. These two groups represent two ends of the spectrum for the impact of label error. We observe that across all datasets, the minority group incurs higher percentage change in group calibration compared to the majority group.

label error has more pronounced effects for minority groups. Further, these results imply that - for calibration error - one can expect a change at least as large as the percentage of label error present in the test dataset.

6.3.2 Training-time Label Sensitivity

Overview & Experimental Setup. The goal of the training-time empirical tests is to measure the impact of label error on a trained model. More specifically, given a training set in which a fraction of the samples' labels have been flipped, what effect does the label error have on the calibration error compared to a model trained on data without label error? We simulate this setting by creating multiple copies of each of the datasets where a percentage of the training labels have been flipped uniformly at random. We then assess the model calibration of these different model using the same fixed test dataset. Under similar experimental training conditions for these models, we are then able to quantify the effect of training label error on a model's test calibration error.

Results & Implications We show the results of the training-time experiments in Figure 6-2. Similar to the test-time experiments, we observe a higher sensitivity for the minority groups. A conjecture for the higher sensitivity to extreme training-time error is that

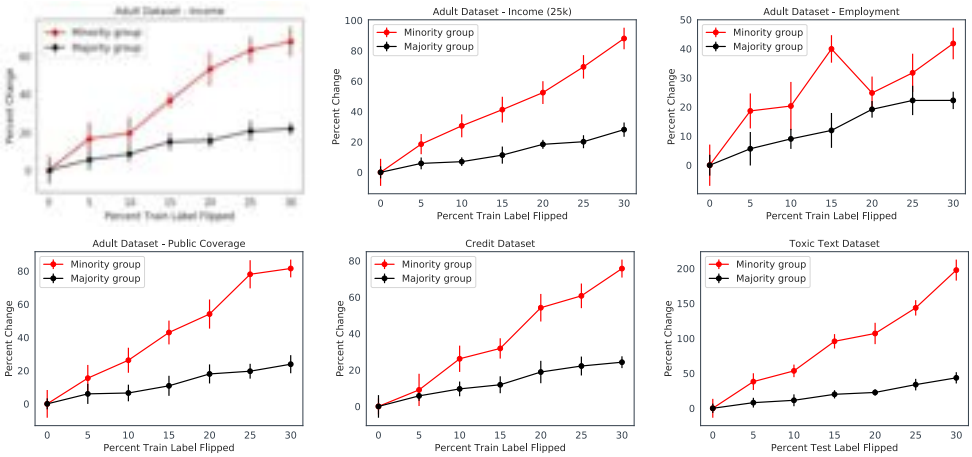

Figure 6-2: **Training-time Label Flipping Results across 6 datasets**. For each dataset, we plot the percent change in calibration error versus the corresponding percentage change in label error for the training set. Here, we plot the minority (smallest) group as well as the majority (largest) groups by size. Similar to the test-time setting, we observe that across all datasets, the minority group incurs higher percentage change in group calibration compared to the majority group. However, we observe a larger magnitude change for the minority groups.

a model trained on significant label error might have a more difficult time learning patterns in the minority class where there are not enough samples to begin with. Consequently, the generalization performance of this model worsens for inputs that belong to the minority group. Alternatively, in the majority group, the proportion of corrupted labels due to label error is smaller. This might mean that uniform flipping does not affect the proportion of true labels compared to the minority group. Even though the majority group exhibits label error, there still exists enough samples with true labels such that a model can learn the underlying signal for the majority class. Additionally, we observe a lower sensitivity for the overparametrized ResNet-18 model, which might imply that overparameterization might add resilience to label training error. We discuss these findings for the other metrics in the Appendix as well.

6.4 Identifying 'Fairness Violations' with Influence Functions

In this section, we present an approach for estimating the 'influence' of perturbing a training point's label on a disparity metric of interest. The approach is based on a modification to the influence functions method of Koh and Liang [2017]. We consider two primary effects: 1) up-weighting a training point, and 2) perturbing the training label.

Upweighting a training point. Let $\hat{\theta}_{-z_i}$ be the ERM solution when a model is trained on all data points, $\{z_j\}_{j=1}^n$, except z_i. The influence, $\mathcal{I}_{\text{up,params}}$, of datapoint, z_i, on the model parameters is then defined as the change: $\hat{\theta}_{-z_i} - \hat{\theta}$. This measure indicates how much the parameters change when the model is 'refit' on all training data points except z_i. Koh and Liang [2017] give a closed-form estimate of this quantity as:

$$\mathcal{I}_{\text{up,params}} \stackrel{\text{def}}{=} \left.\frac{d\hat{\theta}_{\epsilon,\ z_i}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_\theta \ell(z_i, \hat{\theta}), \tag{6.1}$$

where H is the hessian, i.e., $H_{\hat{\theta}} \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n \nabla_\theta^2 \ell(z_i, \theta)$.

The loss on a test example, $\ell(z_t, \hat{\theta})$, is a function of the model parameters, so using the chain-rule, we can estimate the influence, $\mathcal{I}_{\text{up,loss}}(z_i, z_t)$, of a training point, z_i, on $\ell(z_t, \hat{\theta})$ as:

$$\mathcal{I}_{\text{up,loss}}(z_i, z_t) \stackrel{\text{def}}{=} \left.\frac{d\ell(z_t, \hat{\theta}_{\epsilon,\ z_i})}{d\epsilon}\right|_{\epsilon=0} = -\nabla_\theta \ell(z_t, \hat{\theta})^\top H_{\hat{\theta}}^{-1}\nabla_\theta \ell(z_i, \hat{\theta}). \tag{6.2}$$

Perturbing a training point's label. A second notion of influence that Koh and Liang [2017] study is how perturbing a training point leads to changes in the model parameters. Specifically, given a training input, z_i, that is a tuple (x_i, y_i), how would the perturbation, $z_i \to z_{i,\delta}$, which is defined as $(x_i, y_i) \to (x_i, y_i + \delta)$, change the model's predictions? Koh and Liang [2017] give a closed-form estimate of this quantity as:

$$\mathcal{I}_{\text{pert,loss,y}}(z_j, z_t) \approx -\nabla_\theta \ell(z_t, \hat{\theta}_{z_{j,\delta}, -z_j})^\top H_{\hat{\theta}}^{-1}\nabla_y \nabla_\theta \ell(z_j, \hat{\theta}). \tag{6.3}$$

Adapting influence functions to group disparity metrics. We now propose modifications that allow us to compute the influence of a training point on a test group disparity metric. Let S_t be a set of test examples. We can then denote $\mathcal{GD}(S_t, \hat{\theta})$ as the group disparity metric of interest, e.g., the estimated ECE for the set S_t given parameter setting $\hat{\theta}$.

Influence of upweighting a training point on a test group disparity metric. A group disparity metric on the test set is a function of the model parameters; consequently, we can apply the chain rule to $\mathcal{I}_{\text{up,params}}$ (from Equation 6.1) to estimate the influence,

$\mathcal{I}_{\text{up,disparity}}$, of up-weighting a training point on the disparity metric as follows:

$$\mathcal{I}_{\text{up,disparity}}(z_i, S_t) \stackrel{\text{def}}{=} \frac{d\mathcal{GD}(S_t, \hat{\theta}_{\epsilon,\ z_i})}{d\epsilon}\bigg|_{\epsilon=0} = -\nabla_\theta \mathcal{GD}(S_t, \hat{\theta})^\top \frac{d\hat{\theta}_{\epsilon,\ z_i}}{d\epsilon}\bigg|_{\epsilon=0},$$
$$= -\nabla_\theta \mathcal{GD}(S_t, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta \ell(z_i, \hat{\theta}). \tag{6.4}$$

We now have a closed-form expression for a training point's influence on a test group disparity metric.

Influence of perturbing a training point's label on a test group disparity metric. We now consider the influence of a training label perturbation on a group disparity metric of interest. To do this, we simply consider the group disparity metric as the quantity of interest instead of the test loss. Consequently, the closed-form expression for the influence of a modification to the training label on disparity for a given test set is:

$$\mathcal{I}_{\text{pert,disparity,y}}(z_j, S_t) \approx -\nabla_\theta \mathcal{GD}(S_t, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_y \nabla_\theta \ell(z_j, \hat{\theta}). \tag{6.5}$$

With Equations 6.4 and 6.5, we have the key quantities of interest that allows us to rank training points, in terms of influence, on the test group disparity metric. We can then use these quantities to prioritize samples that should be more carefully inspected.

6.5 Empirical Assessment

In this section, we empirically assess the modified influence expressions for calibration across these datasets for prioritizing mislabelled samples. We find a 10-40 percent improvement, compared to alternative approaches that estimate influence on test loss [Kong et al., 2021].

Overview & Experimental Question. We are interested in surfacing training points whose change in label will induce a concomitant change in a test disparity metric like group calibration. Specifically, we ask: When the training points are ranked by influence on test calibration, are the most highly influential training points most likely to have the wrong labels? We conduct our experiments to directly measure a method's ability to answer this question.

Experimental Setup. For each dataset, we randomly flip the labels of 10 percent of the

training samples. We then train our standard logistic regression classifier on this modified dataset. In this task, we have direct access to the ground-truth and knowledge of the exact samples whose labels were flipped. This allows us to directly compare the performance of our proposed methods to each of the baselines on this task. We then rank training points using a number of baseline approaches as well as the modified influence approaches. For the top 50 examples, we consider what fraction of these examples had flipped labels in the training set.

Approaches & Baselines. We consider the following methods: 1) **IF-Calib**: The closed-form approximation to the influence of a training point on the test calibration; 2) **IF-Calib-Label**: The closed-form approximation to the influence of a training point's label on the test calibration; 3) **Loss**: A baseline method which is the training loss evaluated at each data point in the training set. The intuition is that, presumably, more difficult training samples will have higher training loss; and **Training Point Influence of Test Loss (IF-Norm)**: A baseline method which is the (average) training point influence of the

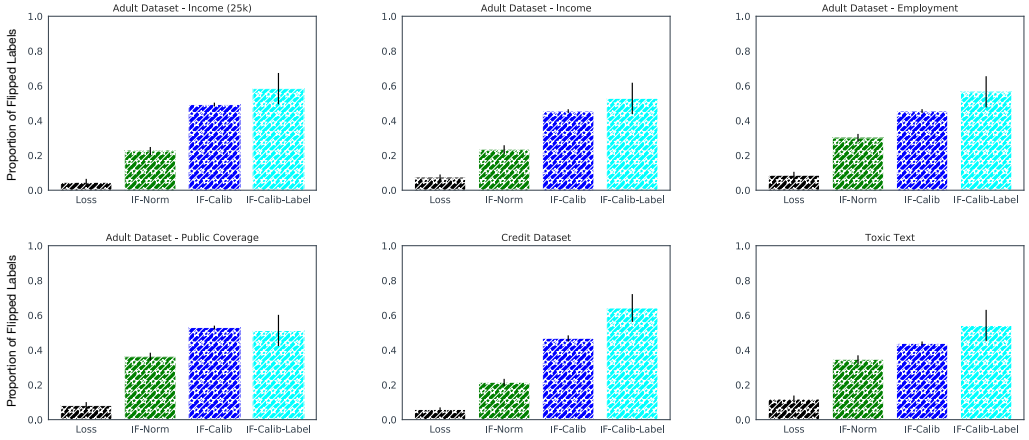

Figure 6-3: **Empirical Results for Training Point Ranking Across 6 datasets**. For the top 50 most influential examples, we show the proportion of samples whose labels were flipped in the training data. Confidence intervals is computed for N=5 different models.

Results: Prioritizing Samples. In Figure 6-3, we show the performance of the two approximations that we consider in this work as well as two baselines. We plot the fraction of inputs, out of the top ranked 50 ranked training points, whose labels were flipped in the training set. The higher this proportion, then the more effective an approach is in identifying the samples that likely have wrong labels. In practice, the goal is to surface these training samples and have a domain expert inspect them. If a larger proportion of the items to be

inspected are mislabeled, then a higher proportion of training set mistakes, i.e. label error, can be fixed. Across the different datasets, we find a 10-40 percent improvement, compared to baseline approaches, in identifying critical training data points whose labels need to be reexamined.

We find the loss baseline to be ineffective for ranking in our experiments. A possible reason is that modern machine learning models can typically be trained to 'memorize' the training data; resulting in settings where a model has low loss even on outliers or mislabeled examples. In such a case, ranking by training loss for a sample is an ineffective ranking strategy.

6.6 Related Work

This discussion in this chapter is based on Adebayo et al. [2022a].

Impact of Label Error/Noise on Model Accuracy. Learning under label error falls under the category more commonly known as *learning under noise* [Bootkrajang and Kabán, 2012, Frénay and Verleysen, 2013, Natarajan et al., 2013]. *Noise* in learning can come from different components of the underlying data generation process, which include the input features and the labels. In this work, we focus exclusively on categorization mistakes associated with the label of a training input, which we term *label error*. Model resilience to training label error has been studied for both synthetic [Arpit et al., 2017, Rolnick et al., 2017, Zhang et al., 2021] and real-world noise settings [Jiang et al., 2020]. Along these lines, a major line of inquiry is the development of mitigation strategies; these seek to produce accurate models and learning algorithms that are resilient to training label error. These strategies can be in the form of model regularization [Srivastava et al., 2014, Zhang et al., 2017], bootstrap [Reed et al., 2014], knowledge distillation [Jiang et al., 2020], instance weighting [Jiang and Nachum, 2020, Ren et al., 2018], robust loss functions [Ghosh et al., 2017, Ma et al., 2020], or trusted data [Hendrycks et al., 2018]. In contrast to this line of work, instead of making the model resilient to label error, we take the approach of finding the 'problematic' training points, whose careful relabeling will most likely improve a model's disparity metric.

Impact of Label Error on Model *'Fairness'*. This work contributes to the burgeoning area that studies the impact of label error on a model's 'fairness' (termed 'group-based

disparity' in this paper) metrics. Fogliato et al. [2020] studied a setting in which the labels used for model training are a noisy proxy for the true label of interest, e.g., predicting rearrest as a proxy for rearrest. Wang et al. [2021] introduce a method to account for group-dependent label noise by using new surrogate loss functions that weight group-based disparity metrics. Similarly, Konstantinov and Lampert [2021] study the effect of adversarial data corruptions on fair learning in a PAC model and show impossibility results under the adversarial model. Jiang and Nachum [2020] propose a re-weighting scheme that is able to correct for label noise. They then show that training a classifier on the re-weighted objective recovers models with group-disparity properties similar to a model trained on the unobserved noiseless labels.

Influence Functions & Their Uses. Influence functions originate from robust statistics where it is used as a tool to identify outliers [Cook, 1986, Cook and Weisberg, 1982, Hampel, 1974]. Koh and Liang [2017] introduced influence functions for modern machine learning models, and used them for various model debugging tasks. Most similar to our work, Kong et al. [2021] propose RDIA, a relabelling scheme based on the influence function that is able to provably correct for label error in the training data. RDIA identifies training samples whose change in label has a high influence on the test loss for a validation set; however, we focus on identifying training samples whose change in label has a high influence on a group-disparity metric on a test/audit set.

In recent work, De-Arteaga et al. [2021] study expert consistency in data labeling and use influence functions to estimate the impact of labelers on a model's predictions. Along similar direction, Brunet et al. [2019] adapt the influence function approach to measure how removing a small part of a training corpus, in a word embedding task, affects test bias as measured by the word embedding association test [Caliskan et al., 2017]. Feldman and Zhang [2020] use influence functions to estimate how likely a training point is to have been memorized by a model. More generally, influence functions are gaining widespread use as a tool for debugging model predictions [Barshan et al., 2020, Han et al., 2020, Pruthi et al., 2020, Yeh et al., 2018].

Low Performance Subgroup Identification We use influence functions to identify training points that have the most effect on the model's disparity metric; however, there are other approaches for surfacing training points that need to be prioritized. For example, Kim et al. [2019] propose an algorithm to identify groups in the data where a model has high test errors, and to 'boost' the model performance for these groups. Similarly, Creager et al. [2021]

propose a two-stage scheme to identify critical subgroups in the data when demographic labels are not known ahead of time. Our approach differs from the aforementioned since we focus principally on the effect to the model's disparity metric instead of the test loss.

Label Flipping. We use labeling flipping as a primary tool in our empirical tests to measure the sensitivity of a model's disparity metrics to label error. Label flipping has also previously been used for alternative purposes [Arpit et al., 2017, Zhang et al., 2021]. In now seminal work, Zhang et al. [2021] used label flipping and shuffling to show that deep neural networks easily memorize data with random labels. More generally, label flipping can also constitute an 'adversarial attack' against an ML model, for which there are increasingly new methods to help defend against such attacks [Rosenfeld et al., 2020].

Explainability Increasingly, 'explanations' derived from a trained model can point to the reason why an input has been misclassified. Ultimately, one holy grail use-case for explanations has been help identify and suggest potential fixes for model performance disparities. Towards this end, Pradhan et al. [2021] propose an approach that intervenes on the training data and measures the changes to downstream performance on the basis of these changes.

Chapter 7

Conclusion

In the lead up to DeepMind's famous Go match against Lee Seedol, members of the AlphaGo team invited Fan Hui, the European Go champion, to help test AlphaGo. A version of AlphaGo had played against Fan Hui, in October 2015, and defeated him.

Regarding the experience Fan Hui said:

> *I played with AlphaGo to* ***understand where is the strong points of AlphaGo and where is maybe the weakness****. I played in the morning, afternoon, all time. And I find something,* ***I find big weakness about AlphaGo. It's a big one [Kohs].***

In response to Fan Hui's finding, David Silver, the team lead said,

> *We can think of there being the space of all the things it [AlphaGo] knows about. And it knows about most of it extremely well, but there will be these* ***tricky lumps of knowledge that it understands very poorly****. It is really hard for us to characterize when it is going to enter into one of these lumps. But if it does, it can be completely delusional, thinking that it is alive in one part of the board, but in fact is dead or vice versa. There is a real risk that we could lose the match. [Kohs]*

Ultimately, the AlphaGo team didn't fix the weakness before the Lee Sedol match. Since, the techniques underlying AlphaGo have been extended to AlphaFold [Jumper et al., 2021] for learning protein structure. Similarly, foundation models [Bommasani et al., 2021], DNNs with billions of parameters trained in a self-supervised manner, can now generate coherent text [Brown et al., 2020], and images [Ramesh et al., 2021, 2022] with remarkable ingenuity.

Would it be acceptable if AlphaFold or these Foundation Models had poorly understood hidden weaknesses? Despite tremendous progress in the capabilities, the ability to debug these systems lags behind.

Thesis Goals. Towards addressing the challenge, this thesis take steps towards developing tools that are effective for model debugging. We addressed the following two questions:

1. Of the post hoc explanation methods available, which ones are effective for model debugging?

2. Second, given the limitations of current approaches, can we develop model debugging approaches that can incorporate expert expertise via interaction?

Key Thesis Contributions. We make the following contributions:

1. **Part I: Empirical Assessment of Current Methods.** We show that current approaches struggle to detect a model's reliance on spurious signals, are unable to identify training inputs with wrong labels, and provide no direct avenue for fixing model errors.

2. **Part II: New tools for model debugging.** In the second part of the thesis, we introduce two new approaches that directly address the limitations of current methods.

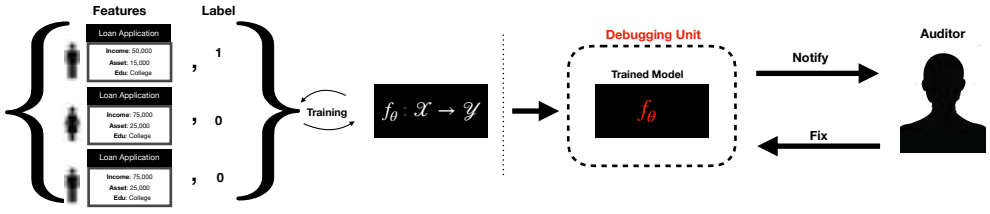

Figure 7-1: **Hypothetical model debugging workflow.**

A model debugging Ideal In software development, it is now standard practice to perform automated testing before deploying software. These tests often catch critical bugs. The hope is that the practice of ML model development will also approach reliability levels similar to the practice of software development. As shown in Figure 7-1, a future where a trained model is (semi)automatically tested, and the model bugs in its ML pipeline are easily detected and fixed prior to deployment would be ideal. This thesis takes a step in that direction.

Bibliography

Hammaad Adam, Ming Ying Yang, Kenrick Cato, Ioana Baldini, Charles Senteio, Leo Anthony Celi, Jiaming Zeng, Moninder Singh, and Marzyeh Ghassemi. Write it like you see it: Detectable differences in clinical notes by race lead to differential model recommendations. *arXiv preprint arXiv:2205.03931*, 2022. 18

Julius Adebayo and Hal Abelson. Guiding models with expert annotations. 2022. 93

Julius Adebayo, Justin Gilmer, Ian Goodfellow, and Been Kim. Local explanation methods for deep neural networks lack sensitivity to parameter values. 2018a. 45

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, pages 9525–9536, 2018b. 55, 64, 65, 75

Julius Adebayo, Michael Muelly, Ilaria Liccardi, and Been Kim. Debugging tests for model explanations. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/075b051ec3d22dac7b33f788da631fd4-Abstract.html. 59, 77, 131

Julius Adebayo, Melissa Hall, Bowen Yu, and Bobbie Chern. From label quality to model fairness: Quantifying the sensitivity of model disparity metrics to label errors. 2022a. 107

Julius Adebayo, Michael Muelly, Hal Abelson, and Been Kim. Post hoc explanations may be ineffective for detecting unknown spurious correlation. In *International Conference on Learning Representations*, 2022b. URL https://openreview.net/forum?id=xNOVfCCvDpM. 77

Emily Aiken, Suzanne Bellue, Dean Karlan, Christopher R Udry, and Joshua Blumenstock. Machine learning and mobile phone data can improve the targeting of humanitarian assistance. Technical report, National Bureau of Economic Research, 2021. 17

Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. innvestigate neural networks! *CoRR*, abs/1808.04260, 2018. URL http://arxiv.org/abs/1808.04260. 46

Ahmed Alqaraawi, Martin Schuessler, Philipp Weiß, Enrico Costanza, and Nadia Berthouze. Evaluating saliency map explanations for convolutional neural networks: a user study. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, pages 275–285, 2020. 20, 75

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016. 42

Christopher Anders, Plamen Pasliev, Ann-Kathrin Dombrowski, Klaus-Robert Müller, and Pan Kessel. Fairwashing explanations with off-manifold detergent. In *International Conference on Machine Learning*, pages 314–323. PMLR, 2020. 76

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019. 24, 39

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *International Conference on Machine Learning*, pages 233–242. PMLR, 2017. 101, 107, 109

Nishanth Arun, Nathan Gaw, Praveer Singh, Ken Chang, Mehak Aggarwal, Bryan Chen, Katharina Hoebel, Sharut Gupta, Jay Patel, Mishka Gidwani, et al. Assessing the trustworthiness of saliency maps for localizing abnormalities in medical imaging. *Radiology: Artificial Intelligence*, 3(6):e200267, 2021. 77

Sean Augenstein, H Brendan McMahan, Daniel Ramage, Swaroop Ramaswamy, Peter Kairouz, Mingqing Chen, Rajiv Mathews, et al. Generative models for effective ml on private, decentralized datasets. *arXiv preprint arXiv:1911.06679*, 2019. 42

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 36

Marley Bacelar. Monitoring bias and fairness in machine learning models: A review. *ScienceOpen Preprints*, 2021. 42

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015. 46

Dinesh Bacham and Janet Zhao. Machine learning: Challenges, lessons, and opportunities in credit risk modeling. https://www.moodysanalytics.com/risk-perspectives-magazine/managing-disruption/spotlight/machine-learning-challenges-lessons-and-opportunities-in-credit-risk-modeling, 2017. Accessed: 2022-01-30. 17

Marcus A Badgeley, John R Zech, Luke Oakden-Rayner, Benjamin S Glicksberg, Manway Liu, William Gale, Michael V McConnell, Bethany Percha, Thomas M Snyder, and Joel T Dudley. Deep learning predicts hip fracture using confounding patient and healthcare variables. *NPJ digital medicine*, 2(1):1–10, 2019. 17

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831, 2010. 44

Mohammad Taha Bahadori and David E Heckerman. Debiasing concept-based explanations with causal analysis. 2021. 76

Chloé Bakalar, Renata Barreto, Stevie Bergman, Miranda Bogen, Bobbie Chern, Sam Corbett-Davies, Melissa Hall, Isabel Kloumann, Michelle Lam, Joaquin Quiñonero Candela, et al. Fairness on the ground: Applying algorithmic fairness approaches to production systems. *arXiv preprint arXiv:2103.06172*, 2021. 97

Aparna Balagopalan, Haoran Zhang, Kimia Hamidieh, Thomas Hartvigsen, Frank Rudzicz, and Marzyeh Ghassemi. The road to explainability is paved with bias: Measuring the fairness of explanations. *arXiv preprint arXiv:2205.03295*, 2022. 20

Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning*. fairmlbook.org, 2019. http://www.fairmlbook.org. 30, 97

Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. Relatif: Identifying explanatory training samples via relative influence. In *International Conference on Artificial Intelligence and Statistics*, pages 1899–1909. PMLR, 2020. 108

Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020. 76, 130

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017. 26, 46

David Bau, Steven Liu, Tongzhou Wang, Jun-Yan Zhu, and Antonio Torralba. Rewriting a deep generative model. In *European conference on computer vision*, pages 351–369. Springer, 2020. 94

Sahely Bhadra and Matthias Hein. Correction of noisy labels via mutual consistency check. *Neurocomputing*, 160:34–52, 2015. 42

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021. 87, 88

Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. An interpretability illusion for bert. *arXiv preprint arXiv:2104.07143*, 2021. 20

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. 111

Jakramate Bootkrajang and Ata Kabán. Label-noise robust logistic regression and its applications. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 143–158. Springer, 2012. 107

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 111

Marc-Etienne Brunet, Colleen Alkalay-Houlihan, Ashton Anderson, and Richard Zemel. Understanding the origins of bias in word embeddings. In *International Conference on Machine Learning*, pages 803–811. PMLR, 2019. 108

Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018. 18, 97

Gabriel Cadamuro, Ran Gilad-Bachrach, and Xiaojin Zhu. Debugging machine learning models. In *ICML Workshop on Reliable Machine Learning in the Wild*, 2016. 42

Aylin Caliskan, Joanna J Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, 2017. 108

Edwin Chen. 30% of google's emotions dataset is mislabeled, 2022. URL https://www.surgehq.ai/blog/30-percent-of-googles-reddit-emotions-dataset-is-mislabeled. 25

Pingjun Chen, Linlin Gao, Xiaoshuang Shi, Kyle Allen, and Yang Lin. Fully automatic knee osteoarthritis severity grading using deep neural networks with a novel ordinal loss. *Computerized Medical Imaging and Graphics*, 75:84–92, 2019. doi: https://doi.org/10.1016/j.compmedimag.2019.06.002. 91, 131

Valerie Chen, Jeffrey Li, Joon Sik Kim, Gregory Plumb, and Ameet Talwalkar. Towards connecting use cases and methods in interpretable machine learning. *arXiv preprint arXiv:2103.06254*, 2021. 20, 36

Zhi Chen, Yijie Bei, and Cynthia Rudin. Concept whitening for interpretable image recognition. *Nature Machine Intelligence*, 2(12):772–782, 2020. 76

Eric Chu, Deb Roy, and Jacob Andreas. Are visual explanations useful? a case study in model-in-the-loop prediction. *arXiv preprint arXiv:2007.12248*, 2020. 20, 75

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007. 86

EU Commission. 2018 reform of eu data protection rules, 2018. URL https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf. 18

European Commission. The artificial intelligence act. 2022. URL https://artificialintelligenceact.eu/the-act/. 18

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019. 149

R Dennis Cook. Assessment of local influence. *Journal of the Royal Statistical Society: Series B (Methodological)*, 48(2):133–155, 1986. 108

R Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982. 42, 108

Elliot Creager, Jörn-Henrik Jacobsen, and Richard Zemel. Environment inference for invariant learning. In *International Conference on Machine Learning*, pages 2189–2200. PMLR, 2021. 92, 108

Maria De-Arteaga, Artur Dubrawski, and Alexandra Chouldechova. Leveraging expert consistency to improve algorithmic decision support. *arXiv preprint arXiv:2101.09648*, 2021. 108

Yves-Alexandre De Montjoye, Laura Radaelli, Vivek Kumar Singh, and Alex "Sandy" Pentland. Unique in the shopping mall: On the reidentifiability of credit card metadata. *Science*, 347(6221):536–539, 2015. 100

Alex J DeGrave, Joseph D Janizek, and Su-In Lee. Ai for radiographic covid-19 detection selects shortcuts over signal. *medRxiv*, 2020. 74

Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan Cowen, Gaurav Nemade, and Sujith Ravi. Goemotions: A dataset of fine-grained emotions. *arXiv preprint arXiv:2005.00547*, 2020. 25

Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. *Advances in Neural Information Processing Systems*, 34, 2021. 100, 146

Ann-Kathrin Dombrowski, Maximillian Alber, Christopher Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. Explanations can be manipulated and geometry is to blame. In *Advances in Neural Information Processing Systems*, pages 13567–13578, 2019. 76

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. 2017. 75

Finale Doshi-Velez, Mason Kortz, Ryan Budish, Chris Bavitz, Sam Gershman, David O'Brien, Stuart Schieber, James Waldo, David Weinberger, and Alexandra Wood. Accountability of ai under the law: The role of explanation. *arXiv preprint arXiv:1711.01134*, 2017. 33

ECOA. Equal credit opportunity act (ecoa). 1974. URL https://www.fdic.gov/resources/supervision-and-examinations/consumer-compliance-examination-manual/documents/5/v-7-1.pdf. 18

Gabriel Erion, Joseph D Janizek, Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Learning explainable models using attribution priors. *arXiv preprint arXiv:1906.10670*, 2019. 45, 74

Sabri Eyuboglu, Maya Varma, Khaled Saab, Jean-Benoit Delbrouck, Christopher Lee-Messer, Jared Dunnmon, James Zou, and Christopher Ré. Domino: Discovering systematic errors with cross-modal embeddings. *arXiv preprint arXiv:2203.14960*, 2022. 94

Vitaly Feldman. Does learning require memorization? a short tale about a long tail. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 954–959, 2020. 24

Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. *arXiv preprint arXiv:2008.03703*, 2020. 108

Riccardo Fogliato, Alexandra Chouldechova, and Max G'Sell. Fairness evaluation in presence of biased noisy labels. In *International Conference on Artificial Intelligence and Statistics*, pages 2325–2336. PMLR, 2020. 108

John Fox. *Regression diagnostics: An introduction*. Sage publications, 2019. 42

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR, 2018. 86

Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869, 2013. 107

Pratyush Garg, John Villasenor, and Virginia Foggo. Fairness metrics: A comparative analysis. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 3662–3666. IEEE, 2020. 30, 97

Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018. 18

Marzyeh Ghassemi, Luke Oakden-Rayner, and Andrew L Beam. The false hope of current approaches to explainable artificial intelligence in health care. *The Lancet Digital Health*, 3(11):e745–e750, 2021. 20

Amirata Ghorbani, Abubakar Abid, and James Y. Zou. Interpretation of neural networks is fragile. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3681–3688. AAAI Press, 2019a. doi: 10.1609/aaai.v33i01.33013681. URL https://doi.org/10.1609/aaai.v33i01.33013681. 76

Amirata Ghorbani, James Wexler, James Zou, and Been Kim. Towards automatic concept-based explanations. *arXiv preprint arXiv:1902.03129*, 2019b. 75, 76

Aritra Ghosh, Himanshu Kumar, and PS Sastry. Robust loss functions under label noise for deep neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017. 97, 107

Judy Wawira Gichoya, Imon Banerjee, Ananth Reddy Bhimireddy, John L Burns, Leo Anthony Celi, Li-Ching Chen, Ramon Correa, Natalie Dullerud, Marzyeh Ghassemi, Shih-Cheng Huang, et al. Ai recognition of patient race in medical imaging: a modelling study. *The Lancet Digital Health*, 2022. 25

Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018. 19

Naman Goel and Boi Faltings. Crowdsourcing with fairness, diversity and budget constraints. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 297–304, 2019. 97

Roger Grosse. Neural net training dynamics: Bilevel optimization, February 2022. 87, 88

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017. 99, 147, 148

Han Guo, Nazneen Fatema Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. Fastif: Scalable influence functions for efficient model interpretation and debugging. *arXiv preprint arXiv:2012.15781*, 2020. 74, 76, 130, 131

Safwan S Halabi, Luciano M Prevedello, Jayashree Kalpathy-Cramer, Artem B Mamonov, Alexander Bilbily, Mark Cicero, Ian Pan, Lucas Araújo Pereira, Rafael Teixeira Sousa, Nitamar Abdala, et al. The rsna pediatric bone age machine learning challenge. *Radiology*, 290(2):498–503, 2019. 91

Frank R Hampel. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393, 1974. 108

Xiaochuang Han, Byron C Wallace, and Yulia Tsvetkov. Explaining black box predictions and unveiling data artifacts through influence functions. *arXiv preprint arXiv:2005.06676*, 2020. 74, 108

Kazuaki Hanawa, Sho Yokoi, Satoshi Hara, and Kentaro Inui. Evaluation of similarity-based explanations. *arXiv preprint arXiv:2006.04528*, 2020. 58

Moritz Hardt, Eric Price, Nati Srebro, et al. Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, pages 3315–3323, 2016. 30, 97

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 36

Dan Hendrycks, Mantas Mazeika, Duncan Wilson, and Kevin Gimpel. Using trusted data to train deep networks on labels corrupted by severe noise. *Advances in neural information processing systems*, 31, 2018. 97, 107

Juyeon Heo, Sunghwan Joo, and Taesup Moon. Fooling neural network interpretations via adversarial model manipulation. In *Advances in Neural Information Processing Systems*, pages 2921–2932, 2019. 76

Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 9737–9748, 2019. 45, 75

Pei-Yun Hsueh, Prem Melville, and Vikas Sindhwani. Data quality from crowdsourcing: a study of annotation selection criteria. In *Proceedings of the NAACL HLT 2009 workshop on active learning for natural language processing*, pages 27–35, 2009. 97

Indu Ilanchezian, Dmitry Kobak, Hanna Faber, Focke Ziemssen, Philipp Berens, and Murat Seçkin Ayhan. Interpretable gender classification from retinal fundus images using bagnets. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 477–487. Springer, 2021. 25

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022. 94

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 36

Heinrich Jiang and Ofir Nachum. Identifying and correcting label bias in machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 702–712. PMLR, 2020. 97, 107, 108

Lu Jiang, Di Huang, Mason Liu, and Weilong Yang. Beyond synthetic noise: Deep learning on controlled noisy labels. In *International Conference on Machine Learning*, pages 4804–4815. PMLR, 2020. 101, 107

Eric Jonas and Konrad Paul Kording. Could a neuroscientist understand a microprocessor? *PLoS computational biology*, 13(1):e1005268, 2017. 33, 34, 35

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. 17, 111

Dmitry Kazhdan, Botty Dimanov, Mateja Jamnik, Pietro Liò, and Adrian Weller. Now you see me (cme): Concept-based model extraction. *arXiv preprint arXiv:2010.13233*, 2020. 76

Fereshte Khani and Percy Liang. Removing spurious features can hurt accuracy and affect groups disproportionately. *arXiv preprint arXiv:2012.04104*, 2020. 75

Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, volume 2, 2011. 67, 91, 133

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International Conference on Machine Learning*, pages 2673–2682, 2018. 26, 46, 47, 75

Michael P Kim, Amirata Ghorbani, and James Zou. Multiaccuracy: Black-box post-processing for fairness in classification. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 247–254, 2019. 108

Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Sven Dähne. Investigating the influence of noise and distractors on the interpretation of neural networks. *arXiv preprint arXiv:1611.07270*, 2016. 45

Pieter-Jan Kindermans, Kristof T. Schütt, Maximilian Alber, Been Kim Klaus-Robert Müller, Dumitru Erhan, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hkn7CBaTW. 45

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 36

Jon Kleinberg, Himabindu Lakkaraju, Jure Leskovec, Jens Ludwig, and Sendhil Mullainathan. Human decisions and machine predictions. *The quarterly journal of economics*, 133(1): 237–293, 2018. 17

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894. PMLR, 2017. URL http://proceedings.mlr.press/v70/koh17a.html. 30, 47, 48, 50, 51, 55, 75, 88, 92, 98, 100, 103, 104, 108, 130

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International Conference on Machine Learning*, pages 5338–5348. PMLR, 2020. 76, 84

Greg Kohs. Alphago - the movie. URL https://www.youtube.com/watch?v=WXuK6gekU1Y. 111

Zico Kolter, Duvenaud David, and Johnson Matt. Deep implicity layers - neural odes, deep equilibrium models, and beyond, December 2021. 87

Shuming Kong, Yanyan Shen, and Linpeng Huang. Resolving training biases via influence-based data relabeling. In *International Conference on Learning Representations*, 2021. 98, 105, 108

Nikola Konstantinov and Christoph H Lampert. Fairness-aware pac learning from corrupted data. *arXiv preprint arXiv:2102.06004*, 2021. 108

Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. Principles of explanatory debugging to personalize interactive machine learning. In *Proceedings of the 20th international conference on intelligent user interfaces*, pages 126–137, 2015. 42

Fabian Küppers, Jan Kronenberger, Amirhossein Shantia, and Anselm Haselhoff. Multivariate confidence calibration for object detection. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020. 148

Himabindu Lakkaraju and Osbert Bastani. " how do i fool you?" manipulating user trust via misleading black box explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 79–85, 2020. 76

Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1–8, 2019. 20, 74

Yuri Lazebnik. Can a biologist fix a radio?—or, what i learned while studying apoptosis. *Cancer cell*, 2(3):179–182, 2002. 35

Klas Leino and Matt Fredrikson. Stolen memories: Leveraging model memorization for calibrated {White-Box} membership inference. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1605–1622, 2020. 23

Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018. 33

Raoni Lourenço, Juliana Freire, and Dennis Shasha. Debugging machine learning pipelines. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, pages 1–10, 2019. 42

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4768–4777, 2017. 46

Xingjun Ma, Hanxun Huang, Yisen Wang, Simone Romano, Sarah Erfani, and James Bailey. Normalized loss functions for deep learning with noisy labels. In *International Conference on Machine Learning*, pages 6543–6553. PMLR, 2020. 97, 107

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015. 86

Aravindh Mahendran and Andrea Vedaldi. Salient deconvolutional networks. In *European Conference on Computer Vision*, pages 120–135. Springer, 2016. 65, 75

David Marr. Vision: A computational investigation into the human representation and processing of visual information, henry holt and co. *Inc., New York, NY*, 2(4.2), 1982. 35

Morgan P McBee, Omer A Awan, Andrew T Colucci, Comeron W Ghobadi, Nadja Kadom, Akash P Kansagra, Srini Tridandapani, and William F Auffermann. Deep learning in radiology. *Academic radiology*, 25(11):1472–1480, 2018. 17

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *arXiv preprint arXiv:2202.05262*, 2022. 94

Qingjie Meng, Christian Baumgartner, Matthew Sinclair, James Housden, Martin Rajchl, Alberto Gomez, Benjamin Hou, Nicolas Toussaint, Jeremy Tan, Jacqueline Matthew, et al. Automatic shadow detection in 2d ultrasound. 2018. 74

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021. 94

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. Memory-based model editing at scale. In *International Conference on Machine Learning*, pages 15817–15831. PMLR, 2022. 94

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017. 46

Edwin Murdoch. How i found nearly 300,000 errors in ms coco, 2022. URL https://medium.com/@jamie_34747/how-i-found-nearly-300-000-errors-in-ms-coco-79d382edf22b. 25

Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. 99, 147

Vaishnavh Nagarajan, Anders Andreassen, and Behnam Neyshabur. Understanding the failure modes of out-of-distribution generalization. *arXiv preprint arXiv:2010.15775*, 2020. 17, 56, 75

Kshirasagar Naik and Priyadarshi Tripathy. *Software testing and quality assurance*. Wiley-Blackwell, 2008. 35

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010. 36

Junhyun Nam, Jaehyung Kim, Jaeho Lee, and Jinwoo Shin. Spread spurious attribute: Improving worst-group accuracy with spurious attribute estimation. *arXiv preprint arXiv:2204.02070*, 2022. 93

Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. *Advances in neural information processing systems*, 26:1196–1204, 2013. 107

Weili Nie, Yang Zhang, and Ankit Patel. A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. In *ICML*, 2018. 66, 75

Curtis G Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. *arXiv preprint arXiv:2103.14749*, 2021. 18, 97

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2 (11):e7, 2017. 20

Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012a. 63, 133

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012b. 67, 91, 133

Mohammad Pezeshki, Oumar Kaba, Yoshua Bengio, Aaron C Courville, Doina Precup, and Guillaume Lajoie. Gradient starvation: A learning proclivity in neural networks. *Advances in Neural Information Processing Systems*, 34:1256–1272, 2021. 24

Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. On fairness and calibration. *arXiv preprint arXiv:1709.02012*, 2017. 30, 97, 99, 147

Neoklis Polyzotis and Matei Zaharia. What can data-centric ai learn from data and ml engineering? *arXiv preprint arXiv:2112.06439*, 2021. 94

Forough Poursabzi-Sangdeh, Daniel G Goldstein, Jake M Hofman, Jennifer Wortman Vaughan, and Hanna Wallach. Manipulating and measuring model interpretability. *arXiv preprint arXiv:1802.07810*, 2018. 20, 75

Romila Pradhan, Jiongli Zhu, Boris Glavic, and Babak Salimi. Interpretable data-based explanations for fairness debugging. *arXiv preprint arXiv:2112.09745*, 2021. 109

Garima Pruthi, Frederick Liu, Mukund Sundararajan, and Satyen Kale. Estimating training data influence by tracking gradient descent. *arXiv preprint arXiv:2002.08484*, 2020. 51, 108, 130

Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. *arXiv preprint arXiv:1902.07208*, 2019. 131

Inioluwa Deborah Raji and Joy Buolamwini. Actionable auditing: Investigating the impact of publicly naming biased performance results of commercial ai products. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 429–435, 2019. 97

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021. 111

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 111

Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014. 107

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pages 4334–4343. PMLR, 2018. 97, 107

Marco Tulio Ribeiro and Scott Lundberg. Adaptive testing and debugging of nlp models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3253–3267, 2022. 94

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016. 19, 46, 74

Laura Rieger, Chandan Singh, W James Murdoch, and Bin Yu. Interpretations are useful: penalizing explanations to align neural networks with prior knowledge. *International Conference on Machine Learning*, 2020. 20, 74, 94

Christopher Rigano. Using artificial intelligence to address criminal justice needs. *National Institute of Justice Journal*, 280:1–10, 2019. 17

Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15):e2016239118, 2021. 17

David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017. 107

Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning*, pages 8230–8241. PMLR, 2020. 109

Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 2662–2670. ijcai.org, 2017. doi: 10.24963/ijcai.2017/371. URL https://doi.org/10.24963/ijcai.2017/371. 74, 94

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019. 94

Kaspar Rufibach. Use of brier score to assess binary predictions. *Journal of clinical epidemiology*, 63(8):938–939, 2010. 99

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 65, 133

Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*, 2019. 17, 30, 75, 90, 93

Shiori Sagawa, Aditi Raghunathan, Pang Wei Koh, and Percy Liang. An investigation of why overparameterization exacerbates spurious correlations. In *International Conference on Machine Learning*, pages 8346–8356. PMLR, 2020. 17, 75

Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278, 2021. 19

Shibani Santurkar, Dimitris Tsipras, Mahalaxmi Elango, David Bau, Antonio Torralba, and Aleksander Madry. Editing a classifier by rewriting its prediction rules. *Advances in Neural Information Processing Systems*, 34:23359–23373, 2021. 94

Frank Schneider, Felix Dangel, and Philipp Hennig. Cockpit: A practical debugging tool for the training of deep neural networks. *Advances in Neural Information Processing Systems*, 34:20825–20837, 2021. 36, 42

David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pages 2503–2511, 2015. 22, 42

Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020. 24

Harshay Shah, Prateek Jain, and Praneeth Netrapalli. Do input gradients highlight discriminative features? *arXiv preprint arXiv:2102.12781*, 2021. 76, 78

Lloyd S Shapley. A value for n-person games. *The Shapley value*, pages 31–40, 1988. 46

Hua Shen and Ting-Hao Huang. How useful are the machine-generated interpretations to general users? a human evaluation on guessing the incorrectly predicted labels. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 8, pages 168–172, 2020. 75

David Shepardson. Google spends hundreds of millions of dollars on content review: letter. *https://www.reuters.com/article/us-alphabet-google-youtube/google-spends-hundreds-of-millions-of-dollars-on-content-review-letter-idUSKCN1S81OK*, 2019. 97

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016. 45

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6034. 44

Sahil Singla and Soheil Feizi. Causal imagenet: How to discover spurious features in deep learning? *arXiv preprint arXiv:2110.04301*, 2021a. 23

Sahil Singla and Soheil Feizi. Salient imagenet: How to discover spurious features in deep learning? In *International Conference on Learning Representations*, 2021b. 95

Leon Sixt, Maximilian Granz, and Tim Landgraf. When explanations lie: Why modified bp attribution fails. *arXiv preprint arXiv:1912.09818*, 2019. 66, 75, 77

Leon Sixt, Martin Schuessler, Oana-Iuliana Popescu, Philipp Weiß, and Tim Landgraf. Do users benefit from interpretable vision? a user study, baseline, and dataset. *arXiv preprint arXiv:2204.11642*, 2022. 20

Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020. 76

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017. 45

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014. 45, 64

Suraj Srinivas and Francois Fleuret. Rethinking the role of gradient-based attribution methods for model interpretability. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=dYeAHXnpWJ4. 75, 76

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 107

Pierre Stock and Moustapha Cisse. Convnets and imagenet beyond accuracy: Understanding mistakes and uncovering biases. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 498–512, 2018. 18

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/sundararajan17a.html. 45

Josh Tobin, Sergey Karayev, and Pieter Abbeel. Troubleshooting deep neural networks: A field guide to fixing your model. 2019. URL http://josh-tobin.com/assets/pdf/troubleshooting-deep-neural-networks-01-19.pdf. 36, 42

Richard Tomsett, Dan Harborne, Supriyo Chakraborty, Prudhvi Gurram, and Alun D. Preece. Sanity checks for saliency metrics. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 6021–6029. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6064. 75

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 63, 133

Jialu Wang, Yang Liu, and Caleb Levy. Fair classification with group-dependent label noise. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 526–536, 2021. 108

Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016. doi: 10.23915/distill.00002. URL http://distill.pub/2016/misread-tsne. 20

Jenna Wiens and Erica S Shenoy. Machine learning for healthcare: on the verge of a major shift in healthcare epidemiology. *Clinical Infectious Diseases*, 66(1):149–153, 2018. 17

Eric Wong, Shibani Santurkar, and Aleksander Madry. Leveraging sparse linear layers for debuggable deep networks. In *International Conference on Machine Learning*, pages 11205–11216. PMLR, 2021. 94

Kai Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. Noise or signal: The role of image backgrounds in object recognition. *arXiv preprint arXiv:2006.09994*, 2020. 18

Yao-Yuan Yang and Kamalika Chaudhuri. Understanding rare spurious correlations in neural networks. *arXiv preprint arXiv:2202.05189*, 2022. 23

Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. In *Advances in neural information processing systems*, pages 9291–9301, 2018. 51, 108, 130

Chih-Kuan Yeh, Been Kim, Sercan Arik, Chun-Liang Li, Tomas Pfister, and Pradeep Ravikumar. On completeness-aware concept-based explanations in deep neural networks. *Advances in Neural Information Processing Systems*, 33, 2020. 75

Gal Yona and Daniel Greenfeld. Revisiting sanity checks for saliency maps. *arXiv preprint arXiv:2110.14297*, 2021. 77

Mert Yuksekgonul, Maggie Wang, and James Zou. Post-hoc concept bottleneck models. *arXiv preprint arXiv:2205.15480*, 2022. 84, 94

Aleš Završnik. Algorithmic justice: Algorithms and big data in criminal justice settings. *European Journal of criminology*, 18(5):623–642, 2021. 17

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 45

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021. 101, 107, 109

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 107

Xuezhou Zhang, Xiaojin Zhu, and Stephen Wright. Training set debugging using trusted items. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 87, 92, 93

Xiaojin Zhu, Adish Singla, Sandra Zilles, and Anna N Rafferty. An overview of machine teaching. *arXiv preprint arXiv:1801.05927*, 2018. 94

Roland S Zimmermann, Judy Borowski, Robert Geirhos, Matthias Bethge, Thomas Wallis, and Wieland Brendel. How well do feature visualizations support causal understanding of cnn activations? *Advances in Neural Information Processing Systems*, 34:11730–11744, 2021. 20

Zinkevich Martin. Rules of Machine Learning: Best Practices for ML Engineering. http://martin.zinkevich.org/rules_of_ml/rules_of_ml.pdf, 2020. Online; accessed 10 January 2020. 22

Appendix A

Chapter 4 Appendix: Challenges

In chapter, we provide additional details regarding the results presented in Chapter 4.

A.1 Experimental Details for Spurious Correlation

Feature Attributions: Implementation. We implement all of these methods from scratch in the PyTorch framework and also compare our implementations to the output of the Captum (PyTorch) library.

Concept-Based Approaches. We now discuss additional implementation details of our concept based approach. We select the TCAV approach to quantify the sensitivity of a DNN model's class score to user provided inputs represent a particular class. Given hidden representations, h_l, from a particular layer of a DNN for for images belonging to concept class C. We can derive the sensitivity score as: $\nabla h_{l,k}(f_l(x)).\theta_c^l$. The previous expression indicates the sensitivity of the class score (logit) for class k to inputs indicating concept, C, given hidden representations from layer l from the DNN f. The concept vector, θ_c^l, typically corresponds to a the weights of a linear classifier trained to separate the images for a particular concept class from or images.

In the bone-age task, we consider clinical concepts. These are the representative clinical attributes that a radiologist would inspect to ascertain the bone age of a particular input. These concepts are: DIP, PIP, MCP, Radius, Ulna, and Wrist.

Concept Implementation. To compute the TCAV score for each concept, we collect representations from all hidden 'layers' of the model and train linear models to obtains the concept vector for the corresponding attribute. We then compute the class sensitivity score for each concept attribute. We train the linear model 100 times and perform statistical significance testing in order to mitigate the case where a spurious concept is selected. For each concept class, we use 325 images that part of the training, validation, or test sets. These new set of images were annotated by a board certified radiology with the clinical bone age regions (MCP, PIP, DIP etc) that we chose.

Influence Functions for Training Point Ranking. The final kind of interpretation that we consider is training point ranking via influence functions. In the case of training point ranking via influence functions, we rank the training samples, in terms of 'influence', on the loss of a test example. Specifically, if we up-weighted a training point and retrained the model, then by how much would the loss on a given test example change? Koh and Liang [2017] analytically derive the analytically formulas for computing this quantity. Given a test point, x_t, the influence of a training point, x_i, on the test loss is: $I(x_t, x_i) = -\nabla_\theta \ell(x_t, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta \ell(x_i, \hat{\theta})$, where H is the empirical Hessian of the loss.

Estimating the influence requires computing hessian-vector products, so it can be difficult to scale to model with large number of parameters, and recent work has shown that influence estimate for test points for deep networks can be inaccurate due to non-convexity [Basu et al., 2020]. Consequently, we estimate influence on a linear model student network trained to mimic the original DNN. We empirically verify that the predictions of the student network seem to mimic the original DNN.

Implementation Details. There are two other training point ranking methods that we also consider in this work [Pruthi et al., 2020, Yeh et al., 2018]. For 150 inputs in the test set, we compare the Spearman rank correlation of the training point due to Influence Functions to these other two approaches. We obtain mean values of 0.88, and 0.76 respectively, which suggests high similarity amongst these approaches. Ultimately, we chose to present the main results in the draft for the training point ranking due to influence functions approach.

We rely on the fast influence heuristic of [Guo et al., 2020] to speed up the influence ranking computations. We were able to obtain a 5-10X speed in doing so. In addition, we

trained a student multi-class logistic regression model to mimic the original model for each model we want to compute influence for. Here for all training points, we collect embeddings across all layers and pass these embedding through a random projection to obtain a 1000-dimensional approximation. We then train linear models to mimic the original deep network using these features. The correlation between the output of the student models and the original teacher models was found to be 0.87. Ultimately, we went with the fast influence implementation of [Guo et al., 2020].

Knee Dataset We also consider the high stakes task of predicting the Kellgren and Lawrence (KL) grade of osteoarthritis based on Knee Xrays. The KL grade ranges from the integers 0 to 4; 5 classes, so we treat it as a classification task. The Knee Radiographs are obtained from the open source release by Chen et al. [2019] and consists of 5778 training samples, 826 validation samples, and 1656 test samples. Again here, we follow Chen et al. [2019] data splits. The images were resized to be 299 by 299 pixels.

Dog Dataset The third dataset that we use in this work is the dog breed classification dataset that is a combination of Stanford dogs dataset Khosla et al. (2011) and the Oxford Cats and Dogs datasets Parkhi et al. (2012b) following Adebayo et al. [2020]. Similarly, we restrict our attention to 10 dog classes: Beagle, Boxer, Chihuahua, Newfoundland, Saint Bernard, Pugs, Pomeranian, Great Pyrenees, Yorkshire Terrier, Wheaten Terrier. Instead of the background spurious signal ofAdebayo et al. [2020], we focus instead on the three spurious signals tested in this work.

Models We consider two different kinds of models: i) a small vanilla DNN based on Raghu et al. [2019], and a Resnet-50 model. The small DNN consists of: conv-relu-batchnorm-maxpooling operation successively, and two fully connected layers at the end. All convolutional kernels have stride 1, and kernel size 5. We train this model with SGD with momentum (set to 0.9) and an initial learning rate of 0.01. We use a learning rate scheduler that decays the learning rate every 10 epochs by $\gamma = 0.1$.

Hyper-Parameter Tuning for Model Training We used the Ray Tune library for hyper-parameter tuning of all the models used in this work. For the ResNet-50, we tuned with Ray, but the best performing models retained the default parameter settings. In the

case of the Small DNN model, we tune the batch size, and learning rate with the validation set.

A Note about random seeds and Model Runs. We train 5 models each (different random seeds) for each category and in the following tables the average performance metrics for these models. We found that the standard error of the mean for typically between $0.01 - 0.05$, so we omit these in the tables to improve readability.

A.2 Experimental Details Mislabelled Examples, Weight-Reinitialization, & Out-of-Distribution Robustness

Here we provide a detailed overview of the data set and models used in our experiments.

Birds-Vs-Dogs Dataset. We consider a birds vs. dogs binary classification task for all the data contamination experiments. We use dog breeds from the Cats-v-Dogs dataset Parkhi et al. [2012a] and Bird species from the caltech UCSD dataset Wah et al. [2011]. These datasets come with segmentation masks that allows us to manipulate an image. All together, this birds-vs-dogs dataset consists of 10k inputs (5k dog samples and 5k bird samples). We use 4300 data points per class, 8600 in total for training, and split the rest evenly for a validation and test set.

Modified MNIST and Fashion-MNIST Datasets. For the test-time contamination tests, we modify the MNIST and Fashion-MNIST datasets to have 3 channels and derive attributions from this three-channel version of MNIST from a VGG-16 model.

ImageNet Dataset. We use two 200 images from the ImageNet Russakovsky et al. [2015] validation set for the model contamination tests.

User Study Dogs Only Dataset. For the user study alone, we restrict our attention to 10 dog classes. We used a modified combination of the Stanford dogs dataset Khosla et al. [2011] and the Oxford Cats and Dogs datasets Parkhi et al. [2012b]. We restrict to a 10-class classification task consisting of the following breeds: *Beagle, Boxer, Chihuahua, Newfoundland, Saint Bernard, Pugs, Pomeranian, Great Pyrenees, Yorkshire Terrier, Wheaten Terrier.* We are able to create spurious correlation bugs by replacing the background in training set images. We consider 10 different background images representing scenes of: *Water Fall, Bamboo Forest, Wheat Field, Snow, Canyon, Empty room, Road or Highway, Blue Sky, Sand Dunes, and Track.*

BVD-CNN For the data contamination tests, we consider a CNN with 5 convolutional layers and 3 fully-connected layers with a ReLU activation functions but sigmoid non-linearity in the final layer. We train this model with an Adam optimizer for 40 epochs to achieve test

accuracy of 94-percent. For ease of discussion, we refer to this architecture as *BVD-CNN*. We use a learning rate of 0.001 and the ADAM optimizer. This is the standard BVD-CNN architecture setup that we consider.

User Study Model. Here we use a ResNet-50 model that was fine-tuned on the dogs only dataset to generate all the attributions for the images considered. Please see public repository for model training script.

A.2.1 Mislabelled Examples

Bug Implementation. We train a BVD-CNN model on a birds-vs-dogs dataset where 10 percent of training samples have their labels flipped. The model achieves a 94.2, 91.7, 88 percent accuracy on the training, validation and test sets. We show additional examples in later paper of the supplemental material.

A.2.2 Weight Re-initialization

The model contamination tests capture defects that occur in the parameters of a model. Here, we consider the simple setting where a model is accidentally reinitialized during or while it is being used. As expected, such a bug will lead to observable accuracy differences. However, the goal of these classes of tests, especially in the model attribution setting, is to ascertain how well a model attribution is able to identify models in different parameter regimes.

A.3 Additional Details on Mislabelled Examples

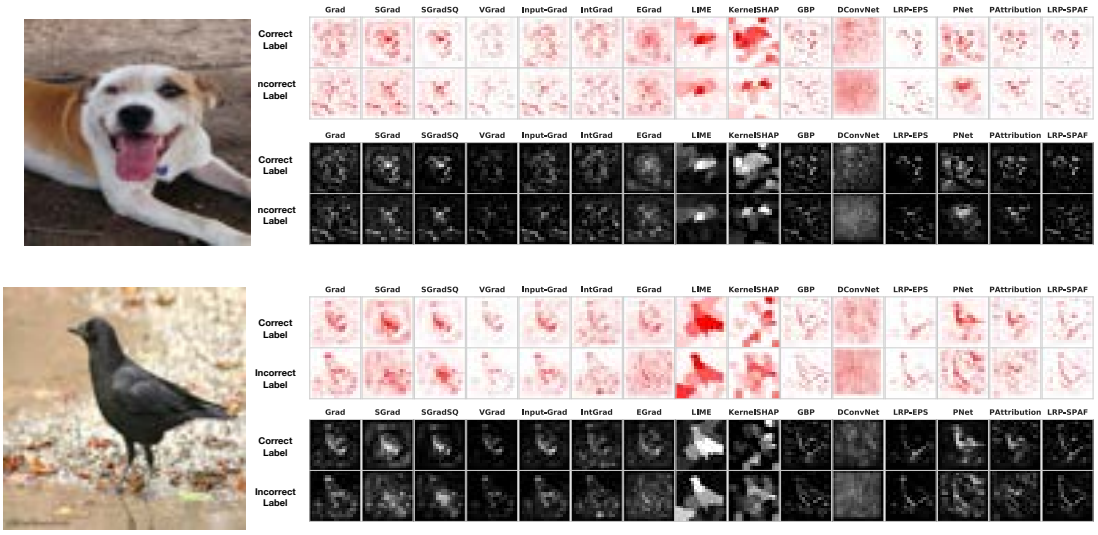

Figure A-1: **Feature Attributions for Mislabeled examples.**

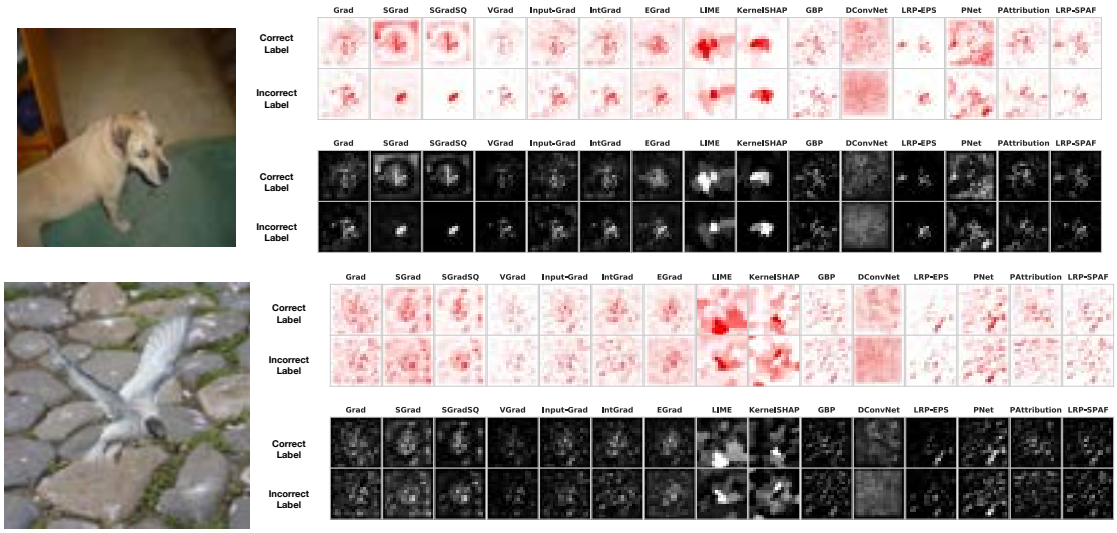

Figure A-2: **Feature Attributions for Mislabeled examples.**

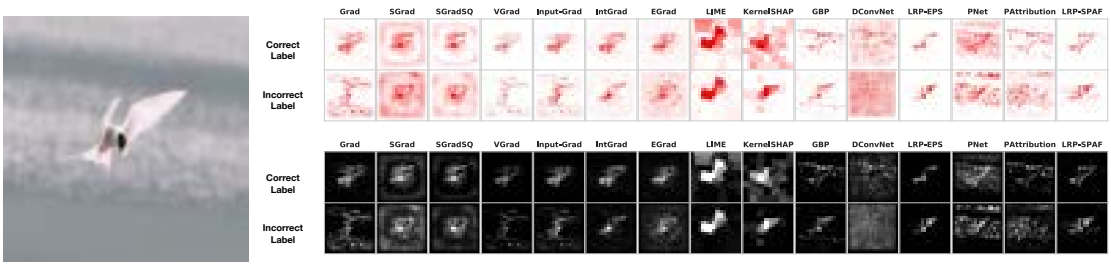

Figure A-3: **Feature Attributions for Mislabeled examples.**

A.4 Additional Details on User Study

We now present additional images that show the samples used as part of the user study conducted in Chapter 4.

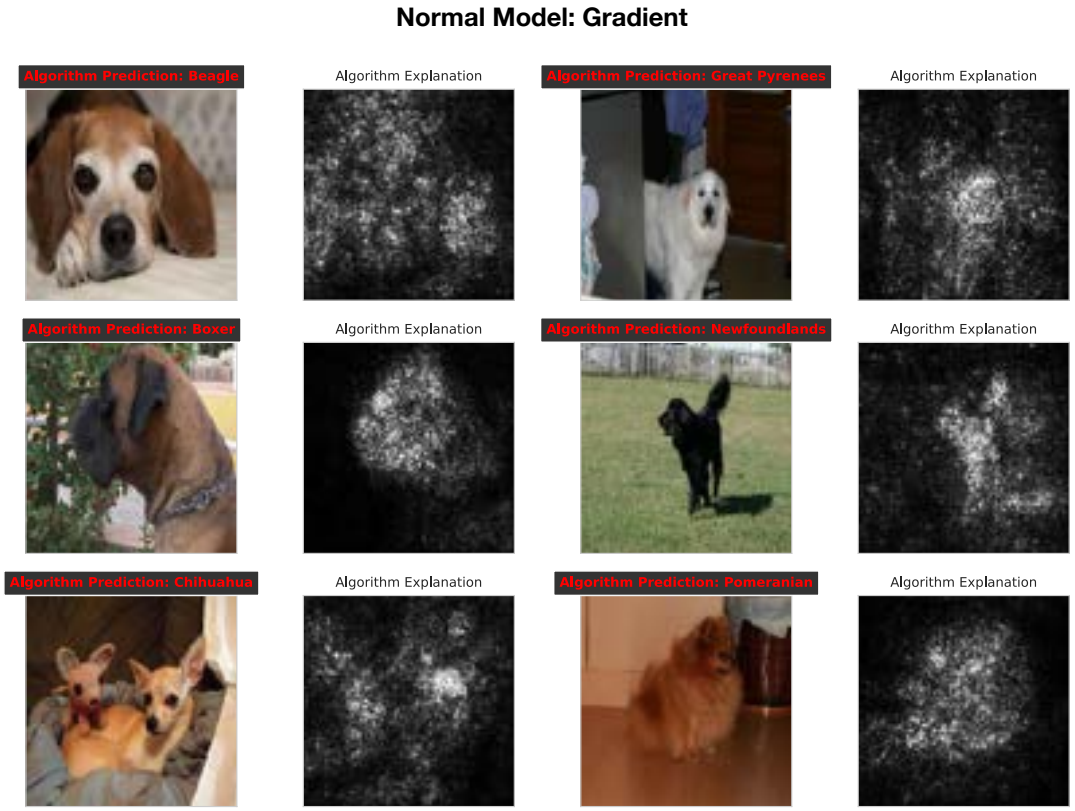

Figure A-4: **Saliency Maps for a Normal Model: Gradient.**

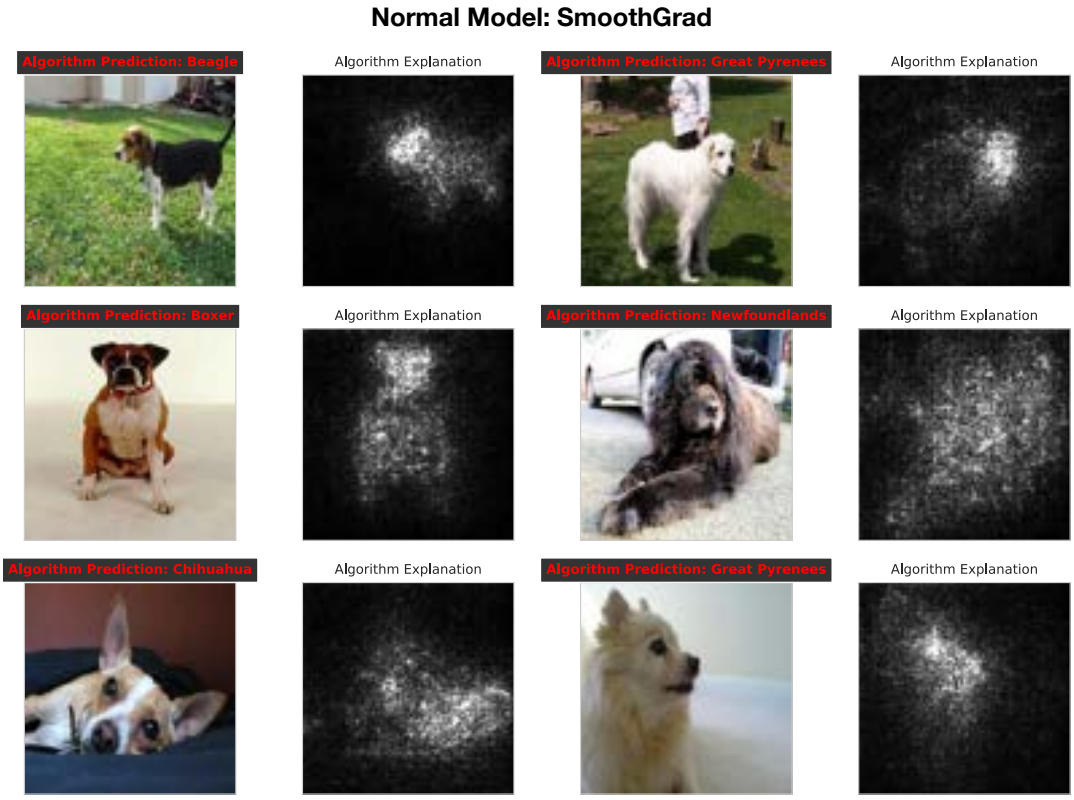

Figure A-5: **Saliency Maps for a Normal Model: SmoothGrad.**

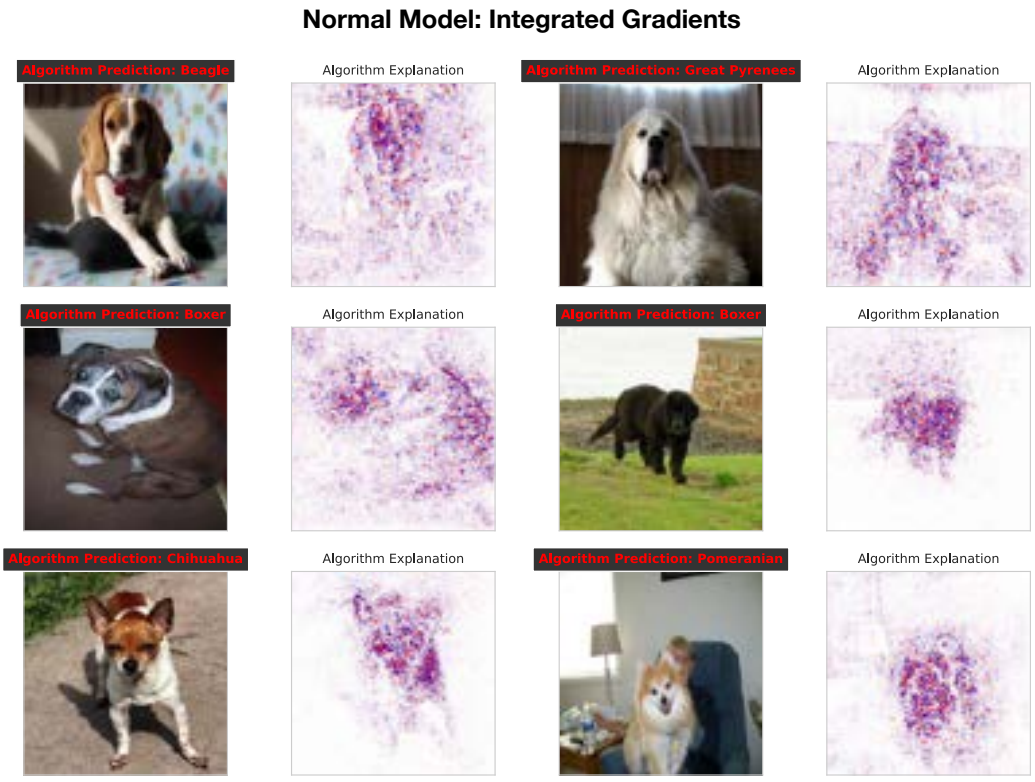

Figure A-6: **Saliency Maps for a Normal Model: Integrated Gradients.**

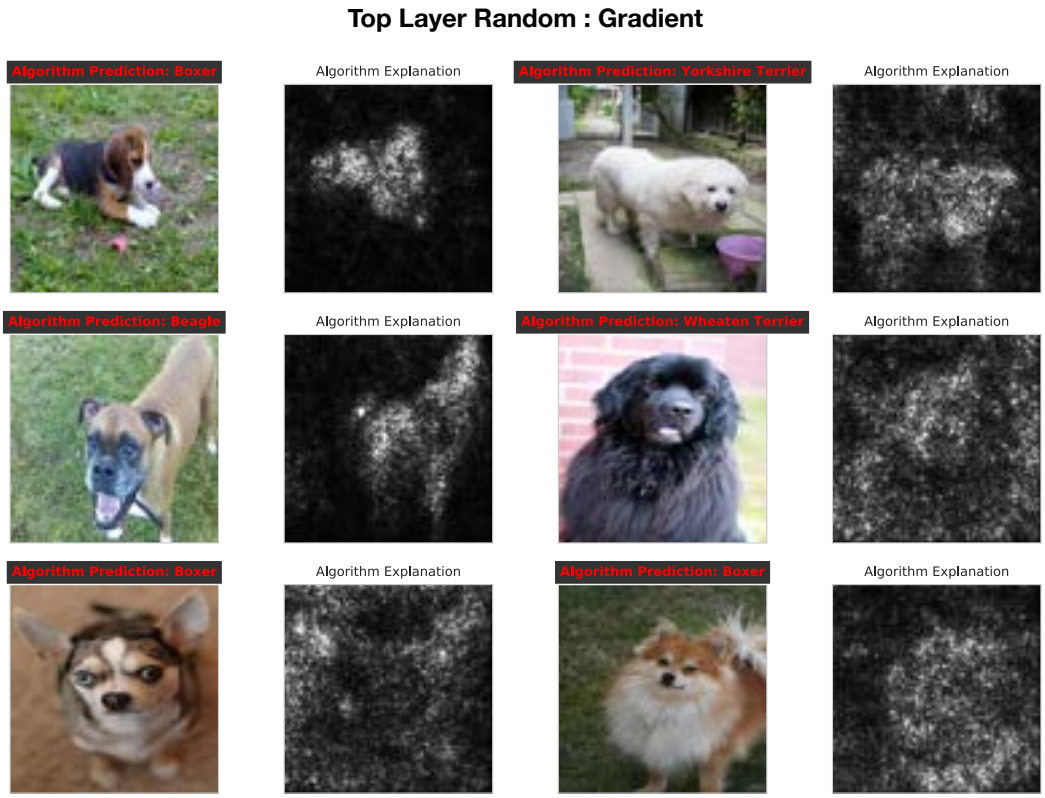

Figure A-7: **Saliency Maps for a Top layer Random Model: Gradient**

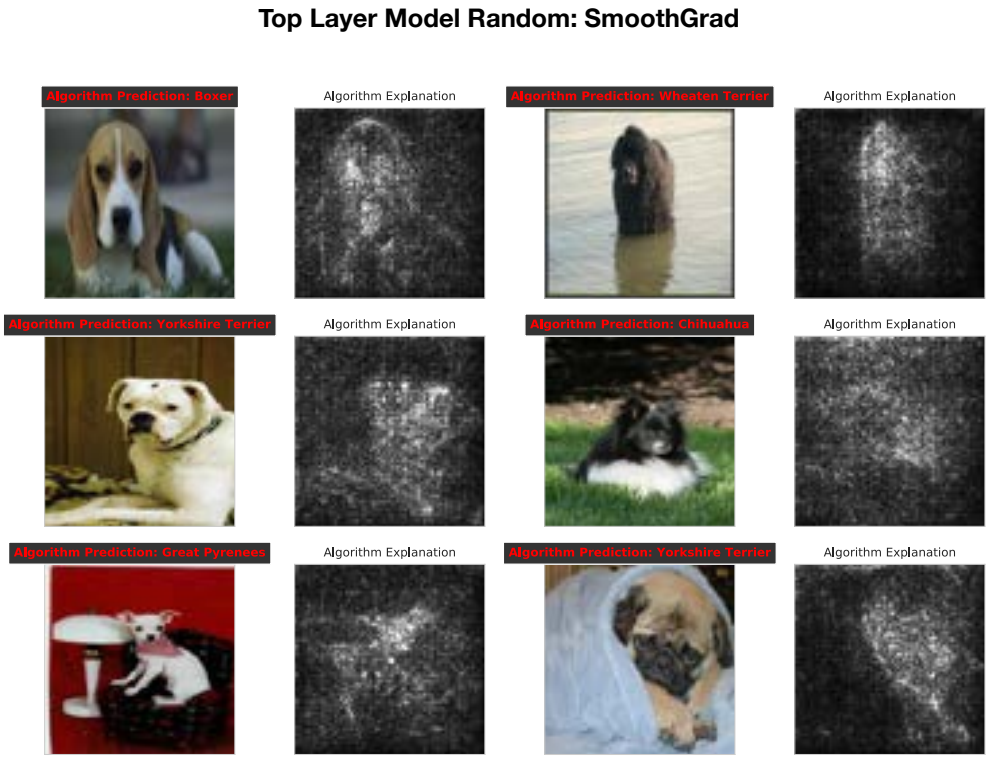

Figure A-8: **Saliency Maps for a Top layer Random Model Model: SmoothGrad.**

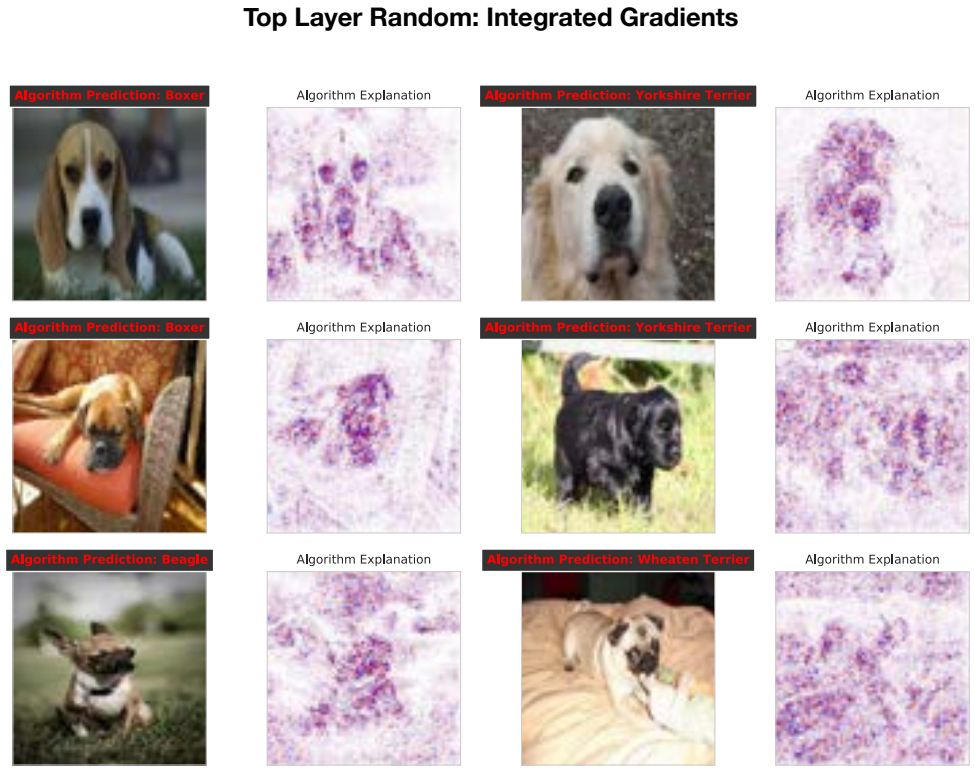

Figure A-9: **Saliency Maps for a Top layer Random Model Model: Integrated**

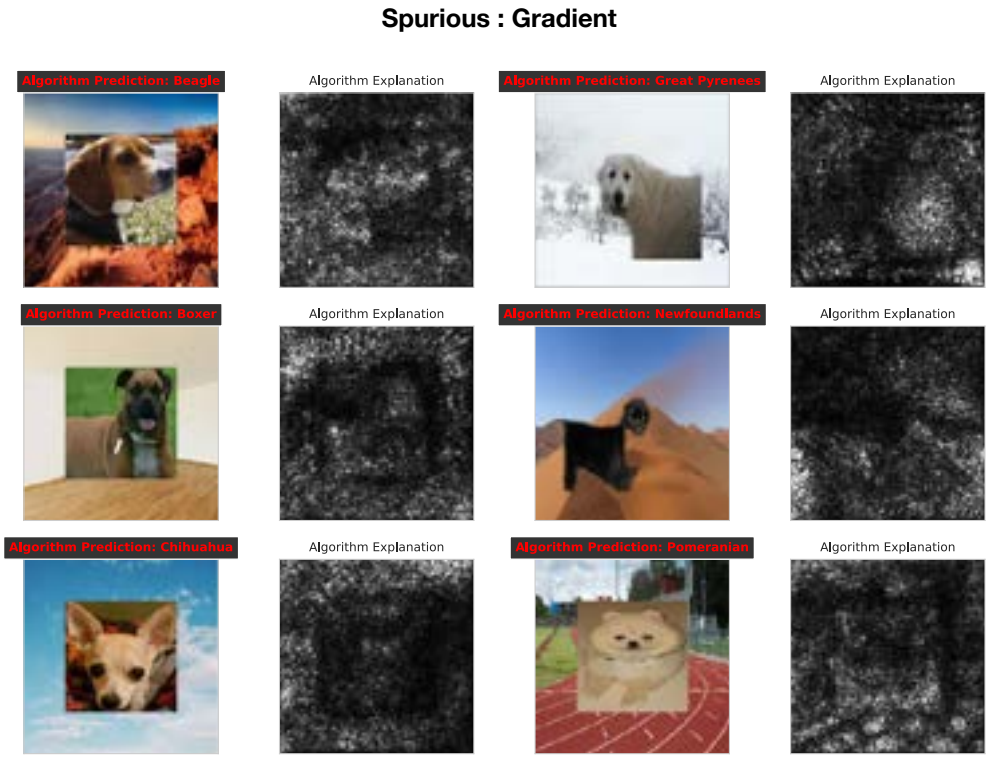

Figure A-10: **Saliency Maps for a Spurious Model: Gradient.**

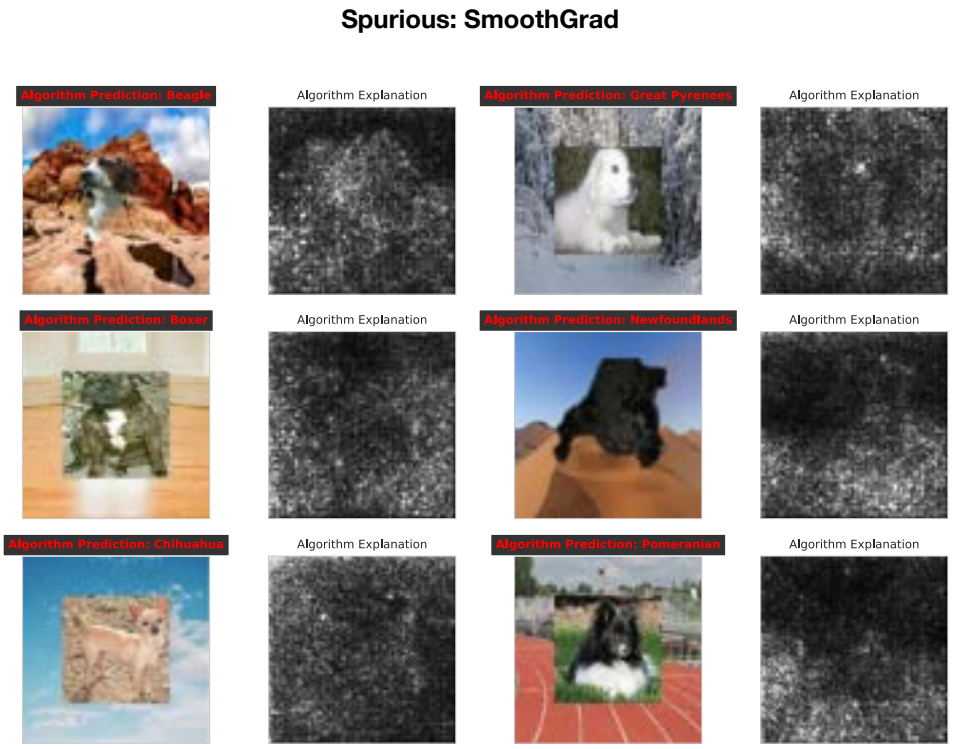

Figure A-11: **Saliency Maps for a Spurious Model: SmoothGrad.**

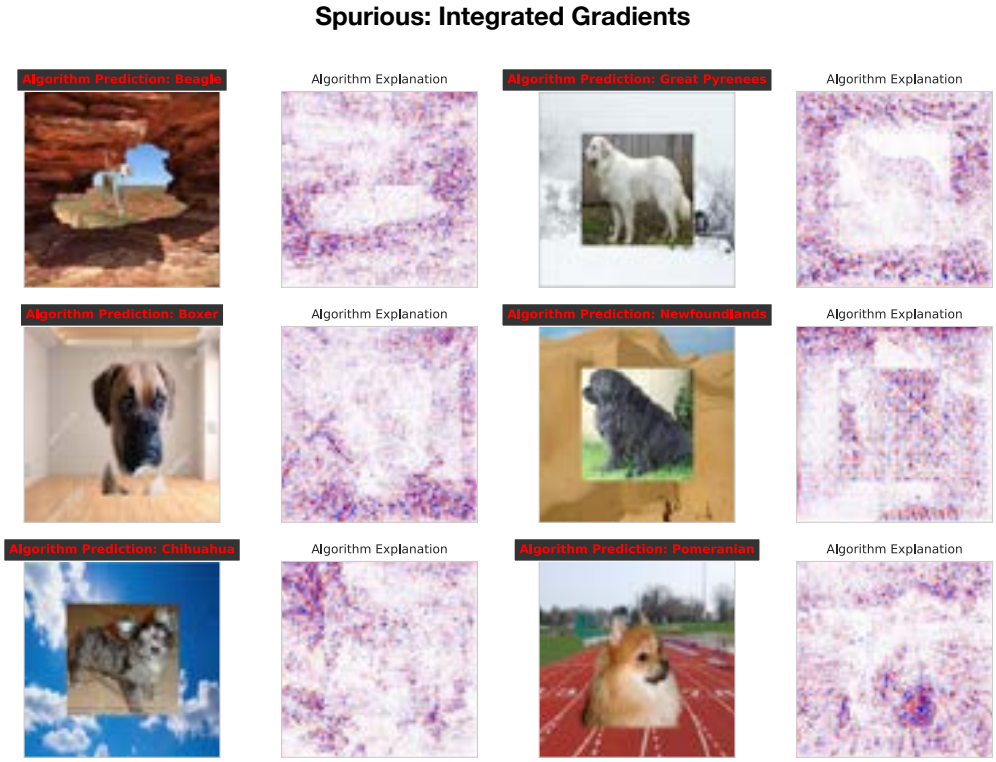

Figure A-12: **Saliency Maps for a Spurious Model: Integrated Gradients.**

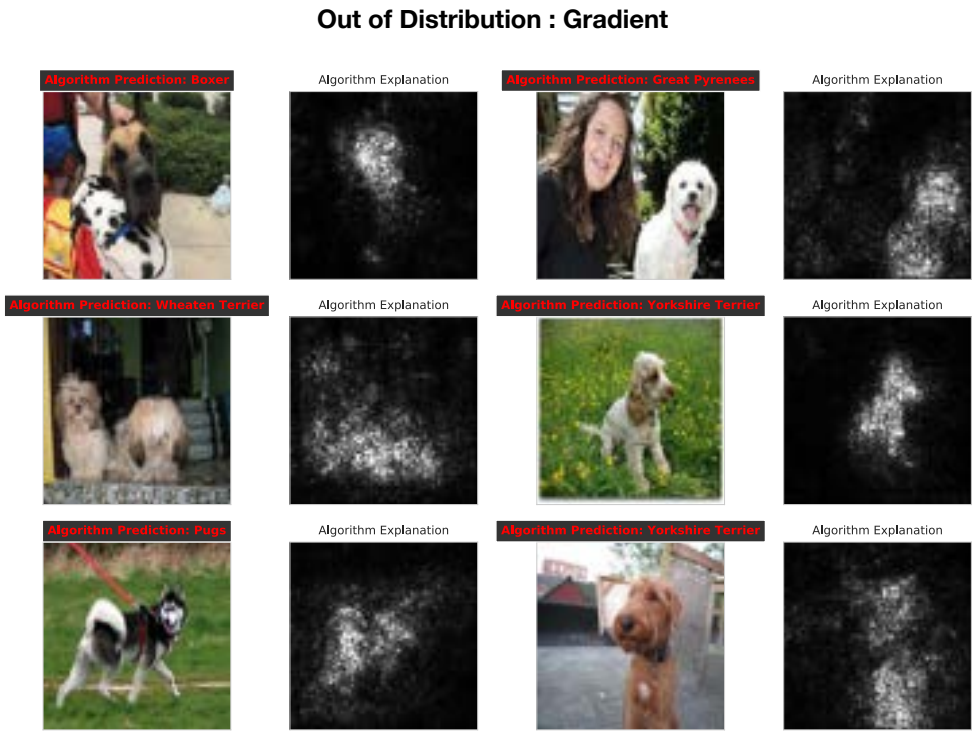

Figure A-13: **Saliency Maps for Out-of-Distribution Examples: Gradient.**

Appendix B

Chapter 6 Appendix: Influence Functions

B.1 Appendix: Datasets, Models, & Experimental Details

We provide additional details.

B.1.1 Datasets

We start with a discussion of the datasets used in this work.

Text Toxicity Classification we use a publicly available dataset from JIGSAW called the *unintended bias in toxicity classification* dataset. The TC dataset contains a subset of comments from the 'Civil Comments' platform that have been annotated by human raters for level of toxic severity. For example, a comment can be tagged as 'benign', 'obscene', 'threat', 'insult', or 'sexually explicit' among several categories. A given text is then indicated as 'toxic' or 'not toxic' on the basis of these tags. The task here is a binary classification one, which is to categorize an input text as either toxic or not.

To obtain the toxicity labels, each comment was shown to up to 10 annotators. Annotators were asked to:"Rate the toxicity of this comment": Very Toxic, Toxic, Hard to Say, and Not Toxic. These ratings were then aggregated with the target value representing the fraction of annotations that annotations fell within the former two categories.

To collect the identity labels, annotators were asked to indicate all identities that were mentioned in the comment. An example question that was asked as part of this annotation

effort was:"What genders are mentioned in the comment?": Male, Female, Transgender, Other Gender, and No gender mentioned. We consider the Gender variable to be the sensitive attribute of interest.

For the dataset, we taken of the raw sentence text and convert them into embedding vectors using the XLM-R models obtaining a vector, for each sentence, that is 768 dimensional. We then further reduce the input dimension of this vector from 768 to 50 via a random projection. We make this reduction to make computing the hessian of the loss function of the logistic regression model trained on this data easy to compute. The 50-dimensional embedding is then used as input for our analyses.

Adult Census Dataset we use a series of tabular datasets more broadly called the 'Adult Dataset'. Specifically, we consider a recently revamped version of the dataset introduced by Ding et al. [2021], which is derived from the broader US Census. We consider the following tasks among the compilation of datasets available in this database:

1. ACSIncome Prediction, where the task is to predict whether an individual has an income above 50000 USD;

2. ACSIncome Prediction (25k) where the task is to predict whether an individual has an income above 25000 USD;

3. ACSEmployment Prediction, where the task is to predict whether the adult individual is employed; and

4. ACSPublic Coverage: where the task is to predict whether a low-income individual has coverage from public health insurance.

The adult dataset comes annotated with race categories, which we take as the key demographic variable in our analyses. We restrict our analyses to data from year 2018 for the state of California across all datasets.

Credit Dataset : a dataset of financial transactions that consists of aggregate demographic and credit information for customers of a large commercial bank. For each customer, information includes their gender, marital status, educational level, employment status, income, age, and asset. Using this information, the bank estimates probability of default on

loan for each customer (a scalar between 0 and 1). We consider the prediction of the default probability as our primary focus. The dataset comes with gender as a sensitive attribute for two categories: Male and Female. While this dataset is not publicly available—it is hosted by the first author's institution, it has been used as part of previous work studying financial well-being classification.

B.1.2 Models

All but one of the datasets we consider are tabular, and mostly low-dimensional. We implement the logistic regression model in PyTorch. We tuned the batch size using the validation set in the range [16, 32, 64, 128] across all datasets, but did not observe substantial across differences, so we set default batch size to be: 128. We use the SGD plain optimizer with a default learning rate of 0.001, we train the all models for 20 epochs. We adapat the standard GBT implementation for sklearn, and standard Resnet-18 from PyTorch to our setting.

B.1.3 Additional Discussion on ECE

The principal metric that we consider in this work is group calibration. We also consider other group-based metrics such as true-positive and false negative rates. Let $\hat{y}_i = h(x_i)$ be the output of a trained model of interest for input x_i. The output can either be an output probability for a binary prediction or a class prediction in a multi-class setting. We denote the ground-truth confidence or probability of correctness as $\hat{p}_i$. We say h is calibrated if $\hat{p}_i$ represents a true probability. As an example, if given 100 predictions with confidence 0.95, then if h is calibrated, 95 of these predictions should be correct given ground-truth labels.

Several recent works have also studied model calibration and found that modern neural network models are uncalibrated. Guo et al. [2017] found that a simple temperature based Platt-scaling post processing was the most effective technique. While the literature abound with disparity metrics, several of which provide conflicting and often counter-intuitive insights simultaneously, group calibration has emerged as a useful metric in practice Pleiss et al. [2017].

There are a variety of metrics for quantifying a model's level of calibration. In this work, we measure calibration with using the metric: Expected Calibration Error (ECE) Naeini et al. [2015]. A calibration metric summarizes the difference between the empirical distribution of

a model's prediction and the ground-truth probabilities for a perfectly calibrated classifier. Here, perfect calibration is defined as:

$$\mathbb{P}\left(\hat{y} = y | \hat{p} = p\right), \forall p \in [0, 1].$$

In practice, the quantity above is impossible to compute, so previous literature made empirical approximations. We follow binning the approach by Guo et al. [2017]. To estimate the accuracy empirically, we group predictions into M interval bins (each of size 1/M) and calculate the accuracy of each bin. Across all experiments, we take $M = 10$—recent work Küppers et al. [2020] found this default setting effective across a range of datasets. We discuss this choice in more detail in the appendix. Let B_m be the set of indices of samples whose prediction confidence falls into the interval $I_m = [\frac{m-1}{M}, \frac{m}{M}]$. Here the accuracy of B_m is then:

$$\text{acc}(\text{B}_\text{m}) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}(\hat{y}_i = y_i),$$

where $\hat{y}_i$, and y_i are the predicted and true class labels for data sample i.

The ECE is the absolute difference in expectation between confidence and accuracy. We can define the average confidence within a bin B_m as:

$$\text{conf}(\text{B}_\text{m}) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i,$$

where $\hat{p}_i$ is the confidence for sample i. Consequently, a perfectly calibrated model will have accuracy and confidence equal for all bins.

A notion of miscalibration is then which is the difference in expectation between confidence and accuracy.

The ECE approximates the above notion of miscalibration, and is defined as:

$$\sum_{m=1}^{M} \frac{|B_m|}{n} |\text{conf}(\text{B}_\text{m}) - \text{acc}(\text{B}_\text{m})|.$$

We use ECE as the primary metric of miscalibration in this work. As we will see, we will compare ECE metrics across various data groups to help identify input groups where a model is miscalibrated.

For the TC dataset, we obtain embedding vectors, 768 dimensional, for all samples using the XLM-R, a state of the art multi-lingual model Conneau et al. [2019]. We then further reduce the input dimension of this vector from 768 to 50 via a random projection. We make this reduction to make computing the hessian of the loss function of the logistic regression model trained on this data easy to compute. The 50-dimensional embedding is then used as input for our analyses. Across all datasets, we take 70 percent of each dataset for training and model validation, while we keep 30 percent as a holdout set for computing the disparity metrics. For the 70 percent portion, we reserve 20 percent as validation set.