Machine Learning Beyond Accuracy: A Features Perspective On Model Generalization

by

Shibani Santurkar

B.Tech & M.Tech., Indian Institute of Technology Bombay (2015)
S.M., Massachusetts Institute of Technology (2017)

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 2021

Author . . . .
Department of Electrical Engineering and Computer Science
July 15, 2021

Certified by . . . .
Aleksander Mądry
Cadence Design Systems Professor of Computing
Thesis Supervisor

Certified by . . . .
Nir Shavit
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

Machine Learning Beyond Accuracy: A Features Perspective On Model Generalization

by

Shibani Santurkar

Submitted to the Department of Electrical Engineering and Computer Science
on July 15, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Prompted by its performance on a variety of benchmark tasks, machine learning (ML) is now being applied to tackle real-world problems. Yet, there is growing evidence that benchmark performance does not convey the full picture. Existing ML models turn out to be remarkably brittle: a striking example of which is their susceptibility to imperceptible input perturbations known as *adversarial examples*.

In the first part of this thesis, we revisit adversarial examples, to use them as a window into current models. Our investigation provides a new perspective on why this susceptibility arises: it is a direct consequence of models' reliance on *predictive*, yet *brittle* input features. In fact, our findings demonstrate that adversarial examples are a manifestation of a deeper problem: the mechanisms by which current models succeed on benchmarks are *fundamentally misaligned* with what humans tend to envision. This prompts the question:

How can we build ML models that generalize not only on the benchmarks used for their development but also to the real world?

To answer this question, we examine the ML pipeline from a "features perspective": focusing not only on *what label* models predict, but also on *what features they use* to do so. To this end, in the second part of this thesis, we develop a suite of tools to get a better grasp on: (i) what features models learn, (ii) why they learn them, and (iii) how one can modify the learned features at train or test time. These tools enable us to gain new insights into crucial design choices made during model development, such as how we create datasets, and train and evaluate models. Equipped with these insights, we then propose concrete refinements to the ML pipeline to improve model generalization in the aforementioned broader sense.

Thesis Supervisor: Aleksander Mądry
Title: Cadence Design Systems Professor of Computing

Thesis Supervisor: Nir Shavit
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I feel truly indebted to many, for the privilege to pursue research and write this thesis, and for their love and support along the way.

First and foremost, I want to thank Nir Shavit and Aleksander Mądry, who have been the best advisors one could ask for. I am grateful to Nir for guiding me through the early stages of my PhD, facilitating my whimsical exploration and helping me discover my research interests, motivational speeches about "the cloud" during rough times, and an unending supply of Lindt truffles. Nothing that I have done in my PhD would have been possible without Aleksander, always happy to provide his insights on any matter from research, to food, vacuums and life. There is so much I have learned from him—and so much more I wish I could—not only technically, but also in terms of his authenticity, dedication and perfectionism. I am indebted to Aleksander for going out of his way to support all my professional and personal endeavors, and treating me no short of family. There is not enough I can say to thank Nir and Aleksander, so I must leave it at that.

Next, I want to thank my thesis committee, Zachary Lipton and Antonio Torralba, for their advice through this process. I am also thankful to my undergraduate advisor Bipin Rajendran, for giving me my first shot at research and encouraging me towards graduate school. During the final stages of my PhD, I was fortunate to be supported by the Google fellowship.

The best part of these past six years has been the the opportunity to collaborate with several phenomenal people: David Bau, David Budden, Shraman Chaudhari, Mahi Elango, Logan Engstrom, Andrew Ilyas, Guillaume LeClerc, Alex Matveev, Hadi Salman, Ludwig Schmidt, Jacob Steinhardt, Kunal Talwar, Antonio Torralba, Brandon Tran, Dimitris Tsipras, Alex Turner, Eric Wong and Kai Xiao. I have learned so much from each of them and they have been instrumental to making this an amazing journey. I also want to thank all the members of our lab (including Natalia, Samanta and Kamila) for creating an incredible environment, full of discussions and dumplings. Some of the most memorable times of my PhD were spent on the sixth floor of Stata, in the theory group. I am thankful to all its members, especially Rati, Ilya, Sepideh, Mira, Jerry, Kyriakos, Pritish, Nishanth, Adam and Saleet. I am also grateful to Debbie, Joanne and Rebecca for everything they have done to support us and for always keeping the snack coffers full.

I would not have been able to survive this journey without my friends, who have had my back all these years and kept me going. I want to thank all my friends from high-school and undergrad, with a special shoutout to Gourav, Murtuza, Rushina, Ishita, Bo, Gangrade, Niladri, Parthe, Arpit, Priyank, Yamini and Upasana. New friends that I made in Cambridge, that made life here amazing: Ian, Nigamaa, Apoorva, Devendra, Namrata, Eva, Chara, Lydia, Lydia, Marinos, Konst, Christina, Manolis, Katerina, Sophie, Konstantina, and Terra. Talking about my life in Cambridge would be incomplete without mentioning Andrew and Logan who are, at this point, family. I also want to thank Shamoyita, Ayra, Rohit and Raisha for dance and providing me with a home away from home.

Being able to come so far from home to pursue a PhD is a privilege, and I owe it entirely to my family. First, my extended family: nani, nana and dodda; my cousins Avni, Aditi, Mihir, Saachi, Vir, Akshay, Apoorva, Pranav, Kritika, Tejas; aunts and uncles Madhu, Ajay, Bhavana, Sumit, Chandru, Shweta, Mangal, Arvind, Vivek and Mamatha; Iraklis and Eftychia. Most of all, I want to thank my parents, Preeti and Vinay, and my sister Ishani—there is just no way to acknowledge all you have done for me. Finally, I want to thank Dimitris for being the best partner in work and life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last decade, the field of machine learning (ML) has witnessed remarkable progress: enabled by the confluence of fast compute and large-scale datasets, and, crucially, renewed interest in a family of models known as *deep neural networks*. Deep learning, in particular, emerged as a popular approach with its success on a classic computer vision task: the ImageNet large scale visual recognition challenge (ILSVRC) [Den+09; Rus+15]. By now, deep networks have greatly improved state-of-the-art on benchmarks across domains—such as natural language [Bow+15; Raj+16; Wan+18], speech [Dah+10; Den+10], robotics [Fin+16], games [Sil+17; Ber+19], and biology [Joh+16; Wan+17; Sen+20]. One of the major appeals of these models stems from the belief that they can extract meaningful high-level feature representations from data [BCV13; Ben19; GBC16]. Consequently, deep networks are viewed as being highly versatile: they can be effectively leveraged in a variety of tasks and settings. But is this indeed so?

1.1 A Dent in the Picture

If we examine deep networks a bit more closely, inconsistencies start to appear: the performance of these models on benchmarks actually turns out to be quite brittle. A striking illustration of this is the phenomenon of *adversarial examples* [Big+13; Sze+14]. Namely, for classification tasks, one can fool a state-of-the-art deep network into making an incorrect prediction by applying a carefully-crafted, yet *imperceptible* perturbation to the input, as shown in Figure 1-1.

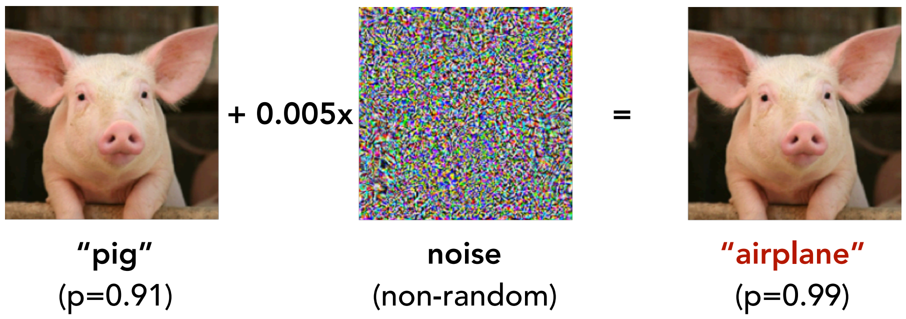


Figure 1-1: An adversarial example: It is possible to fool state-of-the-art vision classifiers on a correctly-classified input (*left*) by adding a carefully-chosen *imperceptible* perturbation to it (*center*). On the resulting adversarial example (*right*), the model makes a highly-confident, yet erroneous prediction.

The existence of adversarial examples clearly indicates that there is a problem with current models: the two "pig" images shown above are indistinguishable to humans, yet high-performing models classify them differently. This problem can, in fact, also cause failures in deep learning systems deployed in the real world—e.g., models misclassify photographs taken with a smartphone camera [KGB16], or of objects that have been intentionally modified [Sha+16; Ath+18; Eyk+18].

In light of this, let us now return to the drawing board and reexamine what we know about these models. To begin with, we need to understand: what causes models to be fooled by adversarial examples?

1.1.1 Why do adversarial examples arise?

Tracing the origins of adversarial examples has, in fact, been a major focus of research in ML [Sze+14; GSS15; Sch+18; Gil+18; MDM18; BPR19; Sha+19a]. The prevalent view in the field has been that they arise due to "unnecessary sensitivities" of the model, caused, for example, by the high dimensionality of the input space, statistical fluctuations in the data and/or overfitting [Sze+14; GSS15; Gil+18; TG16]. That is, adversarial examples stem from glitches in the model that do not affect its behavior under normal test conditions. Fixing these glitches is thus treated as a goal that can be disentangled and pursued independently from maximizing accuracy [Mad+18; SHS19; Sug+19]—through regularization [TG16] or by pre/post-processing the network's inputs/outputs [Ues+18; CW17; He+17].

A new perspective on adversarial examples. In this thesis, we demonstrate that this canonical view on adversarial examples may not be entirely justified. We show that adversarial vulnerability is inextricably linked to how models make predictions. Specifically, that they do so based on *predictive* features in the data that are *imperceptible or unintuitive* to humans (Chapter 2). But why is this the case?

The answer to this lies in the fact that in supervised learning, models are typically trained to maximize accuracy. In this case, a model is incentivized to exploit any available signal in the data—even features that look incomprehensible to humans. After all, a priori, a “tail” is no more natural to a model for detecting a “dog” than another equally predictive feature. Indeed, it turns out that this is the case in practice. That is, standard datasets do contain such predictive yet incomprehensible “non-robust” features. Current models, in turn, end up relying on such features to attain accuracy on the corresponding benchmarks, but do so at the cost of robustness to small input changes.

A key consequence of this is a new perspective on adversarial examples. Namely, that they should no longer be viewed purely as a security concern, but as a marker of a *fundamental misalignment* between how we expect models to behave and how they actually make predictions. In other words, adversarial examples are a “human phenomenon”: we should not be surprised that models exploit predictive features that happen to be brittle under a *human-selected* notion of similarity.

1.1.2 A fundamental misalignment

It turns out that the aforementioned misalignment between humans and ML models goes far beyond adversarial examples. Once we start pulling this thread, we see numerous instances where models rely on brittle features in the data—for example, image texture [Gei+19], backgrounds [Zha+07; RSG16b; RZT18; Bar+19; Xia+20] and high-frequency information [Yin+19]. This misalignment provides a justification for various other inconsistencies observed in deep learning. For instance, it explains the heightened sensitivity of current models not only to adversarial examples but also to fairly benign changes in standard datasets, such as data corruption [Hen+19b], camera angle and lighting [BVP18], object pose [Bar+19] and affine transformations [Eng+19c]. After all, some of the features that are predictive within standard datasets could be useless or even spurious for mild variants of them. Further, it also hints at why models tend to be hard to interpret [Lip18; Ade+18; Ade+20]—they often do rely on features that are unintelligible to humans.

The bigger picture. More broadly, this misalignment hints at deeper issues: not only with current models but also with the way in which we develop them in the first place. Namely, our dominant learning paradigm—optimizing models to solely maximize accuracy—is flawed. Models trained in this fashion tend to rely on *any* predictive patterns in the data, even ones that are misleading or brittle. Since typical test sets are drawn from the same distribution (and thus share the same features) as the training data, benchmark accuracy (or i.i.d. generalization) masks this problem. Overall, this implies that it is possible for models to succeed on (challenging) benchmarks without truly solving the motivating real-world task. This prompts the central question examined in the second part of this thesis:

How can we build ML models that generalize not only on the benchmarks used for their development but to real-world tasks we ultimately care about?

The quest to answer this question makes us rethink basic design choices made during ML development: how we create datasets, and train/evaluate models.

1.2 A Features Perspective on the ML Pipeline

To tackle the question posed above, we start by looking back at our findings on adversarial examples (Section 1.1.1). In particular, we posit that building models that truly generalize requires us to rethink the ML pipeline from what we call a "features perspective". Concretely, we need to look beyond *what label* models predict on benchmarks, and also pay heed to *which features* these predictions are based on. In doing so, our focus should be on assessing and ensuring the pertinence of these features in the real-world task we eventually hope to tackle. Note that aside from improving generalization, by penalizing models for relying on meaningless or brittle features, this perspective can guide the design of models to meet other key considerations, such as interpretability or fairness.

In the second part of this thesis, we thus examine the ML pipeline from such a features perspective, characterizing: (i) the features that models depend on, (ii) where these dependencies stem from, and (iii) how we can modify them.

1.2.1 What features do models learn?

As we discussed in the previous section, existing models often base their predictions on undesirable features in the data. Aside from adversarial examples, other instances of this include visual artifacts used by skin-lesion detectors [BVA20], and image backgrounds exploited by object recognition systems [Zha+07; RSG16b; RZT18; Bar+19; Xia+20]. Thus, to determine whether a model will generalize to real-world settings, we must first characterize the features it relies on. More broadly, as ML becomes pervasive in sensitive domains (e.g., healthcare or recidivism), the need for model auditing mechanisms becomes even more pressing. In principle, one could recruit domain experts to conduct such audits. However, while effective [BG18; SC18; Tsi+20; Sey+20], this approach does not always scale. Thus, there is a rising demand for *general-purpose* model understanding tools.

The challenges with model interpretability. Indeed, the whole field of interpretability is devoted to the problem of characterizing how a model make its predictions. For deep networks, however, this problem turns out to be far from trivial. The major challenges that prior work has focused on are:

1. *Complexity:* Direct model inspection, which is the standard approach in simpler settings (such as for small linear classifiers or decision trees), quickly be-

comes infeasible. To circumvent this problem, prior work attempts to characterize model behavior in a localized or decomposed manner [SVZ13; Yos+15; RSG16a; Bau+17]. However, such interpretations are often hard to aggregate, and can even be misleading [Ade+18; Ade+20; LM20].

2. *Causality:* The high-dimensionality of the data makes it hard to uncover features that are not just correlated with the prediction but are causally linked to it [Goy+19b]. Prior work has found that many interpretability techniques are independent of both the model and data [STY17; Ade+18; Ade+20], and have little impact on human understanding or trust [Cha+17; Alq+20; CRA20].

The work in this thesis makes apparent another, more fundamental challenge with understanding models today. Namely, that their reliance on unintelligible features precludes them from being fully interpretable. This can be seen in practice—raw interpretations of existing models can be hard to parse (Figure 5-3). Furthermore, while the visual appearance of these interpretations can be improved via post-processing [SVZ13; Yos+15; OMS17], our work suggests that this might just be a way to suppress the "non-robust" features that the model depends on.

Overall, these challenges prompt us to reconsider whether the current desideratum of interpretability—completely characterizing the model's decision process in every setting—may simply be too broad and ill-posed. This, in turn, could not only hinder the development of interpretability techniques but also to make it hard to quantify how good an interpretation is—both in terms of constructing metrics and conducting human-in-the-loop assessments. So, where do we go from here?

A toolkit for model debugging. To make progress in this context, and circumvent some of these challenges, we take an alternative approach:

1. We focus on a more quantifiable (and actionable) sub-problem, namely *model debugging*. Specifically, rather than trying to completely characterize how the model makes every decision, our goal is to (semi-)automatically pinpoint some of the problematic features it depends on.

2. We change the model architecture and training process itself, to mitigate the aforementioned issues of scale and reliance on unintuitive features.

3. With specific downstream tasks in mind (e.g., identifying spurious correlations), we design ablation studies and human-in-the-loop experiments to verify that the identified features causally relate to the model's predictions.

Guided by these design principles, we develop two complementary approaches for model debugging:

- **Post-hoc model debugging via counterfactuals.** Our first approach (Chapter 3) is rooted in the primitive of *counterfactuals*: a classic tool to understand the causal structure of a model [Pea10; Goy+19b; Goy+19a; Bau+20b]. In particular, we put forth a procedure to obtain realistic image counterfactuals, i.e., semantically-meaningful image variants that cause the model to change its prediction. Crucially, we are able to do so *without* detailed annotations or human inspection, by leveraging state-of-the-art instance segmentation and style transfer networks. This, in turn, enables us to precisely quantify how particular high-level data features influence a pre-trained model's performance. For instance, we can diagnose (and measure) how sensitive the prediction of a certain class is to background features (e.g., "grass" or "snow") or frequently co-occurring image objects (e.g., "person" for the class "tench").

- **Deep networks that are debuggable by-design.** Our discussion on adversarial examples above suggests that even the most sophisticated post-hoc debugging techniques will invariably only highlight a subset of feature dependencies learned by the model. (After all, these models do rely on features that are not intelligible to humans.) This motivates us to go beyond purely post-hoc analysis, and modify deep networks to be inherently more debuggable. We find that even fairly simple interventions (Chapters 3 and 5) end up being surprisingly effective for building deep networks that are more amenable to debugging—one can easily detect biases in these models, and identify input features that are responsible for misclassifications.

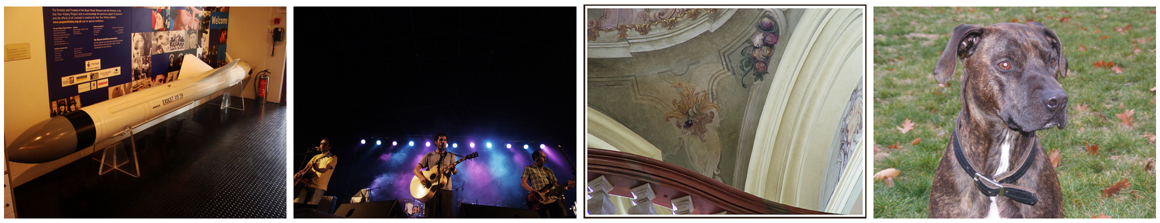
1.2.2 Why do models learn these features?

Next, we might want to understand why models rely on the features they do. We discussed in Section 1.1 that the dominant learning paradigm significantly influences these feature dependencies. In particular, training models to solely maximize accuracy incentivizes them to pick up on all sorts of predictive (in term of test accuracy) correlations in the data—even ones that are meaningless or spurious to humans. For instance, a model might rely on "grass" to detect a "cow"—which while typical, need not be present in every "cow" photo taken in the wild.

Note, however, that it is not only the training algorithm that is at fault here. The core issue lies in our datasets—after all, they contain undesirable input-label associations in the first place. For example, standard dataset contain unintelligible non-robust features, that are highly predictive of the image label (even on the test set)—cf. Section 1.1. So, where do such dataset artifacts stem from?

One natural explanation is that datasets just mirror biases that are already present in the real world—e.g., facial recognition benchmarks under-represent minorities [BG18], and language corpora contain various historical biases [CBN17], both neutral (e.g., "flowers" are more often deemed pleasant than "insects") and problematic (e.g., European American names being deemed pleasant more frequently than African American ones). This explanation has, in fact, guided the progression of datasets in ML, which Torralba and Efros [TE11] aptly phrase as a "revolution" in which "...every new dataset was, in a way, a reaction against the biases and inadequacies of the previous datasets in explaining the (visual) world."

Scalable data collection vs dataset quality. We now discuss how another factor, outside of real-world biases, influences datasets. Specifically, we focus our attention on the data collection pipeline. Our investigation is motivated by the fact that the scale of modern datasets necessitates data collection practices that are very different from what is ideally envisioned—e.g., automated data retrieval and crowdsourced annotation. As a result, the dataset and its corresponding annotations can sometimes be ambiguous, incorrect, or otherwise misaligned with ground truth.



(a) missile (b) stage (c) monastery (d) Staffordshire bull terrier

Figure 1-2: Judging the correctness of ImageNet labels may not be straightforward. While the labels shown above appear valid for the corresponding images, *none* of them actually match the ImageNet labels (which are "projectile", "acoustic guitar", "church", and "American Staffordshire terrier", respectively).

While imperfect and noisy data collection pipelines are used extensively, the extent of the resulting ground truth-label misalignment and how it affects the feature learned by models still remains poorly understood.

In this thesis, we take a step towards bridging this gap in understanding. To this end, we develop a toolkit for collecting fine-grained data annotation via large-scale human studies. These annotations then enable us to precisely quantify the extent to which labels of existing datasets fall short of capturing the underlying ground truth. We demonstrate the utility of our toolkit via a case study on ImageNet [Den+09; Rus+15]—arguably one of most-widely used datasets and a key driver of progress in computer vision.

Indeed, we will see in Chapter 4 that seemingly insignificant design choices made during data collection have a large impact on dataset quality. For instance, the ImageNet labeling task was phrased as a leading question—wherein annotators were not asked to label the image, but only if a single pre-selected label could be valid. Consequently, we see a number of *systemic* annotation errors in ImageNet (cf. Figure 1-2)—e.g., the label often does not correspond to what humans consider the main (ImageNet) object in the image, and there exist pair of classes with overlapping image distributions.

Crucially, we find that these annotation errors greatly impact what ImageNet-trained models learn (and don't learn), and how we perceive model progress. For instance, top-1 accuracy often underestimates model performance by unduly penalizing them for predicting a different, but also valid, label. Further, these models derive part of their accuracy from exploiting ImageNet-specific features that humans are oblivious to, and hence may not generalize well to the real world.

These findings also offer some broader lessons as to the design of the ML pipeline, and how it could cause a mismatch between datasets and the underlying task they are meant to simulate:

- It suggests that the idealized notion of a clear "ground truth" label(s) may be fundamentally at odds with building complex large-scale datasets. Namely, a perfect label(s) may not even exist for every input, and even if it does, a noisy or scalable data collection process may fail to elicit it.
- The canonical approach of treating datasets as the "gold standard"—i.e., training and evaluating models to perfectly match datasets—can cause the model to learn undesirable feature dependencies. Note that this approach not only overlooks biases that our datasets inherit from the real world, but also imperfections that stem from the data collection process. As our study shows, models optimized for performance on these datasets end up relying on dataset-specific spurious features and artifacts.

1.2.3 How can we control the features our models learn?

Our exploration so far underlines the importance of features in determining whether a model will generalize (to the real world). This finding lays bare the need for tools that can *control* the features a model relies on. For instance, mechanisms through which one can ensure that a model does not rely on specific families of features (such as backgrounds or sensitive attributes). The traditional way to specify what models learn is via data collection. That is, gather (more) data that reflects the desired model behavior, and use it to (re-)train the model. However, data collection can be expensive, time-consuming and even infeasible in certain contexts—e.g., think of collecting x-rays from every single hospital. Moreover, as we discussed in

Section 1.2.2, even carefully curated datasets may deviate from the tasks that we hope to model. Can we do something more?

In this thesis, we propose two complementary approaches to *directly* control the features utilized by a model:

Robustness as a feature prior. Recall that one of the major appeals of deep learning is its ability to automatically extract meaningful patterns from the data. Can we somehow direct a model to focus on (or ignore) certain features, without having to manually modify the data? Indeed, we find that the robustness framework, although originally designed for making models secure against adversarial examples [GSS15; Mad+18; CRK19], can provide a means to impose a prior on the features a model learns. As our work shows (Chapter 5), this framework can be leveraged to build models with more general and perceptually-aligned feature representations [Eng+19a]. As a consequence, such robust models generalize better: to vision tasks beyond classification [San+19] and across different domains [Sal+20]. Observe that given the findings of this thesis, these improvements should not be entirely surprising—after all, in order to be insensitive to adversarial examples, robust models must ignore predictive, yet brittle features in the input data.

Direct model editing. In several cases, modifying the training data or objective might not be feasible—e.g., in applications that use pre-trained models trained on large, possibly private datasets [Dev+18; Bro+20]. Additionally, the undesirable feature dependencies learned by models may not even be evident until test time. Retraining the model whenever this occurs may not be an option. In this thesis (Chapter 6), we take a step towards filling this gap in the ML pipeline. To this end, we build on the work of [Bau+20a] to develop a set of tools to *directly edit* the prediction rules—i.e., mappings from input features to labels—learned by classifiers. Our approach can be applied to any pre-trained classifier in a post-hoc manner, *without* additional data collection. Using as few as a *single* exemplar—for instance, a "car" on a "snowy road"—we can improve model generalization *across categories* (e.g., to other vehicles such as "trucks" and "mopeds" on snow). We demonstrate several practical applications of our approach: including improving model generalization to new data subpopulations and eliminating spurious correlations.

1.3 Looking forward: Towards a holistic view on the ML pipeline

A key contribution of this thesis is to shed light on and characterize the disconnect between the conceptual outlook and benchmarks that guide the development of ML models today, and the diverse real-world conditions they will need to eventually tackle. One manifestation of this disconnect is that the impressive performance of existing models quickly starts to degrade under mild deviations from their development conditions (e.g., adversarial examples). However, our investigation uncovers another, perhaps more concerning and fundamental, problem. Namely, even in settings where these models perform well (according to conventional metrics), the mechanisms driving this performance—i.e., the data features models make decisions based on—are very different from what we would expect.

That said, the picture is not all bleak: it is undeniable that current ML models, particularly deep networks, have remarkable potential. Indeed, these models can automatically discover complex predictive patterns in large datasets—the flip side of which is that they, at times, end up extracting unintended ones. This viewpoint suggests that we have some of the right primitives already; the challenge is coupling them together to ensure that models learn the "right" things.

Crucially, as our work shows, tackling this challenge requires us to move beyond a compartmentalized view of ML systems. After all, our investigation constantly reinforces how intertwined the different parts of the ML pipeline and their downstream effects are. Thus, building models that can reliably generalize in practice requires thinking about the entire pipeline—also factoring in the human-ML interactions therein. That is, expanding our focus to consider the societal ramifications of ML models' decisions, including possible feedback loops, and how well these models can collaborate and integrate with humans.

1.4 Thesis Organization

We now outline the organization of this thesis.

Chapter 2 revisits adversarial examples, and discusses our findings as to their origins. The material in this chapter draws upon joint work with Logan Engstrom, Andrew Ilyas, Aleksander Mądry, Brandon Tran and Dimitris Tsipras [Ily+19].

Chapter 3 introduces our toolkit for debugging feature dependencies learned by models: either post-hoc, or by modifying them to be more debuggable. This material is based on joint work with David Bau, Mahalaxmi Elango, Eric Wong, Aleksander Mądry, Antonio Torralba and Dimitris Tsipras [WSM21; San+21].

Chapter 4 examines the effect of scalable data collection procedures on the quality of a dataset and its alignment with the motivating real-world task. The material presented in this chapter is based on joint work with Logan Engstrom, Andrew Ilyas, Aleksander Mądry, and Dimitris Tsipras [Tsi+20].

Chapter 5 demonstrates how the robustness framework can be a viable approach for constraining the features a model can rely on. Our results confirm that models trained using this methodology seem more aligned with perception, and are better-suited for certain downstream tasks. This material is based on joint work with Logan Engstrom, Andrew Ilyas, Aleksander Mądry, Brandon Tran, Dimitris Tsipras, and Alexander Turner [Tsi+19; Eng+19a; San+19].

Chapter 6 introduces a toolkit for directly rewriting prediction-rules learned by models. We demonstrate how this approach can be used to adapt pre-trained models to novel subpopulations and modify them to ignore spurious correlations. The material presented in this chapter is drawn from joint work with David Bau, Mahalaxmi Elango, Aleksander Mądry, Antonio Torralba and Dimitris Tsipras [San+21].

Part I

A Dent in the Picture

Chapter 2

Through the Lens of Adversarial Examples

In this chapter, we investigate one of the most widely-studied failures of models to generalize beyond i.i.d. settings—*adversarial examples* [Big+13; Sze+14; NYC15]. Adversarial examples gained prominence in the security context, as they were found to induce model failures on benchmarks [Sze+14; GSS15; Eng+19c; FFF18b] and in real-world settings [KGB16; Sha+16; Ath+18; Eyk+18]. One of the key contributions to this thesis is to show that adversarial examples are *more* than just a security concern. In particular, we link the adversarial vulnerability of current models to fundamental flaws with their decision making process in standard, non-adversarial contexts. Moreover, we show that these flaws are a direct consequences of the dominant learning paradigm. This chapter is structured as follows: Section 2.1 outlines our contributions, Section 2.2 covers background, and Sections 2.3-2.5 detail our analysis and Section 2.6 discusses its broader implications.

2.1 Summary of our Results

We put forth a new perspective on adversarial examples—imperceptibly perturbed natural inputs that induce erroneous predictions in state-of-the-art classifiers. In particular, the predominant view on adversarial examples previously was that they stem from aberrations in the model (discussed further in Sections 2.2). From this point of view, it is natural to treat adversarial robustness as a goal that can be disentangled and pursued independently from maximizing accuracy [Mad+18; SHS19; Sug+19]. In contrast, we cast this vulnerability as a fundamental consequence of the dominant supervised learning paradigm. Specifically, we claim that:

Adversarial vulnerability is a direct result of our models' sensitivity to well-generalizing features in the data.

Recall that classifiers are usually trained to maximize (distributional) accuracy. Consequently, they tend to use *any* available features in the data to do so, even those that are incomprehensible to humans. We posit that our models rely on such "non-robust" features, and adversarial perturbations exploit this dependence.

We corroborate our theory by showing that it is indeed possible to disentangle robust from non-robust features in standard image classification datasets (cf. Section 2.4). Notably, we show that by adding adversarial perturbations to standard data points, we can construct a dataset wherein the only predictive features are non-robust features. Even though this dataset appears completely mislabeled to humans (cf. Figure 2-1a), standard classifiers are still able to learn from them and perform well on the *original* test set. This demonstrates that adversarial perturbations can arise from flipping features in the data that are useful for classification of correct inputs (hence not being purely aberrations). Additionally, we present a simple setting where the connection between adversarial examples and non-robust features can be studied rigorously (cf. Section 2.5). We discuss broader implications of these findings on model generalization and interpretability in Section 2.6.

2.2 Background: Adversarial Examples

Adversarial examples, originally discovered by [Sze+14] correspond to imperceptible perturbations of natural data points which can "fool" state-of-the-art models into making erroneous predictions. Typically, these perturbations are specially crafted to either maximize the classifier's loss ("untargeted attacks") or to fool it into predicting the desired target class ("targeted attacks"). Formally, an untargeted adversarial perturbation—illustrated in Figure 1-1—can be constructed by solving the following constrained optimization problem:

$$\arg\max_{\delta\in\Delta} \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(x+\delta, y; \theta)] \tag{2.1}$$

where $\mathcal{L}$ is the loss function (e.g., cross entropy), θ denotes model parameters, and (x, y) denote an input-label pair drawn from the distribution $\mathcal{D}$. A key parameter here is the set Δ, which constrains the perturbations the adversary is allowed to apply. In practice, Δ is usually chosen such that the semantic meaning of the input is preserved—e.g., small ℓ_p perturbations [Sze+14; GSS15]; rotations and translations [Eng+19c; Xia+18] or variations in Wasserstein distance [WSK19]. Unless otherwise specified, in this thesis we focus on ℓ_∞ and ℓ_2 perturbations.

This phenomenon has been extensively studied in prior work, giving rise to a range of explanations as to its origins. Broadly, previous work in the field tends to view adversarial examples as aberrations in the model—that are orthogonal to its behavior in standard test conditions. Many of these explanations, however, are often unable to fully capture behaviors we observe in practice, as discussed below.

Concentration of measure in high-dimensions. One line of work argues that the high dimensionality of the input space can present fundamental barriers on classifier robustness [Gil+18; FFF18a; MDM18; Sha+19a]. For certain data distributions, one can show that any decision boundary will be close to a large fraction of inputs and hence no classifier can be robust against small perturbations. However, this model cannot fully explain the fact that in practice, one can train (reasonably) robust classifiers on standard datasets [Mad+18; RSL18; WK18; Xia+19; CRK19].

Boundary Tilting. Tanay and Griffin [TG16] introduce the "boundary tilting" model, and suggest that adversarial examples are a product of over-fitting. Consequently, the authors suggest that mitigating adversarial examples may be a matter of regularization and preventing finite-sample overfitting.

Local Linearity. Goodfellow, Shlens, and Szegedy [GSS15] suggest that the local linearity of DNNs is largely responsible for the existence of small adversarial perturbations. While this conjecture is supported by the effectiveness of adversarial attacks exploiting local linearity (e.g., FGSM [GSS15]), it is not sufficient to fully characterize the phenomena observed in practice. In particular, there exist adversarial examples that violate the local linearity of the classifier [Mad+18], while classifiers that are less linear do not exhibit greater robustness [ACW18].

Piecewise-linear decision boundaries. Shamir et al. [Sha+19b] prove that the geometric structure of the classifier's decision boundaries can lead to sparse adversarial perturbations. However, this result does not take into account the distance to the decision boundary along these direction or feasibility constraints on the input domain. As a result, it cannot meaningfully distinguish between classifiers that are brittle to small adversarial perturbations and classifiers that are moderately robust.

As we discuss in the rest of this chapter, the key differentiating aspect of our model for adversarial perturbations is that they arise as *well-generalizing, yet brittle, features*, rather than statistical anomalies or effects of poor statistical concentration. In particular, we show that adversarial vulnerability does not stem from using a specific model class or a specific training method, since standard training on the "robustified" data distribution of Section 2.4.1 leads to robust models. At the same time, as shown in Section 2.4.2, these non-robust features are sufficient to learn a good standard classifier. We defer the discussion of other theories on adversarial examples that are complementary to our work to Appendix A.4.

2.3 The Robust Features Model

We begin by developing a framework, inspired by the setting in Tsipras et al. [Tsi+19], that enables us to rigorously refer to "robust" and "non-robust" features.

Setup. We consider binary classification[1], where input-label pairs $(x, y) \in \mathcal{X} \times \{\pm 1\}$ are sampled from a distribution $\mathcal{D}$; the goal is to learn a classifier $C : \mathcal{X} \to \{\pm 1\}$ which predicts a label y for a given input x. We define a *feature* to be a function mapping from input space $\mathcal{X}$ to real numbers, with the set of all features being $\mathcal{F} = \{f : \mathcal{X} \to \mathbb{R}\}$. For convenience, we assume that the features in $\mathcal{F}$ are shifted/scaled to be mean-zero and unit-variance (i.e., so that $\mathbb{E}_{(x,y)\sim\mathcal{D}}[f(x)] = 0$ and $\mathbb{E}_{(x,y)\sim\mathcal{D}}[f(x)^2] = 1$), in order to make the following definitions scale-invariant. Note that this formal definition also captures what we abstractly think of as features (e.g., we can construct an f that captures how "furry" an image is).

Useful, robust, and non-robust features. We now define the key concepts required for formulating our framework. To this end, we categorize features as:

- **ρ-useful features**: For a given distribution $\mathcal{D}$, we call a feature f *ρ-useful* ($\rho > 0$) if it is correlated with the true label in expectation, that is if

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}[y \cdot f(x)] \geq \rho. \tag{2.2}$$

  We then define $\rho_{\mathcal{D}}(f)$ as the largest ρ for which feature f is ρ-useful under distribution $\mathcal{D}$. (Note that if a feature f is negatively correlated with the label, then $-f$ is useful instead.) Crucially, a linear classifier trained on ρ-useful features can attain non-trivial generalization performance.

- **γ-robustly useful features**: Suppose we have a ρ-useful feature f ($\rho_{\mathcal{D}}(f) > 0$). We refer to f as a *robust feature* (formally a γ-robustly useful feature for $\gamma > 0$) if, under adversarial perturbation (for some specified set of valid perturbations Δ), f remains γ-useful. Formally, if we have that

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\inf_{\delta\in\Delta(x)} y \cdot f(x+\delta)\right] \geq \gamma. \tag{2.3}$$

[1]Our framework can be straightforwardly adapted though to the multi-class setting.

- **Useful, non-robust features**: This refers to a feature which is ρ-useful for some ρ bounded away from zero, but is not γ-robust for any $\gamma \geq 0$. Such features help with classification in the standard setting, but may hinder accuracy in the adversarial setting, as the correlation with the label can be flipped.

Classification. A classifier $C = (F, w, b)$ is comprised of a set of features $F \subseteq \mathcal{F}$, a weight vector w, and a scalar bias b. For an input x, the classifier predicts

$$C(x) = \operatorname{sgn}\left(b + \sum_{f \in F} w_f \cdot f(x)\right).$$

For convenience, we denote the set of features learned by a classifier C as F_C.

Standard Training. Classifier training is typically performed by minimizing a loss function (via *empirical risk minimization* (ERM)) such as [2]

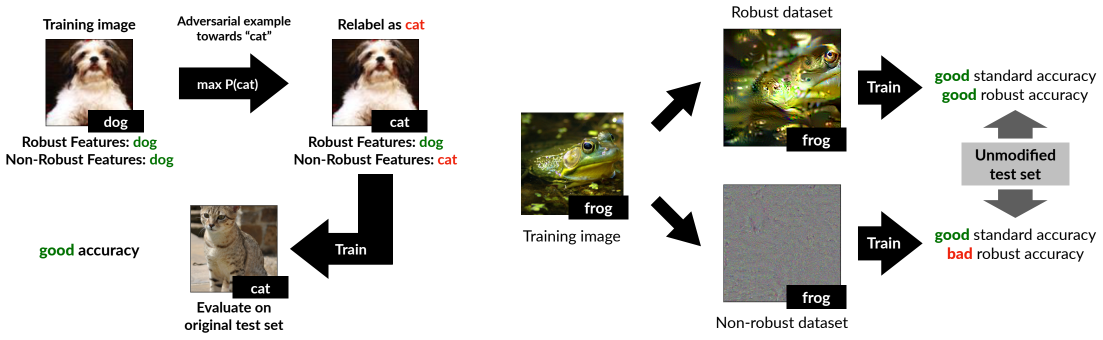
$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathcal{L}_\theta(x,y)\right] = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[y \cdot \left(b + \sum_{f \in F} w_f \cdot f(x)\right)\right]. \tag{2.4}$$

When minimizing classification loss, *no distinction* exists between robust and non-robust features: the only distinguishing factor of a feature is its ρ-usefulness. Namely, the classifier will utilize *any* ρ-useful feature in F to decrease the classifier's loss.

Robust training. In the presence of an *adversary*, any useful but non-robust features can be made *anti-correlated* with the true label, leading to adversarial vulnerability. Therefore, ERM is no longer sufficient to train classifiers that are robust, as we need to explicitly account for the effect of the adversary. To do so, we use an *adversarial* loss function [Mad+18]:

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta \in \Delta(x)} \mathcal{L}_\theta(x+\delta, y)\right], \tag{2.5}$$

[2]Just as for the other parts of this model, we use this loss for simplicity only—it is straightforward to generalize to more practical loss function such as logistic or hinge loss.



(a) (b)

Figure 2-1: A conceptual diagram of the experiments of Section 2.4. In (a) we construct a dataset which appears mislabeled to humans (via adversarial examples) but results in good accuracy on the original test set (Section 2.4.2). In (b) we disentangle features into combinations of robust/non-robust features (Section 2.4.1).

for an appropriately defined set of perturbations Δ. Since the adversary can exploit non-robust features to reduce classification accuracy, minimizing this loss (as in adversarial training [GSS15; Mad+18]) can be viewed as explicitly preventing the classifier from learning a useful but non-robust combination of features.

Remark. Though the framework above enables us to formally describe and predict the outcome of our experiments, it does not completely capture the notion of non-robust features exactly as we might think of them. For instance, it would allow for useful non-robust features to arise as combinations of useful robust features and useless non-robust features [Goh19b], which are however precluded by our experiments in Section 2.4. This shows that our experimental findings capture a stronger, more fine-grained statement than our formal definitions are able to express. We view bridging this gap as an interesting direction for future work.

2.4 Finding Robust (and Non-Robust) Features

Our chief hypothesis is that there exist both robust and non-robust features that constitute useful signals for standard classification. We now provide evidence in support of this by disentangling these two sets of features. A conceptual description of these experiments can be found in Figure 2-1.

2.4.1 A "robustified" version for robust classification

Recall that under our conceptual framework, a classifier learns to rely on features based purely on how useful they are for (standard) generalization. Thus, if we can ensure that the only useful features are robust, standard training should result in a robust classifier. Unfortunately, we cannot directly manipulate the features of very complex, high-dimensional datasets. Instead, we will leverage a robust model and modify our dataset to contain only the features that are relevant to that model.

Dataset construction. At a high level, given a *robust model* (i.e., adversarially trained [Mad+18]) C we aim to construct a distribution $\widehat{\mathcal{D}}_R$ which satisfies:

$$\mathbb{E}_{(x,y)\sim\widehat{\mathcal{D}}_R}\left[f(x)\cdot y\right] = \begin{cases} \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[f(x)\cdot y\right] & \text{if } f \in F_C \\ 0 & \text{otherwise,} \end{cases} \tag{2.6}$$

where F_C again represents the set of features utilized by C. Conceptually, we want features used by C to be as useful as they were on the original distribution $\mathcal{D}$ while ensuring that the rest of the features are not useful under $\widehat{\mathcal{D}}_{NR}$. We will construct a training set for $\widehat{\mathcal{D}}_R$ via a one-to-one mapping $x \mapsto x_r$ from the original training set for $\mathcal{D}$. In the case of a deep neural network, F_C corresponds to exactly the set of activations in the penultimate layer (since these correspond to inputs to a linear classifier). To ensure that features used by the model are equally useful under both training sets, we (approximately) enforce all features in F_C to have similar values for both x and x_r through the following optimization:

$$\min_{x_r} \|g(x_r) - g(x)\|_2, \tag{2.7}$$

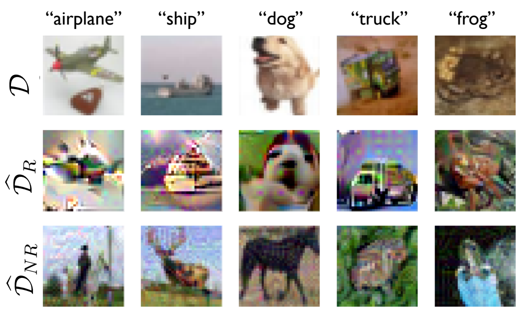
where x is the original input and g is the mapping from x to the representation layer. We optimize this objective using gradient descent in input space. Since we don't have access to features outside F_C, there is no way to ensure that the expectation in (2.6) is zero for all $f \notin F_C$. To approximate this condition, we choose the starting point of gradient descent for the optimization in (2.7) to be an input x_0 which is drawn from $\mathcal{D}$ independently of the label of x. This choice ensures that any feature present in that input will not be useful since they are not correlated with the label in expectation over x_0. The underlying assumption here is that, when performing the optimization in (2.7), features that are not being directly optimized (i.e., features outside F_C) are not affected..

Given the new training set for $\widehat{\mathcal{D}}_R$ (random samples are visualized in Figure 2-2a), we train a classifier using standard (non-robust) training. We then test this classifier on the original test set (i.e. $\mathcal{D}$). The results (Figure 2-2b) indicate that this classifier attains good accuracy in *both standard and adversarial settings*. As a control, we repeat this methodology using a standard (non-robust) model for C in our construction of the dataset. Sample images from the resulting "non-robust dataset" $\widehat{\mathcal{D}}_{NR}$ are shown in Figure 2-2a—they tend to resemble more the source image of the optimization x_0 than the target image x. We find that training on this dataset leads to good standard accuracy, yet yields almost no robustness (Figure 2-2b). We also verify that this procedure is not simply a matter of encoding the weights of the original model—we get the same results for both $\widehat{\mathcal{D}}_R$ and $\widehat{\mathcal{D}}_{NR}$ if we train with different architectures than that of the original models.

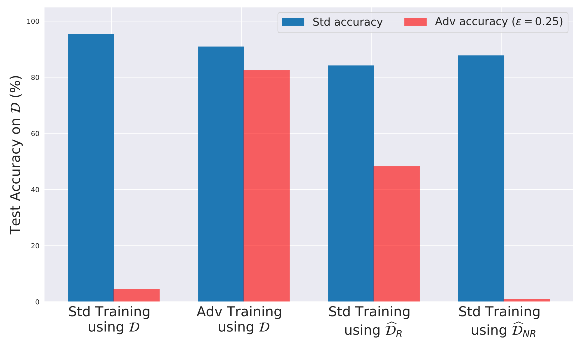
Implications. Overall, our findings corroborate the hypothesis that adversarial examples can arise from (non-robust) features of the data. By filtering these features out of the data (e.g. restricting the set of available features to those used by a robust model), one can train a significantly more robust model using *standard training*. Moreover, this provides evidence that adversarial vulnerability is caused by non-robust features and is not inherently tied to the standard training framework.

2.4.2 Non-robust features suffice for standard classification

The results of the previous section show that by restricting the dataset to only contain features that are used by a robust model, standard training results in clas-



(a)



(b)

Figure 2-2: **Left**: Random samples from our variants of CIFAR-10 [Kri09]: the original training set; the *robust training set* $\widehat{\mathcal{D}}_R$, restricted to features used by a robust model; and the *non-robust training set* $\widehat{\mathcal{D}}_{NR}$, restricted to features used by a standard model (labels appear incorrect to humans). **Right**: Standard and robust accuracy on the CIFAR-10 test set ($\mathcal{D}$) for models trained with: (i) standard training (on $\mathcal{D}$) ; (ii) standard training on $\widehat{\mathcal{D}}_{NR}$; (iii) adversarial training (on $\mathcal{D}$); and (iv) standard training on $\widehat{\mathcal{D}}_R$. Models trained on $\widehat{\mathcal{D}}_R$ and $\widehat{\mathcal{D}}_{NR}$ reflect the original models used to create them: notably, standard training on $\widehat{\mathcal{D}}_R$ yields non-trivial robust accuracy.

sifiers that are significantly more robust. This suggests that when training on the standard dataset, non-robust features take on a large role in the resulting learned classifier. Here we set out to show that this role is not merely incidental or due to finite-sample overfitting. In particular, we demonstrate that non-robust features *alone* suffice for standard generalization— i.e., a model trained solely on non-robust features can perform well on the *standard* test set.

Dataset construction. To this end, we construct a dataset where the only features that are useful for classification are *non-robust* features. To accomplish this, we modify each input-label pair (x, y) as follows. We select a target class t either (a) uniformly at random among classes (hence features become uncorrelated with the labels) or (b) deterministically according to the source class (e.g. using a fixed permutation of labels). Then, we add a small adversarial perturbation to x in order to ensure it is classified as t by a standard model. Formally:

$$x_{adv} = \arg\min_{\|x'-x\|\leq\varepsilon} L_C(x', t), \tag{2.8}$$

where L_C is the loss under a standard (non-robust) classifier C and ε is a small constant. The resulting inputs are nearly indistinguishable from the originals (Appendix Figure A-4)—to a human observer, it thus appears that the label t assigned to the modified input is simply incorrect. The resulting input-label pairs (x_{adv}, t) make up the new training set. Now, since $\|x_{adv} - x\|$ is small, by definition the robust features of x_{adv} are still correlated with class y (and not t) in expectation over the dataset. After all, humans still recognize the original class. On the other hand, since every x_{adv} is strongly classified as t by a standard classifier, it must be that some of the non-robust features are now strongly correlated with t (in expectation).

In the case where t is chosen at random, the robust features are originally uncorrelated with the label t (in expectation), and after the adversarial perturbation can be only slightly correlated (hence being significantly less useful for classification than before Goh [Goh19a]). Formally, we aim to construct a dataset $\widehat{\mathcal{D}}_{rand}$ where:

$$\mathbb{E}_{(x,y)\sim\widehat{\mathcal{D}}_{rand}}\left[y \cdot f(x)\right] \begin{cases} > 0 & \text{if } f \text{ non-robustly useful under } \mathcal{D}, \\ \simeq 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

In contrast, when t is chosen deterministically based on y, the robust features actually point *away* from the assigned label t. In particular, all of the inputs labeled with class t exhibit *non-robust features* correlated with t, but robust features correlated with the original class y. Thus, robust features on the original training set provide significant predictive power on the training set, but will actually hurt generalization on the standard test set. Viewing this case again using the formal model, our goal is to construct $\widehat{\mathcal{D}}_{det}$ such that

$$\mathbb{E}_{(x,y)\sim\widehat{\mathcal{D}}_{det}}\left[y \cdot f(x)\right] \begin{cases} > 0 & \text{if } f \text{ non-robustly useful under } \mathcal{D}, \\ < 0 & \text{if } f \text{ robustly useful under } \mathcal{D} \\ \in \mathbb{R} & \text{otherwise } (f \text{ not useful under } \mathcal{D}) \end{cases} \tag{2.10}$$

For both these datasets, we find that standard training on them actually gener-

Source Dataset	Dataset	
	CIFAR-10	ImageNet$_R$
$\mathcal{D}$	95.3%	96.6%
$\widehat{\mathcal{D}}_{rand}$	63.3%	87.9%
$\widehat{\mathcal{D}}_{det}$	43.7%	64.4%

Table 2.1: Test accuracy (on $\mathcal{D}$) of classifiers trained on the $\mathcal{D}$, $\widehat{\mathcal{D}}_{rand}$, and $\widehat{\mathcal{D}}_{det}$ training sets created using a standard model. For both $\widehat{\mathcal{D}}_{rand}$ and $\widehat{\mathcal{D}}_{det}$, only non-robust features correspond to useful features on both the train set and $\mathcal{D}$.

alizes to the *original* test set—cf. Table 2.1). Remarkably, even training on $\widehat{\mathcal{D}}_{det}$ (where all the robust features are correlated with the wrong class), results in a well-generalizing classifier.

Implications. Our findings indicate that non-robust features are useful for classification in the standard setting, and not merely artifacts of finite-sample overfitting. Further, our experiments show that non-robust featurescan be picked up by models, even in the presence of *predictive robust features* Note that, despite their predictive power on standard datasets, non-robust featuresare brittle, and thus give rise to adversarial vulnerability.[3] .

2.5 A Theoretical Framework for Studying (Non)-Robust Features

The experiments from the previous section demonstrate that the conceptual framework of robust and non-robust features is strongly predictive of the empirical behavior of state-of-the-art models on real-world datasets. In order to further strengthen our understanding of the phenomenon, we instantiate the framework in a concrete setting that allows us to theoretically study various properties of the

[3]It is worth emphasizing that while our findings demonstrate that adversarial vulnerability *does* arise from non-robust features, they do not preclude the possibility of adversarial vulnerability also arising from other phenomena [TG16; Sch+18; Nak19a]. Still, the mere existence of useful non-robust features suffices to establish that without explicitly discouraging models from utilizing these features, adversarial vulnerability will remain an issue.

corresponding model, inspired by the model presented by [Tsi+19].

Setup. We study a simple problem of *maximum likelihood classification* between two Gaussian distributions. Fiven samples (x, y) sampled from $\mathcal{D}$ according to

$$y \overset{\text{u.a.r.}}{\sim} \{-1, +1\}, \qquad x \sim \mathcal{N}(y \cdot \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \tag{2.11}$$

our goal is to learn parameters $\Theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ such that

$$\Theta = \arg\min_{\boldsymbol{\mu}, \boldsymbol{\Sigma}} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\ell(x; y \cdot \boldsymbol{\mu}, \boldsymbol{\Sigma})\right], \tag{2.12}$$

where $\ell(x; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ represents the Gaussian negative log-likelihood (NLL) function. Intuitively, we find the parameters $\boldsymbol{\mu}, \boldsymbol{\Sigma}$ which maximize the likelihood of the sampled data under the given model. Classification under this model can be accomplished via likelihood test: given an unlabeled sample x, we predict y as

$$y = \arg\max_{y} \ell(x; y \cdot \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \text{sign}\left(x^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}\right).$$

In turn, the *robust analogue* of this problem arises from replacing $\ell(x; y \cdot \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with the NLL under adversarial perturbation. The resulting robust parameters Θ_r can be written as

$$\Theta_r = \arg\min_{\boldsymbol{\mu}, \boldsymbol{\Sigma}} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\|\delta\|_2 \leq \varepsilon} \ell(x + \delta; y \cdot \boldsymbol{\mu}, \boldsymbol{\Sigma})\right], \tag{2.13}$$

A detailed analysis of this setting is in Appendix A.3—here we present a high-level overview of the results.

(1) Vulnerability from metric misalignment (non-robust features). Note that in this model, one can rigorously make reference to an *inner product* (and thus a metric) induced by the features. In particular, one can view the learned parameters of a Gaussian $\Theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ as defining an inner product over the input space given by $\langle x, y \rangle_\Theta = (x - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (y - \boldsymbol{\mu})$. This in turn induces the Mahalanobis distance, which represents how a change in the input affects the features learned by the classifier. This metric is not necessarily aligned with the metric in which the adversary is constrained, the ℓ_2-norm. Actually, we show that adversarial vulnerability arises exactly as a *misalignment* of these two metrics.

Theorem 1 (Adversarial vulnerability from misalignment). *Consider an adversary whose perturbation is determined by the "Lagrangian penalty" form of* (2.13), *i.e.*

$$\max_{\delta} \ell(x+\delta; y \cdot \boldsymbol{\mu}, \boldsymbol{\Sigma}) - C \cdot \|\delta\|_2,$$

where $C \geq \frac{1}{\sigma_{min}(\boldsymbol{\Sigma}_*)}$ is a constant trading off NLL minimization and the adversarial constraint[4]. *Then, the adversarial loss $\mathcal{L}_{adv}$ incurred by the non-robustly learned $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by:*

$$\mathcal{L}_{adv}(\Theta) - \mathcal{L}(\Theta) = tr\left[\left(I + (C \cdot \boldsymbol{\Sigma}_* - I)^{-1}\right)^2\right] - d,$$

and, for a fixed $tr(\boldsymbol{\Sigma}_*) = k$ the above is minimized by $\boldsymbol{\Sigma}_* = \frac{k}{d}\boldsymbol{I}$.

In fact, note that such a misalignment corresponds precisely to the existence of *non-robust features*, as it indicates that "small" changes in the adversary's metric along certain directions can cause large changes under the data-dependent notion of distance established by the parameters. This is illustrated in Figure 2-3, where misalignment in the feature-induced metric is responsible for the presence of a non-robust feature in the corresponding classification problem.

(2) Robust Learning. The optimal (non-robust) maximum likelihood estimate is $\Theta = \Theta^*$, and thus the vulnerability for the standard MLE estimate is governed entirely by the true data distribution. The following theorem characterizes the behaviour of the learned parameters in the robust problem. In fact, we can prove that performing (sub)gradient descent on the inner maximization (also known as *adversarial training* [GSS15; Mad+18]) yields exactly Θ_r. We find that as the pertur-

[4]The constraint on C is to ensure the problem is concave.

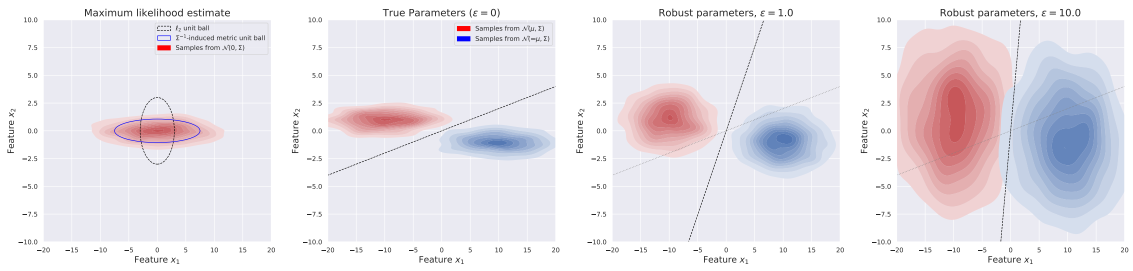


Figure 2-3: An empirical demonstration of the effect illustrated by Theorem 2—as the adversarial perturbation budget ε is increased, the learned mean $\boldsymbol{\mu}$ remains constant, but the learned covariance "blends" with the identity matrix, effectively adding more and more uncertainty onto the non-robust feature.

bation budget ε is increased, the metric induced by the learned features *mixes* ℓ_2 and the metric induced by the features.

Theorem 2 (Robustly Learned Parameters)**.** *Just as in the non-robust case, $\boldsymbol{\mu}_r = \boldsymbol{\mu}^*$, i.e. the true mean is learned. For the robust covariance $\boldsymbol{\Sigma}_r$, there exists an $\varepsilon_0 > 0$, such that for any $\varepsilon \in [0, \varepsilon_0)$,*

$$\boldsymbol{\Sigma}_r = \frac{1}{2}\boldsymbol{\Sigma}_* + \frac{1}{\lambda} \cdot \boldsymbol{I} + \sqrt{\frac{1}{\lambda} \cdot \boldsymbol{\Sigma}_* + \frac{1}{4}\boldsymbol{\Sigma}_*^2},$$
$$\textit{where} \qquad \Omega\left(\frac{1+\varepsilon^{1/2}}{\varepsilon^{1/2}+\varepsilon^{3/2}}\right) \leq \lambda \leq O\left(\frac{1+\varepsilon^{1/2}}{\varepsilon^{1/2}}\right).$$

The effect of robust optimization under an ℓ_2-constrained adversary is visualized in Figure 2-3. As ϵ grows, the learned covariance becomes more aligned with identity. For instance, we can see that the classifier learns to be less sensitive in certain directions, despite their usefulness for natural classification.

(3) Gradient Interpretability. Tsipras et al. [Tsi+19] observe that gradients of robust models tend to look more semantically meaningful. It turns out that under our model, this behaviour arises as a natural consequence of Theorem 2. In particular, we show that the resulting robustly learned parameters cause the gradient of the linear classifier and the vector connecting the means of the two distributions to better align (in a worst-case sense) under the ℓ_2 inner product.

Theorem 3 (Gradient alignment). *Let $f(x)$ and $f_r(x)$ be monotonic classifiers based on the linear separator induced by standard and ℓ_2-robust maximum likelihood classification, respectively. The maximum angle formed between the gradient of the classifier (wrt input) and the vector connecting the classes can be smaller for the robust model:*

$$\min_{\mu} \frac{\langle \boldsymbol{\mu}, \nabla_x f_r(x) \rangle}{\|\boldsymbol{\mu}\| \cdot \|\nabla_x f_r(x)\|} > \min_{\mu} \frac{\langle \boldsymbol{\mu}, \nabla_x f(x) \rangle}{\|\boldsymbol{\mu}\| \cdot \|\nabla_x f(x)\|}.$$

Figure 2-3 illustrates this phenomenon in the two-dimensional case. With ℓ_2-bounded adversarial training the gradient direction (perpendicular to the decision boundary) becomes increasingly aligned under the ℓ_2 inner product with the vector between the means ($\boldsymbol{\mu}$).

2.6 Broader Implications: Robustness, Interpretability and Generalization

We conclude this chapter by discussing some broader implications of our experimental and theoretical analysis.

2.6.1 Adversarial transferability

One of the most intriguing properties of adversarial examples is that they *transfer* across models with different architectures and independently sampled training sets [Sze+14; PMG16; CRP19]. Here, we show that this phenomenon can in fact be viewed as a natural consequence of the existence of non-robust features. Given that such features are inherent to the data distribution, classifiers trained on independent samples from that distribution are likely to utilize similar non-robust features. Consequently, an adversarial example constructed by exploiting the non-robust features learned by one classifier will transfer to any other classifier utilizing these features in a similar manner. To test this, we train models with various architectures on the dataset generated in Section 2.4.2 with respect to a standard ResNet-50 [He+16]. Our hypothesis would suggest that architectures which learn better from this training set (in terms of standard test set accuracy) are more likely to learn similar non-robust features to the original classifier, and hence be amenable to transfer attacks. Indeed, we find this is the case—see Figure 2-4. These findings thus corroborate our hypothesis that adversarial transferability arises when models learn similar brittle features of the underlying dataset.

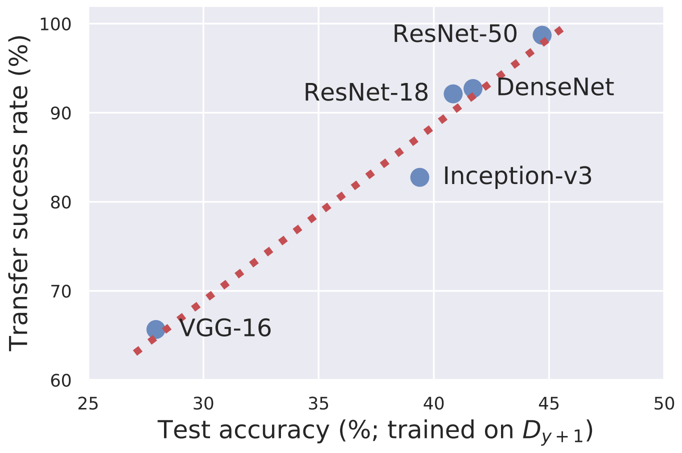


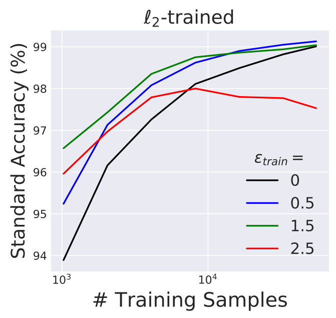
Figure 2-4: Transfer rate of adversarial examples from a ResNet-50 to different architectures alongside test set performance of these architecture when trained on the dataset generated in Section 2.4.2. Architectures more susceptible to transfer attacks also performed better on the standard test set supporting our hypothesis that adversarial transferability arises from utilizing similar *non-robust features*.

2.6.2 The cost of robustness

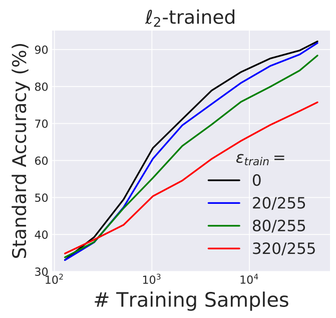
Our new perspective on adversarial examples suggests that building robust models requires preventing said models from utilizing predictive, yet brittle data features. We now discuss the implications of this constraint:

The need for new training objectives. Even in simple settings where a classifier with perfect robustness and accuracy is attainable—e.g., cf. Appendix Figure A-7—it is straightforward to show that any standard loss function will still lead to an accurate yet non-robust classifier. It turns out that this issue arises in practice as well—ERM-trained classifiers are easily fooled by adversarial examples. Consequently, building classifiers that are robust (to adversarial examples) requires fundamentally changing the training methodology—e.g., to robust optimization.

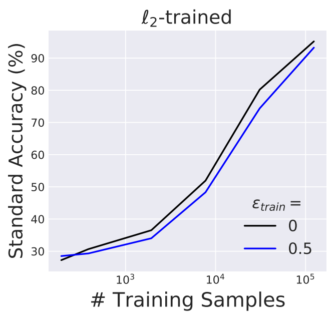
Robustness vs accuracy [Tsi+19]. In practice, it has been observed that robust models tend to have lower *clean* (also known as standard) accuracy than their ERM-trained counterparts—cf. Figure 2-5. Our view on adversarial examples provides a justification for why such a trade-off could arise—after all, training models to be robust prevents them from learning the most accurate classifier.



(a) MNIST



(b) CIFAR-10



(c) Restricted ImageNet

Figure 2-5: Comparison of the standard accuracy of models trained against an ℓ_2-bounded adversary as a function of size of the training dataset. We observe that when training with few samples, adversarial training can have a positive effect on model generalization. However, as training data increase, the standard accuracy of robust models drops below that of the standard model ($\epsilon_{train} = 0$).

The need for more data Schmidt et al. [Sch+18]. In prior work [Sch+18], we proposed a simple setting wherein there is a large sample complexity gap between standard and robust generalization. Namely, in this model, a single sample is sufficient to learn a good, yet non-robust classifier, whereas a good robust classifier requires $O(\sqrt{d})$ samples. This effect can also be viewed as a natural consequence of our conceptual framework. Since training models robustly reduces the effective amount of information in the data (as non-robust features are discarded), more samples may be required to generalize robustly.

2.6.3 Robustness as a feature prior

Our theoretical analysis suggests that rather than offering any quantitative classification benefits, a natural way to view the role of robust optimization is as enforcing a *prior* over the features learned by the classifier. For instance, training with an ℓ_2-bounded adversary prevents the classifier from relying heavily on features which induce a metric dissimilar to the ℓ_2 metric. We explore this connection, as well as its impact on the features learned by models, in Chapter 5.

2.6.4 Interpretability vis-a-vis adversarial examples.

A natural consequence of our findings is that standard models—which rely on human unintelligible features—cannot be fully human interpretable. On one hand, this suggests that approaches aiming to enhance the interpretability of a given model by enforcing "priors" for its explanation [MV15; OMS17; Smi+17] actually hide features that are *"meaningful"* and *predictive* to standard models. On the other hand, it suggests that producing *human*-meaningful explanations that remain faithful to underlying models cannot be pursued independently from the training of the models themselves. In Chapters 3 and 5, we will discuss possible modifications to model training to make models inherently easier to understand.

2.6.5 Human-ML misalignment

Finally, our analysis establishes adversarial vulnerability as a human-centric phenomenon, since, from the standard supervised learning point of view, non-robust features can be as important as robust ones. Viewed differently, adversarial examples stem from a misalignment between the features humans and ML models use to success on a given task.

Part II

A Features Perspective on the ML Pipeline

Chapter 3

A Toolkit for Model Debugging

In this chapter, we develop a set of tools to probe model behavior in a more fine-grained manner: rather than focusing solely on benchmark accuracy, we investigate what data features models use to achieve said accuracy. In part, this exploration is motivated by our findings in Chapter 2—where we saw that the vulnerability of models to adversarial examples stems from their reliance on unintelligible or imperceptible features in the data. However, the failure of current models to generalize in the "right way" goes beyond this single instance. Numerous studies provide evidence that current models rely on other context-dependent or spurious features in the data—ones that are not fundamental to the task they are designed to perform [BVP18; Xia+20; Tsi+20; BVA20].

Precisely characterizing how models make decisions has been a major focus of interpretability research. However, the scale of typical deep networks makes this extremely challenging—limiting one to localized explanations [RSG16a; SVZ13; Yos+15; Bau+17], or requiring specially annotated data [Bau+17; Kim+18; Bau+19a; Bau+20b], or human inspection [Ola+18; WSM21]. Thus, we instead focus on a simpler, more-actionable, problem—*model debugging*. Specifically, rather than aiming for a complete characterization of the model's decision process, our goal is to uncover (unexpected) feature dependencies in models (semi-)automatically. This chapter is structured as follows: Section 3.1 outlines our contributions, Sections 3.2-3.5 detail our approaches for model debugging, Section 3.6 discusses related work and Section 3.7 contextualizes the broader implications of our analysis.

3.1 Summary of Our Results

Our key contribution is a pair of complementary toolkits, that help us better understand the features typical vision and/or language models rely on.

Approach I: Post-hoc debugging (Section 3.2). Here, we attempt to obtain a coarse understanding of a given pre-trained model. To this end, we approximate its behavior with simple, concept-level prediction rules—links between high-level input features (e.g., "road") and the predicted label (e.g., "car"). To do so, we leverage the classic primitive of *counterfactuals*, which we *automatically* synthesize via existing methods for instance segmentation and style transfer. Using these counterfactuals, we can then quantify how individual concepts impact model predictions, without the need for detailed annotations or human inspection.

Approach II: Designing debuggable deep networks (Sections 3.3-3.5). Instead of grappling with understanding models entirely post-hoc, we modify them to be inherently more debuggable. To this end, we train "sparse linear decision layer" on the "deep features" from a given pre-trained deep network. Then, we can probe the behavior of the resulting model by focusing on how a handful of deep features are (linearly) combined by the decision layer. This simple approach ends up being surprisingly effective at building models that are accurate, and at the same time measurably easier for humans to understand and debug.

3.2 Post-hoc debugging via counterfactuals

Our first goal is to develop tools to identify the data features a given pre-trained model relies on to make its predictions. In particular, we would like to be able to query whether (and to what extent) a specific high-level feature or concept in the data affects the model's ability to recognize objects of a certain class—e.g., race or

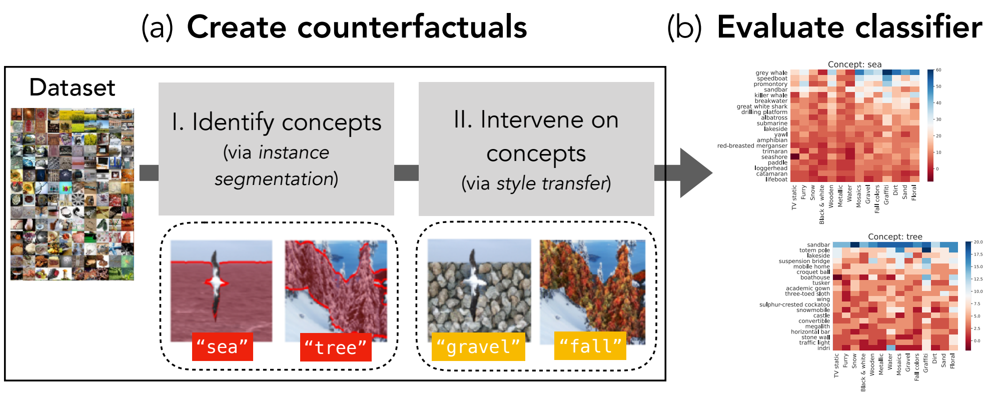


Figure 3-1: Model debugging via counterfactuals. To pinpoint the high-level concepts that have an impact on model predictions, we (a) synthesize counterfactual images in which a given concept in the image (detected via instance segmentation) is modified (via style transfer). Then (b) the impact of this concept-level transformation on classifier predictions is measured. For example, we see that the model relies on "sea" to predict well on "albatros", and on "tree" to detect "sandbar".

gender of a patient while predicting a medical treatment, or "wheel" while detecting "cars". The need for such tools might arise both in the debugging context, and to provide users with recourse [USL19]. We now discuss our approach to automatically discover such prediction-rules—i.e., input feature-class mappings—learned by a given pre-trained classifier.

3.2.1 Leveraging counterfactuals

Our pipeline for discovering such prediction rules revolves around input *counterfactuals*—a primitive commonly used in causal inference [Pea10] and interpretability [Goy+19b; Goy+19a; Bau+20b]. Counterfactuals can be used to identify the data features that the model uses to make its prediction on a given input: by assessing how its prediction changes when the input is modified along a particular axis. In our case, we use counterfactuals to glean how the model's prediction depends on high-level features (used synonymously with concept) in the image. For example, to understand whether the model relies on the "wheel" to recognize a "car" in an image, we will evaluate it on the same image with a different (transformed) "wheel". Our complete rule-discovery pipeline is illustrated in Figure 3-1.

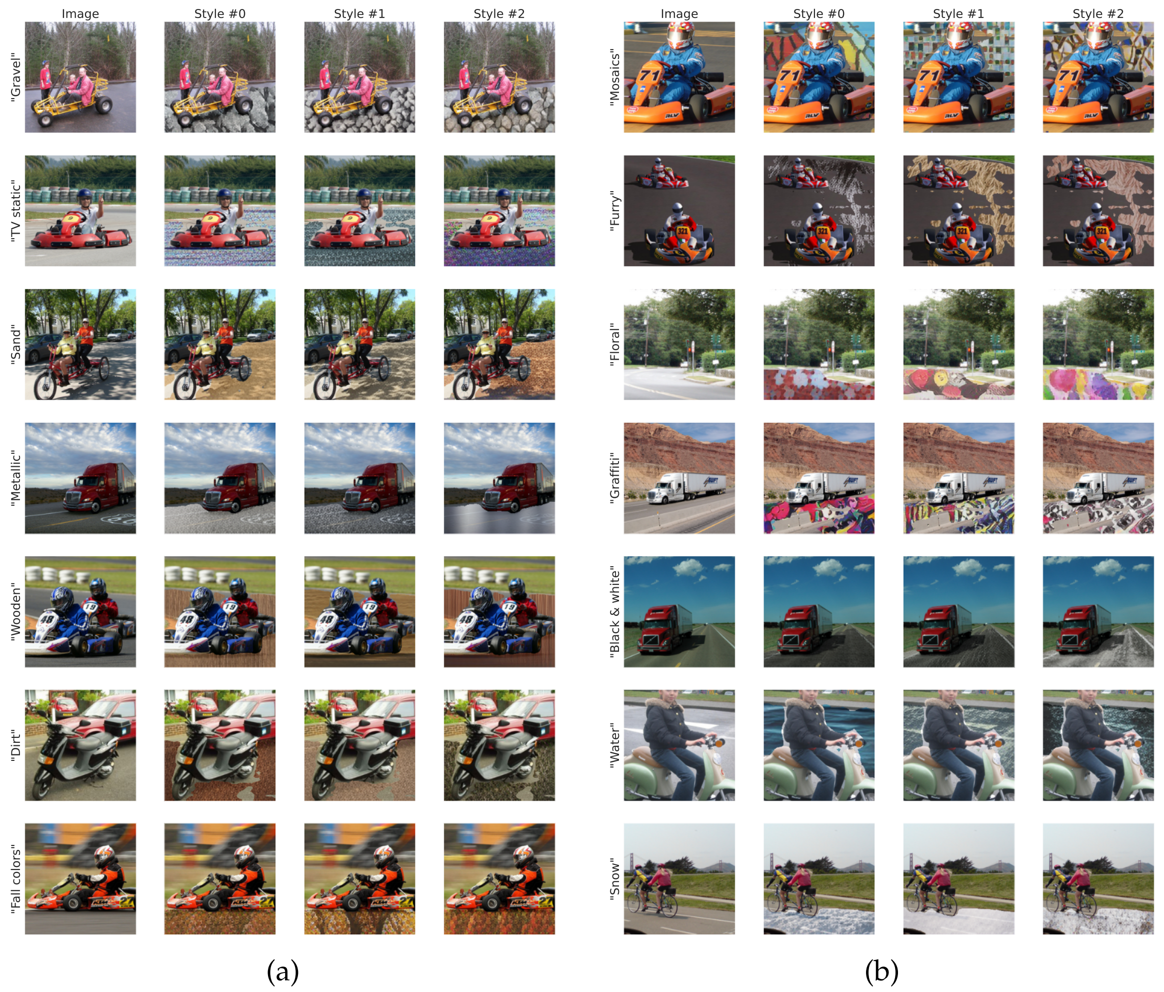


Figure 3-2: Automatically synthesizing concept-level counterfactuals in the ImageNet dataset: We transform the concept "road" in images belonging to various classes via style transfer. Each row (within (a) and (b)) depicts the stylization of a single image with respect to the style described in the label (e.g., "snow").

3.2.2 Synthesizing concept-level counterfactuals

Given a dataset, we generate counterfactuals with respect to a given high-level concept (e.g., "wheel") in two steps:

1. **Concept identification:** First, we find images in which this concept is present, and pinpoint the regions of each image it appears in. In principle, this could be accomplished by manually segmenting images in a fine-grained manner; however this would be quite costly. We instead leverage pre-trained instance segmentation models (e.g., trained on MS-COCO [Lin+14] and LVIS [GDG19])

to automatically obtain approximate concept segmentations.

2. **Concept transformation:** Once we have identified where in a given image the relevant concept is present, we need to transform it. We do so by leveraging existing methods for style transfer [GEB16; Ghi+17] so as to preserve fine-grained image features and realism. For our analysis, we manually collect a set of realistic textures (e.g., “snow” and “graffiti”) which we then use to transform image regions where the concept is present.

These two steps, when combined, allow us to automatically synthesize counterfactuals[1] with realistic *concept-level transformations*, such as “snowy road” or “wooden wheels”—cf. Figure 3-2 for an illustration.

3.2.3 Probing model behavior via counterfactuals

We now evaluate classifiers on counterfactuals created with respect to various concept-style pairs, and measure the change in their performance (relative to unmodified images) on a per-class level. This allows us to pinpoint:

The effect of specific concepts. We can measure and compare the influence of a given high-level concept on model performance for various classes—in terms of the accuracy drop caused by the transformed concept. For instance, in Figure 3-3a, we find that the accuracy of a VGG16 ImageNet classifier drops by 25% on images of “croquet ball” when “grass” is transformed, whereas its accuracy on “collie” does not change. In line with previous studies [Zha+07; RSG16b; RZT18; Bar+19; Xia+20], we also find that background concepts, such as “grass”, “sea” and “sand”, have a large effect model performance. We can contrast this measure of influence across concepts for a single model (Appendix Figure B-1), and across architectures for a single concept (Appendix B.1.4). Finally, we can also examine the effect of the style used to transform a concept—cf. Figure 3-3b.

[1]In other recent work [Lec+21], we propose an alternative approach to construct such counterfactuals with the help of 3D simulators.

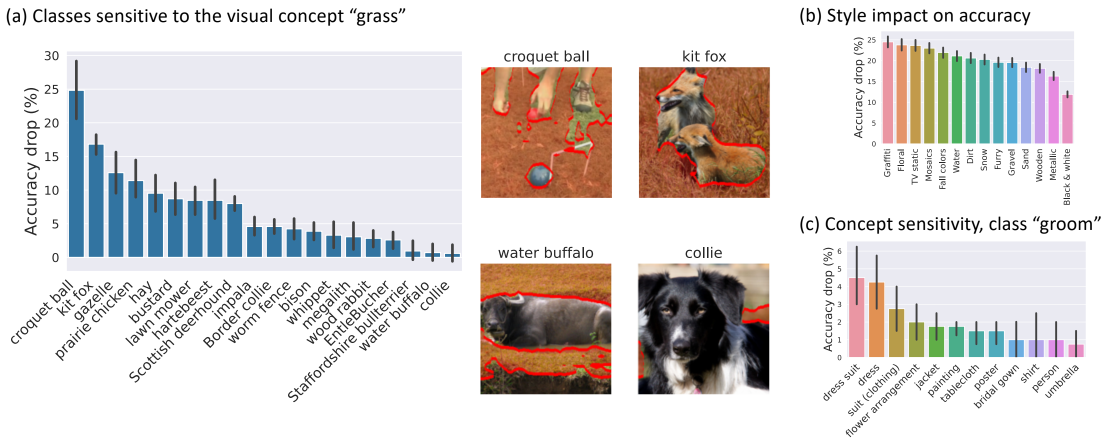


Figure 3-3: Model sensitivities diagnosed using our pipeline in a VGG16 classifier trained on ImageNet. (a) The accuracy drop induced by transformations of the concept "grass" highlights classes for which the model relies on this concept: e.g., a "croquet ball" is not accurately recognized if "grass" is not present, while "collie"s are not affected. (The twenty classes for which the visual concept is most often present are shown.) (b) Applying different styles to visual concepts reduces accuracy by varying amounts. (c) Visual concepts that cause accuracy losses for a given class can highlight context-dependent rules: e.g., the class "groom" is sensitive to the presence of "dress."

Per-class prediction rules. If we focus on the model's predictions for counterfactual inputs belonging to a single class, we can identify high-level concepts that it relies on for performing well on said class. It turns out that aside from the main image object, ImageNet classifiers also heavily depend on commonly co-occurring objects [SC18; Tsi+20; Bey+20] in the image—e.g., the objects "dress" for the class "groom", "person" for the class "tench" (*sic*), and "road" for the class "race car" (cf. Figure 3-3c and Appendix Figure B-3). We can also examine which concept-level transformations hurt model performance the most—e.g., making "plants" "floral" hurts accuracy on the class "damselfly" 15% more than making them "snowy".

Remark. In addition to automatically discovering and validating concept-level sensitivities of a given model, this pipeline has several additional advantages:

- *Versatility:* It does not require *any* annotation effort and can be directly applied to new datasets.
- *Human input:* It makes it easier for humans to encode their prior knowledge

and understanding of how the model *should* behave in the real world into the debugging process. For instance, one can test whether an image classifier relies on specific background features such as "snow" or "grass" that may not always be predictive in the real world.

- *Causality:* There has been growing interest in explaining the computations of deep networks in terms of human-understandable features by identifying neurons [Erh+09; ZF14; OMS17; Bau+17] or activation vectors [Kim+18; Zho+18] that correspond to high-level concepts within the model. However, these works that can identify concepts that are *correlated* with the model's output, we can validate whether and to what extent each of these concepts is *actually used* for classification.

3.3 Designing Debuggable Deep Networks

We now shift gears and discuss an alternative approach to model debugging. This shift is prompted by the fact that debugging networks post-hoc has inherent limitations. Firstly, due to the scale of these networks, it is challenging (and possibly infeasible) to exhaustively identify the data features they depend on. Moreover, as we discussed in Chapter 2, one cannot hope to fully interpret a model that relies on features that humans cannot even understand. This motivates the question: can we modify deep networks to be *intrinsically* easier to debug?

As we will demonstrate, this is indeed possible. Our approach is motivated by an alternative view of deep networks: as a combination of a deep feature[2] representation and a linear decision layer. This view is frequently adopted in other contexts such as transfer learning and domain adaptation [PY09; Yos+14]. However, it also turns out to be very useful in the debugging context. In particular, it allows us to gain insight into a complex, non-linear model by directly examining its deep features, and the coefficients used to aggregate them.

This simplified problem is however still intractable for current deep networks.

[2]Not to be confused with input features.

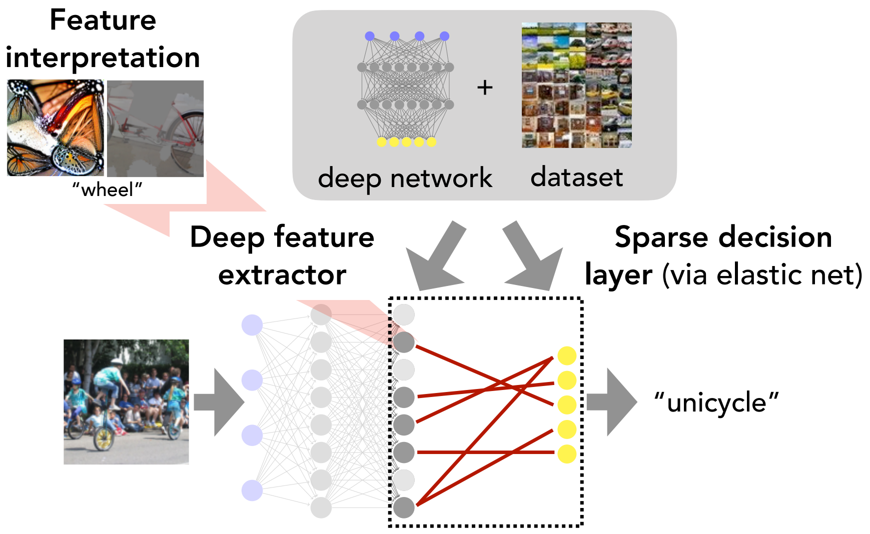


Figure 3-4: Illustration of our pipeline: For a given task, we construct a *sparse decision layer* by training a regularized generalized linear model (via elastic net) on the deep feature representations of a pre-trained deep network. We then debug model behavior by simply inspecting the few relevant deep features (with existing feature interpretation tools), and the linear coefficients used to aggregate them.

Their decision layers can easily have millions of parameters operating on thousands of deep features. To mitigate this, we modify the network itself: we retain the learned feature representation, but combine this with a *sparse* decision layer instead (cf. Figure 3-4). Debugging this sparse decision layer then entails inspecting only the few linear coefficients and deep features that dictate its predictions.

It turns out that the debuggability of such modified networks can be improved even further if we use a different training objective to obtain deep feature representations. The reason for this is that despite significant research, even a single deep feature (or neuron) within a standard network can be hard to interpret—these features often do not correspond to human-recognizable patterns in practice [OMS17]. Based on our findings in Chapter 2, this should not be surprising: after all, these models do rely on features that are incomprehensible to humans. In contrast, the deep feature representations from robust (adversarially-trained) models tend to have more human-aligned features, as we will see in Chapter 5. Thus, for unless otherwise specified, we use robust vision models for our analysis in this chapter.

3.3.1 Constructing sparse decision layers

One possible approach for constructing sparse decision layers is to apply pruning methods from deep learning [LDS90; Han+15; HS93; Li+16a; HMD16; Bla+20]—commonly-used to compress deep networks and speed up inference—to the dense decision layer. It turns out however that for linear classifiers we can actually do better. In particular, the problem of fitting sparse linear models has been extensively studied in statistics, leading to a suite of methods with theoretical optimality guarantees [Tib94; Efr+04; Has+07]. Here, we leverage the classic elastic net formulation [ZH05]—a generalization of LASSO and ridge regression that addresses their corresponding drawbacks.

For simplicity, we present an overview of the elastic net for linear regression, and defer the reader to Friedman, Hastie, and Tibshirani [FHT10] for a more complete presentation on the generalized linear model (GLM) in the classification setting. Let (X, y) be the standardized data matrix (mean zero and variance one) and output respectively. In our setting, X corresponds to the (normalized) deep feature representations of input data points, while y is the target. Our goal is to fit a sparse linear model of the form $\mathbb{E}(Y|X = x) = x^T\beta + \beta_0$. Then, the elastic net is the following convex optimization problem:

$$\min_{\beta} \frac{1}{2N}\|X^T\beta + \beta_0 - y\|_2^2 + \lambda R_\alpha(\beta) \tag{3.1}$$

where

$$R_\alpha(\beta) = (1-\alpha)\frac{1}{2}\|\beta\|_2^2 + \alpha\|\beta\|_1 \tag{3.2}$$

is referred to as the elastic net penalty [ZH05] for given hyperparameters λ and α. Typical elastic net solvers optimize (3.1) for a variety of regularization strengths $\lambda_1 > \cdots > \lambda_k$, resulting in a series of linear classifiers with weights $\beta_1, \ldots, \beta_k$ known as the *regularization path*, where

$$\beta_i = \arg\min_{\beta} \frac{1}{2N}\|X^T\beta - y\|_2^2 + \lambda_i R_\alpha(\beta) \tag{3.3}$$

In particular, a path algorithm for the elastic net calculates the regularization path where sparsity ranges the entire spectrum from the trivial zero model ($\beta = 0$) to completely dense. This regularization path can then be used to select a single linear model to satisfy application-specific sparsity or accuracy thresholds (as measured on a validation set).

Scalable solver for large-scale elastic net. Although the elastic net is widely-used for small-scale GLM problems, existing solvers can not handle the scale (number of samples and input dimensions) that typically arise in deep learning. In fact, at such scales, state-of-the-art solvers struggle to solve the elastic net even for a single regularization value, and cannot be directly parallelized due to their reliance on coordinate descent [FHT10]. We remedy this by creating an optimized GLM solver that combines the path algorithm of Friedman, Hastie, and Tibshirani [FHT10] with recent advancements in variance reduced gradient methods [GGS19]. The speedup in our approach comes from the improved convergence rates of these methods over stochastic gradient descent in strongly convex settings such as the elastic net. Using our approach, we can fit ImageNet-scale regularization paths to numerical precision in the order of hours on a single GPU (cf. Appendix B.2.1).

3.3.2 Interpreting deep features

A sparse linear model allows us to reason about the network's decisions in terms of a significantly smaller set of deep features. When used in tandem with off-the-shelf feature interpretation methods, the end result is a simplified description of how the network makes predictions. For our study, we utilize the following two widely-used techniques (cf. Figure 3-6 for sample visualizations):

1. *LIME [RSG16a]*: Although traditionally used to interpret model outputs, we use it to understand deep features. We fit a local surrogate model around the most activating examples of a deep feature to identify key "superpixels" for images or words for sentences.

2. *Feature visualization [Yos+15]*: Synthesizes inputs that maximally activate a given neuron.

3.4 Are Sparse Decision Layers Better?

We now apply our methodology to widely-used deep networks and assess the resulting sparse decision layers along a number of axes. We demonstrate that:

1. The standard (henceforth referred to as *"dense"*) linear decision layer can be made highly sparse at only a small cost to performance (Section 3.4.1).

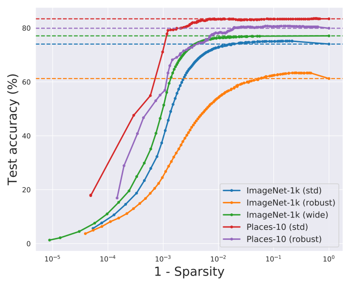
2. The deep features used by sparse decision layers are qualitatively and quantitatively better at summarizing the model's decision process (Section 3.4.2). Note that the dense and sparse decision layers operate on the same deep features—they only differ in the weight (if any) they assign to each one.

3. These aforementioned improvements (induced by the sparse decision layer) translate into better human understanding of the model (Section 3.4.3).

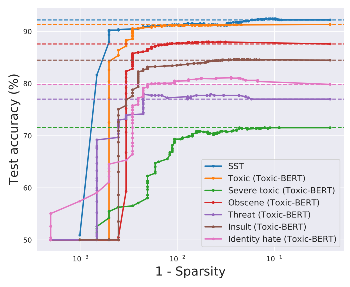
We perform our analysis on: (a) ResNet-50 classifiers [He+16] trained on ImageNet-1k [Den+09; Rus+15] and Places-10 (a 10-class subset of Places365 [Zho+17]); and (b) BERT [Dev+18] for sentiment classification on Stanford Sentiment Treebank (SST) [Soc+13] and toxicity classification of Wikipedia comments [WTD17].

3.4.1 Sparsity vs. performance

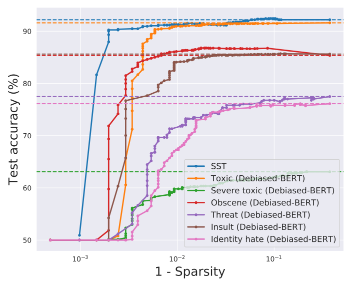
While a substantial reduction in the weights (and features) of a model's decision layer might make it easier to understand, it also limits the model's overall predictive power (and thus its performance). Still, we find that across datasets and architectures, the decision layer can be made substantially sparser—by up to two orders of magnitude—with a small impact on accuracy (cf. Figure 3-5). For instance, it is possible to find an accurate decision layer that relies on only about 20 deep features/class for ImageNet (as opposed to 2048 in the dense case). Toxic



(a)



(b)



(c)

Figure 3-5: Sparsity vs. accuracy trade-offs of models with sparse decision layers for (a) vision and (b,c) language tasks. Each point on the curve corresponds to single linear classifier from the regularization path in (3.3).

comment classifiers can be sparsified even further (<10 features/class), with *improved* generalization over the dense decision layer.

For the rest of our study, we select a single sparse decision layer to balance performance and sparsity—specifically the sparsest model whose accuracy is within 5% of top validation set performance. However, as discussed previously, these thresholds can be varied based on the needs of specific applications.

3.4.2 Sparsity and feature highlighting

Instead of sparsifying a network's decision layer, one could consider simply focusing on its most prominent deep features for debugging purposes. In fact, this is the basis of feature highlighting or principal reason explanations in the credit industry [BSR20]. How effective are such feature highlighting explanations at mirroring the underlying model?

In Table 3.1, we measure the accuracy of the dense/sparse decision layer when it is constrained to utilize only the top-k (5-10) features by weight magnitude. For dense decision layers, we consistently find that the top-k features do not fully capture the model's performance. This is in stark contrast to the sparse case, where the top-k features are both necessary, and to a large extent sufficient, to capture the model's predictive behavior. Note that the top-k features of the dense decision layers in the language setting almost completely fail at near random-chance performance (∼50%). This indicates that there do exist cases where focusing on the

Dataset/Model	k	Dense All	Dense Top-k	Dense Rest	Sparse All	Sparse Top-k	Sparse Rest
ImageNet (std)	10	74.03	58.46	55.22	72.24	69.78	10.84
ImageNet (wide, std)	10	77.07	72.42	48.75	73.48	73.45	0.91
ImageNet (robust)	10	61.23	28.99	34.65	59.99	45.82	19.83
Places-10 (std)	10	83.30	83.60	81.20	77.40	77.40	10.00
Places-10 (robust)	10	80.20	76.10	76.40	77.80	76.60	40.20
SST	5	91.51	53.21	91.17	90.71	90.48	50.92
Toxic-BERT (toxic)	5	83.33	55.35	57.87	82.47	82.33	50.00
Toxic-BERT (severe toxic)	5	71.53	50.00	50.14	67.57	50.00	50.00
Toxic-BERT (obscene)	5	80.41	50.03	50.00	77.32	72.39	50.00
Toxic-BERT (threat)	5	77.01	50.00	50.00	76.30	74.17	50.00
Toxic-BERT (insult)	5	72.72	50.00	50.00	77.14	75.80	50.00
Toxic-BERT (identity hate)	5	79.85	57.87	50.00	74.93	71.49	50.00
Debiased-BERT (toxic)	5	91.61	50.00	83.26	87.59	78.58	50.00
Debiased-BERT (severe toxic)	5	63.08	50.00	50.00	55.86	53.81	50.00
Debiased-BERT (obscene)	5	85.36	50.00	58.36	81.50	81.17	50.00
Debiased-BERT (threat)	5	77.49	50.00	50.00	68.96	50.00	50.00
Debiased-BERT (insult)	5	85.63	50.00	59.95	79.28	71.48	50.00
Debiased-BERT (identity hate)	5	76.12	50.00	50.84	71.98	50.00	50.00

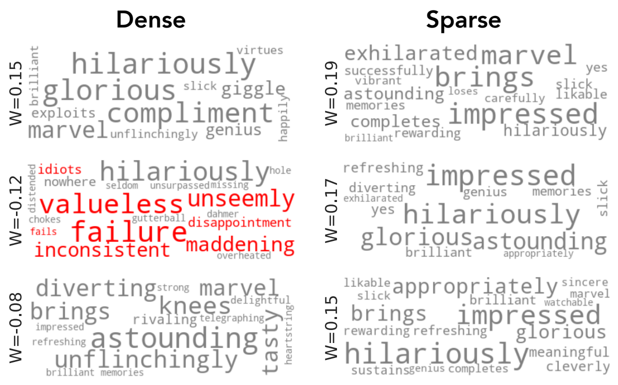
Table 3.1: Comparison of the accuracy of dense/sparse decision layers when they are constrained to utilize only the top-k deep features (based on weight magnitude). We also show overall model accuracy, and the accuracy gained by using the remaining deep features.

most important features (by weight) of a dense decision layer provides a misleading picture of global model behavior.

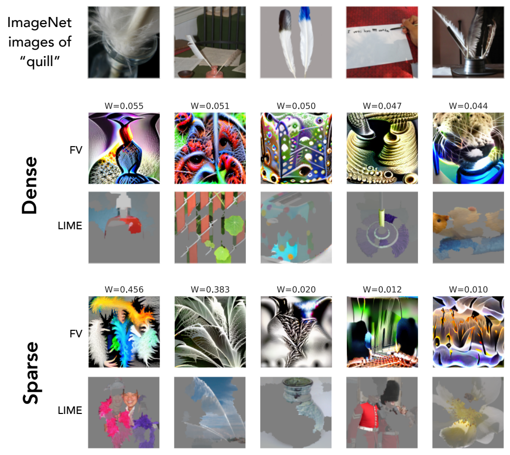
3.4.3 Sparsity and human understanding

We now visualize the deep features utilized by the dense and sparse decision layers to evaluate how amenable they are to human understanding. We show representative examples from sentiment classification (SST) and ImageNet in Figure 3-6a.

Specifically, we present word cloud interpretations of the top three deep features used by both of these decision layers for detecting positive sentiment on the SST dataset [Soc+13]. It is apparent that the sparse one selects features which ac-



(a)



(b)

Figure 3-6: (a) LIME-based word cloud visualizations for the highest-weighted features in the (dense/sparse) decision layers of BERT models for *positive* sentiment detection in the SST dataset. As highlighted in red, some of the key features used by the dense decision layer are actually activated for words with *negative* semantic meaning. (b) Visualization of deep features used by dense and sparse decision layers of a robust ($\varepsilon = 3$) ResNet-50 classifier to detect the ImageNet class "quill". Here we present five deep features used by each decision layer, that are randomly-chosen from the top-k highest-weighted ones—where k is the number of features used by the sparse decision layer for this class. For each (deep) feature, we show its linear coefficient (W), feature visualization (FV) and LIME superpixels.

tivate for words with positive semantic meaning. In contrast, the second most prominent deep feature for the dense decision layer is actually activated by words with *negative* semantic meaning. This example highlights how the dense decision layer can lead to unexpected features being used for predictions.

In Figure 3-6b, we present feature interpretations corresponding to the ImageNet class "quill" for both the dense and sparse decision layers of a ResNet-50 classifier. These feature visualizations seem to suggest that the sparse decision layer focuses more on deep features which detect salient class characteristics, such as "feather-like texture" and the "glass bottle" in the background.

Model simulation study. To validate the perceived differences in the vision setting, and ensure they are not due to confirmation biases, we conduct a human study on Amazon Mechanical Turk (MTurk). Our goal is to assess how well annotators are able to intuit (simulate) overall model behavior when they are exposed to its decision layer. Simulatibility is a standard evaluation criterion in interpretability [RSG16b; Lip18], wherein an interpretation is deemed to be good if it enables humans to reproduce what the model will decide (irrespective of the "correctness" of that decision).

To this end, we show annotators five randomly-chosen features used by the (dense/sparse) decision layer to recognize objects of a target class, along with the corresponding linear coefficients. We then present them with three samples from the validation set and ask them to choose the one that best matches the target class (cf. Appendix Figure B-8 for a sample task). Crucially, annotators are not provided with any information regarding the target class, and must make their prediction based solely on the visualized features.

For both the dense and sparse decision layers, we evaluate how accurate annotators are on average (over 1000 tasks)—based on whether they can correctly identify the image with the highest target class probability according to the corresponding model. For the model with a sparse decision layer, annotators succeed in guessing the predictions in $63.02 \pm 3.02\%$ of the cases. In contrast, they are only able to attain $35.61 \pm 3.09\%$ accuracy—which is near-chance (33.33%)—for the model with a dense decision layer. Crucially, these results hold *regardless* of whether the correct image is actually from the target class or not.

Note that our task setup precludes annotators from succeeding based on any prior knowledge or cognitive biases as we do not provide them with any semantic information about the target label, aside from the feature visualizations. Thus, annotators' success on this task in the sparse setting indicates that the sparse decision layer is actually effective at reflecting the model's internal reasoning process.

3.5 Debugging deep networks

We now demonstrate how deep networks with sparse decision layers can be substantially easier to debug than their dense counterparts. We focus on three problems: detecting biases, creating counterfactuals, and identifying input patterns responsible for misclassifications.

3.5.1 Biases and (spurious) correlations

Our first debugging task is to automatically identify unintended biases or correlations that deep networks extract from their training data.

Toxic comments. We start by examining two BERT models trained to classify comments according to toxicity: (1) Toxic-BERT, a high-performing model that was later found to use identity groups as evidence for toxicity, and (2) Debiased-BERT, which was trained to mitigate this bias [Bor+19].

We find that Toxic-BERT models with sparse decision layers also rely on identity groups to predict comment toxicity (visualizations in Appendix B.6.1 are censored). Words related to nationalities, religions, and sexual identities that are not inherently toxic occur frequently and prominently, and comprise 27% of the word clouds shown for features that detect toxicity. Note that although the standard Toxic-BERT model is known to be biased, this bias is not as apparent in the deep features used by its (dense) decision layer (cf. Appendix B.6.1). In fact, measuring the bias in the standard model required collecting identity and demographic-based subgroup labels [Bor+19].

We can similarly inspect the word clouds for the Debiased-BERT model with sparse decision layers and corroborate that identity-related words no longer appear as evidence for toxicity. But rather than ignoring these words completely, it turns out that this model uses them as strong evidence *against* toxicity. For example, identity words comprise 43% of the word clouds of features detecting non-toxicity. This suggests that the debiasing intervention proposed in Borkan et al. [Bor+19] may not have had the intended effect—Debiased-BERT is still disproportionately sensitive to identity groups, albeit in the opposite way.

We confirm that this is an issue with Debiased-BERT via a simple experiment: we take toxic sentences that this model (with a sparse decision layer) correctly la-

Toxic sentence	Change in score
DJ Robinsin is [redacted]! he [redacted] so much! [+christianity]	0.52 → 0.49
Jeez Ed, you seem like a [redacted] [+christianity]	0.52 → 0.48
Hey [redacted], quit removing FACTS from the article [redacted]!! [+christianity]	0.51 → 0.45

Table 3.2: Bias detection in language models: Using sparse decision layers, we find that Debiased-BERT is *still* disproportionately sensitive to identitity groups—except that it now uses this information as evidence against toxicity. For example, simply adding the word "christianity" to clearly toxic sentences flips the prediction of the model to non-toxic (score < 0.5).

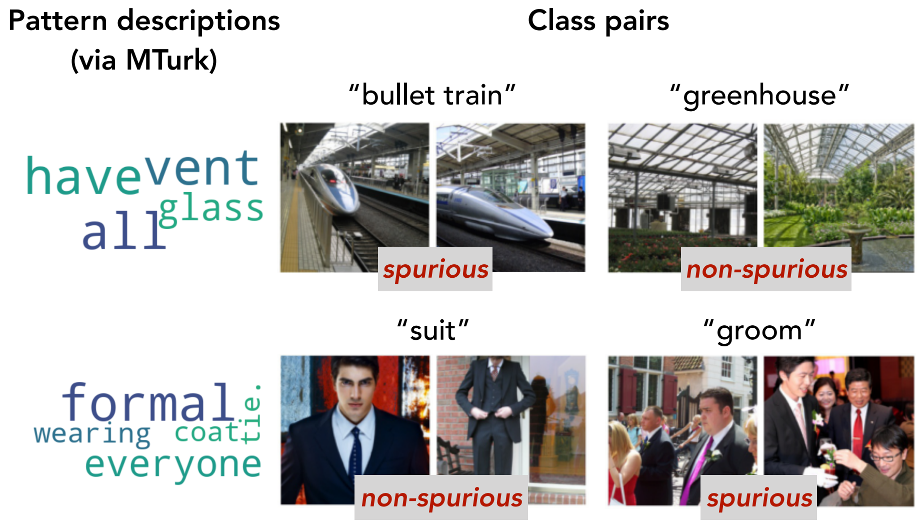
bels as toxic, and simply append an identity related word (as suggested by our word clouds) to the end—see Table 3.2. This modification turns out to strongly impact model predictions: for example, just adding "christianity" to the end of toxic sentences flips the prediction to non-toxic 74.4% of the time. We note that the biases diagnosed via sparse decision layers are also relevant for the standard Debiased-BERT model. In particular, the same toxic sentences with the word "christianity" are classified as non-toxic 62.2% of the time by the standard model, even though this sensitivity is not as readily apparent from inspecting its decision layer.

ImageNet. We now move to the vision setting, with the goal of detecting spurious feature dependencies in ImageNet classifiers. Once again, our approach is based on the following observation: input-class correlations learned by a model can be described as the data patterns (e.g., "dog ears" or "snow") that activate deep features used to recognize objects of that class, according to the decision layer.

Even so, it is not clear how to identify such patterns for image data, without access to fine-grained annotations describing image content. To this end, we rely on a human-in-the-loop approach (via MTurk). Specifically, for a deep feature of interest—used by the sparse decision layer to detect a target class—annotators are shown examples of images that activate it. They are then asked if these "prototyp-

Patterns (%)	Dense	Sparse
Non-spurious	18.43 ± 2.48	34.43 ± 3.38
Spurious	9.56 ± 1.76	12.49 ± 2.02
Total	27.85 ± 2.70	46.97 ± 3.15

(a)



(b)

Figure 3-7: (a) The percentage of class-level correlations identified using our MTurk setup, along with a breakdown of whether annotators believe the pattern to be "non-spurious" (i.e., part of the object) or "spurious" (i.e., part of the surroundings). (b) Examples of correlations in ImageNet models detected using our MTurk study. Each row contains protypical images from a pair of classes, along with the annotator-provided descriptions for the shared deep feature that these images strongly activate. For each class, we also display if annotators marked the feature to be a "spurious correlation".

ical" images have a shared visual pattern, and if so, to describe it using free-text.

However, under this setup, presenting annotators with images from the target class alone can be problematic. After all, these images are likely to have multiple visual patterns in common—not all of which cause the deep feature to activate. Thus, to disentangle the pertinent data pattern, we present annotators with prototypical images drawn from more than one classes. A sample task is presented in Appendix Figure B-11, wherein annotators see three highly-activating images for a specific deep feature from two different classes, along with the respective class labels. Aside from asking annotators to validate (and describe) the presence of a shared pattern between these images, we also ask them whether the pattern (if present) is part of each class object (non-spurious correlation) or its surroundings (spurious correlation)[3].

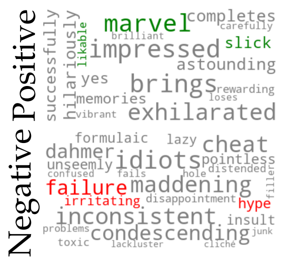
We find that annotators are able to identify a significant number of correlations that standard ImageNet classifiers rely on (cf. Table 3-7a). Once again, sparsity seems to aids the detection of such correlations. Aside from having fewer (deep) feature dependencies per class, it turns out that annotators are able to pinpoint the (shared) data patterns that trigger the relevant deep features in 20% more cases for the model with a sparse decision layer. Interestingly, the fraction of detected patterns that annotators deem spurious is lower for the sparse case. In Figure 3-7b, we present examples of detected correlations.

3.5.2 Counterfactuals

A natural way to probe model behavior is by trying to find small input modifications which cause the model to change its prediction. Such modified inputs, which are (a special case of) *counterfactuals*, can be a useful primitive for pinpointing input features that the model relies on. Aside from debugging, such counterfactuals can also be used to provide users with recourse [USL19] that can guide them to obtaining better outcomes in the future. We now leverage the deep features used by sparse decision layers to inform counterfactual generation.

Sentiment classifiers. Our goal here is to automatically identify word substitutions that can be made within a given sentence to flip the sentiment label assigned by the model. We do this as follows: given a sentence with a positive sentiment prediction, we first identify the set of deep features used by the sparse decision layer that are positively activated for any word in the sentence. For a randomly chosen deep feature from this pool, we then substitute the positive word from the sentence with its negative counterpart. This substitute word is in turn randomly chosen from the set of words that negatively activate the same deep feature (based

[3]We focus on this specific notion of "spurious correlations" as it is easy for humans to verify.



(a)

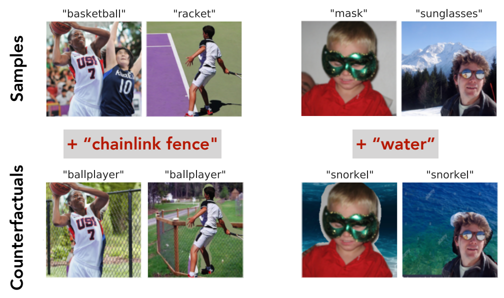
Original sentence	Counterfactual	Change in score
...something *likable* about the marquis...	...something *irritating* about the marquis...	0.73 → 0.34
Slick piece of cross-promotion	*Hype* piece of cross-promotion	0.73 → 0.34
A *marvel* like none you've seen	A *failure* like none you've seen	0.73 → 0.31

(b)

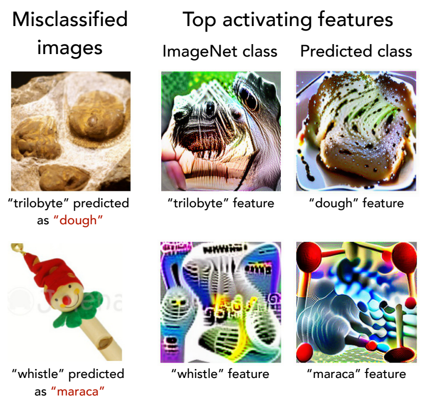
Figure 3-8: (a): Word cloud visualization for tokens that are positively/negatively correlated with the activation of a particular deep feature. (b): Using the word-clouds from (a), we can make word substitutions (as highlighted in green and red) to generate counterfactuals that change the model's predicted sentiment (scores below 0.5 are predicted as negative).

on its word cloud). An example of the positive and negative word clouds for one such deep feature is shown in Figure 3-8a, and the resulting counterfactuals are in Table 3-8b. Such counterfactuals successfully flip the sentiment label assigned by the sparse decision layer $73.1 \pm 3.0\%$ of the time. In contrast, they only have $52.2 \pm 4\%$ efficacy for the dense decision layer. This highlights that for models with sparse decision layers, it can be easier to automatically identify deep features that are causally-linked to model predictions.

ImageNet. We now leverage the annotations collected in Section 3.5.1 to generate counterfactuals for ImageNet classifiers. Concretely, we manually modify images to add or subtract input patterns identified by annotators and verify that they successfully flip the model's prediction. Some representative examples are shown in Figure 3-9a. Here, we alter images from various ImageNet classes to have the pattern "chainlink fence" and "water", so as to fool the sparse decision layer into recognizing them as "ballplayers" and "snorkels" respectively. We find that we are able to consistently change the prediction of the sparse decision layer (and in some cases its dense counterpart) by adding a pattern that was previously identified (cf. Section 3.5.1) to be a spurious correlation.



(a)



(b)

Figure 3-9: (a) Counterfactual images for ImageNet. We manually modify samples (*top row*) to contain the patterns "chainlink fence" and "water", which annotators deem (cf. Section 3.5.1) to be spuriously correlated with the classes "ballplayer" and "snorkel" respectively. We find that these counterfactuals (*bottom row*) succeed in flipping the prediction of the model with a sparse decision layer to the desired class. (b) Examples of misclassified ImageNet images for which annotators deem the top activated feature for the predicted class (*rightmost column*) as a better match than the top activated feature for the ground truth class (*middle column*).

3.5.3 Misclassifications

Our final avenue for diagnosing unintended behaviors in models is through their misclassifications. Concretely, given an image for which the model makes an incorrect prediction (i.e., not the ground truth label as per the dataset), our goal is to pinpoint some aspects of the image that led to this error.

On ImageNet, it turns out that over 30% of misclassifications made by the sparse decision layer can be attributed to a single deep feature—i.e., manually setting this "problematic" feature to zero fixes the misclassification (cf. Figure 3-10 for examples). In these cases, can humans understand why the problematic feature was triggered in the first place? Specifically, can they recognize the *pattern* in the input that caused the error?

To test this, we present annotators on MTurk with misclassified images. Without divulging the ground truth or predicted labels, we show annotators the top activated feature for each of the two classes via feature visualizations. We then ask annotators to select the patterns (i.e., feature visualizations) that match the image,

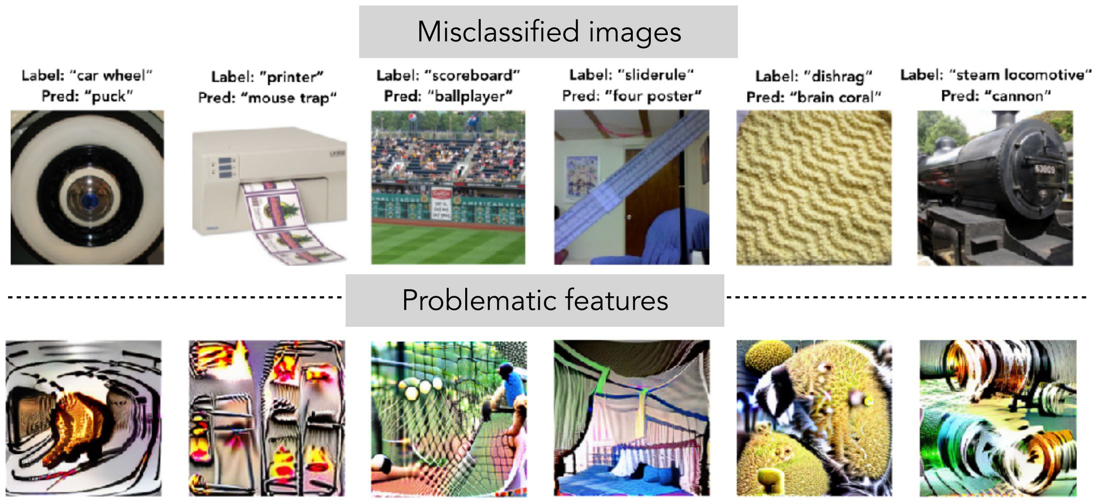

Misclassified images
Label: "car wheel" Pred: "puck"
Label: "printer" Pred: "mouse trap"
Label: "scoreboard" Pred: "ballplayer"
Label: "sliderule" Pred: "four poster"
Label: "dishrag" Pred: "brain coral"
Label: "steam locomotive" Pred: "cannon"
Problematic features

Figure 3-10: Misclassified images (*top*) along with the feature visualization for the corresponding "problematic" deep features from the erroneous class (*bottom*). Manually deactivating this single problematic feature (per image) is sufficient to correct the erroneous model prediction.

and to choose one that is a better match for the image (cf. Appendix B.8.1 for details). As a control, we repeat the same task but replace the problematic feature with a randomly-chosen one.

For about 70% of the misclassified images, annotators select the top feature for the predicted class as being present in the image (cf. Table 3.3). In fact, annotators consider it a better match than the feature for the ground truth class 60% of the time. In contrast, they rarely select randomly-chosen features to be present in the image. Since annotators do not know what the underlying classes are, the high fraction of selections for the problematic feature indicates that annotators actually believe this pattern is present in the image.

We present sample misclassifications validated by annotators in Figure 3-9b, along with the problematic features that led to them. Having access to this information can guide improvements in both models and datasets. For instance, model designers might consider augmenting the training data with examples of "maracas" without "red tips" to correct the second error in Figure 3-9b.

Features	Matches image	Best match
Prediction	70.70% $\pm$ 3.62%	60.12% $\pm$ 3.77%
Random	16.63% $\pm$ 2.91%	10.58% $\pm$ 2.35%

Table 3.3: Fraction of misclassified images for which annotators select the top feature of the predicted class to: (i) match the given image and (ii) be a better match than the top feature for the ground truth class. As a baseline, we also evaluate annotator selections when the top feature for the predicted class is replaced by a randomly-chosen one.

3.6 Further Related Work

We now discuss prior work with similar objectives or primitives to our toolkits.

Regularized GLMs and gradient methods. Estimating GLMs with convex penalties has been studied extensively [Efr+04; PH07; FHT10]. Our solver in Section 3.3 also builds off a line of work in variance reduced proximal gradient methods [JZ13; DBL14; GGS19]. Unlike our approach, prior solvers are best suited for problems with few examples or features, and are not directly amenable to GPU acceleration.

Counterfactual explanations and causal classification. Previous research has observed that causal effects of features can be clarified using counterfactual tests, either through synthetic data [Goy+19b]; by swapping features between individual images [Goy+19a]; or by silencing sets of neurons [Bau+20b]. In Section 3.2, we adopt the approach of generating counterfactual images, introducing a method that can scale to large datasets such as imagenet.

Interpretability tools. There have been extensive efforts towards post-hoc interpretability tools for deep networks. Feature attribution methods provide insight into model predictions for a specific input instance. These include saliency maps [SVZ13; Smi+17; STY17], surrogate models to interpret local decision boundaries [RSG16a], and finding influential [KL17], prototypical [KKK16], or counterfactual inputs [Goy+19a]. However, as noted by various recent studies, these local attributions can be easy to fool [GAZ19; Sla+20] or may otherwise fail to capture global aspects of model behavior [STY17; Ade+18; Ade+20; LM20]. Several methods have been proposed to interpret hidden units within vision networks, for example by generating feature visualizations [Erh+09; Yos+15; Ngu+16; OMS17] or assigning semantic concepts to them [Bau+17; Bau+20b]. Our work is complementary to these methods as we use them as primitives to probe sparse decision layers. Another related line of work is that on concept-based explanations, which seeks to explain the behavior of deep networks in terms of high-level concepts [Kim+18; Gho+19; Yeh+20]. One of the drawbacks of these methods is that the detected concepts need not be causally linked to the model's predictions [Goy+19b]. This is not the case for our approaches—we can quantify the effect of a given concept on the model's prediction in Section 3.2; and for our other approach in Section 3.3 the identified high-level concepts, i.e., the deep features used by the sparse decision layer, entirely determine the model's behavior.

3.7 Broader Implications: Quantifying interpretability

A major challenge in the field of interpretability is evaluating the quality of a given explanation. After all, ultimately, these explanations are intended to assist humans in understanding the models. Defining a metric that captures human understanding, and then measuring it without introducing confirmation biases is a highly non-trivial problem in itself. For instance, it may be hard for annotators to decouple "what they think the model should label the input as" from "what the interpretation suggests the model actually does" (and we are interested in the latter).

These considerations played an important role in the design of our model debugging toolkits. Firstly, rather than aiming for a complete characterization of how models make decisions, we focus on explanations that can help with concrete downstream tasks (e.g., identifying spurious correlations). Second, we verify that the data features we identify are not merely correlated with the model output, but in fact causally influence its predictions—via counterfactuals in Section 3.2 and feature ablations in Section 3.4. Finally, we design a suite of human-in-the-loop experiments to evaluate the quality of our explanations. In particular, these techniques allow us to assess whether an explanation improves human understanding and helps them with specific downstream tasks. Crucially, we are able to do so in practical deep learning settings (e.g., for ResNets trained on ImageNet-1k) and while, to the best of our ability, mitigating human biases.

Chapter 4

The Role of Data (Collection) in Learning

In this chapter, we take a step towards understanding why models learn the features they do. As we saw in Chapters 2 and 3, current models often succeed based on data features that are spurious, or context-dependent. Note, however, that these unintended features ultimately stem from the datasets that models are developed using. After all, despite their role in guiding the development of ML models [KSH12; Sze+16; He+16], these datasets [Eve+10; Den+09; Lin+14; Rus+15] are only proxies for real-world tasks that we actually care about—e.g., object recognition or localization in the wild. Thus, we ask: *How aligned are existing benchmarks, and the features therein, with their motivating real-world tasks?* Here, we explore this question, paying special heed to the *data collection* process. Through a case study on the popular ImageNet dataset [Den+09; Rus+15] we find that seemingly inconsequential design choices made during data collection actually introduce systematic—and fairly pervasive—biases in the dataset. These biases, in turn, percolate into ImageNet-trained models and strongly influence the features they rely on. This chapter is structured as follows: Section 4.1 outlines our contributions, Section 4.2 covers background, Sections 4.3-4.5 detail our analysis , and Section 4.6 discusses its broader implications.

4.1 Summary of Our Results

We first develop a methodology for obtaining fine-grained data annotations via large-scale human studies—to precisely quantify ways in which typical benchmarks, and their features, fall short of capturing the underlying ground truth. We then study how such *benchmark-task misalignment* impacts state-of-the-art models—after all, models are often developed in a way that treats existing datasets as the ground truth. We focus our exploration on the ImageNet dataset [Den+09; Rus+15], one of the most widely used benchmarks in computer vision.

Quantifying benchmark-task alignment. We find that systematic annotation issues pervade ImageNet, and can often be attributed to design choices in the dataset collection pipeline itself. For example, annotators were not asked to classify images, but rather to validate a specific automatically-obtained candidate label without knowledge of other classes in the dataset. Consequently, a large subset of the annotations are ambiguous or systematically biased—with confusion between fine-grained object classes or multiple image objects. Moreover, we find that the ImageNet label often does not even correspond to what humans deem the "main object" in the image. Nevertheless, models still achieve significantly-better-than-chance prediction performance on these images, indicating that they must exploit artifacts or spurious features in the dataset that humans are oblivious to.

Human-based performance evaluation. In light of this, we use our annotation pipeline to more directly measure human-model alignment. We find that more accurate ImageNet models also make predictions that annotators are more likely to agree with. In fact, we find that models have reached a level where non-expert annotators are largely unable to distinguish between predicted labels and ImageNet labels (i.e., even model predictions that don't match the dataset labels are often judged valid as by annotators). While reassuring, this finding highlights a different challenge: non-expert annotations may no longer suffice to tell apart further progress from overfitting to idiosyncrasies of the ImageNet distribution.

4.2 Background: Datasets and Biases

There has been a long line of work on building datasets for ML. For instance, in computer vision, this ranges from the famous *Lena* stock image to more recent constructions with millions of real-world photographs. Torralba and Efros [TE11] reflect on the history of dataset collection, describing it as a “revolution”: where every new benchmark was, to an extent, designed to fix shortcomings and biases in its older counterparts. We now overview prior work on understanding the source of such biases, and how our work fits into it. We refer the reader to Crawford [Cra13] and Friedman and Nissenbaum [FN17] for a more detailed discussion.

Three sources of dataset bias. Friedman and Nissenbaum [FN17] introduce a framework for studying bias in computer systems, breaking it down into:

- *Pre-existing:* This refers to biases that predate and exist independently of the dataset (or computer system), and percolate into it as a result of conscious, unconscious or despite best efforts of creators, and the society on which it is modeled. For instance, these might range from innocuous artifacts like photographer bias [TE11] or more pernicious effects such as gender or racial bias [Bol+16; CBN17; BG18; SC18].

- *Technical:* This includes biases that stem during the formulation of the task—in this case the dataset creation process. For instance, errors in the formulation of the target variable, or imperfections from the utilized software or human resources. The former frequently shows up in contexts such as predictive policing, or medicine, wherein the target variable (number of arrests or medical spending) can be an imperfect proxy for the real-task that one wants to model (crime or health risk) [LI16; Obe+19].

- *Emergent:* These biases arise during deployment, due to changes in society or the population of interest. Such biases have been studied in the context of model robustness and distribution shift—e.g., due to natural or adversarial input corruptions [Sze+14; FF15; FMF16; Eng+19c; For+19; HD19;

Kan+19], differences in data sources [Sae+10; TE11; Kho+12; TT14; Rec+19b], and changes in the relative frequencies of subpopulations [Ore+19; Sag+20].

Scalable data collection pipelines. Prior research discussed above has delved into understanding pre-existing and emergent biases in large-scale ML datasets. However, there are relatively few studies on technical biases in this context. It is worth noting that, despite best efforts, the sheer size of machine learning datasets makes meticulous data curation virtually impossible. Dataset creators thus resort to scalable methods such as automated data retrieval and crowd-sourced annotation [Eve+10; Rus+15; Lin+14; Zho+17], often at the cost of faithfulness to the task being modeled. As a result, the dataset and its corresponding annotations can sometimes be ambiguous, incorrect, or otherwise misaligned with ground truth (cf. Figure 1-2). Still, despite our awareness of these issues [Rus+15; Rec+19b; Hoo+19; NJC19], we lack a precise characterization of their pervasiveness and impact, even for widely-used datasets. In this chapter, we take a step towards bridging this gap in the context of the ImageNet dataset [Den+09; Rus+15].

4.3 A Closer Look at the ImageNet Dataset

We start by briefly describing the original ImageNet data collection and annotation process. As it turns out, several—seemingly innocuous—details of this process have a significant impact on the resulting dataset.

The ImageNet creation pipeline. ImageNet is a prototypical example of a large-scale dataset (1000 classes and millions of images) created through automated data collection and crowd-sourced filtering. Broadly, this process comprised two stages:

1. *Image and label collection:* The ImageNet creators first selected a set of classes using the WordNet hierarchy [Mil95]. Then, for each class, they sourced images by querying several search engines, in multiple languages, with the WordNet synonyms of the class, augmented with those of its parent node(s).

Note that for each of the retrieved images, the label—which we will refer to as the *proposed label*—that will be assigned to this image should it be included in the dataset, is already determined. That is, the label is simply given by the WordNet node that was used for the corresponding search query.

2. *Image validation via the* CONTAINS *task.* To validate the image-label pairs retrieved in the previous stage, the ImageNet creators employed annotators via the Mechanical Turk (MTurk) crowd-sourcing platform. Specifically, for every class, annotators were presented with its description (along with links to the relevant Wikipedia pages) and a grid of candidate images. Their task was then to select all images in that grid that contained an object of that class (with *explicit* instructions to ignore clutter and occlusions). The grids were shown to multiple annotators and only images that received a "convincing majority" of votes (based on per-class thresholds estimated using a small pool of annotators) were included in ImageNet. In what follows, we will refer to this filtering procedure as the CONTAINS task.

Revisiting the ImageNet labels

The automated process described above is a natural method for creating a large-scale dataset, especially if it involves a wide range of classes (as is the case for ImageNet). However, even putting aside occasional annotator errors, the resulting dataset might not accurately capture the ground truth (see Figure 1-2). Indeed, as we discuss below, this pipeline design itself can lead to certain *systematic* errors in the dataset. The root cause for many of these errors is that the image validation stage (i.e., the CONTAINS task) asks annotators only to verify if a *specific* proposed label (i.e., WordNet node for which the image was retrieved), shown in isolation, is valid for a given image. Crucially, annotators are never asked to choose among different possible labels for the image and, in fact, have no knowledge of what the other classes even are. This can introduce discrepancies in the dataset in two ways:

Images with multiple objects. Annotators are instructed to *ignore* the presence of other objects when validating a particular ImageNet label for an image. However, these objects could themselves correspond to other ImageNet classes. This can lead to the selection of images with multiple valid labels or even to images where the dataset label does not correspond to the most prominent object in the image.

Biases in image filtering. Since annotators have no knowledge of what the other classes are, they do not have a sense of the granularity of image features they should pay attention to (e.g., the labels in Figure 1-2 appear reasonable until one becomes aware of the other possible classes in ImageNet). Moreover, the task itself does not necessary account for their expertise (or the lack of thereof). Indeed, one cannot reasonably expect non-experts to distinguish, e.g., between all the 24 terrier breeds that are present in ImageNet. As a result, if annotators are shown images containing objects of a different, yet similar class, they are likely to select them as valid. This implies that potential errors in the collection process (e.g., automated search retrieving images that do not match the query label) are unlikely to be corrected during validation and thus can propagate to the final dataset.

In the light of the above, it is clear that eliciting ground truth information from annotators using the ImageNet creation pipeline may not be straightforward. In the following sections, we present a framework for improving this elicitation (by bootstrapping from and refining the existing labels) and then use that framework to investigate the discrepancies highlighted above (Section 4.4) and their impact on ImageNet-trained models (Section 4.5).

4.4 From Label Validation to Image Classification

We begin our study by obtaining a better understanding of the ground truth for ImageNet data. To achieve this, rather than asking annotators to *validate* a single proposed label for an image (as in the original pipeline), we would like them to *classify* the image, selecting *all* the relevant labels for it. However, asking (un-

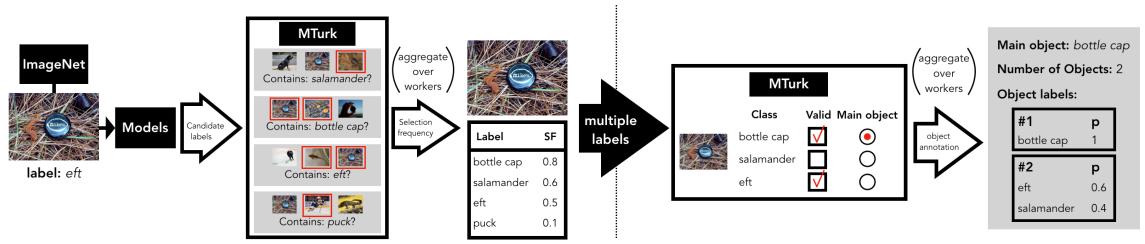


Figure 4-1: Overview of our data annotation pipeline. First, we collect a pool of potential labels for each image using the top-5 predictions of multiple models (Section 4.4.1). Then, we ask annotators to gauge the validity of each label (in isolation) using the CONTAINS task (described in Section 4.3). Next, we present all highly selected labels for each image to a new set of annotators and ask them to select *one* label for every distinct object in the image, as well as a label for the main object according to their judgement, i.e., the CLASSIFY task (Section 4.4.2). Finally, we aggregate their responses to obtain fine-grained image annotations (Section 4.4.2).

trained) annotators to choose from among all 1,000 ImageNet classes is infeasible.

To circumvent this difficulty, our pipeline consists of two phases, illustrated in Figure 4-1. First, we obtain a small set of (potentially) relevant *candidate labels* for each image (Section 4.4.1). Then, we present these labels to annotators and ask them to select one of them for *each* distinct object using what we call the CLASSIFY task (Section 4.4.2). For our analysis, we use 10,000 images from the ImageNet validation set—i.e., 10 randomly selected images per class. Note that since both ImageNet training and validation sets were created using the same procedure, analyzing the latter is sufficient to understand systematic issues in that dataset.

4.4.1 Obtaining candidate labels

To ensure that the per-image annotation task is manageable, we narrow down the candidate labels to a small set. To this end, we first obtain *potential labels* for each image by simply combining the top-5 predictions of 10 models from different parts of the accuracy spectrum with the existing ImageNet label (yields approximately 14 labels per image). Then, to prune this set further, we reuse the ImageNet CONTAINS task—asking annotators whether an image contains a particular class (Section 4.3)—but for *all* potential labels. The outcome of this experiment is a *selection frequency* for each image-label pair, i.e., the fraction of annotators that selected the image as containing the corresponding label.[1] We find that, although images were often selected as valid for many labels, relatively few of these labels had high selection frequency (typically less than five per image). Thus, restricting potential labels to this smaller set of *candidate labels* allows us to hone in on the most likely ones, while ensuring that the resulting annotation task is still cognitively tractable.

4.4.2 Image classification via the CLASSIFY task

Once we have identified a small set of candidate labels for each image, we present them to annotators to obtain fine-grained label information. Specifically, we ask them to identify: (a) *all* labels that correspond to objects in the image, and (b) the label for the *main* object. Crucially, we explicitly instruct annotators to select only *one label per distinct object*—i.e., in case they are confused about the correct label for a specific object, to pick the one they consider most likely. Moreover, since ImageNet contains classes that could describe parts or attributes of a single physical entity (e.g., "car" and "car wheel"), we ask annotators to treat these as distinct objects, since they are not mutually exclusive. We present each image to multiple annotators and then aggregate their responses (per-image) as described below. We refer to this annotation setup as the CLASSIFY task.

Identifying the main label and number of objects. From each annotator's response, we learn what they consider to be the label of the main object, as well as how many objects they think are present in the image. By aggregating these two quantities based on a majority vote over annotators, we can get an estimate of the number of objects in the image, as well as of the main label for that image.

Partitioning labels into objects. Different annotators may choose different labels for the same object and thus we need to map their selections to a single set of distinct objects. To illustrate this, consider an image of a soccer ball and a terrier,

[1]Note that this notion of selection frequency (introduced by Recht et al. [Rec+19b]) essentially mimics the majority voting process used to create ImageNet (cf. Section 4.3), except that it uses a fixed number of annotators per grid instead of the original adaptive process.

where one annotator has selected “Scotch terrier” and “soccer ball” and another “Toy terrier” and “soccer ball”. We would like to partition these selections into objects as [“soccer ball”] and [“Scotch terrier” or “Toy terrier”] since both responses indicate that there are two objects in the image and that the soccer ball and the terrier are distinct objects. More generally, we would like to partition selections in a way that avoids grouping labels together if annotators identified them as distinct objects. To this end, we employ exhaustive search to find a partition that optimizes for this criterion (since there exist only a few possible partitions to begin with). Finally, we label each distinct object with its most frequently selected label.

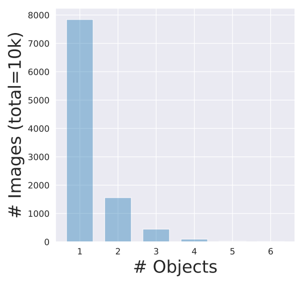
The resulting annotations characterize the content of an image in a more fine-grained manner compared to the original ImageNet labels. Note that these annotations may still not perfectly match the ground truth. After all, we also employ untrained, non-expert annotators, that make occasional errors, and moreover, some images are inherently ambiguous without further context (e.g., Figure 1-2c). Nevertheless, as we will see, these annotations are already sufficient for our examination of ImageNet.

4.5 Quantifying the Benchmark-Task Alignment of ImageNet

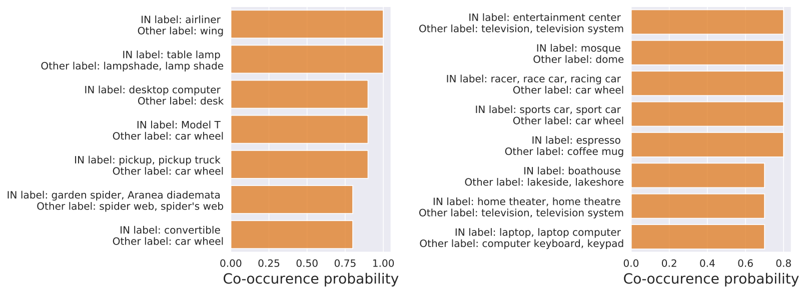
Our goal in this section is two-fold. First, we want to use our refined image annotations to examine potential sources of discrepancy between ImageNet and the motivating object recognition task. Next, we want to assess the impact these deviations have on models developed using ImageNet benchmark. To this end, we will measure how the accuracy of a diverse set of models is affected when they are evaluated on different sub-populations of images identified using our annotations.

4.5.1 Multi-object images

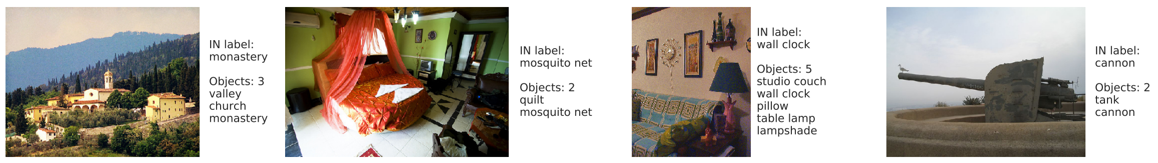
We start by taking a closer look at images which contain objects from more than one ImageNet class—how often these additional objects appear and how salient they are. Recall that if two labels are both simultaneously valid for an image—i.e., they are not mutually exclusive (e.g., "car" and "car wheel")—we refer to them as different objects. Figure 4-2a shows that a significant fraction of images—more than a fifth—contains *at least two* objects. In fact, there are pairs of classes which *consistently co-occur* (see Figure 4-2b). This indicates that multi-object images in ImageNet are not caused solely by irrelevant clutter, but also arise due to systematic issues with the class selection process. Indeed, even though, in principle, ImageNet classes correspond to distinct objects, some of these objects can overlap greatly in terms of how they occur in the real world.



(a)



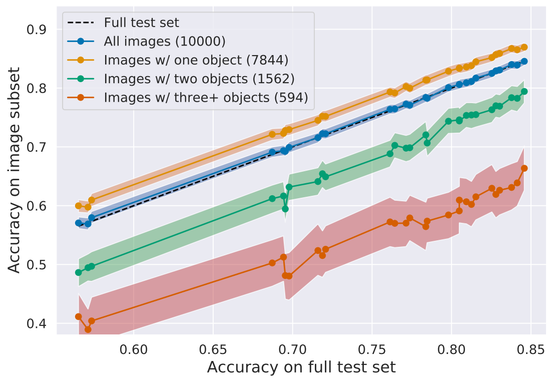
(b)



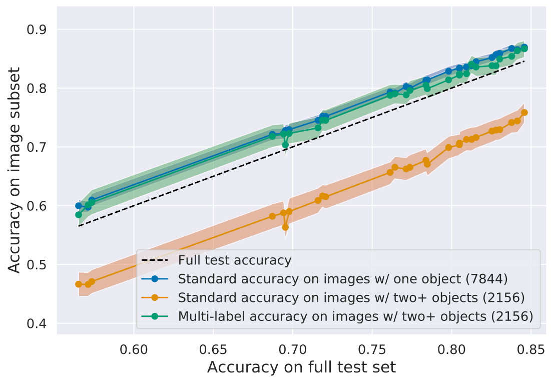
(c)

Figure 4-2: (a) Number of objects per image—more than a fifth of the images contains two or more objects from ImageNet classes. (b) Pairs of classes which consistently co-occur as distinct objects. Here, we visualize the top 15 ImageNet classes based on how often their images contain another *fixed* object ("Other label"). (c) Random examples of multi-label ImageNet images.

Model accuracy on multi-object images. Model performance is typically measured using (top-1 or top-5) accuracy with respect to a *single* ImageNet label, treating it as the ground truth. However, it is not clear what the right notion of ground truth annotation even is when classifying multi-object images. Indeed, we find that models perform significantly worse on multi-label images based on top-1 accuracy (measured w.r.t. ImageNet labels): accuracy drops by more than 10% across all models—see Figure 4-3a.



(a)

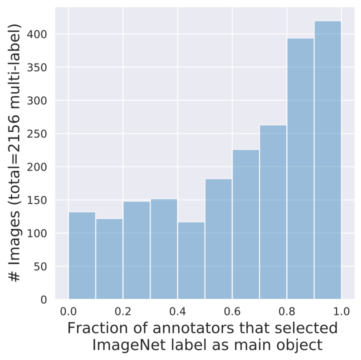


(b)

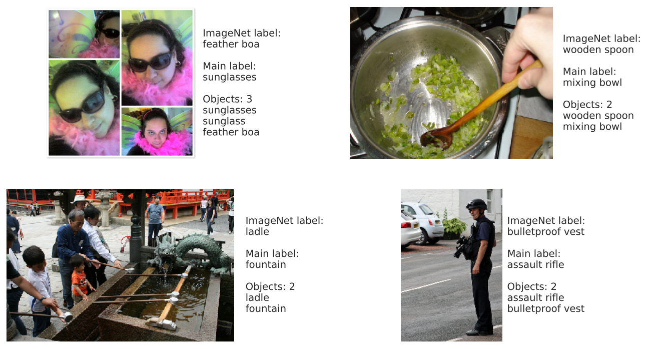
Figure 4-3: (a) Top-1 model accuracy on multi-object images (as a function of overall test accuracy). Accuracy drops by roughly 10% across all models. (b) Evaluating multi-label accuracy on ImageNet: the fraction of images where the model predicts the label of *any* object in the image. Based on this metric, the performance gap between single- and multi-object images virtually vanishes. Confidence intervals: 95% via bootstrap.

In light of this, a more natural notion of accuracy for multi-object images would be to consider a model prediction to be correct if it matches the label of *any* object in the image. On this metric, we find that the aforementioned performance drop essentially disappears—models perform similarly on single- and multi-object images (see Figure 4-3b). This indicates that the way we typically measure accuracy, i.e., with respect to a single label, can be overly pessimistic.Note, however, that while evaluating top-5 accuracy also accounts for most of these multi-object confusions—which was after all the original motivation for considering that measure [Rus+15], it tends to inflate accuracy on *single object* images, by treating several erroneous predictions as valid (cf. Appendix C.3.1).

Human-label disagreement. Although models suffer a sizeable accuracy drop on multi-object images, they are still relatively good at predicting the ImageNet label—much better than the baseline of choosing the label of one object at random. This bias could be justified whenever there is a distinct main object in the image, which also corresponds to the ImageNet label. However, we find that for nearly a third of the multi-object images, the ImageNet label does *not* denote the most likely main object as judged by annotators—see Figure 4-4. Nevertheless, model accuracy (w.r.t. the ImageNet label) on these images is still high—see Figure 4-5a.



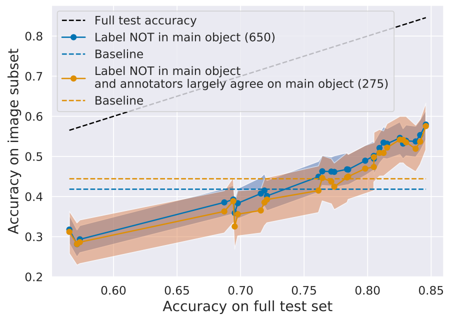
(a)



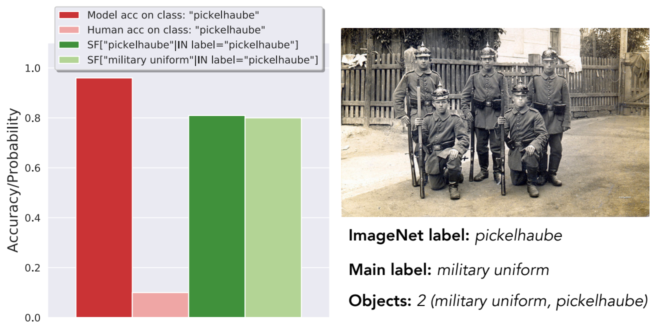
ImageNet label: feather boa
Main label: sunglasses
Objects: 3 sunglasses sunglass feather boa
ImageNet label: wooden spoon
Main label: mixing bowl
Objects: 2 wooden spoon mixing bowl
ImageNet label: ladle
Main label: fountain
Objects: 2 ladle fountain
ImageNet label: bulletproof vest
Main label: assault rifle
Objects: 2 assault rifle bulletproof vest

(b)

Figure 4-4: (a) Fraction of annotators that selected the ImageNet label as denoting the main image object. For 650 images (out of 2156 multi-object images), the majority of annotators select a label *other* than the ImageNet one as the main object. Examples of such images are presented in (b).



(a)



(b)

Figure 4-5: (a) Model accuracy on images where the annotator-selected main object does not match the ImageNet label. Models perform much better than the baseline of randomly choosing one of the objects in the image (dashed line)—potentially by exploiting dataset biases. (b) Example of a class where humans disagree with the label as to the main object, yet models still predict the ImageNet label. Here, for images of that class, we plot both model and annotator accuracy as well as the selection frequency (SF) of both labels.

On these samples, models must thus base their predictions on features that humans do not consider salient. For instance, we find that these disagreements often arise when the ImageNet label corresponds to a very distinctive object (e.g., “pickelhaube”), but the image also contain another more prominent object (e.g., “military uniform”)—see Figure 4-5b. To do well on these classes, the model likely picks up on ImageNet-specific biases, e.g., detecting that military uniforms often occur in images of class “pickelhaube” but not vice versa. While this might be a valid strategy for improving ImageNet accuracy, it causes models to rely on features that may not generalize to the real world.

4.5.2 Bias in label validation

We now turn our attention to assessing the quality of the ImageNet filtering process. Our goal is to understand how likely annotators are to detect incorrect image-label pairs during the image validation process. Recall that they are asked a somewhat leading question, i.e., if a specific label is valid for the image, making them prone to answering positively even for images from a different, yet similar, class.

Indeed, we find that when we replicate the original task setup (i.e., the CONTAINS task) with different proposed labels, annotators consistently select some of these labels, *in addition to* the ImageNet label, as being valid for an image. In fact, for nearly 40% of the images, another label is selected *at least as often* as the ImageNet label (cf. Figure 4-6). Moreover, this phenomenon does not occur only when multiple objects are present in the image—even when annotators observe a single object, theyoften select as many as 10 classes (cf. Figure 4-7a). Thus, even for images where a single ground truth label exists, the ImageNet validation process may fail to elicit this label from annotators.[2]

In fact, we find that this confusion is not just a consequence of using non-expert annotators, but also of the CONTAINS task setup itself. If instead of asking annotators to judge the validity of a specific label(s) in isolation, we ask them to choose the

[2]Note that our estimates for the selection frequency of image-ImageNet label pairs may be biased (underestimates) [Eng+20] as these specific pairs have already been filtered during dataset creation based on their selection frequency. However, we can effectively ignore this bias since: a) our results seem to be robust to varying the number of annotators (Appendix C.3.2), b) most of our results are based on the CLASSIFY task for which this bias does not apply.

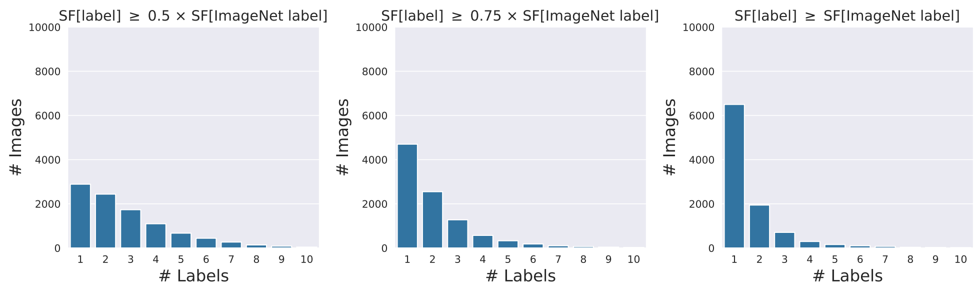
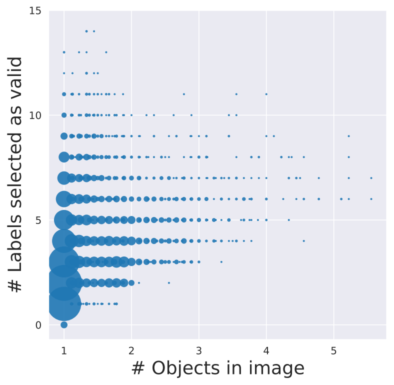
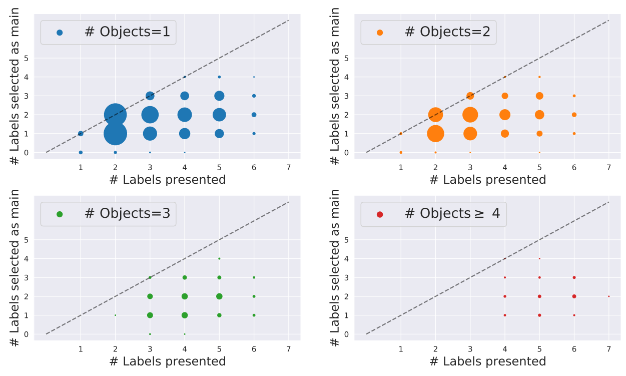


Figure 4-6: Number of labels per image that annotators selected as valid in isolation (determined by the selection frequency of the label relative to that of the ImageNet label). For more than 70% of images, annotators select another label at least half as often as they select the ImageNet label (*leftmost*).



(a)



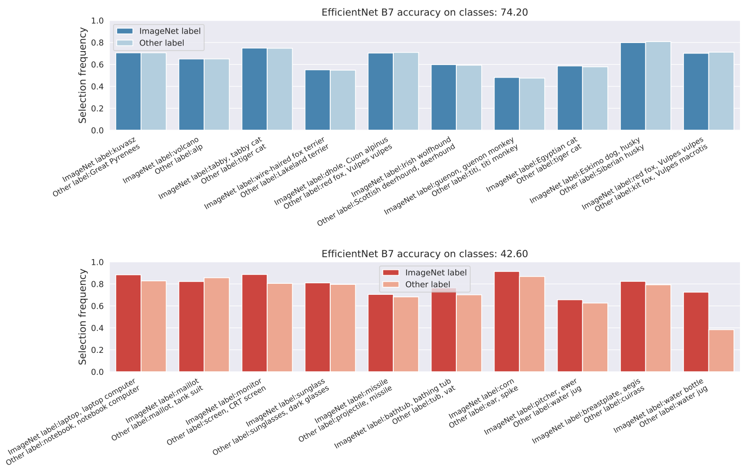
(b)

Figure 4-7: (a) Number of labels selected in the CONTAINS task: the y-axis measures the number of labels that were selected by at least two annotators; the x-axis measures the average number of objects indicated to be present in the image during the CLASSIFY task; the dot size represents the number of images in each 2D bin. Even when annotators view an images as depicting a single object, they often select multiple labels as valid. (b) Number of labels that at least two annotators selected for the main image object (in the CLASSIFY task) as a function of the number of labels presented to them. Annotator confusion decreases significantly when the task setup explicitly involves choosing between multiple labels *simultaneously*.

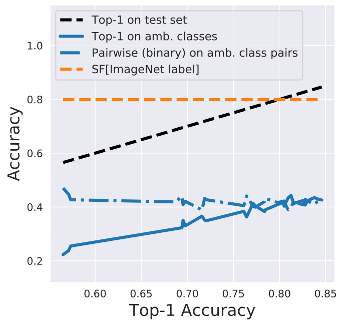
main object among several possible labels simultaneously (i.e., via the CLASSIFY task), they select substantially fewer labels—see Figure 4-7b.

These findings highlight how sensitive annotators are to the data collection and validation pipeline setup, even to aspects of it that may not seem significant at first glance. It also indicates that the existing ImageNet annotations may not have be vetted as carefully as one might expect. ImageNet annotators might have been unable to correct certain errors, and consequently, the resulting class distributions may be determined, to a large extent, by the fidelity (and biases) of the automated image retrieval process.

Confusing class pairs. We find that there are several pairs of ImageNet classes that annotators have trouble telling apart—they consistently select both labels as valid for images of either class—see Figure 4-8a. On some of these pairs we see that models still perform well—likely because the search results from the automated image retrieval process are sufficiently error-free (i.e., the overlap in the actual image search results for the two classes is insignificant enough) to allow the models to disambiguate between these pairs.



(a)



(b)

Figure 4-8: (a) ImageNet class pairs for which annotators often deem both classes as valid. We visualize the top 10 pairs split based on the accuracy of EfficientNet B7 on these pairs being high (*top*) or low (*bottom*). (b) Model progress on ambiguous class pairs (from (a) *bottom*) has been largely stagnant—possibly due to substantial overlap in the class distributions. In fact, models are unable to distinguish between these pairs better than chance (cf. pairwise accuracy).

However, for other pairs, even state-of-the-art models have poor accuracy (below 40%)—see Figure 4-8b. In fact, we can attribute this poor performance to model confusion *within* these pairs—*none* of the models we examined do much better than chance at distinguishing between the two classes. The apparent performance barrier on these ambiguous classes could thus be due to an inherent overlap in their ImageNet distributions. It is likely that the automated image retrieval caused mixups in the images for the two classes, and annotators were unable to rectify these. If that is indeed the case, it is natural to wonder whether accuracy on ambiguous classes can be improved without overfitting to the ImageNet test set.

The annotators' inability to remedy overlaps in the case of ambiguous class pairs could be due to the presence of classes that are semantically similar, e.g., "rifle" and "assault rifle"—which is problematic under the CONTAINS task as discussed before. In some cases, we find that there also were errors in the task setup. For instance, there were occasional overlaps in the class names (e.g., "maillot" and "maillot, tank suit") and Wikipedia links (e.g., "laptop computer" and "notebook computer") presented to the annotators. This highlights that choosing labels that are in principle disjoint (e.g., using WordNet) might not be sufficient to ensure that the resulting dataset has non-overlapping classes—when using noisy validation pipelines, we need to factor human confusion into class selection and description.

Remark. Some of the ImageNet label issues we study have already been identified in prior work. Specifically, Recht et al. [Rec+19b], Northcutt, Jiang, and Chuang [NJC19], and Hooker et al. [Hoo+19] demonstrate the existence of classes that might be inherently ambiguous (similar to our findings in Section 4.5.2). Moreover, the existence of cluttered images was discussed by Russakovsky et al. [Rus+15] as an indication that the dataset mirrors real-world conditions—and hence deemed desirable—and by Stock and Cisse [SC18], Northcutt, Jiang, and Chuang [NJC19], and Hooker et al. [Hoo+19] as a source of label ambiguity (similar to our findings in Section 4.5.1). Additionally, Stock and Cisse [SC18], motivated by the multi-label nature of images, perform human-based model evaluation, similar to our experiments in Section 4.6. Finally, Stock and Cisse [SC18] use manual data collection and saliency maps to identify racial biases that models rely on to make their predictions. However, the focus of all these studies is not on characterizing the extent of these issues and they only provide coarse estimates for their pervasiveness in ImageNet. In particular, none of these studies evaluate how these issues affect model performance, nor do they obtain annotations that are sufficient (in granularity and/or scale) to rigorously draw per-class conclusions.

4.6 Broader Implications: Task Formulation and Model Evaluation

We now discuss implications our findings about annotation issues in ImageNet on model evaluation and dataset design.

4.6.1 Beyond test accuracy: human-in-the-loop evaluation

Our analysis so far makes it clear that using top-1 accuracy as a standalone performance metric can be problematic—issues such as multi-object images and ambiguous classes make ImageNet labels an imperfect proxy for the ground truth. With this in mind, we now focus on augmenting the model evaluation toolkit with metrics that are better aligned with the underlying goal of object recognition.

We start by *directly* employing annotators to assess how good model predictions are. Our goal is to understand whether more accurate models also make higher-quality predictions, i.e., if the labels they predict (including the erroneous ones) also appear more reasonable to humans? Intuitively, this should not only help account for imperfections in ImageNet labels but also to capture improvements in models that might not be reflected by improvements in accuracy alone (e.g., predicting a dog breed that is incorrect but more similar to the correct one). Concretely, given a model prediction for a specific image, we measure how often annotators select the predicted label as being present in the image (determined using the CONTAINS task). Note that this metric accommodates for multiple objects or ambiguous classes as annotators will confirm all valid labels.

We find that models *do* improve consistently along these axes as well (cf. Figure 4-9)—faster than improvements in accuracy (i.e., more predictions matching the ImageNet label) alone could explain. Moreover, we observe that the predictions of state-of-the-art models have gotten, on average, quite close to ImageNet labels with respect to these metrics. That is, annotators are almost *equally likely* to

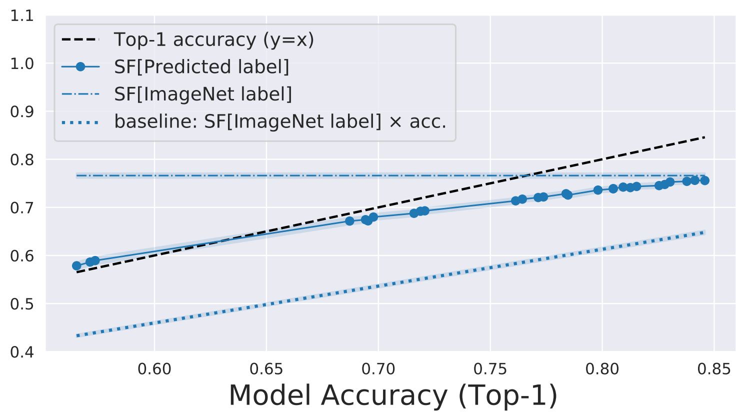


Figure 4-9: Using humans to assess model predictions—we measure how often annotators select the predicted/ImageNet label to be contained in the image (*selection frequency* [SF]), along with 95% confidence intervals (shaded).

select the predicted label as valid for the image as the ImageNet label. This indicates that model predictions might be closer to what non-expert annotators can recognize as the ground truth than accuracy alone suggests.

Implications. These findings do not imply that all of the remaining gap between state-of-the-art model performance and perfect top-1 accuracy is inconsequential. After all, for many images, the labels shown to annotators during the ImageNet creation process (based on automated data retrieval) could have been the ground truth. In these cases, striving for higher accuracy on ImageNet would actually extend to real-world object recognition settings. However, the results in Figure 4-9 hint at a different issue: we are at a point where we may no longer by able to easily identify (e.g., using crowd-sourcing) the extent to which further gains in accuracy correspond to such improvements, as opposed to models simply matching the ImageNet distribution better.

4.6.2 Defining and constructing benchmark tasks.

Our findings highlight an inherent conflict between the goal of building datasets that are large and diverse enough to capture complexities of the real world and the need for the corresponding annotation process to be scalable. Indeed, in the context of ImageNet, we found that some of the very reasons that make the collection pipeline scalable (e.g., resorting to the CONTAINS task, employing non-expert annotators) were also at the core of the afore-mentioned annotation issues. We believe that developing annotation pipelines that better capture the ground truth while remaining scalable is an important avenue for future research.

Taking a step back, our exploration also raises concerns about how benchmark tasks are formulated in the first place. For instance, the assumption that there exists a single "ground truth" label for each input may be fundamentally flawed in many scenarios. It is unclear whether humans can even agree upon such a label, or if such a single-label classification task is even pertinent in the real world. This also calls into question the currently dominant approach to ML model development, which datasets as the "gold standard" for measuring and guiding progress. Our work demonstrates that this practice might not be well-founded—these datasets are often imperfect proxies for the real-world tasks they were meant to capture.

Finally, our analysis highlights the importance of doing away with the black-box view on datasets. Namely, if we want to build models that can generalize reliably, it is crucial that we periodically revisit and scrutinize the datasets that we use to train them. This may be even more pertinent when instead of using curated benchmarks, we must resort to training models on observational data in real-world applications—e.g., medicine or self-driving cars.

Chapter 5

Robustness as a Feature Prior

In this chapter, we introduce a training framework that allows us to control the features that ML models rely on to make their predictions. Recall that our findings so far indicate that while current models perform well on typical benchmarks, the mechanisms underlying this success differ from those that humans employ or envision. For instance, these models tend to utilize unintuitive or undesirable features in the data—e.g., non-robust ones, image backgrounds or sensitive attributes. Crucially, these dependencies are the consequence of core aspects of the ML development pipeline. For instance, the dominant training paradigm of optimizing models to solely maximize (distributional) accuracy causes them to latch onto *any* predictive features in the data—even ones that are brittle or spurious. Thus, the natural question that arises is: can we modify our training objectives to prevent models from relying on specific types of features? Here, we discuss how the classic *robust optimization framework* [Wal45]—which is commonly used to build models that are robust to adversarial examples—can provide a natural solution to this problem. This chapter is structured as follows: Section 5.1 outlines our contributions, Section 5.2 introduces the robust optimization framework, and Sections 5.3-5.4 focus on a closer examination of the resulting robust models (and their features) and finally Section 5.5 discusses the broader implications of our analyses.

5.1 Summary of Our Results

We propose using the robust optimization framework [Wal45] as a tool to enforce (user-specified) *priors* on features that models should learn. In particular, we discuss how this approach can be used to train models that do not depend on ℓ_p non-robust features in input images (Section 5.2).

We then study the effect of this train-time intervention on the resulting "robust models" [GSS15; Mad+18], focusing on their feature representations, as well as their performance on downstream tasks. We find that these models have:

"Better" features (Section 5.3). The feature representations learned by robust models address many of the shortcomings affecting their standard counterparts. For instance, as illustrated in Figure 5-1, they tend to be approximately invertible and amenable to direct feature visualization. Broadly, our results indicate that robust optimization is a promising avenue for learning representations that are more "aligned" with our notion of perception.

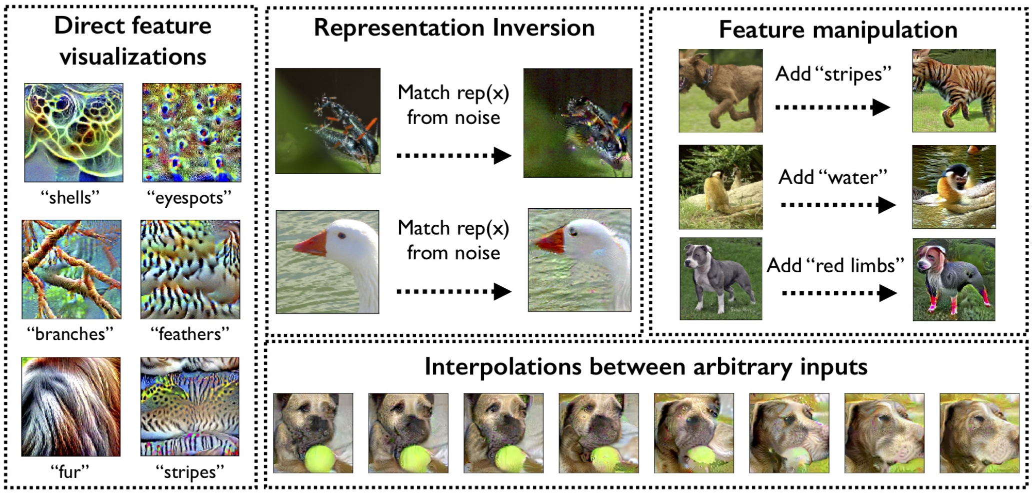


Figure 5-1: Illustration of the properties of "robust representations".

Improved performance on downstream tasks (Section 5.4). Robust models suffice to perform a range of downstream image synthesis tasks—e.g., image generation, interpolation and manipulation (c.f. Figure 5-2 for examples). Moreover, doing so only requires: (i) a generic robust classifier (say, ResNet-50) trained on a standard dataset (say, ImageNet) with minimal tuning, and (ii) performing a simple input manipulation: maximizing predicted class scores with gradient descent.

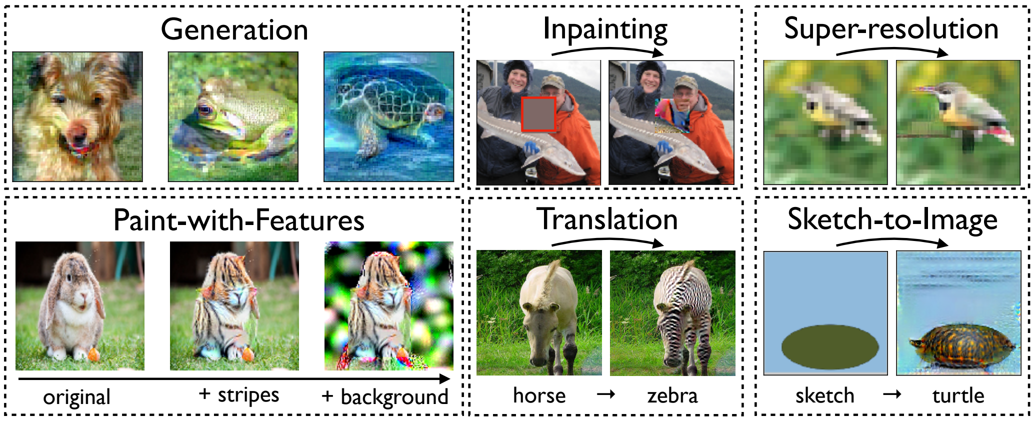


Figure 5-2: Image synthesis tasks performed using a *single* robust classifier.

5.2 The Robust Optimization Framework

In the canonical classification setting, the focus is on maximizing standard accuracy, i.e., the performance on (yet) unseen samples from the underlying distribution. Specifically, to find a set of model parameters that minimizes the *expected loss* (also known as population risk):

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}_\theta(x,y)\right]. \tag{5.1}$$

We refer to (5.1) as the *standard* training objective or empirical risk minimization (ERM). In general, there are several choices for the loss function $\mathcal{L}$ (e.g., hinge, squared), though the most popular one is the cross-entropy loss.

5.2.1 Adversarial robustness

The existence of adversarial examples [Sze+14] largely changed the picture described above. In particular, there has been a lot of interest in developing models that are resistant to them, or, in other words, models that are *adversarially robust*.

A natural approach (and one of the most successful) for doing so is to use the *robust optimization framework*: a classical tool for optimization in the presence of uncertainty [Wal45; Dan67]. In particular, instead of just finding parameters which minimize the expected loss (5.1), a robust optimization objective also requires that the model induced by the parameters θ be robust to worst-case perturbation of the input, or have low *expected adversarial loss*:

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\Delta}\mathcal{L}_\theta(x+\delta,y)\right]. \tag{5.2}$$

We refer to (5.2) as the *robust* training objective. Here, Δ represents the set of perturbations that the adversary can apply to induce misclassification. The most common choice for Δ is the set of ℓ_p-bounded perturbations, i.e., $\Delta = \{\delta \in \mathbb{R}^d \mid \|\delta\|_p \leq \epsilon\}$. It is worth noting though that several other notions of adversarial perturbations have been studied. These include rotations and translations [FF15; Eng+19b], smooth spatial deformations [Xia+18] and Wasserstein distance [WSK19].

5.2.2 A different perspective on robustness

Traditionally, adversarial robustness has been explored as a goal in the context of ML security and reliability [BR18]: to build models that are robust to manipulation in safety-critical settings. Here, we discuss how this framework could be a way to engineer the features models learn to rely on during training.

To motivate this, we first note that a model trained with the robust optimization framework (5.2) must be *invariant* to the set of perturbations Δ. This implies that the resulting robust model cannot depend on an input feature, even if it is predictive of the label, if this correlation can be flipped by a perturbation within the set Δ. Formally, this means that while standard (ERM-trained) models can rely on any ρ-useful features in the data (2.2), robust models can only rely on ones that are γ-robustly useful (2.3). Thus, one can intuitively think of robust optimization as a *feature prior*, where Δ specifies the family of features that one would like the model be agnostic to (e.g., small ℓ_p-norm perturbations). In general, incorporating such human-selected priors and invariances in this fashion has a long history in ML—convolutional layers, for instance, were introduced as a means of introducing an invariance to translations of the input [Fuk80].

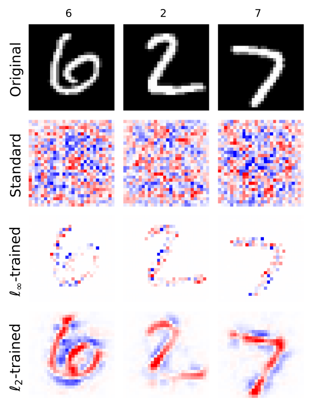
5.2.3 A closer look at robust models

In the context of deep networks, perfectly optimizing (5.2) is infeasible—after all these models are not concave/convex. That being said, approximations of this objective, for instance adversarial training [GSS15; Mad+18]) have arisen as practical ways of obtaining adversarially-robust models. Typically, this entails training the model against a projected gradient descent (PGD; a standard first-order optimization method) adversary.

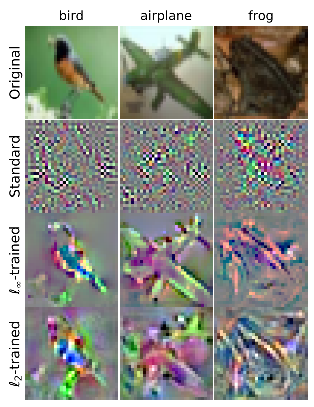
We now shift our focus to better understand the precise effect imposing such a feature prior—specifically invariance to small ℓ_2 perturbations—during training has on the resulting robust models. At a high level, our goal is to understand which input features they rely on to make their predictions, and contrast these the ones utilized by standard classifiers. To this end, we leverage a classic technique from ML interpretability—feature attribution [SVZ13; Smi+17; STY17]. In particular, we visualize the gradients of the loss with respect to individual features (pixels) of the input in Figure 5-3.

We observe that gradients for robust models align well with perceptually relevant features (such as edges) of the input image. In contrast, for standard models, these gradients have no coherent patterns and appear very noisy to humans. We want to emphasize that no preprocessing was applied to the gradients (other than scaling and clipping for visualization). On the other hand, extraction of interpretable information from the gradients of standard networks has only been possible with additional sophisticated techniques [SVZ13; Yos+15; OMS17].

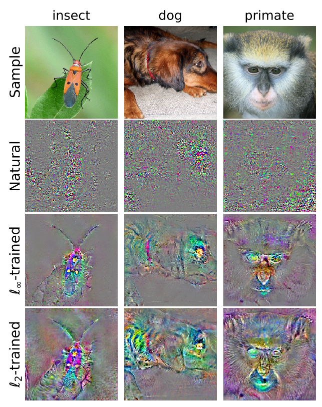
These gradients reveal a surprising side-effect of adversarial training. Namely, constraining models to be invariant to small ℓ_2 changes in the input causes them to depend on features that seemingly align better with human perception. In the rest of this chapter, we demonstrate that this effect goes beyond just qualitative



(a) MNIST



(b) CIFAR-10



(c) Restricted ImageNet

Figure 5-3: Visualization of the loss gradient with respect to input pixels. Recall that these gradients highlight the input features which affect the loss most strongly, and thus are important for the classifier's prediction. For MNIST, blue and red pixels denote positive and negative gradient regions respectively. For CIFAR-10 and ImageNet, we clip gradients to within ± 3 standard deviations of their mean and rescale them to lie in the $[0, 1]$ range.

differences in gradients between standard and robust models. In particular, robust models seem to learn more perceptually-aligned representations (Section 5.3), that are also useful for other downstream tasks beyond classification (Section 5.4).

5.3 Properties and Applications of Robust Representations

In Section 5.2, we saw that models trained via robust optimization seem to differ from their standard counterparts (obtained via ERM) in terms of the input features they rely on to make predictions. We now attempt to get a more precise understanding of these differences by directly examining the feature representations learned by standard and robust models. After all, beyond achieving high accuracy on a variety of tasks [KSH12; He+15b; CW08], a major appeal of deep learning is the ability to learn effective *feature representations* of data. Specifically, deep neural networks can be thought of as linear classifiers acting on *learned feature representations* (also known as *feature embeddings*). A major goal in representation learning is for these embeddings to encode high-level, interpretable features of a given input [GBC16; BCV13; Ben19].

Remark. Following standard convention, for a given deep network we define the *representation* $R(x) \in \mathbb{R}^k$ of a given input $x \in \mathbb{R}^d$ as the activations of the penultimate layer of the network (where usually $k \ll d$). The prediction of the network can thus be viewed as the output of a linear classifier on the representation $R(x)$. In what follows, we refer to "standard representations" as the representation functions induced by standard (non-robust) networks, trained with the objective (5.1). Analogously, "robust representations" refer to the representation functions induced by ℓ_2-adversarially robust networks (5.2). We refer to the *distance in representation space* between two inputs (x_1, x_2) as the ℓ_2 distance between their representations $(R(x_1), R(x_2))$, i.e., $\|R(x_1) - R(x_2)\|_2$.

5.3.1 Inverting representations

To probe the features learned by robust models more closely, we now directly examine their representations. Recall that a common aspiration in representation learning is to have that for any pixel-space input x, $R(x)$ is a vector encoding a set of "human-meaningful" concepts in x [Ben19; GBC16; BCV13]. Intuitively, these high-level features would make different classes linearly separable, allowing the subsequent linear classifier to attain high accuracy.

For standard models, running somewhat counter to this intuition, we find that it is straightforward to construct pairs of images with nearly identical representations yet drastically different content, as shown in Figures 5-4 and 5-5a. Finding such pairs turns out to be as simple as sampling two images $x_1, x_2 \sim \mathcal{D}$, then optimizing one of them to minimize distance in representation space to the other:

$$x_1' = x_1 + \arg\min_{\delta} \|R(x_1 + \delta) - R(x_2)\|_2. \tag{5.3}$$

This process can be seen as recovering an image that maps to the desired target representation, and hence is commonly referred to as *representation inversion* [DB16b; MV15; UVL17]. Ideally, one would expect the result of such inversion to match, in terms of high-level features, the target image being inverted.

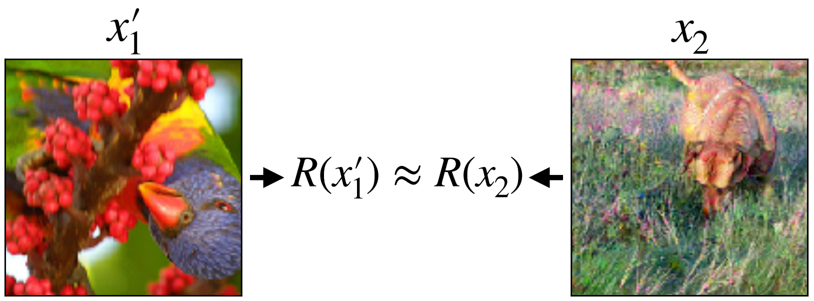
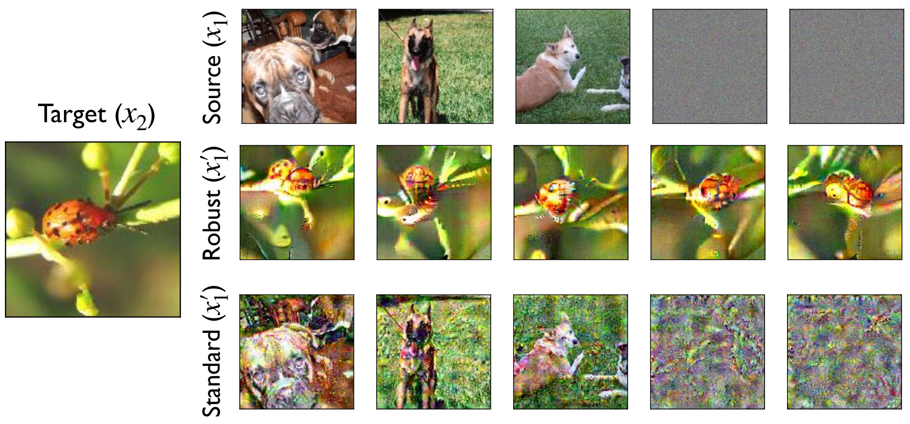


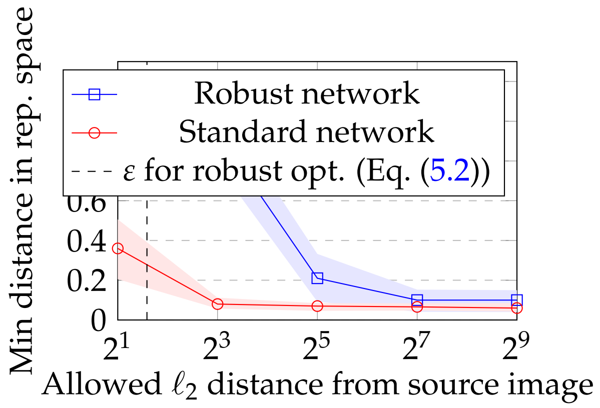
Figure 5-4: A limitation of standard neural network representations: it is straightforward to construct pairs of images (x_1', x_2) that appear completely different yet map to similar representations.

In contrast, we find that for robust models, minimizing (5.3) actually yields images that are actually *semantically similar* to the original (target) images whose representation is being matched. Moreover, this behavior is consistent across multiple samplings of the starting point (source image) x_1 (cf. Figure 5-5a).

Representation proximity seems to entail semantic similarity. In fact, the contrast between the invertibility of standard and robust representations is even stronger. To illustrate this, we will attempt to match the representation of a target image while staying close to the starting image of the optimization in pixel-wise ℓ_2-norm (this is equivalent to putting a norm bound on δ in objective (5.3)). With standard models, we can consistently get close to the target image in representation space, without moving far from the source image x_1. On the other hand, for robust models, we cannot get close to the target representation while staying close to the source image—this is illustrated quantitatively in Figure 5-5b. This indicates that for robust models, semantic similarity may in fact be necessary for representation similarity (and is not, for instance, merely an artifact of the local robustness induced by robust optimization). We also find that even when δ is highly constrained (i.e. when we are forced to stay very close to the source image and thus cannot match the representation of the target well), the solution to the inversion problem still displays some salient features of the target image (c.f. Figure 5-6). Both of these observations suggest that the representations of robust networks function much more like we would expect high-level feature representations to behave.



(a)



(b)

Figure 5-5: Inverting representations (5.3) via PGD learned by standard and robust models trained on the Restricted ImageNet dataset. (a) *Target* (x_2) & *Source* (x_1): random examples image from the test set; *Robust* and *Standard* (x_1'): result of inverting the representation of the target image starting from the corresponding source image for (*top*): a robust (adversarially trained) and (*bottom*): a standard model respectively. (b) Representation inversion with an ℓ_2-norm constraint around the source image. On the x-axis is the radius of the constraint set, and on the y-axis is the normalized distance in representation space between the minimizer of objective (5.3) within the constraint set and the target image, i.e., $y_i = \min_{\|\delta\|_2 \leq x_i} \|R(x+\delta) - R(x_{targ})\|_2 / \|R(x_{targ})\|_2$.

Inversion of out-of-distribution inputs. We find that the inversion properties uncovered above hold even for out-of-distribution inputs, demonstrating that robust representations capture *general* features as opposed to features only relevant for the specific classification task. In particular, we repeat the inversion experiment (simple minimization of distance in representation space) using images from

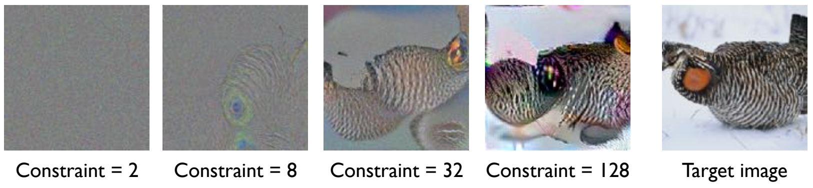


Figure 5-6: A visualization of the final solutions to the optimizing objective (5.3) with PGD when constraining the solution to lie in an ℓ_2 ball around the source image for an adversarially robust neural network.

classes not present in the original dataset used during training (Figure 5-7 right): the reconstructed images consistently resemble the targets.

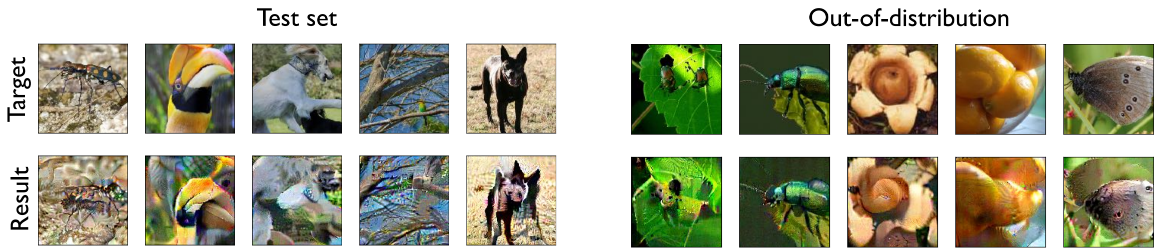


Figure 5-7: Robust representations yield semantically meaningful embeddings. *Target*: random images from the test set (col. 1-5) and from outside of the training distribution (6-10); *Result*: images obtained from optimizing inputs (using Gaussian noise as the source image) to minimize ℓ_2-distance to the representations of the corresponding image in the top row.

Interpolation between arbitrary inputs. Note that this A natural consequence of the "natural invertibility" property of robust representations is the ability to synthesize natural interpolations between any two inputs $x_1, x_2 \in \mathbb{R}^n$. In particular, given two images x_1 and x_2, we define the λ-*interpolate* between them as

$$x_\lambda = \min_x \| (\lambda \cdot R(x_1) + (1-\lambda) \cdot R(x_2)) - R(x) \|_2. \tag{5.4}$$

where, for a given λ, we find x_λ by solving (5.4) with projected gradient descent. Intuitively, this corresponds to linearly interpolating between the points in representation space and then finding a point in image space that has a similar representation. To construct a length-$(T+1)$ interpolation, we choose $\lambda = \{0, \frac{1}{T}, \frac{2}{T}, \ldots 1\}$. The resulting interpolations, shown in Figure 5-8, demonstrate that the λ-interpolates of robust representations correspond to a meaningful feature interpolation between images. (For standard models constructing meaningful interpolations is impossible due to the brittleness identified in Figure 5-4.)

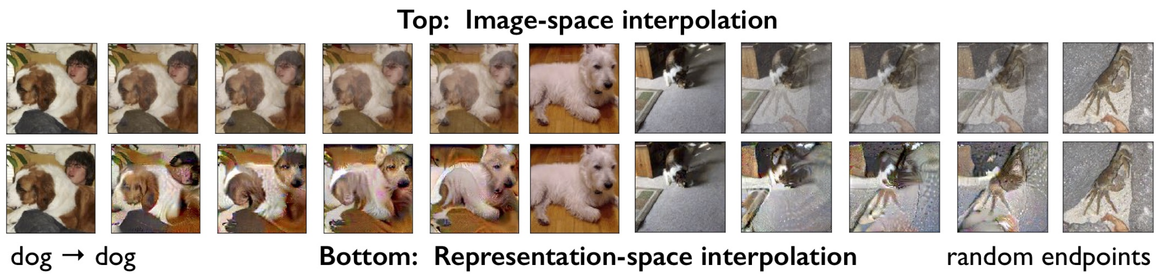

Top: Image-space interpolation
dog → dog
Bottom: Representation-space interpolation
random endpoints

Figure 5-8: Image interpolation using robust representations compared to their image-space counterparts. The former appear perceptually plausible while the latter exhibit ghosting artifacts. For pairs of images from the Restricted ImageNet test set, we solve (5.4) for λ varying between zero and one, i.e., we match linear interpolates in representation space.

We emphasize that linearly interpolating in robust representation space works for *any* two images. This generality is in contrast to interpolations induced by GANs (e.g. [RMC16; BDS19]), which can only interpolate between images generated by the generator. (Reconstructions of out-of-range images tend to be decipherable but rather different from the originals [Bau+19b].) It is worth noting that even for models with analytically invertible representations, interpolating in representation space does not yield semantic interpolations [JSO18].

5.3.2 Direct feature visualization

A common technique for visualizing and understanding the representation function $R(\cdot)$ of a given network is *optimization-based feature visualization* [OMS17], a process in which we maximize a specific feature (component) in the representation with respect to the input, in order to obtain insight into the role of the feature in classification. Concretely, given some $i \in [k]$ denoting a component of the representation vector, and a seed image x_0 (e.g., random noise), we use gradient descent

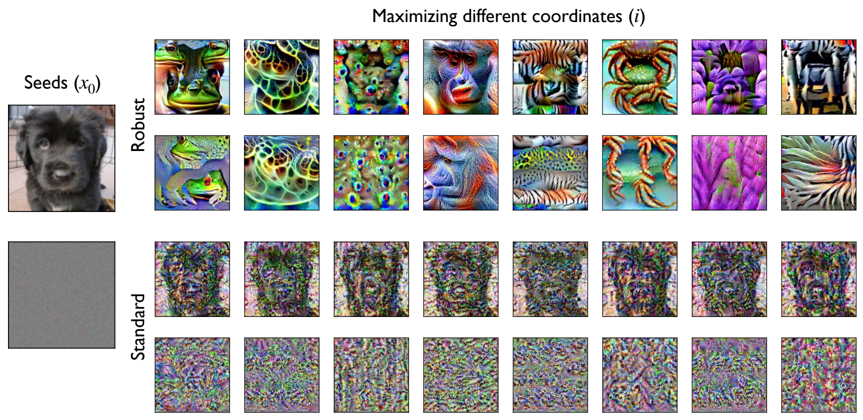


Figure 5-9: Correspondence between image-level patterns and activations learned by standard and robust models on the Restricted ImageNet dataset. Starting from randomly chosen seed inputs (noise/images), we use PGD to find inputs that (locally) maximally activate a given component of the representation vector. In the left column we have the seed inputs x_0 (selected *randomly*), and in subsequent columns we visualize the result of the optimization (5.5), i.e., x', for different activations, with each row starting from the same (far left) input x_0 for (*top*): a robust (adversarially trained) and (*bottom*): a standard model.

to find an input x' that maximally activates it, i.e., we solve:

$$x' = x_0 + \arg\max_{\delta} R(x_0 + \delta)_i \tag{5.5}$$

For standard networks, optimizing this objective typically yields unsatisfying results. While we *can* easily find images for which the i^{th} component of $R(\cdot)$ is large (and thus the optimization problem is tractable), these images tends to look meaningless to humans, often resembling the starting point of the optimization. Even when these images are non-trivial, they tend to contain abstract, hard-to-discern patterns (c.f. Figure 5-9 (bottom)).

For robust representations, however, we find that easily recognizable high-level features emerge from optimizing objective (5.5) directly, *without any regularization or post-processing*—cf. Figure 5-9. Furthermore, these concepts are not merely an artifact of our visualization process, as they consistently appear in the test-set inputs that most strongly activate their corresponding coordinates (Figure 5-10).

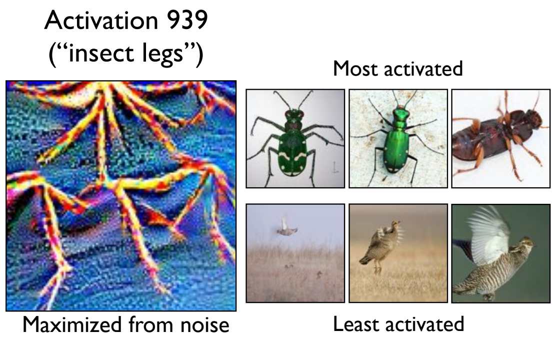


Figure 5-10: Maximizing inputs x' (found by solving (5.5) with x_0 being a gray image) and most or least activating images (from the test set) for a *random* activation of a robust model trained on the Restricted ImageNet dataset. For that activation, we plot the three images from the validation set that had the highest or lowest activation value sorted by the magnitude of the selected activation.

The limitations of regularization for visualization in standard networks. Given that directly optimizing objective (5.5) does not produce human-meaningful images, prior work on visualization usually tries to regularize objective (5.5) through a variety of methods. These methods include applying random transformations during the optimization process [MOT15; OMS17], restricting the space of possible solutions [NYC15; Ngu+16; Ngu+17], or post-processing the input or gradients [Oyg15; Tyk16]. While regularization does in general produce better results qualitatively, it comes with a few notable disadvantages that are well-recognized in the domain of feature visualization. First, when one introduces prior information about what makes images visually appealing into the optimization process, it becomes difficult to disentangle the effects of the actual model from the effect of the prior information introduced through regularization[1] (cf. Chapter 2). Furthermore, while adding regularization does improve the visual quality of the visualizations, the components of the representation still cannot be shown to correspond to any recognizable high-level feature [OMS17].

Natural consequence: feature manipulation The ability to directly visualize high-level, recognizable features reveals another application of robust representations,

[1]In fact, model explanations that enforce priors for purposes of visual appeal have been often found to have little to do with the data or the model itself [Ade+18].

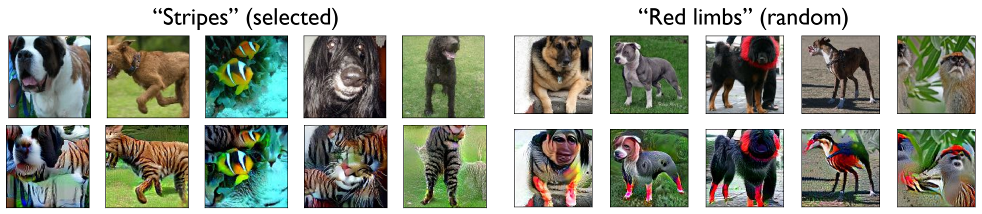


Figure 5-11: Visualization of the results from maximizing a chosen (left) and a *random* (right) representation coordinate starting from *random* images for the Restricted ImageNet dataset. In each figure, the top row has the initial images, and the bottom row has a feature added.

which we refer to as *feature manipulation*. Consider the visualization objective (5.5) shown in the previous section. Starting from some original image, optimizing this objective results in the corresponding feature being introduced in a continuous manner. It is hence possible to stop this process relatively early to ensure that the content of the original image is preserved. As a heuristic, we stop the optimization process as soon as the desired feature attains a larger value than all the other coordinates of the representation. We visualize the result of this process for a variety of input images in Figure 5-11, where *"stripes"* or *"red limbs"* are introduced seamlessly into images without any processing or regularization.

5.4 Image Synthesis via a (Single) Robust Classifier

Our findings so far indicate that robust models exhibit more human-aligned gradients and feature embeddings. Moreover, we saw that these models could be leveraged to manipulate the input image itself, by simply performing gradient descent on their predictions—cf. Figure 5-11 for an illustration. Another instance of this can be seen in Figure 5-12, where salient features of a class emerge when we simply maximize the probability of said class (targeted attacks) for a robust model.

We now demonstrate how this property of robust models makes them well-suited for various downstream *image synthesis tasks*—such as generation (Section 5.4.1), inpainting (Section 5.4.2), translation (Section 5.4.3) and superresolution (Section 5.4.4). Previously, these tasks have largely been tackled by training models in the generative adversarial network (GAN) framework [Goo+14; ISI17; Zhu+17; Yu+18;

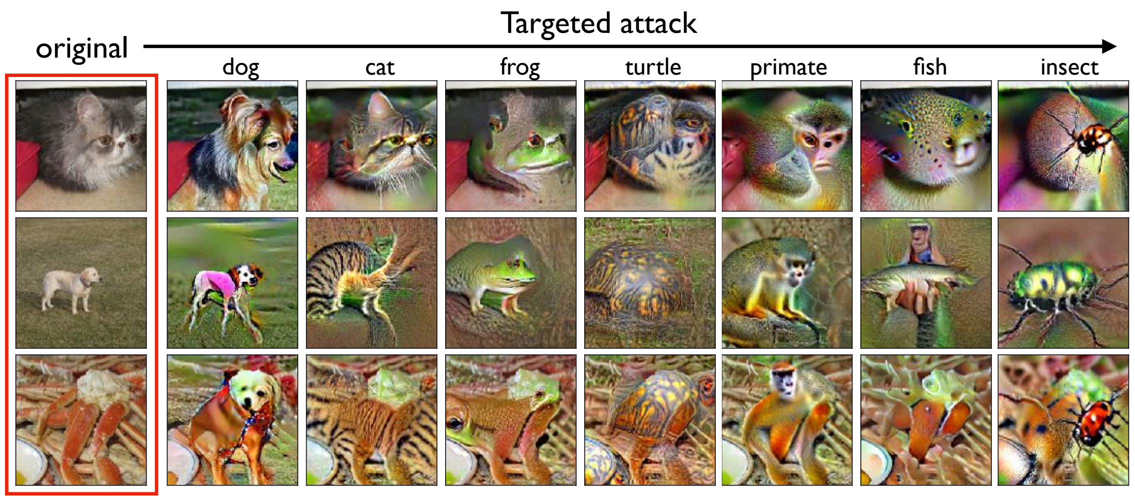


Figure 5-12: Maximizing class scores of a robustly trained classifier. For each image, we visualize the result of performing targeted projected gradient descent. The resulting images actually resemble samples of the target class.

BDS19], using priors obtained from generative models [Ngu+16; Ngu+17; UVL17; Yeh+17], or leveraging standard classifiers via task-specific methods [MOT15; Oyg15; Tyk16; GEB16]. However, it turns out that one can obtain competitive performance on these tasks using only robust (feed-forward) classifiers. In particular, our approach does not involve fine-grained tuning, highlighting the potential of robust classifiers as a versatile primitive for sophisticated vision tasks.

5.4.1 Realistic Image Generation

Synthesizing realistic samples for natural data domains (such as images) has been a long standing challenge in computer vision. Given a set of example inputs, we would like to learn a model that can produce novel perceptually-plausible inputs. The development of deep learning-based methods such as autoregressive models [HS97; Gra13; VKK16], auto-encoders [Vin+10; KW15] and flow-based models [DKB14; RM15; DSB17; KD18] has led to significant progress in this domain. More recently, advancements in GANs [Goo+14] have made it possible to generate high-quality images for challenging datasets [Zha+18a; Kar+18; BDS19]. Many of these methods, however, can be computationally intensive, and tricky to train.

In contrast, we demonstrate that robust classifiers, without any special training or auxiliary networks, can be a powerful tool for synthesizing realistic natural images. At a high level, our generation procedure is based on maximizing the class score of the desired class using a robust model. The purpose of this maximization is to add relevant and semantically meaningful features of that class to a given input image. This approach has been previously used on standard models to perform class visualization—synthesizing prototypical inputs of each class—in combination with domain-specific input priors (either hand-crafted [NYC15] and learned [Ngu+16; Ngu+17]) or regularizers [SVZ13; MOT15; Oyg15; Tyk16].

As this process is deterministic, generating a diverse set of samples requires a random seed as the starting point for class maximization. Formally, to generate a sample of class y, we sample a seed and minimize the loss $\mathcal{L}$ of label y

$$x = \underset{\|x'-x_0\|_2 \leq \varepsilon}{\arg\min}\ \mathcal{L}(x', y), \qquad x_0 \sim \mathcal{G}_y,$$

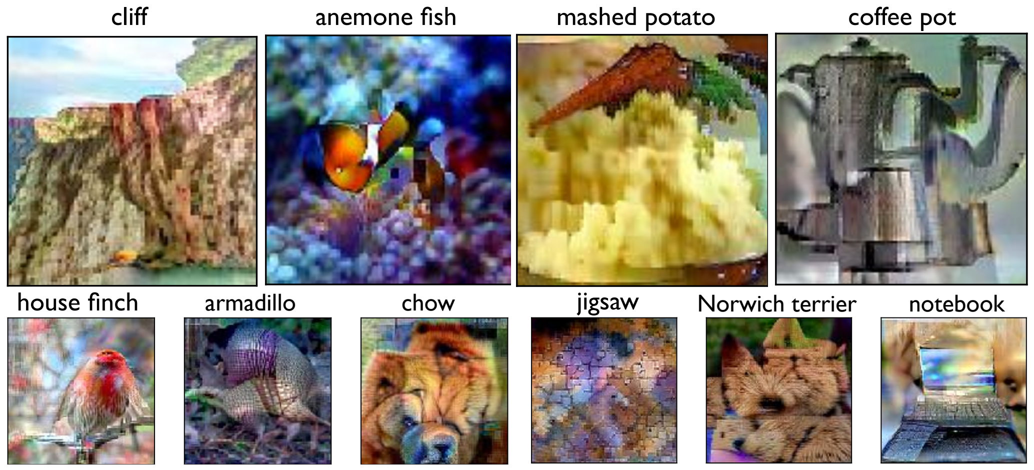
for some class-conditional seed distribution $\mathcal{G}_y$, using PGD. Ideally, samples from $\mathcal{G}_y$ should be diverse and statistically similar to the data distribution. Here, we use a simple (but already sufficient) choice for $\mathcal{G}_y$—a multivariate normal distribution fit to the empirical class-conditional distribution and $\mathcal{D}_y$ is the distribution of natural inputs conditioned on the label y.

This approach enables us to perform *conditional* image synthesis given any target class—cf. Figure 5-13. The resulting images are diverse and realistic, despite the fact that they are generated using targeted PGD on off-the-shelf robust models without any additional optimizations. [2]

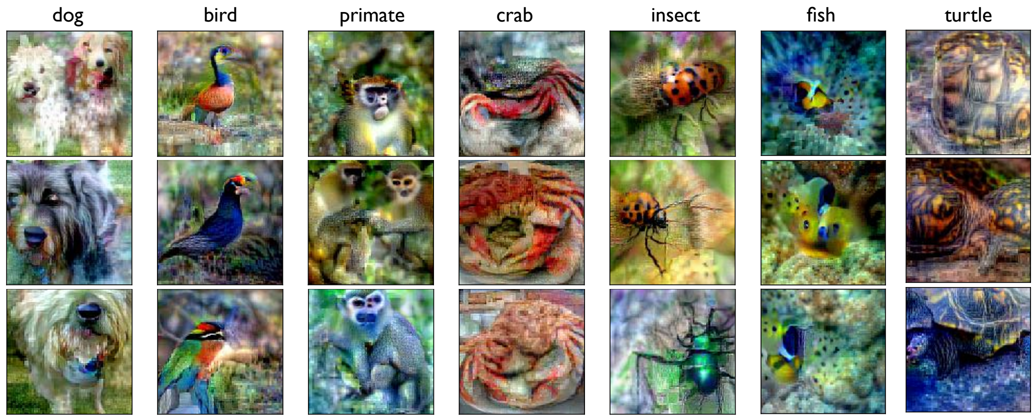
5.4.2 Inpainting

Image inpainting is the task of recovering images with large corrupted regions [EL99; Ber+00; HE07]. Given an image x, corrupted in a region corresponding to a binary mask $m \in \{0, 1\}^d$, the goal of inpainting is to recover the missing pixels in a man-

[2]Interestingly, the robust model used to generate these ImageNet samples is only 45% accurate, yet has a sufficiently rich representation to synthesize semantic features for 1000 classes.



(a)

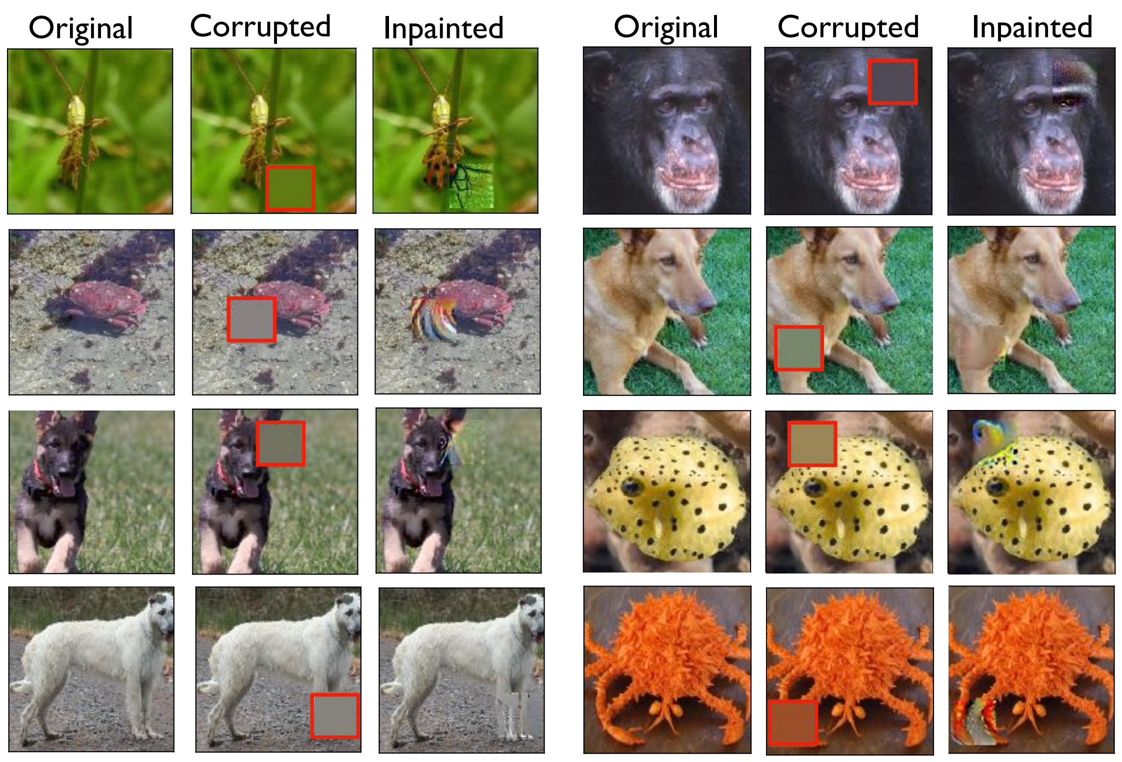


(b)

Figure 5-13: *Random* samples (of resolution 224×224) produced using a robust classifier. We show samples from several (random) classes of the (a) ImageNet and (b) restricted ImageNet datasets.

ner that is perceptually plausible with respect to the rest of the image. We find that simple feed-forward classifiers, when robustly trained, can be a powerful tool for such image reconstruction tasks.

From our perspective, the goal is to use robust models to restore missing features of the image. To this end, we will optimize the image to maximize the score of the underlying true class, while also forcing it to be consistent with the original in the uncorrupted regions. Concretely, given a robust classifier trained on



(a) *random* samples (b) select samples

Figure 5-14: Image inpainting using robust models – *left:* original, *middle:* corrupted and *right:* inpainted samples. We use PGD to maximize the predicted class score while penalizing changes to the uncorrupted image regions.

uncorrupted data, and a corrupted image x with label y, we solve

$$x_I = \arg\min_{x'} \mathcal{L}(x', y) + \lambda ||(x - x') \odot (1 - m)||_2 \tag{5.6}$$

where $\mathcal{L}$ is the cross-entropy loss, $\odot$ denotes element-wise multiplication, and λ is an appropriately chosen constant. Note that while we require knowing the underlying label y for the input, it can typically be accurately predicted by the classifier itself given the corrupted image.

In Figure 5-14, we show sample reconstructions obtained by optimizing (5.6) using PGD. We can observe that these reconstructions look remarkably similar to the uncorrupted images in terms of semantic content. Interestingly, even when this approach fails (reconstructions differ from the original), the resulting images do tend to be perceptually plausible to a human, as shown in Figure 5-15.

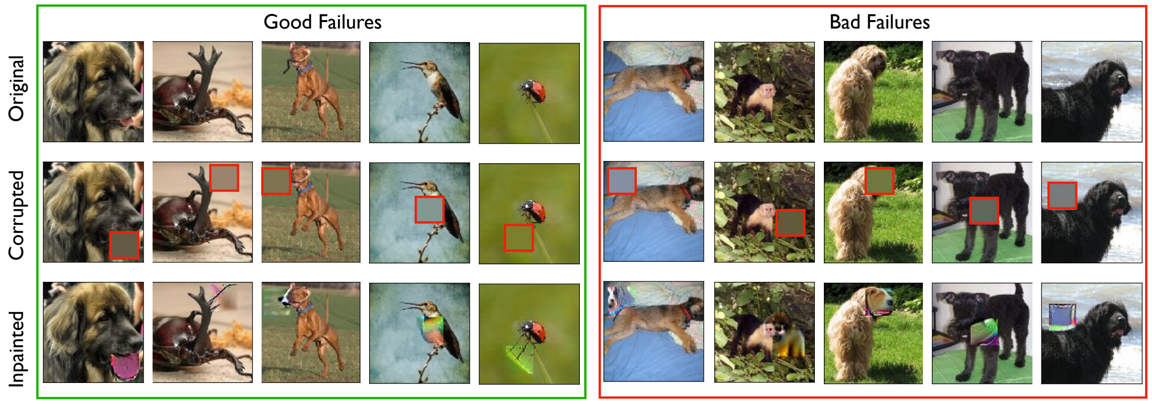


Figure 5-15: Failure cases for image inpainting using robust models – *top:* original, *middle:* corrupted and *bottom:* inpainted samples. The failure modes can be categorized into "good" failures – where the infilled region is semantically consistent with the rest of the image but differs from the original; and "bad" failures – where the inpainting is clearly erroneous to a human.

5.4.3 Image-to-Image Translation

In this section, we demonstrate that robust classifiers give rise to a new methodology for performing *image-to-image translation*: where the goal is to translate an image from a source to a target domain in a semantic manner [Her+01]. The key is to (robustly) train a classifier to distinguish between the source and target domain. Conceptually, such a classifier will extract salient characteristics of each domain in order to make accurate predictions. We can then translate an input from the source domain by directly maximizing the predicted score of the target domain.

In Figure 5-16, we provide sample translations produced by our approach using robust models—each trained only on the source and target domains for the Horse $\leftrightarrow$ Zebra, Apple $\leftrightarrow$ Orange, and Summer $\leftrightarrow$ Winter datasets [Zhu+17] respectively. In general, we find that this procedure yields meaningful translations by directly modifying characteristics of the image that are strongly tied to the corresponding domain (e.g., color, texture, stripes).

Note that, in order to manipulate such features, the model must have learned them in the first place—for example, we want models to distinguish between horses and zebras based on salient features such as stripes. For overly simple tasks, models might extract little salient information (e.g., by relying on backgrounds instead

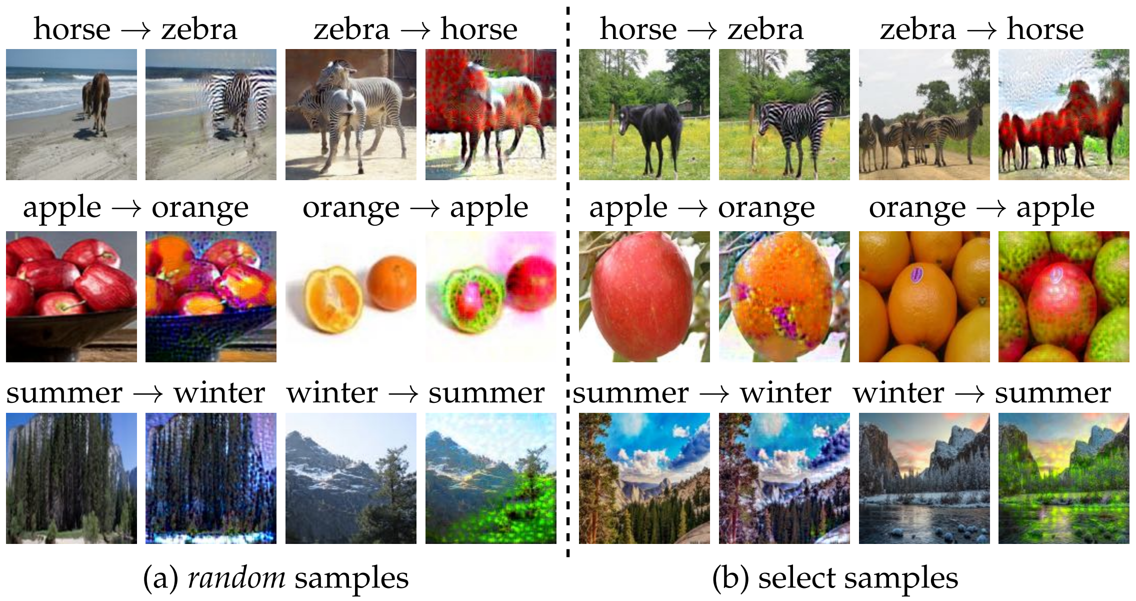


Figure 5-16: Image-to-image translation on the Horse ↔ Zebra, Apple ↔ Orange, and Summer ↔ Winter datasets [Zhu+17] using PGD on the input of an ℓ_2-robust model trained on that dataset.

of objects) in which case our approach would not lead to meaningful translations. Nevertheless, this not a fundamental barrier and can be addressed by training on richer, more challenging datasets. From this perspective, scaling to larger datasets (which can be difficult for state-of-the-art methods such as GANs) is actually easy and advantageous for our approach.

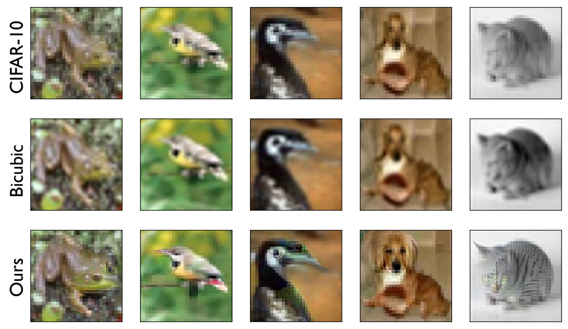
Unpaired datasets. Datasets for translation tasks often comprise source-target domain pairs [Iso+17]. In contrast, our method operates in the *unpaired* setting, where samples from the source and target domain are provided without an explicit pairing [Zhu+17]. This is due to the fact that our method only requires a classifier capable of distinguishing between the source and target domains.

5.4.4 Super-Resolution

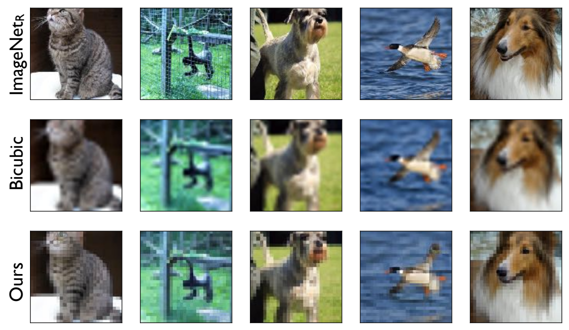
Super-resolution refers to the task of recovering high-resolution images given their low-resolution version [DFE07; BSH12]. While this goal is underspecified, our aim is to produce an image that is consistent with the input and plausible to a human. In order to adapt our framework to this problem, we cast super-resolution as the task of accentuating the salient features of low-resolution images. This can be achieved by maximizing the score predicted by a robust classifier (trained on the original high-resolution dataset) for the underlying class. At the same time, to ensure that the structure and high-level content is preserved, we penalize large deviations from the original low-resolution image. Formally, given a robust classifier and a low-resolution image x_L belonging to class y, we use PGD to solve

$$\hat{x}_H = \underset{||x' - \uparrow(x_L)|| < \varepsilon}{\arg\min} \; \mathcal{L}(x', y) \tag{5.7}$$

where $\uparrow(\cdot)$ denotes the up-sampling operation based on nearest neighbors, and ε is a small constant.



(a) 7x on CIFAR-10



(b) 8x on restricted ImageNet

Figure 5-17: Comparing approaches for super-resolution. *Top: random* samples from the test set; *middle:* upsampling using bicubic interpolation; and *bottom:* super-resolution using robust models.

We use this approach to upsample *random* 32 × 32 CIFAR-10 images to full ImageNet size (224 × 224)—cf. Figure 5-17a. For comparison, we also show upsampled images obtained from bicubic interpolation. In Figure 5-17b, we visualize the results for super-resolution on *random* 8-fold down-sampled images from the restricted ImageNet dataset. Since in the latter case we have access to ground truth high-resolution images (actual dataset samples), we can compute the Peak Signal-to-Noise Ratio (PSNR) of the reconstructions. Over the Restricted ImageNet test set, our approach yields a PSNR of 21.53 (95% CI [21.49, 21.58]) compared to 21.30 (95% CI [21.25, 21.35]) from bicubic interpolation. In general, our approach produces high-resolution samples that are substantially sharper, particularly in regions of the image that contain salient class information.

Note that the pixelation of the resulting images can be attributed to using a very crude upsampling of the original, low-resolution image as a starting point for our optimization. Combining this method with a more sophisticated initialization scheme (e.g., bicubic interpolation) is likely to yield better overall results.

5.5 Broader Implications: Robustness Beyond Security

Before concluding our discussion, we provide a brief overview of other contexts in which robust models, and their "better" feature representations, may be desirable. We also discuss how priors beyond just ℓ_p perturbations can be imposed via the robust optimization framework.

5.5.1 Towards interpretability by-design

Recall from Chapter 2 that there are some fundamental roadblocks in interpreting standard ERM-trained classifiers. In particular, these models tend to rely on input features that are imperceptible or unintuitive to make their predictions. Consequently, directly applying interpretability methods to these models tends to yield explanations that are hard to parse (e.g., Figure 5-3). While one can improve the quality of interpretations via post-processing [SVZ13; Yos+15; OMS17], this could suppress input features that are key in determining the model's prediction. We also saw models trained via the robust optimization framework rely on a different set than their standard counterparts—seemingly ones that are more-aligned with human perception. This prompts the question: is robustness a path[3] to building models that are *interpretable by-design?*

The gradients of robust models (cf. Section 5.3) provide some hints that this

[3]In Chapter 3 we discussed how modifying networks to be *sparse* is another viable approach.

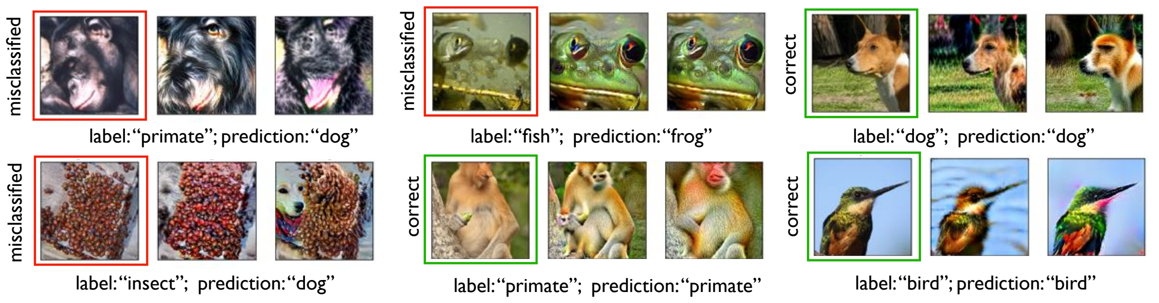

misclassified
label:"primate"; prediction:"dog"
misclassified
label:"insect"; prediction:"dog"
misclassified
label:"fish"; prediction:"frog"
correct
label:"primate"; prediction:"primate"
correct
label:"dog"; prediction:"dog"
correct
label:"bird"; prediction:"bird"

Figure 5-18: Accentuating the highest-weight features for correctly classified (right) and incorrectly (left) classified images.

might be the case. We now explore this question through the lens of another common interpretation technique—*interpretation via perturbation* [FV17]. Here, for an image x, the model explanation consists of a similar image x' (or several such images) along with its (or their) classification(s). Typically, the qualitative differences between x and x' are used to assess the input features used by the model. This explanation technique is akin to counterfactuals from causal inference [Pea10].

In applying this technique to standard models, we once again run into the aforementioned roadblock: one must either apply priors at interpretation time (e.g. using regularization), or else end up with unintelligible explanations [FV17; Cha+19; DG17]. The picture for robust models, however, looks quite different—cf. Figure 5-18. Here, for a given input image x, we find the three most activated components of the representation layer (appropriately weighted and scaled). We then construct counterfactuals by magnifying these coordinates in the input by performing the feature addition objective described in Section 5.3.2.

In Figure 5-18, accentuating the highest-weight features reveals a natural transformation from the ear of a monkey to the eye of a dog (top), from negative space to the face of a dog (middle), and from the heads of two different fish to the two eyes of a single frog, with a reed transforming into a mouth (bottom). These transformations hint at the most sensitive directions of robust models on specific (misclassified) inputs, thus could provide insight into model decisions or errors. These findings suggest that robustness could indeed be an avenue to build deep networks that are inherently more interpretable. We view exploring this direction further—e.g., via human-in-the-loop studies to evaluate these explanations—as an interesting direction for future work [RD18].

5.5.2 Other downstream tasks

As discussed previously, one of the major appeals of deep networks—aside from their impressive performance on benchmarks—is that they are able to learn meaningful high-level representations of the data. Indeed, this belief is what drives the use of learned deep feature representations in transfer learning [Gir+14; Don+14], and similarity metrics such as VGG distance [DB16a; JAF16; Zha+18b]. From this viewpoint, robust models could be particularly advantageous: after all, their representations overcome many shortcomings exhibited by standard ones? But how far do these benefits of robust representations extend?

Follow up work has shown that robust models also perform better on other downstream tasks, outside of image synthesis and manipulation, than their standard counterparts. These include: (i) transfer learning [Sal+20; Utr+20], (ii) zero-shot learning [Agg+20], (iii) weakly-supervised object localization [Agg+20] and (iv) better robustness to data corruptions [For+19; Kan+19; Tao+20]. These demonstrations indicate that robust models may indeed be a step towards learning more general data representations—that transfer between different tasks and domains.

5.5.3 Diverse feature priors

Finally, our analysis so far has been limited largely to ℓ_p feature priors. However, in practice, one might also want to enforce other forms of invariance into models—e.g., to backgrounds, lighting, affine transformations, protected attributes. While some of these notions have been formalized [FF15; Xia+18; Eng+19b; WSK19], a significant design space still remains to be explored. In particular, designing methods to translate high-level features (e.g., invariance to "people") into simulatable perturbations and extending the robustness framework to handle these more complex perturbations could be valuable directions for future work.

Chapter 6

Post-hoc Model Editing

In this chapter, we introduce a toolkit to *directly* rewrite a model's prediction rules *post-hoc*. We have seen so far that in the process of discovering predictive correlations in the data, models often latch features that are not entirely meaningful [TE11; BVP18; Ily+19; SSF19; ASF20; Xia+20; BVA20; Gei+20; WSM21]. In particular, such correlations can arise from biases in the training data, and thus may not reflect the real world—e.g., a pasture, while typical, need not be present in every cow image [BVP18]. In Chapter 5, we discussed how one might train models that do not depend on specific families of features—e.g., ones that are sensitive to small ℓ_p changes in the input. However, performing such train-time interventions requires a priori knowledge of the invariances we would like our models to satisfy.

This might not always be feasible in practice—for instance, some of the undesirable correlations learned by a model may only come to light upon deployment. This raises the question: *How can we modify, post-hoc, the way in which a given model makes its predictions?* The canonical approach doing so is to intervene at the data level: collect additional input-label pairs that capture the desired behavior and use them to further train the model. Unfortunately, collecting such data can be challenging: how do we get cows to pose for us in a variety of environments? Furthermore, data collection is ultimately a very indirect way of specifying the intended behavior of a model. Even when the data has been carefully curated to reflect a given real-world task, models still end up learning unintended prediction

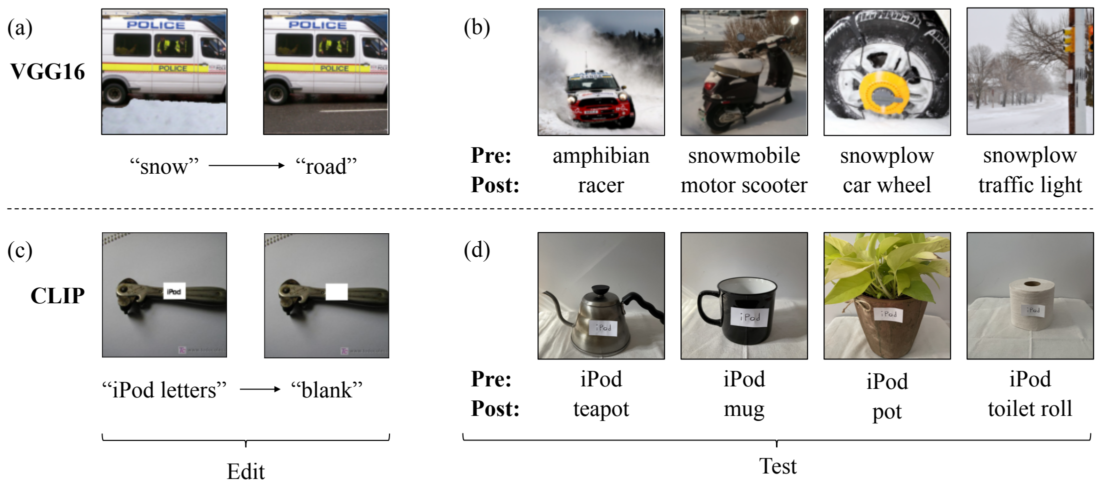

(a)
VGG16
"snow" → "road"
(b)
Pre: amphibian snowmobile snowplow snowplow
Post: racer motor scooter car wheel traffic light
(c)
CLIP
"iPod letters" → "blank"
(d)
Pre: iPod iPod iPod iPod
Post: teapot mug pot toilet roll
Edit
Test

Figure 6-1: Editing prediction rules in pre-trained classifiers using a *single* exemplar. (a) We edit a VGG-16 ImageNet classifier to map the representation of the concept "snow" to that of "asphalt road". (b) This edit corrects systematic classification errors on snowy scenes for various ImageNet classes. (c) We edit an OpenAI CLIP model such that the text "iPod" maps to a blank area. (d) This change makes the model robust to the typographic attacks from Goh et al. [Goh+21].

rules from it [Pon+06; TE11; Tsi+20; Bey+20]. Thus, our focus in this chapter is on directly modifying the prediction rules learned by a model instead. This chapter is structured as follows: Section 6.1 outlines our contributions, Sections 6.2-6.3 detail and evaluate our model editing toolkit, Section 6.4 presents relevant related work and Section 6.5 discusses the broader implications of our findings.

6.1 Summary of Our Results

We build on the recent work of Bau et al. [Bau+20a] to develop a method for modifying a model's prediction rules in a targeted and controlled manner, with essentially *no* additional data collection. Fundamentally, our method changes the behavior of the model on occurrences of a particular concept beyond just the specific classes it encounters during the edit. We demonstrate the effectiveness of this approach in two different settings.

Synthetic transformations (Section 6.3.2). First, we apply our approach to correct model sensitivities to concept-level transformations identified using our debugging pipeline in Section 3.2. For instance, we can modify a pre-trained ImageNet classifier to correctly recognize vehicles with unusual "wooden wheels" by teaching it to treat a "wooden wheel" in a "car" image as it would a regular wheel—see Figure 6-2 for an illustration. In contrast, the typical approach of adapting models via fine-tuning consistently fails to generalize in the same way.

Real-world demonstrations (Section 6.3.3). Then, we apply our approach to scenarios reflecting concept-level transformations that could arise in the real world. First, we focus on adapting an ImageNet classifier to a new environment, namely, recognizing actual photographs of vehicles on snowy roads. Second, we consider the recent "typographic attack" of Goh et al. [Goh+21] on a zero-shot CLIP [Rad+21] classifier: attaching a piece of paper with "iPod" written on it to various household items causes them to be incorrectly classified as "iPod." We find that our approach significantly improves model performance in both settings, using only a *single synthetic* example—cf. Figure 6-1.

Overall, our findings demonstrate that there is a rich design space around *directly* rewriting a model's prediction rules and that this approach can lead to a powerful model debugging toolkit.

6.2 A Toolkit for Editing Prediction Rules

As we saw in Section 3.2, models often pick up context-specific correlations in the data—e.g., using the presence of "road" or a "wheel" to predict "car". Such prediction rules could hinder models when they encounter novel environments (e.g., snow-covered roads), and confusing or adversarial test conditions (e.g., cars with wooden wheels). As a result, after identifying undesirable prediction rules, a user might want to modify them before deploying their model in the wild.

The canonical approach to modify a classifier post hoc is to collect additional data that captures the desired deployment scenario, and use it to retrain the model. However, even setting aside the challenges of collecting such data, it is not obvious a priori exactly what effect such adaptation (e.g., via fine-tuning) will have on the model's prediction rules. For instance, if we fine-tune our model on "cars" with wooden wheels, will it now recognize "scooters" or "trucks" with such wheels?

Our goal here is to instead develop a more direct way to modify a model's behavior by rewriting its prediction rules in a targeted manner. Ideally, in our previous example, we would like to enable the classifier to recognize vehicles with wooden wheels by simply teaching it to treat *any* wooden wheel as it would a standard one. Before describing our approach, we first provide a brief overview of recent work by Bau et al. [Bau+20a] which forms the basis for our approach.

6.2.1 Background: Rewriting generative models

Bau et al. [Bau+20a] developed an approach for rewriting a deep generative model, enabling a user to replace all occurrences of one selected object (say, "dome") in the generated images with another (say, "tree"), without changing the model's behavior in other contexts. The approach is built on the observation that, using a handful of example images, we can identify vectors in the model's representation space that encode a specific high-level concept [Kim+18; Bau+20a]. Leveraging this, Bau et al. [Bau+20a] treat each layer of the model as an *associative memory*, which maps the concept vector at each spatial location in its input (which we will refer to as the *key*) to a concept vector in its output (which we will call the *value*).

In the simplest case, a linear layer with weights $W \in \mathbb{R}^{mxn}$ transforms the key $k \in \mathbb{R}^n$ to the value $v \in \mathbb{R}^m$. Observe that in this setting, one could perform a rewrite by modifying the layer weights from W to W' so that $v^* = W'k^*$, where k^* corresponds to the old concept that we want to replace, and v^* the new concept. For instance, if we wanted to replace "domes" with "trees" in the generated images, we would modify the layer so that the key k^* for "dome" maps to the value v^* for "tree". Consequently, when this value is fed into the downstream layers of the network it would result in a *tree* in the final image. Crucially, this update should change the model's behavior to *every* instance of the concept encoded in k^*—i.e., all "domes" in the images should now be "trees".

To extend this approach to typical deep generative models, two challenges remain: (1) handling non-linear layers, and (2) ensuring that the edit doesn't significantly hurt model behavior in other scenarios. With these considerations in mind, Bau et al. [Bau+20a] propose making the following rank-one updates to the parameters W of an arbitrary non-linear layer f:

$$\min_{\Lambda} \quad \sum_{(i,j)\in S} \left\| v^*_{ij} - f(k^*_{ij}; W') \right\| \tag{6.1}$$

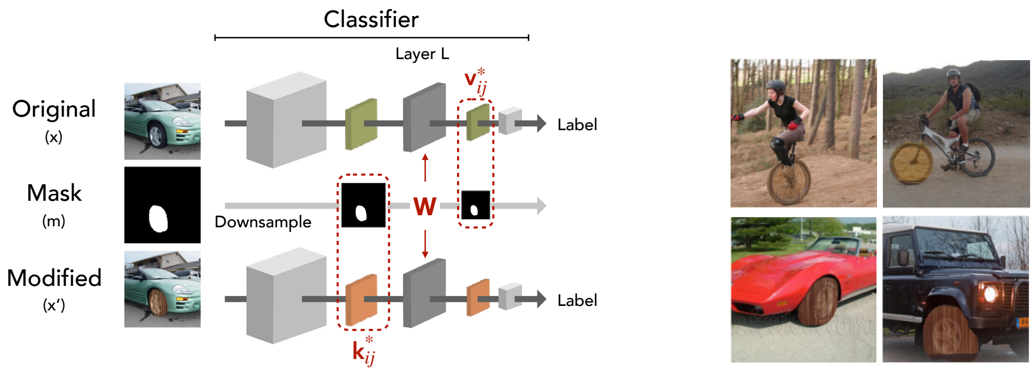
$$\text{s.t.} \quad W' = W + \Lambda (C^{-1} d)^\top. \tag{6.2}$$

Here, S denotes the set of spatial locations in representation space corresponding to the concept of interest, d is the top eigenvector of the keys k^*_{ij} corresponding to locations $(i,j) \in S$ and $C = \sum_d k_d {k_d}^\top$ captures the second-order statistics for other keys k_d. In general, the keys and values in (6.1) can be obtained not just from various spatial locations in the representations of a single image, but over multiple images containing the concept as well. Intuitively, the goal of this update is to minimally modify the layer parameters to rewrite the desired key-value mapping. We refer the reader to Bau et al. [Bau+20a] for additional details.

6.2.2 Editing classifiers

We now shift our attention to the focus of this work: editing classifiers. To describe our approach, we will use as a running example the task of enabling classifiers to recognize vehicles with "wooden wheels". Specifically, let us start with a single image x from the dataset, say, from class "car", that contains the concept "wheel". Moreover, let the location of the "wheel" in the image be denoted by a binary mask m.[1] We first create a transformed image x' of a "car" with a "wooden wheel"—i.e., by manually replacing the wheel, or by applying the counterfactual-generation procedure described in Section 3.2. Next, we would like to apply the approach described above to modify a chosen (potentially non-linear) layer L of the network

[1]Such a mask can either be obtained manually or automatically (cf. Section 3.2).



(a) Edit (b) Test

Figure 6-2: Overview of our pipeline for directly editing the prediction-rules of a classifier. The edit in (a) seeks to modify the network to perceive wooden wheels as standard ones, using a small set of exemplar images (say from class "car"). To achieve this, we to first obtain the keys k^*_{ij} corresponding to the new concept (here, "wooden wheel"), and the values v^*_{ij} corresponding to the original concept (here, "standard wheel") in the input and output representation space of a layer L respectively. We then update the weights W of the layer to enforce this new key-value association (6.1). (b) To test our editing technique, we measure the improvement in model performance on test instances (from any class) containing the new concept—in this case, example images of vehicles with "wooden wheels".

to rewrite a suitable key-value association. But, we need to first determine what the relevant keys and values are.

Intuitively, we want the classifier to perceive a "wooden wheel" in the image as it would a standard one. To achieve this, we must map the keys for wooden wheels to the value corresponding to their standard counterparts. Thus, the keys that we want to rewrite correspond to the network's representation of the concept in the *transformed* image directly *before* layer L. Similarly, the values that we want to map these keys to correspond to the network's representation of the concept in the *original* images directly *after* layer L. (The relevant spatial regions in the representation space are simply determined by downsampling the mask to the appropriate dimensions.) Finally, the actual edit is performed by feeding the resulting key-value pairs into the optimization problem (6.2) to determine the updated layer weights W'—see Figure 6-2 for an illustration of the overall process.

6.3 Does Editing Generalize?

We now demonstrate how the methodology described above can be applied to edit vision classifiers—specifically, VGG [SZ15] and ResNet [He+15a] models trained on the ImageNet [Den+09; Rus+15] and Places-365 [Zho+17] datasets.

6.3.1 Evaluation Setup

We start by describing our evaluation setup, using as a running example the task of editing the model to recognize vehicles with "wooden wheels" akin to their standard counterparts.

Train-test split. To edit the model with respect to a particular concept-style pair (say "wheel"-"wooden"), our training set comprises N *exemplars*, i.e., pairs of original and transformed images, (x, x') (cf. Section 6.2.2) that belong to a single (randomly-chosen) target class in the dataset (e.g., "car"). All other transformed images containing the concept, including those belonging to classes other than the target one, are used for validation and testing (30-70 split). We select the best hyperparameters—including the choice of the layer to modify—based on the validation set performance (cf. Appendix E.1.4).

Baselines. We comparing editing to the canonical fine-tuning approach, i.e., directly minimizing the cross-entropy loss on the new data (in this case the transformed images) with respect to the target label. We consider two variants of fine-tunin: (i) *local* fine-tuning, where we only train the weights of a single layer L (similar to our editing approach); and (ii) *global* fine-tuning, where we also train all other layers between L and the output of the model. It is also worth noting that unlike fine-tuning, our editing approach does not utilize class labels in any way.

Evaluation criteria. To evaluate the effect of the modification, we need to measure the change in model performance on the transformed examples (e.g., vehicles with "wooden wheel" in Figure 6-2b). We will only focus on the subset of examples

D that were correctly classified before the transformation, since we cannot expect to correct mistakes that do not stem from the transformation itself. Concretely, we will measure the change in the number of misclassifications made by the model on the transformed examples:

$$\frac{N_{pre}(D) - N_{post}(D)}{N_{pre}(D)} \tag{6.3}$$

where $N_{pre/post}(D)$ denotes the number of transformed examples misclassified by the model before and after the modification, respectively. Note that this metric can range from 100% when rewriting leads to perfect classification on the transformed examples, to even a negative value when the rewriting process causes more mistakes that it fixes.

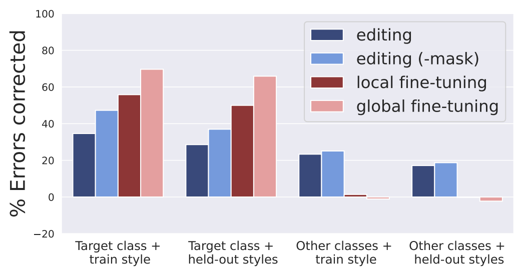
To quantify the effect of the modification on overall model behavior, we also measure the change in its (standard) test set performance. Since we are interested in rewrites that do not significantly hurt the overall model performance, we only consider hyperparameters that do not cause a large accuracy drop ($\leq$0.25% unless otherwise specified). We found that the exact accuracy threshold did not have significant impact on the results.

6.3.2 Synthetic Transformations

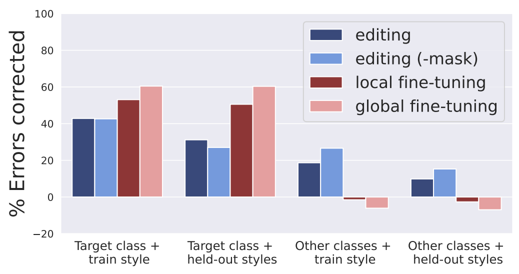
Our first use case will be improving model generalization to the concept-level transformations from Section 3.2. Recall that our analysis pinpointed a number of concepts in the data that when transformed—even in fairly natural ways—caused the model performance to drop significantly We will now use our editing approach to correct these sensitivities.[2]

Crucially, note that we want the resulting prediction-rule rewrites to generalize. That is, if we modify the way that our model treats a specific concept, we want

[2]Note that some of these cases might not be suitable for editing, i.e., when the transformed concept is critical for recognizing the label of an input image (e.g., transforming concept "dog" in images of class "poodle") . We thus manually exclude such concept-class pairs from our analysis—cf. Appendix E.1.4.



(a) ImageNet-trained VGG16 (concepts from COCO)



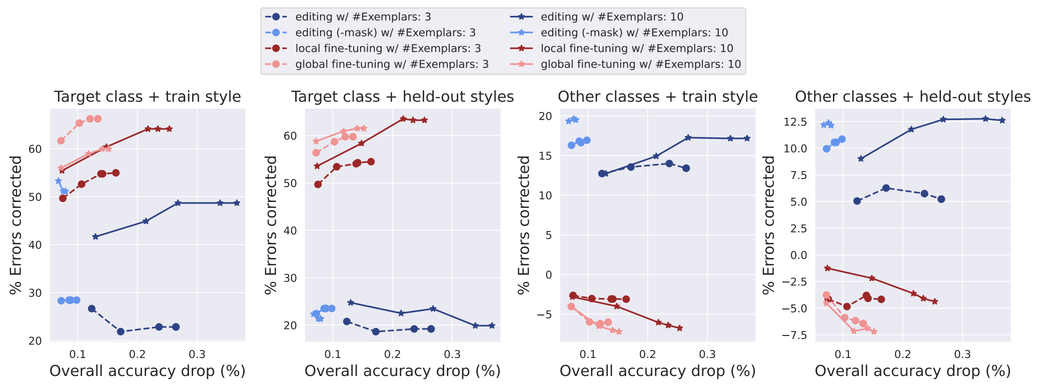
(b) Places-trained ResNet-18 (concepts from LVIS)

Figure 6-3: Editing vs. fine-tuning, averaged over concept-style pairs. Both methods (and their variants) are fairly successful at correcting misclassifications on the target class (examples of which are used to perform the modification). This holds even when the transformation applied during testing is different from the one present in the train exemplars (e.g., a different texture of "wood"). However, crucially, only the improvements induced by editing generalize to other classes where the transformed concept is present, while fine-tuning fails in this setting—typically, causing more errors than it fixes. See Appendix Figures E-1-E-4 for other experimental settings.

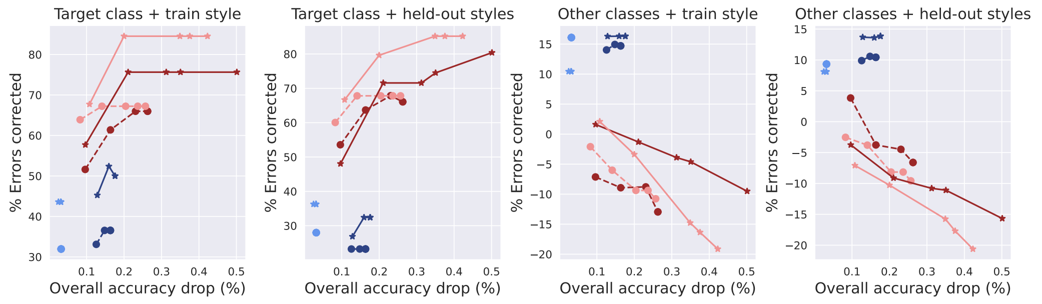
this modification to apply to *every* occurrence of that concept. For instance, if we edit a model to enforce that "wooden wheels" should be treated the same as regular "wheels" in the context of "car" images, we want the model to do the same when encountering other vehicles with "wooden wheels". Thus, while analyzing model performance, we treat inputs belonging to the (target) class used to perform the modification separately. Additionally, to test generalization across styles (for a particular transformation), we create two variants of the test set: one using the same style image as the exemplars (i.e., same wooden texture) for the transformation; and another using held-out style images (i.e., other wooden textures).

Editing. We find that editing is able to consistently correct mistakes in a manner that *generalizes across classes*—cf. Figures 6-3 and 6-4. That is, editing is able to reduce errors in non-target classes by often more than 20 percentage points, even though it is performed using only three exemplars from the target class.

Fine-tuning. In contrast, while the two fine-tuning baselines are able to correct mistakes on transformed inputs of the target class used to perform the modifica-



(a) Concepts derived from an instance segmentation model trained on MS-COCO.

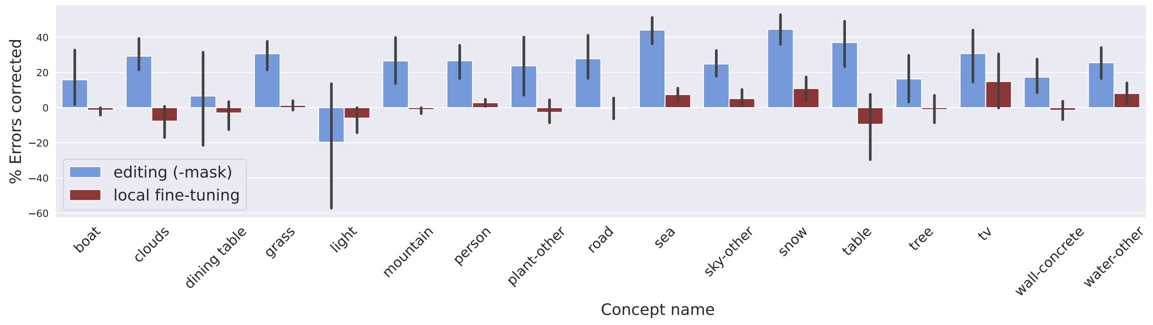


(b) Concepts derived from an instance segmentation model trained on LVIS.

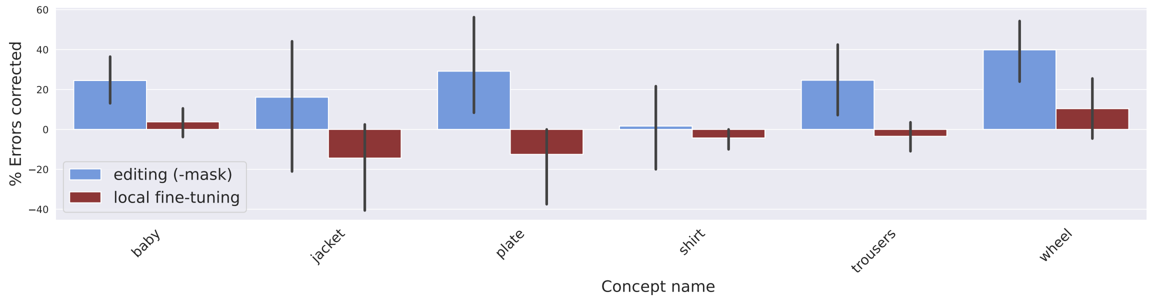
Figure 6-4: Performance vs. drop in overall test set accuracy: Here, we visualize average number of misclassifications corrected by editing and fine-tuning when applied to an ImageNet-trained ResNet-50 classifier—where the average is computed over different concept-transformation pairs. See Appendix Figures E-8-E-10 for additional configurations.

tion, they typically *decrease* the model's performance on *other* classes—i.e., they cause more errors than they fix. Moreover, even when we allow a larger drop in the model's accuracy, or use more training exemplars, their performance often becomes *worse* on inputs from other classes. This suggests that fine-tuning causes the model to overfit to the class it was trained on.

Interestingly, we find that in all cases where a method improves performance, this improvement extends to transformations using other variants of the style. For instance, the modification generalizes to textures of "wood" other than those present in exemplars used to perform the modification (cf. Figure 6-2b). We provide a per-concept/style break down in Appendix Figures 6-5 and 6-6.

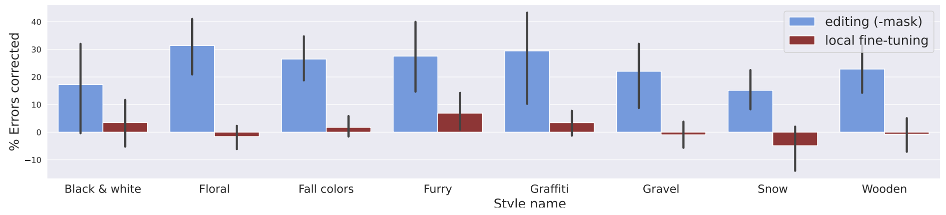


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

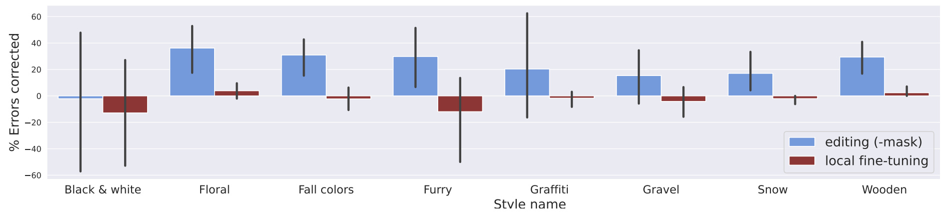


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure 6-5: Performance of editing and fine-tuning on test examples from non-target classes containing a given concept, averaged across transformations.



(a) Concepts derived from an instance segmentation model trained on MS-COCO.



(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure 6-6: Performance of editing and fine-tuning on test examples from non-target classes transformed using a given style, averaged across concepts.

Ablation studies. In order to get a better understanding of the core factors that affect performance in this setting, we conduct a set of ablation studies. Note that we can readily perform these ablations as, in contrast to the setting of Bau et al. [Bau+20a] we have access to a quantitative performance metric that does not rely on human evaluation.

- *Layer:* We compare both editing and local fine-tuning when they are applied to different layers of the model in Appendix Figure 6-7. For editing, we find a consistent increase in performance—on examples from both the target and other classes—as we edit deeper into the model. For (local) fine-tuning, a similar trend is observed with regards to performance on the target class, with the second last layer being optimal overall. However, at the same time, the fine-tuned model's performance on examples from other classes containing the concept seems to get worse.

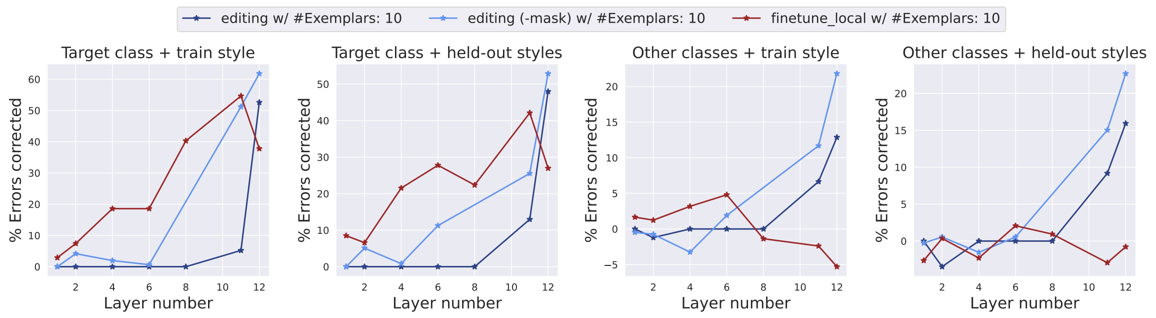


Figure 6-7: Editing vs. fine-tuning performance (with 10 exemplars) on an ImageNet-trained VGG-16 classifier, as a function of the layer that is modified. Wwe visualize the average number of misclassifications corrected over different concept-transformation pairs, with concepts derived from instance segmentation modules trained on MS-COCO; and transformations "snow" and "graffiti".

- *Number of exemplars:* Increasing the exemplars used for each method typically leads to qualitatively the same impact, just more significant, cf. Figure 6-4. We also perform a more fine-grained ablation for a single model (ImageNet-trained VGG16 on COCO-concepts) in Figure 6-8. In general, for editing, using more exemplars tends to improve the number of mistakes corrected on both the target and non-target classes. For fine-tuning, this improves its effectiveness on the target class alone, albeit the trends are more noisy.

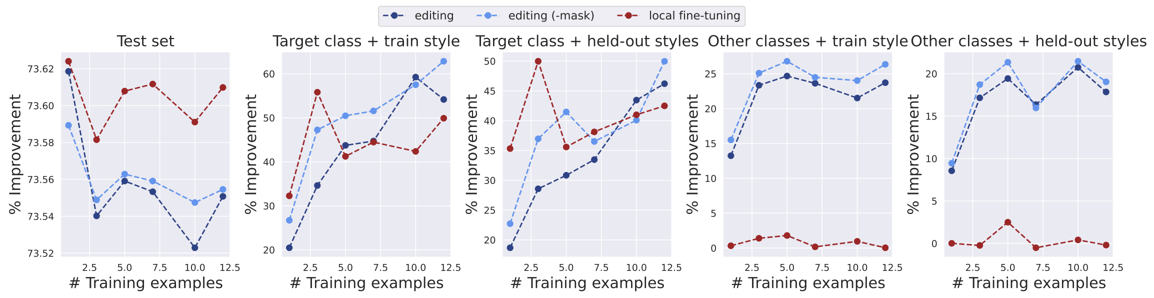


Figure 6-8: Editing vs. fine-tuning on an ImageNet-trained VGG-16 classifier, with concepts derived from MS-COCO, as a function of the number of train exemplars.

- *Rank restriction.* We evaluate the performance of editing when the weight update is not restricted to a rank-one modification. We find that this change significantly reduces the efficacy of editing on examples from both the target and non-target classes—cf. curves corresponding to '-proj' in Figure 6-9. This suggests that the rank restriction is necessary to prevent the model from overfitting to the few exemplars used.

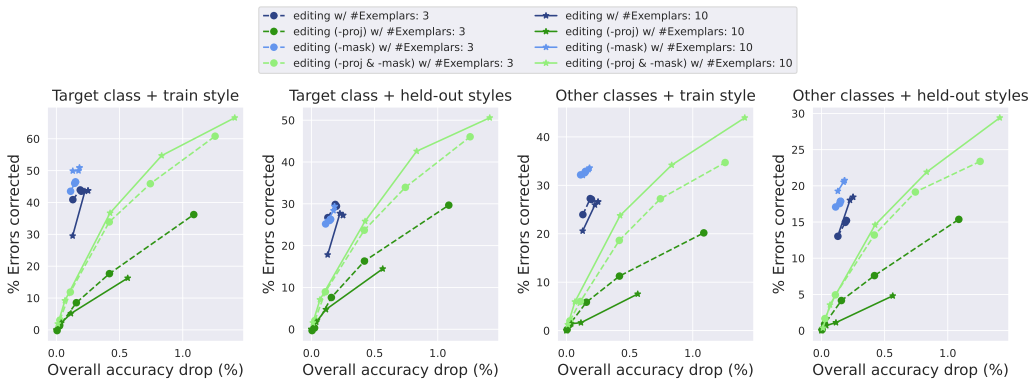
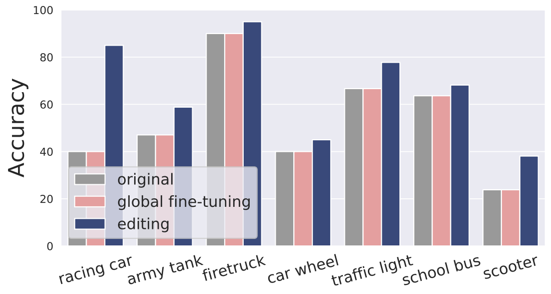
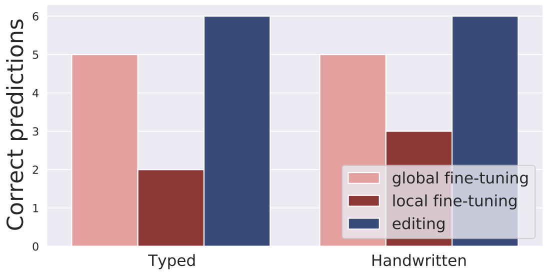


Figure 6-9: Ablating constraints in the editing procedure for a Places365-trained VGG16 classifier, with concepts derived from MS-COCO. See Appendix Figures E-8-E-10 for additional configurations.

- *Mask.* During editing, Bau et al. [Bau+20b] focus on rewriting only the key-value pairs that correspond to the concept of interest. We find, however, that imposing the editing constraints on the entirety of the image leads to even better performance—cf. curves corresponding to '-mask' in Appendix Figures 6-3-6-4. We hypothesize that this has a regularizing effect as it constrains



(a) Vehicles in snowy weather



(b) Typographic attacks

Figure 6-10: (a) Adapting a pre-trained ImageNet classifier to images of vehicles on snowy roads with a single exemplar. Fine-tuning (both local and global) only slightly improves accuracy, while editing to map "snowy road"→"road" leads to a consistent improvement across multiple classes. (b) Improving the robustness of CLIP [Rad+21] models to typographic attacks [Goh+21]. Editing the model to map the text "iPod"→"blank" using a single exemplar—either based on handwritten text on a physical teapot or from pasting typed text on an image of a "can opener"—completely corrects this vulnerability. While global fine-tuning can also improve model performance in this setting, it requires more careful hyperparameter tuning and typically hurts model performance in other contexts.

the weights to preserve the original mapping between keys and values in regions that do not contain the concept.

6.3.3 Real-World Demonstrations

So far, we have seen that our rule rewriting methodology can significantly improve model generalization to concept-level transformations synthesized using our prediction rule-discovery pipeline (cf Section 3.2). To test the versatility of our approach, we now shift our attention to real-world applications of machine learning models, where similar generalization might be desirable.

Tackling new environments: Vehicles on snow. Our first use-case is adapting pre-trained classifiers to image subpopulations that are under-represented in the training data. Specifically, we will focus on the task of recognizing vehicles under heavy snow conditions—a setting that could be pertinent to self-driving cars. To study this problem, we collect a set of real photographs from road-related ImageNet classes using Flickr (details in Appendix E.1.5). To improve model performance under these conditions, we rewrite its prediction rules: to map "snowy roads" to "road". To do so, we first create a *single synthetic* exemplar: by manually annotating the concept "road" in an ImageNet image from a *different* class (here, "police van"), and transforming it using a snow image obtained from Flickr. We then apply our editing methodology (cf. Section 6.2), using this single snow-to-road exemplar—see Figure 6-1.

In Figure 6-10a, we measure the error rate of the model on the new test set (vehicles in snow) before and after performing the rewrite. We find that our edits significantly improve the model's error rate on these images, despite the fact that we did not use any real "snowy road" photographs to edit the model. In contrast, fine-tuning the model under the same setup does not improve its performance.

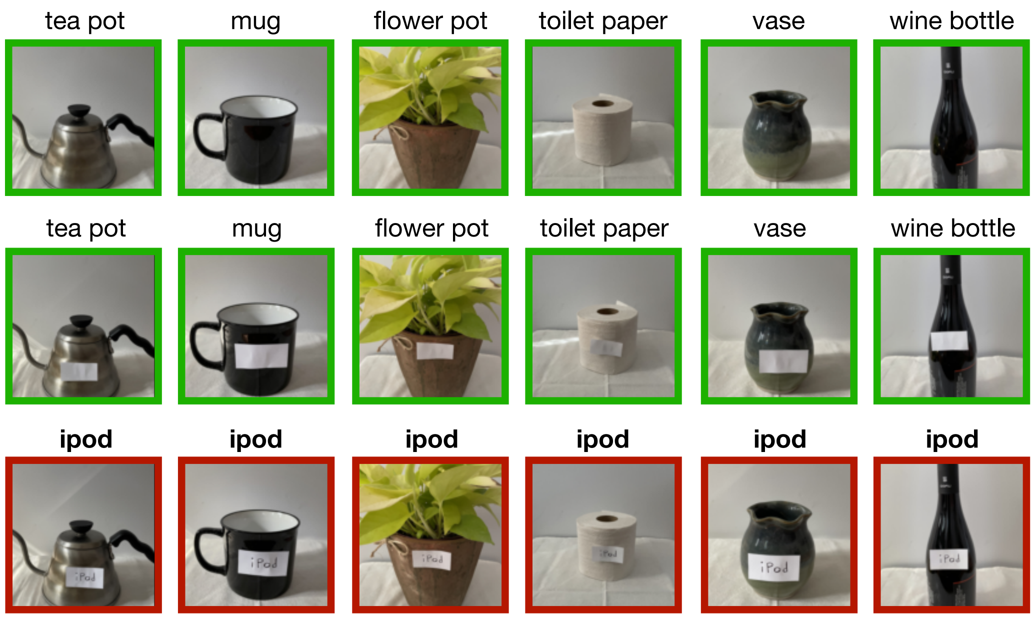
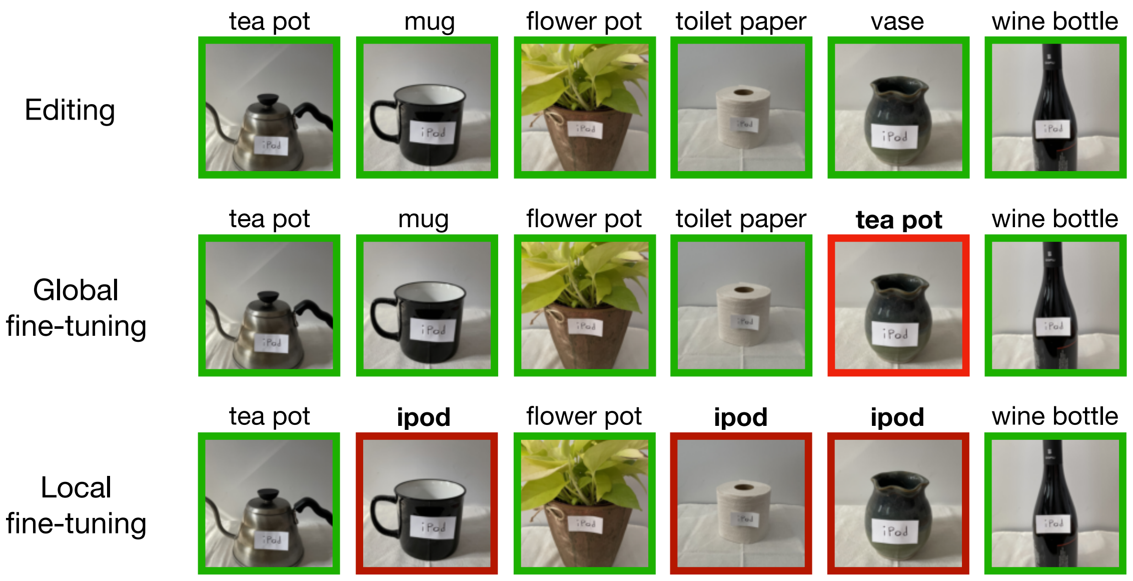
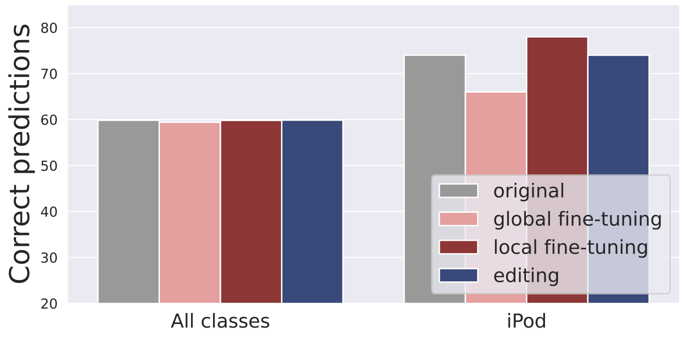


Figure 6-11: Typographic attacks on CLIP: We reproduce the results of Goh et al. [Goh+21] by taking photographs of household objects with a paper containing handwritten text "iPod" attached to them (third row). We see that these attacks consistently fool the zero-shot CLIP classifier (ResNet50)—compare the predictions (shown in the title) for the first and third row. In contrast, if we instead use a blank piece of paper (second row), the model predicts correctly.

Ignoring a spurious feature: Typographic attacks. Our second real-world setting involves modifying a model to ignore a spurious feature. We focus on the



(a)



(b)

Figure 6-12: Effectiveness of different modification procedures in preventing typographic attacks. (a) Model predictions after the rewrite—local fine-tuning often fails to prevent such attacks, while global fine-tuning causes the model to associate "iPod" with the target class used for fine-tuning ("teapot"). (b) Accuracy on the test set and specifically on clean samples from class "ipod" before and after the rewrite. While global fine-tuning is fairly effective at mitigating these attacks, it disproportionately reduces model accuracy on clean images from this class.

recently-discovered typographic attacks from Goh et al. [Goh+21]: simply attaching a piece of paper with the text "iPod" on it is enough to make a zero-shot CLIP [Rad+21] classifier predict an assortment of other objects to be iPods. We start by reproducing these attacks—see Figure 6-11 for an illustration. We now rewrite the model's prediction rules: to map the text "iPod" to "blank" (as the latter does not cause misclassifications). For the choice of our transformed exemplar x', we consider two variants: either a real photograph of a "teapot" with the typographic attack (Figure 6-11); or an ImageNet image from of a "can opener" (randomly-chosen) with the typed text "iPod" programatically pasted on it (Figure 6-1). The original image x for our approach is then obtained by replacing the handwritten/typed text with a white mask—cf. Figure 6-1. We then use this single training exemplar to perform the network modification.

In both cases, we find that editing is able to fix *all* the errors caused by the typographic attacks, see Figure 6-10b. Interestingly, global fine-tuning also helps to correct many of these errors (potentially by adjusting class biases), albeit less reliably (for specific hyperparameters). However, unlike editing, fine-tuning also ends up damaging the model behavior in other scenarios—causing it to now spuriously associate the text "iPod" with the target class used for training or significantly reducing the accuracy on normal "iPod" images from the test set (Figure 6-12).

6.4 Further Related Work

We now discuss previous research in interpretability, robustness, and domain adaptation that our work builds upon.

Ignoring spurious features. Prior work on preventing models from relying on spurious correlations is based on constraining model predictions to satisfy certain invariances. This can be done by: creating counterfactuals that add or remove objects from scenes [SFS18; SSF19; ASF20], learning representations that are simultaneously optimal across domains [Arj+19], ensuring comparable performance across data subpopulations [Sag+20], or enforcing consistency in predictions across inputs that depict the same physical entity [HM17]. In this work, we take an alternative approach that does not rely on collecting new inputs or annotations, and instead directly rewrites the model's prediction rules.

Model interventions. Direct manipulations of latent representations inside generative models have been used to create human-understandable changes in synthesized images [Bau+19a; JCI19; Goe+19; She+20; Här+20; WLS20]. Our work is inspired by that line of work as well as a recent finding that parameters of a generative model can be directly changed to alter generalized behavior [Bau+20a]. Unlike previous work, we edit classification models, changing rules that govern predictions rather than image synthesis.

Previous work on direct interventions within a deep network has mostly focused on activating or deactivating neurons for a single stimulus at a time. Objects can be added or removed to image generators [Bau+19a]; predictions can be changed in image classifiers [Bau+20b]; and state and biases can be altered in NLP models [Bau+18; Vig+20] by raising or lowering the values of neurons that represent high-level concepts. Unlike this work, we ask how the parameters of a model can be changed to modify the high-level rules of a classifier.

Model robustness. A long line of work has been devoted to discovering and correcting failure modes of models. These studies focus on simulating variations in testing conditions that can arise during deployment, including: adversarial or natural input corruptions [Sze+14; FF15; FMF16; Eng+19c; For+19; HD19; Kan+19], changes in the data collection process [Sae+10; TE11; Kho+12; TT14; Rec+19b], or variations in the data subpopulations present [BVP18; Ore+19; Sag+20; STM21; Koh+20]. Typical approaches for improving robustness in these contexts include robust optimization [Mad+18; Yin+19; Sag+20] and additional data augmentation [Lop+19; Hen+19a; Zha+21]. Our rule-editing pipeline can be viewed as complementary to this work as they allows us to preemptively adjust the model's prediction rules in anticipation of deployment conditions.

Domain adaptation. The goal of domain adaptation is to adapt a model to a specific deployment environment using (potentially unlabeled) samples from it. This is typically achieved by either fine-tuning the model on the new domain [Don+14; Sha+14; KML20], by learning the correspondence between the source and target domain, often in a latent representation space [Ben+07; Sae+10; GL15; Cou+16;

Gon+16], or by updating the model's batch normalization statistics [Li+16b; BS21]. These approaches all require a non-trivial amount of data from the target domain. The question of adaptation from a handful of samples has been explored [Mot+17], but in a setting that requires samples across all target classes. In contrast, our method allows for generalization to new (potentially unknown) classes with even a single example.

6.5 Broader Implications: Human-in-the-loop Model Correction

We conclude by discussing more broadly how our model editing toolkit fits in within the ML pipeline. We have seen consistently that current deep networks are hard to interact with and probe. Thus, the dominant approach in ML today is to specify model behavior *implicitly* via data. But, aside from being expensive, precise data collection can be bias-ridden or even infeasible. The editing approach developed in this chapter offers an alternative path forward: wherein model designers can *directly* modify, and in a targeted manner, the behavior of their models. As we saw through various examples, these edits they can be guided by as few as a single (synthetically-created) exemplar. In a sense, such targeted model editing can offer the best of both worlds: it still leverages the fact that deep networks can automatically extract powerful features from complex datasets, while at the same time allowing developers to use their prior knowledge and preferences to manipulate these features. We believe that this primitive opens up new avenues to interact with and correct our models before or during deployment.

Bibliography

[ACW18] Anish Athalye, Nicholas Carlini, and David A. Wagner. "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples". In: *International Conference on Machine Learning (ICML)*. 2018.

[Ade+18] Julius Adebayo et al. "Sanity checks for saliency maps". In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[Ade+20] Julius Adebayo et al. "Debugging Tests for Model Explanations". In: 2020.

[Agg+20] Gunjan Aggarwal et al. "On the Benefits of Models with Perceptually-Aligned Gradients". In: *Towards Trustworthy ML Workshop (ICLR)*. 2020.

[Alq+20] Ahmed Alqaraawi et al. "Evaluating saliency map explanations for convolutional neural networks: a user study". In: *Proceedings of the 25th International Conference on Intelligent User Interfaces*. 2020.

[Arj+19] Martin Arjovsky et al. "Invariant risk minimization". In: *arXiv preprint arXiv:1907.02893* (2019).

[ASF20] Vedika Agarwal, Rakshith Shetty, and Mario Fritz. "Towards causal vqa: Revealing and reducing spurious correlations by invariant and covariant semantic editing". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9690–9698.

[Ath+18] Anish Athalye et al. "Synthesizing Robust Adversarial Examples". In: *International Conference on Machine Learning (ICML)*. 2018.

[Bar+19] Andrei Barbu et al. “ObjectNet: A large-scale bias-controlled dataset for pushing the limits of object recognition models”. In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[Bau+17] David Bau et al. “Network dissection: Quantifying interpretability of deep visual representations”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2017.

[Bau+18] Anthony Bau et al. “Identifying and Controlling Important Neurons in Neural Machine Translation”. In: *International Conference on Learning Representations*. 2018.

[Bau+19a] David Bau et al. “GAN Dissection: Visualizing and Understanding Generative Adversarial Networks”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[Bau+19b] David Bau et al. “Inverting Layers of a Large Generator”. In: *ICLR Debugging Machine Learning Models Workshop*. 2019.

[Bau+20a] David Bau et al. “Rewriting a deep generative model”. In: *European Conference on Computer Vision (ECCV)*. 2020.

[Bau+20b] David Bau et al. “Understanding the role of individual units in a deep neural network”. In: *Proceedings of the National Academy of Sciences (PNAS)* (2020).

[BCV13] Y. Bengio, A. Courville, and P. Vincent. “Representation Learning: A Review and New Perspectives”. In: (2013).

[BDS19] Andrew Brock, Jeff Donahue, and Karen Simonyan. “Large Scale GAN Training for High Fidelity Natural Image Synthesis”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[Ben+07] Shai Ben-David et al. “Analysis of representations for domain adaptation”. In: *Neural Information Processing Systems (NeurIPS)*. 2007.

[Ben19] Yoshua Bengio. *Talk Abstract: Learning High-Level Representations for Agents*. Abstract for talk given at MIT. 2019. URL: `https://calendar.mit.edu/event/yoshua_bengio_learning_high-level_representations_for_agents%5C#.XYozli2ZNhF`.

[Ber+00] Marcelo Bertalmio et al. "Image inpainting". In: *Computer graphics and interactive techniques*. 2000.

[Ber+19] Christopher Berner et al. "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680* (2019).

[Bey+20] Lucas Beyer et al. "Are we done with ImageNet?" In: *arXiv preprint arXiv:2006.07159* (2020).

[BG18] Joy Buolamwini and Timnit Gebru. "Gender shades: Intersectional accuracy disparities in commercial gender classification". In: *Conference on fairness, accountability and transparency (FAccT)*. 2018.

[Big+13] Battista Biggio et al. "Evasion attacks against machine learning at test time". In: *Joint European conference on machine learning and knowledge discovery in databases (ECML-KDD)*. 2013.

[Bla+20] Davis Blalock et al. "What is the state of neural network pruning?" In: *arXiv preprint arXiv:2003.03033* (2020).

[Bol+16] Tolga Bolukbasi et al. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings". In: *arXiv preprint arXiv:1607.06520* (2016).

[Bor+19] Daniel Borkan et al. "Nuanced metrics for measuring unintended bias with real data for text classification". In: *Companion Proceedings of The 2019 World Wide Web Conference*. 2019, pp. 491–500.

[Bow+15] Samuel R Bowman et al. "A large annotated corpus for learning natural language inference". In: *arXiv preprint arXiv:1508.05326* (2015).

[BPR19] Sébastien Bubeck, Eric Price, and Ilya Razenshteyn. "Adversarial examples from computational constraints". In: *International Conference on Machine Learning*. 2019.

[BR18] Battista Biggio and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning". In: 2018.

[Bro+20] Tom B Brown et al. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[BS21] Collin Burns and Jacob Steinhardt. "Limitations of Post-Hoc Feature Alignment for Robustness". In: 2021.

[BSH12] Harold C Burger, Christian J Schuler, and Stefan Harmeling. "Image denoising: Can plain neural networks compete with BM3D?" In: *Computer Vision and Pattern Recognition (CVPR)*. 2012.

[BSR20] Solon Barocas, Andrew D Selbst, and Manish Raghavan. "The hidden assumptions behind counterfactual explanations and principal reasons". In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. 2020.

[BVA20] Alceu Bissoto, Eduardo Valle, and Sandra Avila. "Debiasing Skin Lesion Datasets and Models? Not So Fast". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 740–741.

[BVP18] Sara Beery, Grant Van Horn, and Pietro Perona. "Recognition in terra incognita". In: *European Conference on Computer Vision (ECCV)*. 2018.

[CBN17] Aylin Caliskan, Joanna J Bryson, and Arvind Narayanan. "Semantics derived automatically from language corpora contain human-like biases". In: *Science* (2017).

[Cha+17] Arjun Chandrasekaran et al. "It takes two to tango: Towards theory of ai's mind". In: *arXiv preprint arXiv:1704.00717* (2017).

[Cha+19] Chun-Hao Chang et al. “Explaining Image Classifiers by Counterfactual Generation”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[Cou+16] Nicolas Courty et al. “Optimal transport for domain adaptation”. In: *Transactions on Pattern Analysis and Machine Intelligence*. 2016.

[Cra13] Kate Crawford. “The hidden biases in big data”. In: *Harvard business review* (2013).

[CRA20] Eric Chu, Deb Roy, and Jacob Andreas. “Are visual explanations useful? a case study in model-in-the-loop prediction”. In: *arXiv preprint arXiv:2007.12248* (2020).

[CRK19] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. “Certified adversarial robustness via randomized smoothing”. In: *International Conference on Machine Learning (ICML)*. 2019.

[CRP19] Zachary Charles, Harrison Rosenberg, and Dimitris Papailiopoulos. “A Geometric Perspective on the Transferability of Adversarial Directions”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2019.

[CW08] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning”. In: *International Conference on Machine Learning (ICML)*. 2008, pp. 160–167.

[CW17] Nicholas Carlini and David Wagner. “Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods”. In: *Workshop on Artificial Intelligence and Security (AISec)*. 2017.

[Dah+10] George Dahl et al. “Phone recognition with the mean-covariance restricted Boltzmann machine”. In: (2010).

[Dan67] John M. Danskin. *The Theory of Max-Min and its Application to Weapons Allocation Problems*. 1967.

[Das+19] Constantinos Daskalakis et al. “Efficient Statistics, in High Dimensions, from Truncated Samples”. In: *Foundations of Computer Science (FOCS)*. 2019.

[DB16a] Alexey Dosovitskiy and Thomas Brox. “Generating images with perceptual similarity metrics based on deep networks”. In: *neural information processing systems (NeurIPS)*. 2016.

[DB16b] Alexey Dosovitskiy and Thomas Brox. “Inverting visual representations with convolutional networks”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2016.

[DBL14] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. “SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives”. In: *Advances in neural information processing systems (NeurIPS)*. 2014.

[Den+09] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2009.

[Den+10] Li Deng et al. “Binary coding of speech spectrograms using a deep auto-encoder”. In: *Eleventh Annual Conference of the International Speech Communication Association*. 2010.

[Dev+18] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[DFE07] Kostadin Dabov, Alessandro Foi, and Karen Egiazarian. “Video denoising by sparse 3D transform-domain collaborative filtering”. In: *European Signal Processing Conference*. 2007.

[DG17] Piotr Dabkowski and Yarin Gal. “Real time image saliency for black box classifiers”. In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[DKB14] Laurent Dinh, David Krueger, and Yoshua Bengio. "Nice: Non-linear independent components estimation". In: *arXiv preprint arXiv:1410.8516*. 2014.

[Don+14] Jeff Donahue et al. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International conference on machine learning (ICML)*. 2014.

[DSB17] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using real NVP". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Efr+04] Bradley Efron et al. "Least angle regression". In: *The Annals of statistics* (2004).

[EL99] Alexei A Efros and Thomas K Leung. "Texture synthesis by non-parametric sampling". In: *conference on computer vision (CVPR)*. 1999.

[Eng+19a] Logan Engstrom et al. "Adversarial Robustness as a Prior for Learned Representations". In: *ArXiv preprint arXiv:1906.00945*. 2019.

[Eng+19b] Logan Engstrom et al. "Adversarial robustness as a prior for learned representations". In: *arXiv preprint arXiv:1906.00945* (2019).

[Eng+19c] Logan Engstrom et al. "Exploring the Landscape of Spatial Robustness". In: *International Conference on Machine Learning (ICML)*. 2019.

[Eng+20] Logan Engstrom et al. "Identifying Statistical Bias in Dataset Replication". In: *International Conference on Machine Learning (ICML)*. 2020.

[Erh+09] Dumitru Erhan et al. "Visualizing higher-layer features of a deep network". In: (2009).

[Eve+10] M. Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision*. 2010.

[Eyk+18] Kevin Eykholt et al. "Physical Adversarial Examples for Object Detectors". In: *CoRR* (2018).

[FF15] Alhussein Fawzi and Pascal Frossard. “Manitest: Are classifiers really invariant?” In: *British Machine Vision Conference (BMVC)*. 2015.

[FFF18a] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. “Adversarial vulnerability for any classifier”. In: *Advances in Neural Information Processing Systems (NeuRIPS)*. 2018.

[FFF18b] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. “Analysis of classifiers’ robustness to adversarial perturbations”. In: *Machine Learning* 107.3 (2018), pp. 481–508.

[FHT10] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. “Regularization paths for generalized linear models via coordinate descent”. In: *Journal of statistical software* (2010).

[Fin+16] Chelsea Finn et al. “Deep spatial autoencoders for visuomotor learning”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016.

[FMF16] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. “Robustness of classifiers: from adversarial to random noise”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016.

[FN17] Batya Friedman and Helen Nissenbaum. *Bias in computer systems*. Routledge, 2017.

[For+19] Nic Ford et al. “Adversarial Examples Are a Natural Consequence of Test Error in Noise”. In: *arXiv preprint arXiv:1901.10513*. 2019.

[Fuk80] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* (1980).

[FV17] Ruth C Fong and Andrea Vedaldi. “Interpretable explanations of black boxes by meaningful perturbation”. In: *International Conference on Computer Vision (ICCV)*. 2017.

[GAZ19] Amirata Ghorbani, Abubakar Abid, and James Zou. "Interpretation of neural networks is fragile". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2019.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GDG19] Agrim Gupta, Piotr Dollar, and Ross Girshick. "LVIS: A Dataset for Large Vocabulary Instance Segmentation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[GEB16] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks". In: *computer vision and pattern recognition (CVPR)*. 2016.

[Gei+19] Robert Geirhos et al. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness." In: *International Conference on Learning Representations (ICLR)*. 2019.

[Gei+20] Robert Geirhos et al. "Shortcut learning in deep neural networks". In: *Nature Machine Intelligence*. 2020.

[GGS19] Nidham Gazagnadou, Robert M Gower, and Joseph Salmon. "Optimal mini-batch and step sizes for SAGA". In: *arXiv preprint arXiv:1902.00071* (2019).

[Ghi+17] Golnaz Ghiasi et al. "Exploring the structure of a real-time, arbitrary neural artistic stylization network". In: *arXiv preprint arXiv:1705.06830* (2017).

[Gho+19] Amirata Ghorbani et al. "Towards automatic concept-based explanations". In: *arXiv preprint arXiv:1902.03129* (2019).

[Gil+18] Justin Gilmer et al. "Adversarial spheres". In: *Workshop of International Conference on Learning Representations (ICLR)*. 2018.

[Gir+14] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *computer vision and pattern recognition (CVPR)*. 2014, pp. 580–587.

[Gir+18] Ross Girshick et al. *Detectron*. `https://github.com/facebookresearch/detectron`. 2018.

[GL15] Yaroslav Ganin and Victor Lempitsky. "Unsupervised domain adaptation by backpropagation". In: 2015.

[Goe+19] Lore Goetschalckx et al. "Ganalyze: Toward visual definitions of cognitive image properties". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5744–5753.

[Goh+21] Gabriel Goh et al. "Multimodal neurons in artificial neural networks". In: *Distill* (2021).

[Goh19a] Gabriel Goh. "A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Robust Feature Leakage". In: *Distill* (2019). https://distill.pub/2019
bugs-discussion/response-2. DOI: `10.23915/distill.00019.2`.

[Goh19b] Gabriel Goh. "A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Two Examples of Useful, Non-Robust Features". In: *Distill* (2019). https://distill.pub/2019/advex-bugs-discussion/response-3. DOI: `10.23915/distill.00019.3`.

[Gon+16] Mingming Gong et al. "Domain adaptation with conditional transferable components". In: *International conference on machine learning (ICML)*. 2016.

[Goo+14] Ian Goodfellow et al. "Generative adversarial nets". In: *neural information processing systems (NeurIPS)*. 2014.

[Goy+19a] Yash Goyal et al. "Counterfactual visual explanations". In: *arXiv preprint arXiv:1904.07451* (2019).

[Goy+19b] Yash Goyal et al. "Explaining classifiers with causal concept effect (cace)". In: *arXiv preprint arXiv:1907.07165* (2019).

[Gra13] Alex Graves. “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850*. 2013.

[GSS15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *International Conference on Learning Representations (ICLR)*. 2015.

[Han+15] Song Han et al. “Learning both weights and connections for efficient neural networks”. In: *arXiv preprint arXiv:1506.02626* (2015).

[Här+20] Erik Härkönen et al. “GANSpace: Discovering Interpretable GAN Controls”. In: *Advances in Neural Information Processing Systems* 33 (2020).

[Has+07] Trevor Hastie et al. “Forward stagewise regression and the monotone lasso”. In: *Electronic Journal of Statistics* 1 (2007), pp. 1–29.

[HD19] Dan Hendrycks and Thomas G. Dietterich. “Benchmarking Neural Network Robustness to Common Corruptions and Surface Variations”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[He+15a] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015.

[He+15b] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *international conference on computer vision (ICCV)*. 2015.

[He+16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[He+17] Warren He et al. “Adversarial example defense: Ensembles of weak defenses are not strong”. In: *USENIX Workshop on Offensive Technologies (WOOT)*. 2017.

[He+19] Tong He et al. “Bag of tricks for image classification with convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 558–567.

[HE07] James Hays and Alexei A Efros. “Scene completion using millions of photographs”. In: *ACM Transactions on Graphics (TOG)*. 2007.

[Hen+19a] Dan Hendrycks et al. “Augmix: A simple data processing method to improve robustness and uncertainty”. In: *arXiv preprint arXiv:1912.02781* (2019).

[Hen+19b] Dan Hendrycks et al. “Natural adversarial examples”. In: *arXiv preprint arXiv:1907.07174* (2019).

[Her+01] Aaron Hertzmann et al. “Image analogies”. In: *Computer graphics and interactive techniques*. 2001.

[HM17] Christina Heinze-Deml and Nicolai Meinshausen. “Conditional variance penalties and domain shift robustness”. In: *arXiv preprint arXiv:1710.11469* (2017).

[HMD16] Song Han, Huizi Mao, and William J. Dally. “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”. In: *International Conference on Learning Representations (ICLR)*. 2016.

[Hoo+19] Sara Hooker et al. “Selective Brain Damage: Measuring the Disparate Impact of Model Pruning”. In: *arXiv preprint arXiv:1911.05248* (2019).

[HS93] Babak Hassibi and David Stork. “Second order derivatives for network pruning: Optimal Brain Surgeon”. In: *Advances in Neural Information Processing Systems*. 1993.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation*. 1997.

[HSS18] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-excitation networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

[Hua+17] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[Ian+16] Forrest N Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016).

[Ily+19] Andrew Ilyas et al. “Adversarial Examples Are Not Bugs, They Are Features”. In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[ISI17] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. “Globally and Locally Consistent Image Completion”. In: *ACM Trans. Graph.* 2017.

[Iso+17] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *conference on computer vision and pattern recognition (CVPR)*. 2017.

[JAF16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. “Perceptual losses for real-time style transfer and super-resolution”. In: *European conference on computer vision (ECCV)*. 2016.

[JCI19] Ali Jahanian, Lucy Chai, and Phillip Isola. “On the" steerability" of generative adversarial networks”. In: *International Conference on Learning Representations*. 2019.

[Joh+16] Alistair EW Johnson et al. “MIMIC-III, a freely accessible critical care database”. In: *Scientific data* (2016).

[JSO18] Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. “i-RevNet: Deep Invertible Networks”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[JZ13] Rie Johnson and Tong Zhang. “Accelerating stochastic gradient descent using predictive variance reduction”. In: *Advances in neural information processing systems* 26 (2013), pp. 315–323.

[Kan+19] Daniel Kang et al. “Testing Robustness Against Unforeseen Adversaries”. In: *ArXiv preprint arxiv:1908.08016*. 2019.

[Kar+18] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: *International Conference on Learning Representations*. 2018.

[KD18] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative flow with invertible 1x1 convolutions”. In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. “Adversarial examples in the physical world”. In: *arXiv preprint arXiv:1607.02533* (2016).

[Kho+12] Aditya Khosla et al. “Undoing the damage of dataset bias”. In: *European Conference on Computer Vision (ECCV)*. 2012.

[Kim+18] Been Kim et al. “Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)”. In: *International conference on machine learning (ICML)*. 2018.

[KKK16] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. “Examples are not enough, learn to criticize! criticism for interpretability”. In: *Advances in neural information processing systems (NeurIPS*. 2016.

[KL17] Pang Wei Koh and Percy Liang. “Understanding Black-box Predictions via Influence Functions”. In: *ICML*. 2017.

[KML20] Ananya Kumar, Tengyu Ma, and Percy Liang. “Understanding Self-Training for Gradual Domain Adaptation”. In: *International Conference on Machine Learning (ICML)*. 2020.

[Koh+20] Pang Wei Koh et al. “WILDS: A Benchmark of in-the-Wild Distribution Shifts”. In: *arXiv preprint arXiv:2012.07421* (2020).

[Kri09] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: *Technical report*. 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2012.

[KSJ19] Beomsu Kim, Junghoon Seo, and Taegyun Jeon. "Bridging Adversarial Robustness and Gradient Interpretability". In: *International Conference on Learning Representations Workshop on Safe Machine Learning (ICLR SafeML)*. 2019.

[KW15] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2015.

[LDS90] Yann LeCun, John Denker, and Sara Solla. "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Morgan-Kaufmann, 1990.

[Lec+19] Mathias Lecuyer et al. "Certified robustness to adversarial examples with differential privacy". In: *Symposium on Security and Privacy (SP)*. 2019.

[Lec+21] Guillaume Leclerc et al. "3DB: A Framework for Debugging Computer Vision Models". In: *arXiv preprint arXiv:2106.03805* (2021).

[LeC98] Yann LeCun. "The MNIST database of handwritten digits". In: *Technical report*. 1998.

[Li+16a] Hao Li et al. "Pruning filters for efficient convnets". In: *arXiv preprint arXiv:1608.08710* (2016).

[Li+16b] Yanghao Li et al. "Revisiting batch normalization for practical domain adaptation". In: *arXiv preprint arXiv:1603.04779* (2016).

[LI16] Kristian Lum and William Isaac. "To predict and serve?" In: *Significance* (2016).

[Lin+14] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision (ECCV)*. 2014.

[Lip18] Zachary C Lipton. "The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery." In: (2018).

[Liu+18] Chenxi Liu et al. “Progressive neural architecture search”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 19–34.

[LM00] Beatrice Laurent and Pascal Massart. “Adaptive estimation of a quadratic functional by model selection”. In: *Annals of Statistics*. 2000.

[LM20] Matthew L Leavitt and Ari Morcos. “Towards falsifiable interpretability research”. In: *arXiv preprint arXiv:2010.12016* (2020).

[Lop+19] Raphael Gontijo Lopes et al. “Improving robustness without sacrificing accuracy with patch gaussian augmentation”. In: *arXiv preprint arXiv:1906.02611* (2019).

[Mad+18] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[MDM18] Saeed Mahloujifar, Dimitrios I Diochnos, and Mohammad Mahmoody. “The curse of concentration in robust learning: Evasion and poisoning attacks from concentration of measure”. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2018.

[Mil95] George A Miller. “WordNet: a lexical database for English”. In: *Communications of the ACM* (1995).

[Mot+17] Saeid Motiian et al. “Few-Shot Adversarial Domain Adaptation”. In: *NIPS*. 2017.

[MOT15] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. *Inceptionism: Going deeper into neural networks*. 2015. URL: `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`.

[MV15] Aravindh Mahendran and Andrea Vedaldi. “Understanding deep image representations by inverting them”. In: *computer vision and pattern recognition (CVPR)*. 2015.

[Nak19a] Preetum Nakkiran. “A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Adversarial Examples are Just Bugs, Too”. In: *Distill* (2019). https://distill.pub/2019/advex-bugs-discussion/response-5. DOI: `10.23915/distill.00019.5`.

[Nak19b] Preetum Nakkiran. “Adversarial robustness may be at odds with simplicity”. In: *arXiv preprint arXiv:1901.00532*. 2019.

[Ngu+16] Anh Nguyen et al. “Synthesizing the preferred inputs for neurons in neural networks via deep generator networks”. In: *Neural Information Processing Systems (NeurIPS)*. 2016.

[Ngu+17] Anh Nguyen et al. “Plug & play generative networks: Conditional iterative generation of images in latent space”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[NJC19] Curtis G Northcutt, Lu Jiang, and Isaac L Chuang. “Confident Learning: Estimating Uncertainty in Dataset Labels”. In: *arXiv preprint arXiv:1911.00068* (2019).

[NYC15] Anh Nguyen, Jason Yosinski, and Jeff Clune. “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images”. In: *Conference on computer vision and pattern recognition (CVPR)*. 2015.

[Obe+19] Ziad Obermeyer et al. “Dissecting racial bias in an algorithm used to manage the health of populations”. In: *Science* (2019).

[Ola+18] Chris Olah et al. “The Building Blocks of Interpretability”. In: *Distill*. 2018.

[OMS17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. “Feature Visualization”. In: *Distill*. 2017.

[Ore+19] Yonatan Oren et al. “Distributionally Robust Language Modeling”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2019.

[Oyg15] Audun Oygard. *Visualizing GoogLeNet Classes*. 2015. URL: `https://www.auduno.com/2015/07/29/visualizing-googlenet-classes/`.

[Pea10] Judea Pearl. "Causal inference". In: *Causality: Objectives and Assessment* (2010).

[PH07] Mee Young Park and Trevor Hastie. "L1-regularization path algorithm for generalized linear models". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* (2007).

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. "Transferability in Machine Learning: from Phenomena to Black-box Attacks using Adversarial Samples". In: *ArXiv preprint arXiv:1605.07277*. 2016.

[Pon+06] Jean Ponce et al. "Dataset issues in object recognition". In: *Toward category-level object recognition*. 2006.

[PY09] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* (2009).

[Rad+21] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *arXiv preprint arXiv:2103.00020* (2021).

[Raj+16] Pranav Rajpurkar et al. "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250* (2016).

[RD18] Andrew Slavin Ross and Finale Doshi-Velez. "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[Rec+19a] Benjamin Recht et al. "Do CIFAR-10 Classifiers Generalize to CIFAR-10?" In: *International Conference on Machine Learning (ICML)*. 2019.

[Rec+19b] Benjamin Recht et al. "Do ImageNet Classifiers Generalize to ImageNet?" In: *International Conference on Machine Learning (ICML)*. 2019.

[RM15] Danilo Jimenez Rezende and Shakir Mohamed. "Variational inference with normalizing flows". In: *International Conference on Machine Learning (ICML)*. 2015.

[RMC16] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *International Conference on Learning Representations (ICLR)*. 2016.

[RSG16a] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "" Why should I trust you?" Explaining the predictions of any classifier". In: *International Conference on Knowledge Discovery and Data Mining (KDD)*. 2016.

[RSG16b] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *International Conference on Knowledge Discovery and Data Mining (KDD)*. 2016.

[RSL18] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. "Certified defenses against adversarial examples". In: *International Conference on Learning Representations (ICLR)*. 2018.

[Rus+15] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)*. 2015.

[RZT18] Amir Rosenfeld, Richard Zemel, and John K. Tsotsos. "The Elephant in the Room". In: *arXiv preprint arXiv:1808.03305*. 2018.

[Sae+10] Kate Saenko et al. "Adapting visual category models to new domains". In: *European conference on computer vision (ECCV)*. 2010.

[Sag+20] Shiori Sagawa et al. "Distributionally Robust Neural Networks for Group Shifts: On the Importance of Regularization for Worst-Case Generalization". In: *International Conference on Learning Representations*. 2020.

[Sal+20] Hadi Salman et al. “Do Adversarially Robust ImageNet Models Transfer Better?” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[San+18] Mark Sandler et al. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[San+19] Shibani Santurkar et al. “Image Synthesis with a Single (Robust) Classifier”. In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[San+21] Shibani Santurkar et al. “Editing a classifier by rewriting its prediction rules”. In: *Preprint*. 2021.

[SC18] Pierre Stock and Moustapha Cisse. “Convnets and imagenet beyond accuracy: Understanding mistakes and uncovering biases”. In: *European Conference on Computer Vision (ECCV)*. 2018.

[Sch+18] Ludwig Schmidt et al. “Adversarially Robust Generalization Requires More Data”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[Sen+20] Andrew W Senior et al. “Improved protein structure prediction using potentials from deep learning”. In: *Nature* (2020).

[Sey+20] Laleh Seyyed-Kalantari et al. “CheXclusion: Fairness gaps in deep chest X-ray classifiers”. In: *BIOCOMPUTING 2021: Proceedings of the Pacific Symposium*. 2020.

[SFS18] Rakshith Shetty, Mario Fritz, and Bernt Schiele. “Adversarial Scene Editing: Automatic Object Removal from Weak Supervision”. In: *NeurIPS*. 2018.

[Sha+14] Ali Sharif Razavian et al. “CNN features off-the-shelf: an astounding baseline for recognition”. In: *conference on computer vision and pattern recognition (CVPR) workshops*. 2014.

[Sha+16] Mahmood Sharif et al. "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 2016, pp. 1528–1540.

[Sha+19a] Ali Shafahi et al. "Are adversarial examples inevitable?" In: *International Conference on Learning Representations (ICLR)*. 2019.

[Sha+19b] Adi Shamir et al. "A Simple Explanation for the Existence of Adversarial Examples with Small Hamming Distance". In: *arXiv preprint arXiv:1901.10861*. 2019.

[She+20] Yujun Shen et al. "Interpreting the latent space of gans for semantic face editing". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9243–9252.

[SHS19] David Stutz, Matthias Hein, and Bernt Schiele. "Disentangling Adversarial Robustness and Generalization". In: *Computer Vision and Pattern Recognition (CVPR)*. 2019.

[Sil+17] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature*. 2017.

[Sla+20] Dylan Slack et al. "Fooling lime and shap: Adversarial attacks on post hoc explanation methods". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. 2020.

[Smi+17] D. Smilkov et al. "SmoothGrad: removing noise by adding noise". In: *ICML workshop on visualization for deep learning*. 2017.

[Soc+13] Richard Socher et al. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642.

[SSF19] Rakshith Shetty, Bernt Schiele, and Mario Fritz. "Not Using the Car to See the Sidewalk–Quantifying and Controlling the Effects of Context in Classification and Segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8218–8226.

[STM21] Shibani Santurkar, Dimitris Tsipras, and Aleksander Madry. "Breeds: Benchmarks for subpopulation shift". In: *International Conference on Learning Representations (ICLR)*. 2021.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic attribution for deep networks". In: *International Conference on Machine Learning (ICML)*. 2017.

[Sug+19] Arun Sai Suggala et al. "Revisiting Adversarial Risk". In: *Conference on Artificial Intelligence and Statistics (AISTATS)*. 2019.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *arXiv preprint arXiv:1312.6034* (2013).

[SZ15] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations (ICLR)*. 2015.

[Sze+14] Christian Szegedy et al. "Intriguing properties of neural networks". In: *International Conference on Learning Representations (ICLR)*. 2014.

[Sze+15] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[Sze+16] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Sze+17] Christian Szegedy et al. “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[Tao+20] Rohan Taori et al. “Measuring Robustness to Natural Distribution Shifts in Image Classification”. In: *arXiv preprint arXiv:2007.00644* (2020).

[TE11] Antonio Torralba and Alexei A Efros. “Unbiased look at dataset bias”. In: *CVPR 2011*. 2011.

[TG16] Thomas Tanay and Lewis Griffin. “A Boundary Tilting Perspective on the Phenomenon of Adversarial Examples”. In: *ArXiv preprint arXiv:1608.07690*. 2016.

[Tib94] Robert Tibshirani. “Regression Shrinkage and Selection Via the Lasso”. In: *Journal of the Royal Statistical Society, Series B*. 1994.

[TL19] Mingxing Tan and Quoc V Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *arXiv preprint arXiv:1905.11946* (2019).

[Tsi+19] Dimitris Tsipras et al. “Robustness May Be at Odds with Accuracy”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[Tsi+20] Dimitris Tsipras et al. “From ImageNet to Image Classification: Contextualizing Progress on Benchmarks”. In: *International Conference on Machine Learning (ICML)*. 2020.

[TT14] Tatiana Tommasi and Tinne Tuytelaars. “A testbed for cross-dataset analysis”. In: *European Conference on Computer Vision (ECCV)*. 2014.

[TW17] Ryan Tibshirani and Larry Wasserman. “Sparsity, the Lasso, and Friends”. In: *Lecture notes from “Statistical Machine Learning,” Carnegie Mellon University, Spring* (2017).

[Tyk16] Mike Tyka. *Class visualization with bilateral filters*. 2016. URL: `https://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html`.

[Ues+18] Jonathan Uesato et al. "Adversarial Risk and the Dangers of Evaluating Against Weak Attacks". In: *International Conference on Machine Learning (ICML)*. 2018.

[USL19] Berk Ustun, Alexander Spangher, and Yang Liu. "Actionable recourse in linear classification". In: *Proceedings of the Conference on Fairness, Accountability, and Transparency*. 2019.

[Utr+20] Francisco Utrera et al. "Adversarially-Trained Deep Nets Transfer Better". In: *ArXiv preprint arXiv:2007.05869*. 2020.

[UVL17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Deep Image Prior". In: *ArXiv preprint arXiv:1711.10925*. 2017.

[Vig+20] Jesse Vig et al. "Causal mediation analysis for interpreting neural nlp: The case of gender bias". In: *arXiv preprint arXiv:2004.12265* (2020).

[Vin+10] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of machine learning research (JMLR)*. 2010.

[VKK16] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *International Conference on Machine Learning (ICML)*. 2016.

[Wal45] Abraham Wald. "Statistical Decision Functions Which Minimize the Maximum Risk". In: *Annals of Mathematics*. 1945.

[Wan+17] Xiaosong Wang et al. "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases". In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2017.

[Wan+18] Alex Wang et al. "GLUE: A multi-task benchmark and analysis platform for natural language understanding". In: *arXiv preprint arXiv:1804.07461* (2018).

[WK18] Eric Wong and J Zico Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: *International Conference on Machine Learning (ICML)*. 2018.

[WLS20] Zongze Wu, Dani Lischinski, and Eli Shechtman. "StyleSpace Analysis: Disentangled Controls for StyleGAN Image Generation". In: *Computer Vision and Pattern Recognition (CVPR)*. 2020.

[WSK19] Eric Wong, Frank Schmidt, and Zico Kolter. "Wasserstein adversarial examples via projected sinkhorn iterations". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6808–6817.

[WSM21] Eric Wong, Shibani Santurkar, and Aleksander Madry. "Leveraging Sparse Linear Layers for Debuggable Deep Networks". In: *Arxiv preprint arXiv:2105.04857*. 2021.

[WTD17] Ellery Wulczyn, Nithum Thain, and Lucas Dixon. "Ex machina: Personal attacks seen at scale". In: *Proceedings of the 26th International Conference on World Wide Web*. 2017, pp. 1391–1399.

[Wu+16] Yuxin Wu et al. *Tensorpack*. `https://github.com/tensorpack/`. 2016.

[Xia+18] Chaowei Xiao et al. "Spatially Transformed Adversarial Examples". In: *International Conference on Learning Representations (ICLR)*. 2018.

[Xia+19] Kai Y. Xiao et al. "Training for Faster Adversarial Robustness Verification via Inducing ReLU Stability". In: *International Conference on Learning Representations (ICLR)*. 2019.

[Xia+20] Kai Xiao et al. "Noise or signal: The role of image backgrounds in object recognition". In: *arXiv preprint arXiv:2006.09994* (2020).

[Yeh+17] Raymond A Yeh et al. "Semantic image inpainting with deep generative models". In: *Computer Vision and Pattern Recognition (CVPR)*. 2017.

[Yeh+20] Chih-Kuan Yeh et al. "On Completeness-aware Concept-Based Explanations in Deep Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2020).

[Yin+19] Dong Yin et al. "A fourier perspective on model robustness in computer vision". In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[Yos+14] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems (NeurIPS)*. 2014.

[Yos+15] Jason Yosinski et al. "Understanding neural networks through deep visualization". In: *arXiv preprint arXiv:1506.06579*. 2015.

[Yu+18] Jiahui Yu et al. "Generative image inpainting with contextual attention". In: *Computer Vision and Pattern Recognition (CVPR)*. 2018.

[ZF14] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: (2014).

[ZH05] Hui Zou and Trevor Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the royal statistical society: series B (statistical methodology)* (2005).

[Zha+07] Jianguo Zhang et al. "Local features and kernels for classification of texture and object categories: A comprehensive study". In: *International journal of computer vision*. 2007.

[Zha+18a] Han Zhang et al. "Self-attention generative adversarial networks". In: *arXiv preprint arXiv:1805.08318*. 2018.

[Zha+18b] Richard Zhang et al. "The unreasonable effectiveness of deep features as a perceptual metric". In: *Computer Vision and Pattern Recognition (CVPR)*. 2018.

[Zha+21] Linjun Zhang et al. "How Does Mixup Help With Robustness and Generalization?" In: *International Conference on Learning Representations (ICLR)*. 2021.

[Zho+17] Bolei Zhou et al. “Places: A 10 million image database for scene recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* (2017).

[Zho+18] Bolei Zhou et al. “Interpretable basis decomposition for visual explanation”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 119–134.

[Zhu+17] Jun-Yan Zhu et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: *international conference on computer vision(ICCV)*. 2017.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).

[Zop+18] Barret Zoph et al. “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.

Part III

Appendices

Appendix A

Details for Chapter 2

A.1 Experimental Setup

A.1.1 Datasets

For our experimental analysis, we use the CIFAR-10 [Kri09] and (restricted) ImageNet [Rus+15] datasets. Attaining robust models for the complete ImageNet dataset is known to be a challenging problem, both due to the hardness of the learning problem itself, as well as the computational complexity. We thus restrict our focus to a subset of the dataset which we denote as restricted ImageNet. To this end, we group together semantically similar classes from ImageNet into 9 super-classes shown in Table A.1. We train and evaluate only on examples corresponding to these classes.

A.1.2 Models

We use the ResNet-50 architecture for our baseline standard and adversarially trained classifiers on CIFAR-10 and restricted ImageNet. For each model, we grid search over three learning rates (0.1, 0.01, 0.05), two batch sizes (128, 256) including/not including a learning rate drop (a single order of magnitude) and data augmentation. We use the standard training parameters for the remaining parameters. The hyperparameters used for each model are given in Tables A.2 and A.3.

Class	**Corresponding ImageNet Classes**
"Dog"	151 to 268
"Cat"	281 to 285
"Frog"	30 to 32
"Turtle"	33 to 37
"Bird"	80 to 100
"Primate"	365 to 382
"Fish"	389 to 397
"Crab"	118 to 121
"Insect"	300 to 319

Table A.1: Classes used in the Restricted ImageNet model. The class ranges are inclusive.

Dataset	LR	Batch Size	LR Drop	Data Aug.	Momentum	Weight Decay
$\widehat{\mathcal{D}}_R$	0.1	128	Yes	Yes	0.9	$5 \cdot 10^{-4}$
$\widehat{\mathcal{D}}_{NR}$	0.1	128	Yes	Yes	0.9	$5 \cdot 10^{-4}$
$\widehat{\mathcal{D}}_{rand}$	0.01	128	Yes	Yes	0.9	$5 \cdot 10^{-4}$
$\widehat{\mathcal{D}}_{det}$	0.1	128	Yes	No	0.9	$5 \cdot 10^{-4}$

Table A.2: Hyperparameters for the models trained on the CIFAR dataset. All hyperparameters were obtained through a grid search.

Dataset	LR	Batch Size	LR Drop	Data Aug.	Momentum	Weight Decay
$\widehat{\mathcal{D}}_R$	0.01	128	No	Yes	0.9	$5 \cdot 10^{-4}$
$\widehat{\mathcal{D}}_{rand}$	0.01	256	No	No	0.9	$5 \cdot 10^{-4}$
$\widehat{\mathcal{D}}_{det}$	0.05	256	No	No	0.9	$5 \cdot 10^{-4}$

Table A.3: Hyperparameters for the models trained on the Restricted ImageNet dataset. All hyperparameters were obtained through a grid search.

A.1.3 Adversarial training

To obtain robust classifiers, we employ the adversarial training methodology proposed in [Mad+18]. Specifically, we train against a projected gradient descent (PGD) adversary constrained in ℓ_2-norm starting from the original image. Following Madry et al. [Mad+18] we normalize the gradient at each step of PGD to ensure that we move a fixed distance in ℓ_2-norm per step. Unless otherwise specified, we use the values of ϵ provided in Table A.4 to train/evaluate our models. We used 7 steps of PGD with a step size of $\varepsilon/5$.

Adversary	**CIFAR-10**	**Restricted Imagenet**
ℓ_2	0.5	3

Table A.4: Value of ϵ used for ℓ_2 adversarial training/evaluation of each dataset.

A.1.4 Constructing a Robust Dataset

In Section 2.4.1, we describe a procedure to construct a dataset that contains features relevant only to a given (standard/robust) model. To do so, we optimize the training objective in (2.7). Unless otherwise specified, we initialize x_r as a different randomly chosen sample from the training set. (For the sake of completeness, we also try initializing with a Gaussian noise instead as shown in Table A.7.) We then perform normalized gradient descent (ℓ_2-norm of gradient is fixed to be constant at each step). At each step we clip the input x_r to in the $[0, 1]$ range so as to ensure that it is a valid image. Details on the optimization procedure are shown in Table A.5. We provide the pseudocode for the construction in Figure A-1.

	CIFAR-10	**Restricted Imagenet**
step size	0.1	1
iterations	1000	2000

Table A.5: Parameters used for optimization procedure to construct dataset in Section 2.4.1.

```
GetRobustDataset($D$)

  1. $C_R \leftarrow$ AdversarialTraining($D$)
     $g_R \leftarrow$ mapping learned by $C_R$ from the input to the representation layer

  2. $D_R \leftarrow \{\}$

  3. For $(x, y) \in D$
         $x' \sim D$
         $x_R \leftarrow \arg\min_{z \in [0,1]^d} \|g_R(z) - g_R(x)\|_2$
     # Solved using $\ell_2$-PGD starting from $x'$
         $D_R \leftarrow D_R \cup \{(x_R, y)\}$

  4. Return $D_R$
```

Figure A-1: Algorithm to construct a "robust" dataset, by restricting to features used by a robust model.

A.1.5 Non-robust features suffice for standard classification

To construct the dataset as described in Section 2.4.2, we use the standard projected gradient descent (PGD) procedure described in [Mad+18] to construct an adversarial example for a given input from the dataset (2.8). Perturbations are constrained in ℓ_2-norm while each PGD step is normalized to a fixed step size. The details for our PGD setup are described in Table A.6. We provide pseudocode in Figure A-2.

```
GetNonRobustDataset($(D, \varepsilon)$)
  1. $D_{NR} \leftarrow \{\}$
  2. $C \leftarrow$ StandardTraining($D$)
  3. For $(x, y) \in D$
         $t \overset{\text{uar}}{\sim} [C]$        # or $t \leftarrow (y+1) \mod C$
         $x_{NR} \leftarrow \min_{||x'-x|| \leq \varepsilon} L_C(x', t)$ # Solved using $\ell_2$ PGD
         $D_{NR} \leftarrow D_{NR} \cup \{(x_{NR}, t)\}$
  4. Return $D_{NR}$
```

Figure A-2: Algorithm to construct a dataset where input-label association is based entirely on non-robust features.

Attack Parameters	CIFAR-10	Restricted Imagenet
ε	0.5	3
step size	0.1	0.1
iterations	100	100

Table A.6: Projected gradient descent parameters used to construct constrained adversarial examples in Section 2.4.2.

A.2 Omitted Experiments and Figures

A.2.1 Detailed evaluation of models trained on "robust" dataset

In Section 2.4.1, we generate a "robust" training set by restricting the dataset to only contain features relevant to a robust model (robust dataset) or a standard model (non-robust dataset). This is performed by choosing either a random input from the training set or random noise[1] and then performing the optimization procedure described in (2.7). The performance of these classifiers along with various baselines is shown in Table A.7. We observe that while the robust dataset constructed from noise resembles the original, the corresponding non-robust does not (Figure A-3). This also leads to suboptimal performance of classifiers trained on this dataset (only 46% standard accuracy) potentially due to a distributional shift.

		Robust Accuracy	
Model	**Accuracy**	$\varepsilon = 0.25$	$\varepsilon = 0.5$
Standard Training	95.25 %	4.49%	0.0%
Robust Training	90.83%	82.48%	70.90%
Trained on non-robust dataset (constructed from images)	**87.68%**	0.82%	0.0%
Trained on non-robust dataset (constructed from noise)	**45.60%**	1.50%	0.0%
Trained on robust dataset (constructed from images)	85.40%	**48.20 %**	21.85%
Trained on robust dataset (constructed from noise)	84.10%	**48.27 %**	29.40%

Table A.7: Standard and robust classification performance on the CIFAR-10 test set of: an (i) ERM classifier; (ii) ERM classifier trained on a dataset obtained by distilling features relevant to ERM classifier in (i); (iii) adversarially trained classifier ($\varepsilon = 0.5$); (iv) ERM classifier trained on dataset obtained by distilling features used by robust classifier in (iii). Simply restricting the set of available features during ERM to features used by a standard model yields non-trivial robust accuracy.

[1]We use 10k steps to construct the dataset from noise, instead to using 1k steps done when the input is a different training set image (cf. Table A.5).

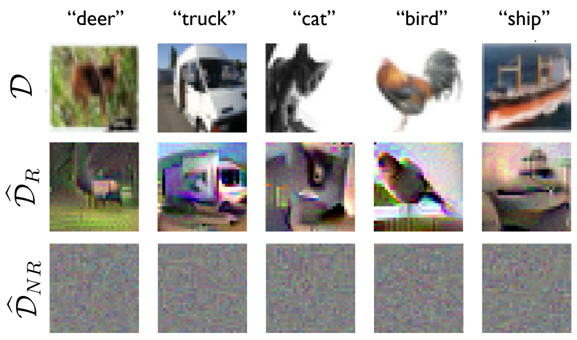


Figure A-3: Robust and non-robust datasets for CIFAR-10 when the process starts from noise (as opposed to random images as in Figure 2-2a).

A.2.2 Performance of "robust" training and test set

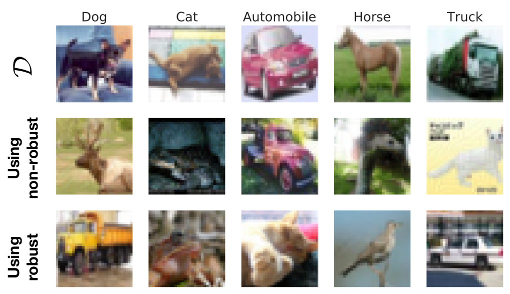
In Section 2.4.1, we observe that an ERM classifier trained on a "robust" training dataset $\widehat{\mathcal{D}}_R$ (obtained by restricting features to those relevant to a robust model) attains non-trivial robustness (cf. Figure 2-1 and Table A.7). In Table A.8, we evaluate the adversarial accuracy of the model on the corresponding robust training set (the samples which the classifier was trained on) and test set (unseen samples from $\widehat{\mathcal{D}}_R$, based on the test set). We find that the drop in robustness comes from a combination of generalization gap (the robustness on the $\widehat{\mathcal{D}}_R$ test set is worse than it is on the robust training set) and distributional shift (the model performs better on the robust test set consisting of unseen samples from $\widehat{\mathcal{D}}_R$ than on the standard test set containing unseen samples from $\mathcal{D}$).

Dataset	Robust Accuracy
Robust training set	77.33%
Robust test set	62.49%
Standard test set	48.27%

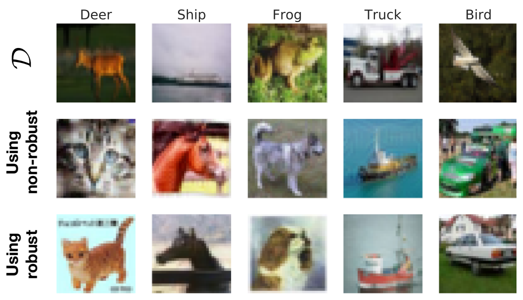
Table A.8: Performance of model trained on the *robust dataset* on the robust training and test sets as well as the standard CIFAR-10 test set. We observe that the drop in robust accuracy stems from a combination of generalization gap and distributional shift. The adversary is constrained to $\varepsilon = 0.25$ in ℓ_2-norm.

A.2.3 Classification based on non-robust features

Figure A-4 shows sample images from $\mathcal{D}$, $\widehat{\mathcal{D}}_{rand}$ and $\widehat{\mathcal{D}}_{det}$ constructed using a standard (non-robust) ERM classifier, and an adversarially trained (robust) classifier.



(a) $\widehat{\mathcal{D}}_{rand}$



(b) $\widehat{\mathcal{D}}_{det}$

Figure A-4: Random samples from datasets where the input-label correlation is entirely based on non-robust features. Samples are generated by performing small adversarial perturbations using either random ($\widehat{\mathcal{D}}_{rand}$) or deterministic ($\widehat{\mathcal{D}}_{det}$) label-target mappings for every sample in the training set. Each image shows: *top*: original; *middle*: adversarial perturbations using a standard ERM-trained classifier; *bottom*: adversarial perturbations using a robust classifier (adversarially trained against $\varepsilon = 0.5$).

In Table A.9, we repeat the experiments in Table 2.1 based on datasets constructed using a robust model. Note that using a robust model to generate the $\widehat{\mathcal{D}}_{det}$ and $\widehat{\mathcal{D}}_{rand}$ datasets will not result in non-robust features that are strongly predictive of t (since the prediction of the classifier will not change). Thus, training a model on these datasets leads to poor accuracy on the standard test set from $\mathcal{D}$.

Model used to construct dataset	**Dataset used in training**		
	$\mathcal{D}$	$\widehat{\mathcal{D}}_{rand}$	$\widehat{\mathcal{D}}_{det}$
Robust	95.3%	25.2 %	5.8%
Standard	95.3%	63.3 %	43.7%

Table A.9: Repeating the experiments of Table 2.1 using a robust model to construct the datasets $\mathcal{D}$, $\widehat{\mathcal{D}}_{rand}$ and $\widehat{\mathcal{D}}_{det}$. Results in Table 2.1 are reiterated for comparison.

A.2.4 Performance of ERM classifiers on relabeled test set

In Table A.10), we evaluate the performance of classifiers trained on $\widehat{\mathcal{D}}_{det}$ on both the original test set drawn from $\mathcal{D}$, and the test set relabelled using $t(y) = (y+1)$ mod C. Observe that the classifier trained on $\widehat{\mathcal{D}}_{det}$ constructed using a robust model actually ends up learning permuted labels based on robust features (indicated by high test accuracy on the relabelled test set).

Model used to construct training dataset for $\widehat{\mathcal{D}}_{det}$	**Dataset used in testing**	
	$\mathcal{D}$	relabelled-$\mathcal{D}$
Standard	43.7%	16.2%
Robust	5.8%	65.5%

Table A.10: Performance of classifiers trained using $\widehat{\mathcal{D}}_{det}$ training set constructed using either standard or robust models. The classifiers are evaluated both on the standard test set from $\mathcal{D}$ and the test set relabeled using $t(y) = (y+1) \mod C$. We observe that using a robust model for the construction results in a model that largely predicts the permutation of labels, indicating that the dataset does not have strongly predictive non-robust features.

A.2.5 Generalization to CIFAR-10.1

Recht et al. [Rec+19a] have constructed an unseen but distribution-shifted test set for CIFAR-10. They show that for many previously proposed models, accuracy on the CIFAR-10.1 test set can be predicted as a linear function of performance on the CIFAR-10 test set.

As a sanity check (and a safeguard against any potential adaptive overfitting to the test set via hyperparameters, historical test set reuse, etc.) we note that the classifiers trained on $\widehat{\mathcal{D}}_{det}$ and $\widehat{\mathcal{D}}_{rand}$ achieve 44% and 55% generalization on the CIFAR-10.1 test set, respectively. This demonstrates non-trivial generalization, and actually perform better than the linear fit would predict (given their accuracies on the CIFAR-10 test set).

A.2.6 Omitted Results for Restricted ImageNet

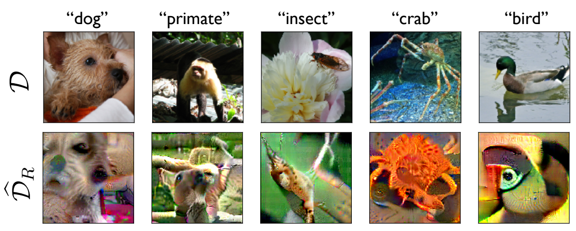


Figure A-5: Repeating the experiments shown in Figure 2-2 for the Restricted ImageNet dataset. Sample images from the resulting dataset.

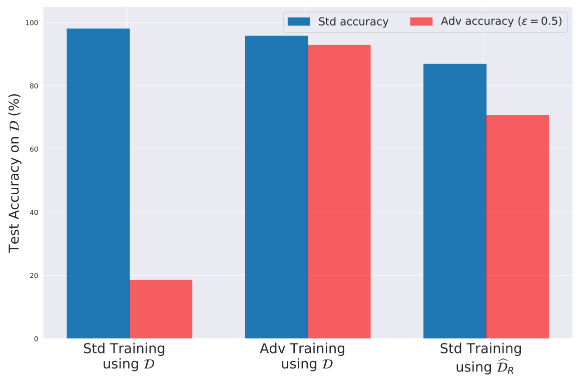


Figure A-6: Repeating the experiments shown in Figure 2-2 for the Restricted ImageNet dataset. Standard and robust accuracy of models trained on these datasets.

A.2.7 Robustness vs. Accuracy

A.3 Gaussian MLE under Adversarial Perturbation

In this section, we develop a framework for studying non-robust features by studying the problem of *maximum likelihood classification* between two Gaussian distributions. We first recall the setup of the problem, then present the main theorems from Section 2.5. First we build the techniques necessary for their proofs.

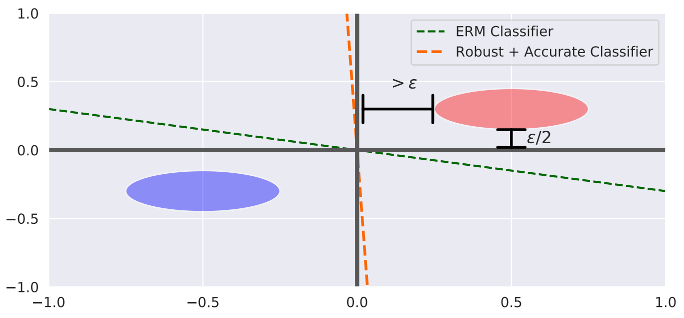


Figure A-7: An example where adversarial vulnerability can arise from ERM training on any standard loss function due to non-robust features (the green line shows the ERM-learned decision boundary). There exists, however, a classifier that is both perfectly robust *and* accurate, resulting from robust training, which forces the classifier to ignore the x_2 feature despite its predictiveness.

A.3.1 Setup

We consider the setup where a learner receives labeled samples from two distributions, $\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$, and $\mathcal{N}(-\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$. The learner's goal is to be able to classify new samples as being drawn from $\mathcal{D}_1$ or $\mathcal{D}_2$ according to a maximum likelihood (MLE) rule.

A simple coupling argument demonstrates that this problem can actually be reduced to learning the parameters $\widehat{\boldsymbol{\mu}}, \widehat{\boldsymbol{\Sigma}}$ of a single Gaussian $\mathcal{N}(-\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$, and then employing a linear classifier with weight $\widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\boldsymbol{\mu}}$. In the standard setting, maximum likelihoods estimation learns the true parameters, $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$, and thus the learned classification rule is $C(x) = \mathbb{1}\{x^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} > 0\}$.

In this work, we consider the problem of *adversarially robust* maximum likelihood estimation. In particular, rather than simply being asked to classify samples, the learner will be asked to classify *adversarially perturbed* samples $x + \delta$, where $\delta \in \Delta$ is chosen to maximize the loss of the learner. Our goal is to derive the parameters $\boldsymbol{\mu}, \boldsymbol{\Sigma}$ corresponding to an adversarially robust maximum likelihood estimate of the parameters of $\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$. Note that since we have access to $\boldsymbol{\Sigma}_*$ (indeed, the learner can just run non-robust MLE to get access), we work in the space where $\boldsymbol{\Sigma}^*$ is a diagonal matrix, and we restrict the learned covariance $\boldsymbol{\Sigma}$ to the set of diagonal matrices.

Notation. We denote the parameters of the sampled Gaussian by $\boldsymbol{\mu}_* \in \mathbb{R}^d$, and $\boldsymbol{\Sigma}_* \in \{\mathrm{diag}(\boldsymbol{u}) | \boldsymbol{u} \in \mathbb{R}^d\}$. We use $\sigma_{min}(X)$ to represent the smallest eigenvalue of a square matrix X, and $\ell(\cdot; x)$ to represent the Gaussian negative log-likelihood for a single sample x. For convenience, we often use $\boldsymbol{v} = x - \boldsymbol{\mu}$, and $R = \|\boldsymbol{\mu}_*\|$. We also define the $\otimes$ operator to represent the vectorization of the diagonal of a matrix. In particular, for a matrix $X \in \mathbb{R}^{d \times d}$, we have that $X_{\otimes} = v \in \mathbb{R}^d$ if $v_i = X_{ii}$.

A.3.2 Outline and Key Results

We focus on the case where $\Delta = \mathcal{B}_2(\epsilon)$ for some $\epsilon > 0$, i.e. the ℓ_2 ball, corresponding to the following minimax problem:

$$\min_{\boldsymbol{\mu}, \boldsymbol{\Sigma}} \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)} \left[\max_{\delta : \|\delta\| = \varepsilon} \ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}; x + \delta) \right] \tag{A.1}$$

We first derive the optimal adversarial perturbation for this setting (Section A.3.3), and prove Theorem 1 (Section A.3.3). We then propose an alternate problem, in which the adversary picks a linear operator to be applied to a fixed vector, rather than picking a specific perturbation vector (Section A.3.3). We argue via Gaussian concentration that the alternate problem is indeed reflective of the original model (and in particular, the two become equivalent as $d \to \infty$). In particular, we propose studying the following in place of (A.1):

$$\begin{aligned} &\min_{\boldsymbol{\mu}, \boldsymbol{\Sigma}} \max_{M \in \mathcal{M}} \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)} \left[\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}; x + M(x - \boldsymbol{\mu})) \right] \qquad (A.2) \\ &\text{where } \mathcal{M} = \left\{ M \in \mathbb{R}^{d \times d} : \ M_{ij} = 0 \, \forall \, i \neq j, \ \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)} \left[\|M\boldsymbol{v}\|_2^2 \right] = \epsilon^2 \right\}. \end{aligned}$$

Our goal is to characterize the behavior of the robustly learned covariance $\boldsymbol{\Sigma}$ in terms of the true covariance matrix $\boldsymbol{\Sigma}_*$ and the perturbation budget ε. The proof is through Danskin's Theorem, which allows us to use any maximizer of the inner problem M^* in computing the subgradient of the inner minimization. After showing the applicability of Danskin's Theorem (Section A.3.3) and then applying it (Section A.3.3) to prove our main results (Section A.3.3). Our three main results, which we prove in the following section, are presented below.

First, we consider a simplified version of (A.1), in which the adversary solves a maximization with a fixed Lagrangian penalty, rather than a hard ℓ_2 constraint. In this setting, we show that the loss contributed by the adversary corresponds to a misalignment between the data metric (the Mahalanobis distance, induced by $\mathbf{\Sigma}^{-1}$), and the ℓ_2 metric:

Theorem 1 (Adversarial vulnerability from misalignment)**.** *Consider an adversary whose perturbation is determined by the "Lagrangian penalty" form of* (2.13)*, i.e.*

$$\max_{\delta} \ell(x+\delta; y \cdot \boldsymbol{\mu}, \mathbf{\Sigma}) - C \cdot \|\delta\|_2,$$

where $C \geq \frac{1}{\sigma_{min}(\mathbf{\Sigma}_*)}$ is a constant trading off NLL minimization and the adversarial constraint[2]*. Then, the adversarial loss $\mathcal{L}_{adv}$ incurred by the non-robustly learned $(\boldsymbol{\mu}, \mathbf{\Sigma})$ is given by:*

$$\mathcal{L}_{adv}(\Theta) - \mathcal{L}(\Theta) = tr\left[\left(I + (C \cdot \mathbf{\Sigma}_* - I)^{-1}\right)^2\right] - d,$$

and, for a fixed $tr(\mathbf{\Sigma}_*) = k$ the above is minimized by $\mathbf{\Sigma}_* = \frac{k}{d}\boldsymbol{I}$.

We then return to studying (A.2), where we provide upper and lower bounds on the learned robust covariance matrix $\mathbf{\Sigma}$:

Theorem 2 (Robustly Learned Parameters)**.** *Just as in the non-robust case, $\boldsymbol{\mu}_r = \boldsymbol{\mu}^*$, i.e. the true mean is learned. For the robust covariance $\mathbf{\Sigma}_r$, there exists an $\varepsilon_0 > 0$, such that for any $\varepsilon \in [0, \varepsilon_0)$,*

$$\mathbf{\Sigma}_r = \frac{1}{2}\mathbf{\Sigma}_* + \frac{1}{\lambda} \cdot \boldsymbol{I} + \sqrt{\frac{1}{\lambda} \cdot \mathbf{\Sigma}_* + \frac{1}{4}\mathbf{\Sigma}_*^2},$$
$$\text{where} \qquad \Omega\left(\frac{1+\varepsilon^{1/2}}{\varepsilon^{1/2}+\varepsilon^{3/2}}\right) \leq \lambda \leq O\left(\frac{1+\varepsilon^{1/2}}{\varepsilon^{1/2}}\right).$$

Finally, we show that in the worst case over mean vectors $\boldsymbol{\mu}_*$, the gradient of the adversarial robust classifier aligns more with the inter-class vector:

[2]The constraint on C is to ensure the problem is concave.

Theorem 3 (Gradient alignment). *Let $f(x)$ and $f_r(x)$ be monotonic classifiers based on the linear separator induced by standard and ℓ_2-robust maximum likelihood classification, respectively. The maximum angle formed between the gradient of the classifier (wrt input) and the vector connecting the classes can be smaller for the robust model:*

$$\min_{\mu} \frac{\langle \boldsymbol{\mu}, \nabla_x f_r(x) \rangle}{\|\boldsymbol{\mu}\| \cdot \|\nabla_x f_r(x)\|} > \min_{\mu} \frac{\langle \boldsymbol{\mu}, \nabla_x f(x) \rangle}{\|\boldsymbol{\mu}\| \cdot \|\nabla_x f(x)\|}.$$

A.3.3 Proofs

In the first section, we have shown that the classification between two Gaussian distributions with identical covariance matrices centered at $\boldsymbol{\mu}^*$ and $-\boldsymbol{\mu}^*$ can in fact be reduced to learning the parameters of a single one of these distributions.

Thus, in the standard setting, our goal is to solve the following problem:

$$\min_{\boldsymbol{\mu},\boldsymbol{\Sigma}} \mathbb{E}_{x\sim\mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)} \left[\ell(\boldsymbol{\mu},\boldsymbol{\Sigma}; x)\right] := \min_{\boldsymbol{\mu},\boldsymbol{\Sigma}} \mathbb{E}_{x\sim\mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)} \left[-\log\left(\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma}; x)\right)\right].$$

Note that in this setting, one can simply find differentiate ℓ with respect to both $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, and obtain closed forms for both (indeed, these closed forms are, unsurprisingly, $\boldsymbol{\mu}^*$ and $\boldsymbol{\Sigma}^*$). Here, we consider the existence of a *malicious adversary* who is allowed to perturb each sample point x by some δ. The goal of the adversary is to *maximize* the same loss that the learner is minimizing.

Motivating example: ℓ_2-constrained adversary

We first consider, as a motivating example, an ℓ_2-constrained adversary. That is, the adversary is allowed to perturb each sampled point by $\delta : \|\delta\|_2 = \varepsilon$. In this case, the minimax problem being solved is the following:

$$\min_{\boldsymbol{\mu},\boldsymbol{\Sigma}} \mathbb{E}_{x\sim\mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)} \left[\max_{\|\delta\|=\varepsilon} \ell(\boldsymbol{\mu},\boldsymbol{\Sigma}; x+\delta)\right]. \tag{A.3}$$

The following Lemma captures the optimal behaviour of the adversary:

Lemma 1. *In the minimax problem captured in* (A.3) *(and earlier in* (A.1)*), the optimal adversarial perturbation δ^* is given by*

$$\delta^* = \left(\lambda \boldsymbol{I} - \boldsymbol{\Sigma}^{-1}\right)^{-1} \boldsymbol{\Sigma}^{-1} \boldsymbol{v} = \left(\lambda \boldsymbol{\Sigma} - \boldsymbol{I}\right)^{-1} \boldsymbol{v}, \tag{A.4}$$

where $\boldsymbol{v} = x - \boldsymbol{\mu}$, and λ is set such that $\|\delta^*\|_2 = \varepsilon$.

Proof. In this context, we can solve the inner maximization problem with Lagrange multipliers. In the following we write $\Delta = \mathcal{B}_2(\varepsilon)$ for brevity, and discard terms not containing δ as well as constant factors freely:

$$\begin{aligned}\arg\max_{\delta \in \Delta}\ \ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}; x + \delta) - &= \arg\max_{\delta \in \Delta} \left(x + \delta - \boldsymbol{\mu}\right)^\top \boldsymbol{\Sigma}^{-1} \left(x + \delta - \boldsymbol{\mu}\right) \\ &= \arg\max_{\delta \in \Delta}\ (x - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (x - \boldsymbol{\mu}) + 2\delta^\top \boldsymbol{\Sigma}^{-1} (x - \boldsymbol{\mu}) + \delta^\top \boldsymbol{\Sigma}^{-1} \delta \\ &= \arg\max_{\delta \in \Delta}\ \delta^\top \boldsymbol{\Sigma}^{-1} (x - \boldsymbol{\mu}) + \frac{1}{2} \delta^\top \boldsymbol{\Sigma}^{-1} \delta. \end{aligned} \tag{A.5}$$

Now we can solve (A.5) using the aforementioned Lagrange multipliers. In particular, note that the maximum of (A.5) is attained at the boundary of the ℓ_2 ball Δ. Thus, we can solve the following system of two equations to find δ, rewriting the norm constraint as $\frac{1}{2}\|\delta\|_2^2 = \frac{1}{2}\varepsilon^2$:

$$\begin{cases} \nabla_\delta \left(\delta^\top \boldsymbol{\Sigma}^{-1} (x - \boldsymbol{\mu}) + \frac{1}{2} \delta^\top \boldsymbol{\Sigma}^{-1} \delta\right) = \lambda \nabla_\delta \left(\|\delta\|_2^2 - \varepsilon^2\right) \implies \boldsymbol{\Sigma}^{-1}(x - \boldsymbol{\mu}) + \boldsymbol{\Sigma}^{-1}\delta = \lambda\delta \\ \|\delta\|_2^2 = \varepsilon^2. \end{cases} \tag{A.6}$$

For clarity, we write $\boldsymbol{v} = x - \boldsymbol{\mu}$: then, combining the above, we have that

$$\delta^* = \left(\lambda \boldsymbol{I} - \boldsymbol{\Sigma}^{-1}\right)^{-1} \boldsymbol{\Sigma}^{-1} \boldsymbol{v} = \left(\lambda \boldsymbol{\Sigma} - \boldsymbol{I}\right)^{-1} \boldsymbol{v}, \tag{A.7}$$

our final result for the maximizer of the inner problem, where λ is set according to the norm constraint. □

Variant with Fixed Lagrangian (Theorem 1)

To simplify the analysis of Theorem 1, we consider a version of (A.3) with a fixed Lagrangian penalty, rather than a norm constraint:

$$\max \ell(x+\delta; y \cdot \boldsymbol{\mu}, \boldsymbol{\Sigma}) - C \cdot \|\delta\|_2.$$

Note then, that by Lemma 1, the optimal perturbation δ^* is given by

$$\delta^* = \left(C\boldsymbol{\Sigma} - \boldsymbol{I}\right)^{-1}.$$

We now proceed to the proof of Theorem 1.

Theorem 1 (Adversarial vulnerability from misalignment)**.** *Consider an adversary whose perturbation is determined by the "Lagrangian penalty" form of* (2.13)*, i.e.*

$$\max_{\delta} \ell(x+\delta; y \cdot \boldsymbol{\mu}, \boldsymbol{\Sigma}) - C \cdot \|\delta\|_2,$$

where $C \geq \frac{1}{\sigma_{min}(\boldsymbol{\Sigma}_*)}$ is a constant trading off NLL minimization and the adversarial constraint[3]*. Then, the adversarial loss $\mathcal{L}_{adv}$ incurred by the non-robustly learned $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by:*

$$\mathcal{L}_{adv}(\Theta) - \mathcal{L}(\Theta) = tr\left[\left(I + \left(C \cdot \boldsymbol{\Sigma}_* - I\right)^{-1}\right)^2\right] - d,$$

and, for a fixed $tr(\boldsymbol{\Sigma}_*) = k$ the above is minimized by $\boldsymbol{\Sigma}_* = \frac{k}{d}\boldsymbol{I}$.

Proof. We begin by expanding the Gaussian negative log-likelihood for the relaxed

[3]The constraint on C is to ensure the problem is concave.

problem:

$$\begin{aligned}\mathcal{L}_{adv}(\Theta) - \mathcal{L}(\Theta) &= \mathbb{E}_{x\sim\mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)}\Big[2\cdot \boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}-\boldsymbol{I}\right)^{-\top}\boldsymbol{\Sigma}^{-1}\boldsymbol{v} \\ &\quad + \boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}-\boldsymbol{I}\right)^{-\top}\boldsymbol{\Sigma}^{-1}\left(C\cdot\boldsymbol{\Sigma}-\boldsymbol{I}\right)^{-1}\boldsymbol{v}\Big] \\ &= \mathbb{E}_{x\sim\mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)}\Big[2\cdot \boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}\boldsymbol{\Sigma}-\boldsymbol{\Sigma}\right)^{-1}\boldsymbol{v} \\ &\quad + \boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}-\boldsymbol{I}\right)^{-\top}\boldsymbol{\Sigma}^{-1}\left(C\cdot\boldsymbol{\Sigma}-\boldsymbol{I}\right)^{-1}\boldsymbol{v}\Big]\end{aligned}$$

Recall that we are considering the vulnerability at the MLE parameters $\boldsymbol{\mu}^*$ and $\boldsymbol{\Sigma}^*$:

$$\begin{aligned}\mathcal{L}_{adv}(\Theta) - \mathcal{L}(\Theta) &= \mathbb{E}_{\boldsymbol{v}\sim\mathcal{N}(0,I)}\Big[2\cdot \boldsymbol{v}^\top\boldsymbol{\Sigma}_*^{1/2}\left(C\cdot\boldsymbol{\Sigma}_*^2-\boldsymbol{\Sigma}_*\right)^{-1}\boldsymbol{\Sigma}_*^{1/2}\boldsymbol{v} \\ &\quad + \boldsymbol{v}^\top\boldsymbol{\Sigma}_*^{1/2}\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-\top}\boldsymbol{\Sigma}_*^{-1}\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-1}\boldsymbol{\Sigma}_*^{1/2}\boldsymbol{v}\Big] \\ &= \mathbb{E}_{\boldsymbol{v}\sim\mathcal{N}(0,I)}\Big[2\cdot \boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-1}\boldsymbol{v} \\ &\quad + \boldsymbol{v}^\top\boldsymbol{\Sigma}_*^{1/2}\left(C^2\boldsymbol{\Sigma}_*^3-2C\cdot\boldsymbol{\Sigma}_*^2+\boldsymbol{\Sigma}_*\right)^{-1}\boldsymbol{\Sigma}_*^{1/2}\boldsymbol{v}\Big] \\ &= \mathbb{E}_{\boldsymbol{v}\sim\mathcal{N}(0,I)}\Big[2\cdot \boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-1}\boldsymbol{v}+\boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-2}\boldsymbol{v}\Big] \\ &= \mathbb{E}_{\boldsymbol{v}\sim\mathcal{N}(0,I)}\Big[-\|v\|_2^2+\boldsymbol{v}^\top\boldsymbol{I}\boldsymbol{v}+2\cdot \boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-1}\boldsymbol{v} \\ &\quad + \boldsymbol{v}^\top\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-2}\boldsymbol{v}\Big] \\ &= \mathbb{E}_{\boldsymbol{v}\sim\mathcal{N}(0,I)}\Big[-\|v\|_2^2+\boldsymbol{v}^\top\left(\boldsymbol{I}+\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-1}\right)^2\boldsymbol{v}\Big] \\ &= \operatorname{tr}\left[\left(\boldsymbol{I}+\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-1}\right)^2\right]-d\end{aligned}$$

This shows the first part of the theorem. It remains to show that for a fixed $k = \operatorname{tr}(\boldsymbol{\Sigma}_*)$, the adversarial risk is minimized by $\boldsymbol{\Sigma}_* = \frac{k}{d}\boldsymbol{I}$:

$$\begin{aligned}\min_{\boldsymbol{\Sigma}_*}\ \mathcal{L}_{adv}(\Theta) - \mathcal{L}(\Theta) &= \min_{\boldsymbol{\Sigma}_*}\ \operatorname{tr}\left[\left(\boldsymbol{I}+\left(C\cdot\boldsymbol{\Sigma}_*-\boldsymbol{I}\right)^{-1}\right)^2\right] \\ &= \min_{\{\sigma_i\}}\ \sum_{i=1}^d\left(1+\frac{1}{C\cdot\sigma_i-1}\right)^2,\end{aligned}$$

where $\{\sigma_i\}$ are the eigenvalues of $\boldsymbol{\Sigma}_*$. Now, we have that $\sum\sigma_i = k$ by assumption, so by optimality conditions, we have that $\boldsymbol{\Sigma}_*$ minimizes the above if $\nabla_{\{\sigma_i\}} \propto \vec{1}$, i.e.

if $\nabla_{\sigma_i} = \nabla_{\sigma_j}$ for all i, j. Now,

$$\begin{aligned}\nabla_{\sigma_i} &= -2 \cdot \left(1 + \frac{1}{C \cdot \sigma_i - 1}\right) \cdot \frac{C}{(C \cdot \sigma_i - 1)^2} \\ &= -2 \cdot \frac{C^2 \cdot \sigma_i}{(C \cdot \sigma_i - 1)^3}.\end{aligned}$$

Then, by solving analytically, we find that

$$-2 \cdot \frac{C^2 \cdot \sigma_i}{(C \cdot \sigma_i - 1)^3} = -2 \cdot \frac{C^2 \cdot \sigma_j}{(C \cdot \sigma_j - 1)^3}$$

admits only one real solution, $\sigma_i = \sigma_j$. Thus, $\boldsymbol{\Sigma}_* \propto \boldsymbol{I}$. Scaling to satisfy the trace constraint yields $\boldsymbol{\Sigma}_* = \frac{k}{d}\boldsymbol{I}$, which concludes the proof. □

Real objective

Our motivating example (Section A.3.3) demonstrates that the optimal perturbation for the adversary in the ℓ_2-constrained case is actually a linear function of $\boldsymbol{v}$, and in particular, that the optimal perturbation can be expressed as $D\boldsymbol{v}$ for a diagonal matrix D. Note, however, that the problem posed in (A.3) is not actually a minimax problem, due to the presence of the expectation between the outer minimization and the inner maximization. Motivated by this and (A.7), we define the following robust problem:

$$\begin{aligned}&\min_{\boldsymbol{\mu},\boldsymbol{\Sigma}} \max_{M \in \mathcal{M}} \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)} \left[\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}; x + M\boldsymbol{v})\right], \qquad \text{(A.8)} \\ &\text{where } \mathcal{M} = \left\{ M \in \mathbb{R}^{d \times d} :\ M_{ij} = 0\ \forall\, i \neq j,\ \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)} \left[\|M\boldsymbol{v}\|_2^2\right] = \epsilon^2 \right\}.\end{aligned}$$

First, note that this objective is slightly different from that of (A.3). In the motivating example, δ is constrained to *always* have ε-norm, and thus is normalizer on a per-sample basis inside of the expectation. In contrast, here the classifier is concerned with being robust to perturbations that are linear in $\boldsymbol{v}$, and of ε^2 squared norm *in expectation*.

Note, however, that via the result of Laurent and Massart [LM00] showing strong concentration for the norms of Gaussian random variables, in high dimensions this bound on expectation has a corresponding high-probability bound on the norm. In particular, this implies that as $d \to \infty$, $\|M\boldsymbol{v}\|_2 = \varepsilon$ almost surely, and thus the problem becomes identical to that of (A.3). We now derive the optimal M for a given $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

Lemma 2. *Consider the minimax problem described by (A.8), i.e.*

$$\min_{\boldsymbol{\mu},\boldsymbol{\Sigma}} \max_{M \in \mathcal{M}} \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)} \left[\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}; x + M\boldsymbol{v}) \right].$$

Then, the optimal action M^* of the inner maximization problem is given by

$$M = (\lambda \boldsymbol{\Sigma} - \boldsymbol{I})^{-1}, \tag{A.9}$$

where again λ is set so that $M \in \mathcal{M}$.

Proof. We accomplish this in a similar fashion to what was done for δ^*, using Lagrange multipliers:

$$\begin{aligned}
\nabla_M \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)} \left[\boldsymbol{v}^\top M \boldsymbol{\Sigma}^{-1} \boldsymbol{v} + \frac{1}{2} \boldsymbol{v}^\top M \boldsymbol{\Sigma}^{-1} M \boldsymbol{v} \right] &= \lambda \nabla_M \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)} \left[\|M\boldsymbol{v}\|_2^2 - \varepsilon^2 \right] \\
\mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)} \left[\boldsymbol{\Sigma}^{-1} \boldsymbol{v}\boldsymbol{v}^\top + \boldsymbol{\Sigma}^{-1} M \boldsymbol{v}\boldsymbol{v}^\top \right] &= \mathbb{E}_{x \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)} \left[\lambda M \boldsymbol{v}\boldsymbol{v}^\top \right] \\
\boldsymbol{\Sigma}^{-1} \boldsymbol{\Sigma}^* + \boldsymbol{\Sigma}^{-1} M \boldsymbol{\Sigma}^* &= \lambda M \boldsymbol{\Sigma}^* \\
M &= (\lambda \boldsymbol{\Sigma} - \boldsymbol{I})^{-1},
\end{aligned}$$

where λ is a constant depending on $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ enforcing the expected squared-norm constraint. □

Indeed, note that the optimal M for the adversary takes a near-identical form to the optimal δ (A.7), with the exception that λ is not sample-dependent but rather varies only with the parameters.

Danskin's Theorem

The main tool in proving our key results is Danskin's Theorem [Dan67], a powerful theorem from minimax optimization which contains the following key result:

Theorem 4 (Danskin's Theorem)**.** *Suppose $\phi(x, z) : \mathbb{R} \times Z \to \mathbb{R}$ is a continuous function of two arguments, where $Z \subset \mathbb{R}^m$ is compact. Define $f(x) = \max_{z \in Z} \phi(x, z)$. Then, if for every $z \in Z$, $\phi(x, z)$ is convex and differentiable in x, and $\frac{\partial \phi}{\partial x}$ is continuous:*

The subdifferential of $f(x)$ is given by

$$\partial f(x) = \operatorname{conv}\left\{ \frac{\partial \phi(x, z)}{\partial x} : z \in Z_0(x) \right\},$$

where $\operatorname{conv}(\cdot)$ represents the convex hull operation, and Z_0 is the set of maximizers defined as

$$Z_0(x) = \left\{ \overline{z} : \phi(x, \overline{z}) = \max_{z \in Z} \phi(x, z) \right\}.$$

In short, given a minimax problem of the form $\min_x \max_{y \in C} f(x, y)$ where C is a compact set, if $f(\cdot, y)$ is convex for all values of y, then rather than compute the gradient of $g(x) := \max_{y \in C} f(x, y)$, we can simply find a maximizer y^* for the current parameter x; Theorem 4 ensures that $\nabla_x f(x, y^*) \in \partial_x g(x)$. Note that $\mathcal{M}$ is trivially compact (by the Heine-Borel theorem), and differentiability/continuity follow rather straightforwardly from our reparameterization (c.f. (A.10)), and so it remains to show that the outer minimization is convex for any fixed M.

Convexity of the outer minimization. Note that even in the standard case (i.e. non-adversarial), the Gaussian negative log-likelihood is not convex with respect to $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Thus, rather than proving convexity of this function directly, we employ the parameterization used by [Das+19]: in particular, we write the problem in terms of $\boldsymbol{T} = \boldsymbol{\Sigma}^{-1}$ and $\boldsymbol{m} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$. Under this parameterization, we show that the robust problem is convex for any fixed M.

Lemma 3. *Under the aforementioned parameterization of $\boldsymbol{T} = \boldsymbol{\Sigma}^{-1}$ and $\boldsymbol{m} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$, the* *following "Gaussian robust negative log-likelihood" is convex:*

$$\mathbb{E}_{x\sim\mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)}\left[\ell(\boldsymbol{m},\boldsymbol{T};x+M\boldsymbol{v})\right].$$

Proof. To prove this, we show that the likelihood is convex even with respect to a single sample x; the result follows, since a convex combination of convex functions remains convex. We begin by looking at the likelihood of a single sample $x\sim\mathcal{N}(\boldsymbol{\mu}_*,\boldsymbol{\Sigma}_*)$:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\mu},\boldsymbol{\Sigma};x+M(x-\boldsymbol{\mu})) &= \frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}(x-\boldsymbol{\mu})^\top(I+M)^2\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})\right)\\
&= \frac{\frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}(x-\boldsymbol{\mu})^\top(I+M)^2\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})\right)}{\int\frac{1}{\sqrt{(2\pi)^k|(\boldsymbol{I}+\boldsymbol{M})^{-2}\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}(x-\boldsymbol{\mu})^\top(I+M)^2\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})\right)}\\
&= \frac{|I+M|^{-1}\exp\left(-\frac{1}{2}x^\top(I+M)^2\boldsymbol{\Sigma}^{-1}x+\boldsymbol{\mu}^\top(I+M)^2\boldsymbol{\Sigma}^{-1}x\right)}{\int\exp\left(-\frac{1}{2}x^\top(I+M)^2\boldsymbol{\Sigma}^{-1}x+\boldsymbol{\mu}^\top(I+M)^2\boldsymbol{\Sigma}^{-1}x\right)}
\end{aligned}$$

In terms of the aforementioned $\boldsymbol{T}$ and $\boldsymbol{m}$, and for convenience defining $A=(I+M)^2$:

$$\begin{aligned}
\ell(x) &= |A|^{-1/2}+\left(\frac{1}{2}x^\top A\boldsymbol{T}x-\boldsymbol{m}^\top Ax\right)-\log\left(\int\exp\left(\frac{1}{2}x^\top A\boldsymbol{T}x-\boldsymbol{m}^\top Ax\right)\right)\\
\nabla\ell(x) &= \begin{bmatrix}\frac{1}{2}(Axx^\top)_{\obslash}\\ -Ax\end{bmatrix}-\frac{\int\begin{bmatrix}\frac{1}{2}(Axx^\top)_{\obslash}\\ -Ax\end{bmatrix}\exp\left(\frac{1}{2}x^\top A\boldsymbol{T}x-\boldsymbol{m}^\top Ax\right)}{\int\exp\left(\frac{1}{2}x^\top A\boldsymbol{T}x-\boldsymbol{m}^\top Ax\right)}\\
&= \begin{bmatrix}\frac{1}{2}(Axx^\top)_{\obslash}\\ -Ax\end{bmatrix}-\mathbb{E}_{z\sim\mathcal{N}(\boldsymbol{T}^{-1}\boldsymbol{m},(\boldsymbol{AT})^{-1})}\begin{bmatrix}\frac{1}{2}(Azz^\top)_{\obslash}\\ -Az\end{bmatrix}.
\end{aligned} \tag{A.10}$$

From here, following an identical argument to [Das+19] Equation (3.7), we find that

$$\boldsymbol{H}_\ell = \mathrm{Cov}_{z\sim\mathcal{N}\left(\boldsymbol{T}^{-1}\boldsymbol{m},(A\boldsymbol{T})^{-1}\right)}\left[\left(\begin{matrix}\left(-\frac{1}{2}Azz^T\right)_{\obslash} \\ z\end{matrix}\right),\left(\begin{matrix}\left(-\frac{1}{2}Azz^T\right)_{\obslash} \\ z\end{matrix}\right)\right] \succcurlyeq \mathbf{0},$$

i.e. that the log-likelihood is indeed convex with respect to $\begin{bmatrix}\boldsymbol{T}\\ \boldsymbol{m}\end{bmatrix}$, as desired. □

Applying Danskin's Theorem

The previous two parts show that we can indeed apply Danskin's theorem to the outer minimization, and in particular that the gradient of f at $M = M^*$ is in the subdifferential of the outer minimization problem. We proceed by writing out this gradient explicitly, and then setting it to zero (note that since we have shown f is convex for all choices of perturbation, we can use the fact that a convex function is globally minimized $\iff$ its subgradient contains zero). We continue from above, plugging in (A.9) for M and using (A.10) to write the gradients of ℓ with respect to $\boldsymbol{T}$ and $\boldsymbol{m}$.

$$\begin{aligned}
0 = \nabla_{\begin{bmatrix}\boldsymbol{T}\\ \boldsymbol{m}\end{bmatrix}}\ell &= \mathbb{E}_{x\sim\mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)}\left[\begin{bmatrix}\frac{1}{2}(Axx^\top)_{\obslash}\\ -Ax\end{bmatrix} - \mathbb{E}_{z\sim\mathcal{N}(\boldsymbol{T}^{-1}\boldsymbol{m},(A\boldsymbol{T})^{-1})}\begin{bmatrix}\frac{1}{2}(Azz^\top)_{\obslash}\\ -Az\end{bmatrix}\right] \\
&= \mathbb{E}_{x\sim\mathcal{N}(\boldsymbol{\mu}^*,\boldsymbol{\Sigma}^*)}\begin{bmatrix}\frac{1}{2}(Axx^\top)_{\obslash}\\ -Ax\end{bmatrix} - \mathbb{E}_{z\sim\mathcal{N}(\boldsymbol{T}^{-1}\boldsymbol{m},(A\boldsymbol{T})^{-1})}\begin{bmatrix}\frac{1}{2}(Azz^\top)_{\obslash}\\ -Az\end{bmatrix} \\
&= \begin{bmatrix}\frac{1}{2}(A\boldsymbol{\Sigma}_*)_{\obslash}\\ -A\boldsymbol{\mu}_*\end{bmatrix} - \mathbb{E}_{z\sim\mathcal{N}(\boldsymbol{T}^{-1}\boldsymbol{m},(A\boldsymbol{T})^{-1})}\begin{bmatrix}\frac{1}{2}(A(A\boldsymbol{T})^{-1})_{\obslash}\\ -A\boldsymbol{T}^{-1}\boldsymbol{m}\end{bmatrix} \\
&= \begin{bmatrix}\frac{1}{2}A\boldsymbol{\Sigma}_*\\ -A\boldsymbol{\mu}_*\end{bmatrix} - \begin{bmatrix}\frac{1}{2}A(A\boldsymbol{T})^{-1}\\ -A\boldsymbol{T}^{-1}\boldsymbol{m}\end{bmatrix} \\
&= \begin{bmatrix}\frac{1}{2}A\boldsymbol{\Sigma}_* - \frac{1}{2}\boldsymbol{T}^{-1}\\ A\boldsymbol{T}^{-1}\boldsymbol{m} - A\boldsymbol{\mu}_*\end{bmatrix} \qquad \text{(A.11)}
\end{aligned}$$

Using this fact, we derive an *implicit* expression for the robust covariance matrix

$\boldsymbol{\Sigma}$. Note that for the sake of brevity, we now use M to denote the optimal adversarial perturbation (previously defined as M^* in (A.9)). This implicit formulation forms the foundation of the bounds given by our main results.

Lemma 4. *The minimax problem discussed throughout this work admits the following (implicit) form of solution:*

$$\boldsymbol{\Sigma} = \frac{1}{\lambda} I + \frac{1}{2}\boldsymbol{\Sigma}_* + \sqrt{\frac{1}{\lambda}\boldsymbol{\Sigma}_* + \frac{1}{4}\boldsymbol{\Sigma}_*^2},$$

where λ is such that $M \in \mathcal{M}$, and is thus dependent on $\boldsymbol{\Sigma}$.

Proof. Rewriting (A.11) in the standard parameterization (with respect to $\boldsymbol{\mu}, \boldsymbol{\Sigma}$) and re-expanding $A = (I+M)^2$ yields:

$$0 = \nabla_{\begin{bmatrix} \boldsymbol{T} \\ \boldsymbol{m} \end{bmatrix}} \ell = \begin{bmatrix} \frac{1}{2}(I+M)^2\boldsymbol{\Sigma}_* - \frac{1}{2}\boldsymbol{\Sigma} \\ (I+M)^2\boldsymbol{\mu} - (I+M)^2\boldsymbol{\mu}_* \end{bmatrix}$$

Now, note that the equations involving $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are completely independent, and thus can be solved separately. In terms of $\boldsymbol{\mu}$, the relevant system of equations is $A\boldsymbol{\mu} - A\boldsymbol{\mu}_* = 0$, where multiplying by the inverse A gives that

$$\boldsymbol{\mu} = \boldsymbol{\mu}_*. \tag{A.12}$$

This tells us that the mean learned via ℓ_2-robust maximum likelihood estimation is precisely the true mean of the distribution.

Now, in the same way, we set out to find $\boldsymbol{\Sigma}$ by solving the relevant system of equations:

$$\boldsymbol{\Sigma}_*^{-1} = \boldsymbol{\Sigma}^{-1}(M+I)^2. \tag{A.13}$$

Now, we make use of the Woodbury Matrix Identity in order to write $(I+M)$

as

$$I + (\lambda \mathbf{\Sigma} - I)^{-1} = I + \left(-I - \left(\frac{1}{\lambda} \mathbf{\Sigma}^{-1} - I \right)^{-1} \right) = - \left(\frac{1}{\lambda} \mathbf{\Sigma}^{-1} - I \right)^{-1}.$$

Thus, we can revisit (A.13) as follows:

$$\begin{aligned} \mathbf{\Sigma}_*^{-1} &= \mathbf{\Sigma}^{-1} \left(\frac{1}{\lambda} \mathbf{\Sigma}^{-1} - I \right)^{-2} \\ \frac{1}{\lambda^2} \mathbf{\Sigma}_*^{-1} \mathbf{\Sigma}^{-2} - \left(\frac{2}{\lambda} \mathbf{\Sigma}_*^{-1} + I \right) \mathbf{\Sigma}^{-1} + \mathbf{\Sigma}_*^{-1} &= 0 \\ \frac{1}{\lambda^2} \mathbf{\Sigma}_*^{-1} - \left(\frac{2}{\lambda} \mathbf{\Sigma}_*^{-1} + I \right) \mathbf{\Sigma} + \mathbf{\Sigma}_*^{-1} \mathbf{\Sigma}^2 &= 0 \end{aligned}$$

We now apply the quadratic formula to get an implicit expression for $\mathbf{\Sigma}$ (implicit since technically λ depends on $\mathbf{\Sigma}$):

$$\begin{aligned} \mathbf{\Sigma} &= \left(\frac{2}{\lambda} \mathbf{\Sigma}_*^{-1} + I \pm \sqrt{\frac{4}{\lambda} \mathbf{\Sigma}_*^{-1} + I} \right) \frac{1}{2} \mathbf{\Sigma}_* \\ &= \frac{1}{\lambda} I + \frac{1}{2} \mathbf{\Sigma}_* + \sqrt{\frac{1}{\lambda} \mathbf{\Sigma}_* + \frac{1}{4} \mathbf{\Sigma}_*^2}. \end{aligned} \tag{A.14}$$

This concludes the proof. □

Bounding λ

We now attempt to characterize the shape of λ as a function of ε. First, we use the fact that $\mathbb{E}[\|Xv\|^2] = \operatorname{tr}(X^2)$ for standard normally-drawn v. Thus, λ is set such that $\operatorname{tr}(\mathbf{\Sigma}_* M^2) = \varepsilon$, i.e:

$$\sum_{i=0} \frac{\mathbf{\Sigma}_{ii}^*}{(\lambda \mathbf{\Sigma}_{ii} - 1)^2} = \varepsilon \tag{A.15}$$

Now, consider ε^2 as a function of λ. Observe that for $\lambda \geq \frac{1}{\sigma_{min}(\mathbf{\Sigma})}$, we have that M must be positive semi-definite, and thus ε^2 decays smoothly from ∞ (at $\lambda = \frac{1}{\sigma_{min}}$) to zero (at $\lambda = \infty$). Similarly, for $\lambda \leq \frac{1}{\sigma_{max}(\mathbf{\Sigma})}$, ε decays smoothly as λ *decreases*. Note, however, that such values of λ would necessarily make M *negative semi-definite*, which would actually *help* the log-likelihood. Thus, we can exclude this case; in particular, for the remainder of the proofs, we can assume $\lambda \geq \frac{1}{\sigma_{max}(\mathbf{\Sigma})}$.

Also observe that the zeros of ε in terms of λ are only at $\lambda = \pm\infty$. Using this, we can show that there exists some ε_0 for which, for all $\varepsilon < \varepsilon_0$, the only corresponding possible valid value of λ is where $\lambda \geq \frac{1}{\sigma_{min}}$. This idea is formalized in the following Lemma.

Lemma 5. *For every $\mathbf{\Sigma}_*$, there exists some $\varepsilon_0 > 0$ for which, for all $\varepsilon \in [0, \varepsilon_0)$ the only admissible value of λ is such that $\lambda \geq \frac{1}{\sigma_{min}(\mathbf{\Sigma})}$, and thus such that M is positive semi-definite.*

Proof. We prove the existence of such an ε_0 by lower bounding ε (in terms of λ) for any finite $\lambda > 0$ that does not make M PSD. Providing such a lower bound shows that for small enough ε (in particular, less than this lower bound), the only corresponding values of λ are as desired in the statement[4].

In particular, if M is not PSD, then there must exist at least one index k such that $\lambda\mathbf{\Sigma}_{kk} < 1$, and thus $(\lambda\mathbf{\Sigma}_{kk} - 1)^2 \leq 1$ for all $\lambda > 0$. We can thus lower bound (A.15) as:

$$\varepsilon = \sum_{i=0} \frac{\mathbf{\Sigma}^*_{ii}}{(\lambda\mathbf{\Sigma}_{ii} - 1)^2} \geq \frac{\mathbf{\Sigma}^*_{kk}}{(\lambda\mathbf{\Sigma}_{kk} - 1)^2} \geq \mathbf{\Sigma}^*_{kk} \geq \sigma_{min}(\mathbf{\Sigma}^*) > 0 \tag{A.16}$$

By contradiction, it follows that for any $\varepsilon < \sigma_{min}(\mathbf{\Sigma}_*)^2$, the only admissible λ is such that M is PSD, i.e. according to the statement of the Lemma. $\square$

In the regime $\varepsilon \in [0, \varepsilon_0)$, note that λ is inversely proportional to ε (i.e. as ε grows, λ decreases). This allows us to get a qualitative view of (A.14): as the allowed perturbation value increases, the robust covariance $\mathbf{\Sigma}$ resembles the identity matrix more and more, and thus assigns more and more variance on initially low-variance features. The $\sqrt{\mathbf{\Sigma}_*}$ term indicates that the robust model also adds uncertainty proportional to the square root of the initial variance—thus, low-variance features will have (relatively) more uncertainty in the robust case. Indeed, our main result actually follows as a (somewhat loose) formalization of this intuition.

[4]Since our only goal is existence, we lose many factors from the analysis that would give a tighter bound on ε_0.

Proof of main theorems

First, we give a proof of Theorem 2, providing lower and upper bounds on the learned robust covariance $\boldsymbol{\Sigma}$ in the regime $\varepsilon \in [0, \varepsilon_0)$.

Theorem 2 (Robustly Learned Parameters)**.** *Just as in the non-robust case, $\boldsymbol{\mu}_r = \boldsymbol{\mu}^*$, i.e. the true mean is learned. For the robust covariance $\boldsymbol{\Sigma}_r$, there exists an $\varepsilon_0 > 0$, such that for any $\varepsilon \in [0, \varepsilon_0)$,*

$$\boldsymbol{\Sigma}_r = \tfrac{1}{2}\boldsymbol{\Sigma}_* + \tfrac{1}{\lambda} \cdot \boldsymbol{I} + \sqrt{\tfrac{1}{\lambda} \cdot \boldsymbol{\Sigma}_* + \tfrac{1}{4}\boldsymbol{\Sigma}_*^2},$$
$$\textit{where} \qquad \Omega\left(\tfrac{1+\varepsilon^{1/2}}{\varepsilon^{1/2}+\varepsilon^{3/2}}\right) \leq \lambda \leq O\left(\tfrac{1+\varepsilon^{1/2}}{\varepsilon^{1/2}}\right).$$

Proof. We have already shown that $\boldsymbol{\mu} = \boldsymbol{\mu}_*$ in the robust case (c.f. (A.12)). We choose ε_0 to be as described, i.e. the largest ε for which the set $\{\lambda : \mathrm{tr}(\boldsymbol{\Sigma}_*^2 M) = \varepsilon, \lambda \geq 1/\sigma_{\max}(\boldsymbol{\Sigma})\}$ has only one element λ (which, as we argued, must not be less than $1/\sigma_{min}(\boldsymbol{\Sigma})$). We have argued that such an ε_0 must exist.

We prove the result by combining our early derivation (in particular, (A.13) and (A.14)) with upper and lower bound on λ, which we can compute based on properties of the trace operator. We begin by deriving a lower bound on λ. By linear algebraic manipulation (given in Appendix A.3.3), we get the following bound:

$$\lambda \geq \frac{d}{\mathrm{tr}(\boldsymbol{\Sigma})}\left(1 + \sqrt{\frac{d \cdot \sigma_{min}(\boldsymbol{\Sigma}_*)}{\varepsilon}}\right) \tag{A.17}$$

Now, we can use (A.13) in order to remove the dependency of λ on $\boldsymbol{\Sigma}$:

$$\begin{aligned}
\boldsymbol{\Sigma} &= \boldsymbol{\Sigma}_*(M + I)^2 \\
\mathrm{tr}(\boldsymbol{\Sigma}) &= \mathrm{tr}\left[(\boldsymbol{\Sigma}_*^{1/2}M + \boldsymbol{\Sigma}_*^{1/2})^2\right] \\
&\leq 2 \cdot \mathrm{tr}\left[(\boldsymbol{\Sigma}_*^{1/2}M)^2 + (\boldsymbol{\Sigma}_*^{1/2})^2\right] \\
&\leq 2 \cdot (\varepsilon + \mathrm{tr}(\boldsymbol{\Sigma}_*)).
\end{aligned}$$

Applying this to (A.17) yields:

$$\lambda \geq \frac{d/2}{\varepsilon + \text{tr}(\mathbf{\Sigma}_*)} \left(1 + \sqrt{\frac{d \cdot \sigma_{min}(\mathbf{\Sigma}_*)}{\varepsilon}}\right).$$

Note that we can simplify this bound significantly by writing $\varepsilon = d \cdot \sigma_{min}(\mathbf{\Sigma}_*)\varepsilon' \leq \text{tr}(\mathbf{\Sigma}_*)\varepsilon'$, which does not affect the result (beyond rescaling the valid regime $(0, \varepsilon_0)$), and gives:

$$\lambda \geq \frac{d/2}{(1+\varepsilon')\text{tr}(\mathbf{\Sigma}_*)} \left(1 + \frac{1}{\sqrt{\varepsilon'}}\right) \geq \frac{d \cdot (1+\sqrt{\varepsilon'})}{2\sqrt{\varepsilon'}(1+\varepsilon')\text{tr}(\mathbf{\Sigma}_*)}$$

Next, we follow a similar methodology (Appendix A.3.3) in order to upper bound λ:

$$\lambda \leq \frac{1}{\sigma_{min}(\mathbf{\Sigma})} \left(\sqrt{\frac{\|\mathbf{\Sigma}_*\|_F \cdot d}{\varepsilon}} + 1\right).$$

Note that by (A.13) and positive semi-definiteness of M, it must be that $\sigma_{min}(\mathbf{\Sigma}) \geq \sigma_{min}(\mathbf{\Sigma}_*)$. Thus, we can simplify the previous expression, also substituting $\varepsilon = d \cdot \sigma_{min}(\mathbf{\Sigma}_*)\varepsilon'$:

$$\lambda \leq \frac{1}{\sigma_{min}(\mathbf{\Sigma}_*)} \left(\sqrt{\frac{\|\mathbf{\Sigma}_*\|_F}{\sigma_{min}(\mathbf{\Sigma}_*)\varepsilon'}} + 1\right) = \frac{\|\mathbf{\Sigma}_*\|_F + \sqrt{\varepsilon \cdot \sigma_{min}(\mathbf{\Sigma}_*)}}{\sigma_{min}(\mathbf{\Sigma}_*)^{3/2}\sqrt{\varepsilon}}$$

These bounds can be straightforwardly combined with Lemma 4, which concludes the proof. □

Using this theorem, we can now show Theorem 3:

Theorem 3 (Gradient alignment). *Let $f(x)$ and $f_r(x)$ be monotonic classifiers based on the linear separator induced by standard and ℓ_2-robust maximum likelihood classification, respectively. The maximum angle formed between the gradient of the classifier (wrt input) and the vector connecting the classes can be smaller for the robust model:*

$$\min_{\mu} \frac{\langle \boldsymbol{\mu}, \nabla_x f_r(x) \rangle}{\|\boldsymbol{\mu}\| \cdot \|\nabla_x f_r(x)\|} > \min_{\mu} \frac{\langle \boldsymbol{\mu}, \nabla_x f(x) \rangle}{\|\boldsymbol{\mu}\| \cdot \|\nabla_x f(x)\|}.$$

Proof. To prove this, we make use of the following Lemmas:

Lemma 6. *For two positive definite matrices A and B with $\kappa(A) > \kappa(B)$, we have that $\kappa(A+B) \leq \max\{\kappa(A), \kappa(B)\}$.*

Proof. We proceed by contradiction:

$$\begin{aligned}
\kappa(A+B) &= \frac{\lambda_{max}(A) + \lambda_{max}(B)}{\lambda_{min}(A) + \lambda_{min}(B)} \\
\kappa(A) &= \frac{\lambda_{max}(A)}{\lambda_{min}(A)} \\
\kappa(A) &\geq \kappa(A+B) \\
\iff \lambda_{max}(A)\left(\lambda_{min}(A) + \lambda_{min}(B)\right) &\geq \lambda_{min}(A)\left(\lambda_{max}(A) + \lambda_{max}(B)\right) \\
\iff \lambda_{max}(A)\lambda_{min}(B) &\geq \lambda_{min}(A)\lambda_{max}(B) \\
\iff \frac{\lambda_{max}(A)}{\lambda_{min}(A)} &\geq \frac{\lambda_{min}(A)}{\lambda_{max}(B)},
\end{aligned}$$

which is false by assumption. This concludes the proof. □

Lemma 7 (Straightforward). *For a positive definite matrix A and $k > 0$, we have that*

$$\kappa(A + k \cdot I) < \kappa(A) \qquad \kappa(A + k \cdot \sqrt{A}) \leq \kappa(A).$$

Lemma 8 (Angle induced by positive definite matrix; folklore). [5] *For a positive definite matrix $A \succ 0$ with condition number κ, we have that*

$$\min_x \frac{x^\top A x}{\|Ax\|_2 \cdot \|x\|_2} = \frac{2\sqrt{\kappa}}{1+\kappa}. \tag{A.18}$$

These two results can be combined to prove the theorem. First, we show that

[5] A proof can be found in `https://bit.ly/2L6jdAT`

$\kappa(\mathbf{\Sigma}) \leq \kappa(\mathbf{\Sigma}_*)$:

$$\begin{aligned}
\kappa(\mathbf{\Sigma}) &= \kappa\left(\frac{1}{\lambda}I + \frac{1}{2}\mathbf{\Sigma}_* + \sqrt{\frac{1}{\lambda}\mathbf{\Sigma}_* + \frac{1}{4}\mathbf{\Sigma}_*^2}\right) \\
&< \max\left\{\kappa\left(\frac{1}{\lambda}I + \frac{1}{2}\mathbf{\Sigma}_*\right), \kappa\left(\sqrt{\frac{1}{\lambda}\mathbf{\Sigma}_* + \frac{1}{4}\mathbf{\Sigma}_*^2}\right)\right\} \\
&< \max\left\{\kappa\left(\mathbf{\Sigma}_*\right), \sqrt{\kappa\left(\frac{1}{\lambda}\mathbf{\Sigma}_* + \frac{1}{4}\mathbf{\Sigma}_*^2\right)}\right\} \\
&= \max\left\{\kappa\left(\mathbf{\Sigma}_*\right), \sqrt{\kappa\left(\frac{2}{\lambda}\sqrt{\frac{1}{4}\mathbf{\Sigma}_*^2} + \frac{1}{4}\mathbf{\Sigma}_*^2\right)}\right\} \\
&\leq \kappa\left(\mathbf{\Sigma}_*\right).
\end{aligned}$$

Finally, note that (A.18) is a strictly decreasing function in κ, and as such, we have shown the theorem. □

Bounds for λ

Lower bound.

$$\begin{aligned}
\varepsilon &= \operatorname{tr}(\mathbf{\Sigma}_* M^2) \\
&\geq \sigma_{min}(\mathbf{\Sigma}_*) \cdot \operatorname{tr}(M^2) && \text{by the definition of tr}(\cdot) \\
&\geq \frac{\sigma_{min}(\mathbf{\Sigma}_*)}{d} \cdot \operatorname{tr}(M)^2 && \text{by Cauchy-Schwarz} \\
&\geq \frac{\sigma_{min}(\mathbf{\Sigma}_*)}{d} \cdot \left[\operatorname{tr}\left((\lambda\mathbf{\Sigma} - \boldsymbol{I})^{-1}\right)\right]^2 && \text{Expanding } M \text{ (A.9)} \\
&\geq \frac{\sigma_{min}(\mathbf{\Sigma}_*)}{d} \cdot \left[\operatorname{tr}\left(\lambda\mathbf{\Sigma} - \boldsymbol{I}\right)^{-1} \cdot d^2\right]^2 && \text{AM-HM inequality} \\
&\geq d^3 \cdot \sigma_{min}(\mathbf{\Sigma}_*) \cdot [\lambda \cdot \operatorname{tr}(\mathbf{\Sigma}) - d]^{-2} \\
[\lambda \cdot \operatorname{tr}(\mathbf{\Sigma}) - d]^2 &\geq \frac{d^3 \cdot \sigma_{min}(\mathbf{\Sigma}_*)}{\varepsilon} \\
\lambda \cdot \operatorname{tr}(\mathbf{\Sigma}) - d &\geq \frac{d^{3/2} \cdot \sqrt{\sigma_{min}(\mathbf{\Sigma}_*)}}{\sqrt{\varepsilon}} && \text{since } M \text{ is PSD} \\
\lambda &\geq \frac{d}{\operatorname{tr}(\mathbf{\Sigma})}\left(1 + \sqrt{\frac{d \cdot \sigma_{min}(\mathbf{\Sigma}_*)}{\varepsilon}}\right)
\end{aligned}$$

Upper bound

$$\begin{aligned}
\varepsilon &= \operatorname{tr}(\mathbf{\Sigma}_* M^2) \\
&\leq \|\mathbf{\Sigma}_*\|_F \cdot d \cdot \sigma_{max}(M)^2 \\
&\leq \|\mathbf{\Sigma}_*\|_F \cdot d \cdot \sigma_{min}(M)^{-2} \\
\lambda \cdot \sigma_{min}(\mathbf{\Sigma}) - 1 &\leq \sqrt{\frac{\|\mathbf{\Sigma}_*\|_F \cdot d}{\varepsilon}} \\
\lambda &\leq \frac{1}{\sigma_{min}(\mathbf{\Sigma})} \left(\sqrt{\frac{\|\mathbf{\Sigma}_*\|_F \cdot d}{\varepsilon}} + 1 \right).
\end{aligned}$$

A.4 Connections to Other Models of Adversarial Examples

Here, we describe other models for adversarial examples and how they relate to the model presented in this work.

Insufficient data. As discussed in Section 2.6, we note that our model does not explicitly contradict the main thesis of Schmidt et al. [Sch+18].

Boundary Tilting. Inspired by this hypothesis [TG16] and concurrently to our work, Kim, Seo, and Jeon [KSJ19] present a simple classification task comprised of two Gaussian distributions in two dimensions. They experimentally show that the decision boundary tends to better align with the vector between the two means for robust models. This is a special case of our theoretical results in Section 2.5. (Note that this exact statement is not true beyond two dimensions, as discussed in Section 2.5.)

Test Error in Noise. Fawzi, Moosavi-Dezfooli, and Frossard [FMF16] and Ford et al. [For+19] argue that the adversarial robustness of a classifier can be directly connected to its robustness under (appropriately scaled) random noise. While this constitutes a natural explanation of adversarial vulnerability given the classifier robustness to noise, these works do not attempt to justify the source of the latter.

At the same time, recent work [Lec+19; CRK19; For+19] utilizes random noise during training or testing to construct adversarially robust classifiers. In the context of our framework, we can expect the added noise to disproportionately affect non-robust features and thus hinder the model's reliance on them.

Piecewise-linear decision boundaries. Shamir et al. [Sha+19b] prove that the geometric structure of the classifier's decision boundaries can lead to sparse adversarial perturbations. However, this result does not take into account the distance to the decision boundary along these direction or feasibility constraints on the input domain. As a result, it cannot meaningfully distinguish between classifiers that are brittle to small adversarial perturbations and classifiers that are moderately robust.

Theoretical constructions which incidentally exploit non-robust features. Bubeck, Price, and Razenshteyn [BPR19] and Nakkiran [Nak19b] propose theoretical models where the barrier to learning robust classifiers is, respectively, due to computational constraints or model complexity. In order to construct distributions that admit accurate yet non-robust classifiers they (implicitly) utilize the concept of non-robust features. Namely, they add a low-magnitude signal to each input that encodes the true label. This allows a classifier to achieve perfect standard accuracy, but cannot be utilized in an adversarial setting as this signal is susceptible to small adversarial perturbations.

Appendix B

Details for Chapter 3

B.1 Post-hoc Debugging Setup & Addition Results

B.1.1 Datasets

We use the ImageNet-1k [Den+09; Rus+15] and Places-365 [Zho+17] datasets which contain images from 1,000 and 365 categories respectively. In particular, both prediction-rule discovery is performed on (or using) samples from the standard test sets to avoid overlap with the training data used to develop the models.

B.1.2 Models

Here, we describe the exact architecture and training process for each model we use. We utilize two canonical, yet relatively diverse model architectures for our study: namely, VGG [SZ15] and ResNet [He+16]. We use the standard PyTorch implementation [1] and train the models from scratch on the ImageNet and Places365 datasets. The accuracy of each model on the corresponding test set is provided in Table B.1.

ImageNet classifiers. We study: (i) a VGG16 variant with batch normalization and (ii) a ResNet-50. Both models are trained using standard hyperparameters: SGD for 90 epochs with an initial learning rate of 0.1 that drops by a factor of 10

[1]https://pytorch.org/vision/stable/models.html

every 30 epochs. We use a momentum of 0.9, a weight decay of 10^{-4} and a batch size of 256 for the VGG16 and 512 for the ResNet-50.

Places365 classifiers. We study: (i) a VGG16 and (ii) a ResNet-18. Both models are trained for 131072 iterations using SGD with a single-cycle learning rate schedule peaking at 2e-2 and descending to 0 at the end of training. We use a momentum 0.9, a weight decay 5e-4 and a batch size of 256 for both models.

	Test Accuracy (%)	
Architecture \ Dataset	ImageNet	Places
VGG	73.70	54.02
ResNet	75.77	54.24

Table B.1: Accuracy of each model architecture on the datasets used in our analysis.

B.1.3 Prediction rule discovery

Recall that our pipeline for probing prediction rules consists of two steps: concept detection and concept transformation. We describe each step below.

We detect concepts using object detectors trained on MS-COCO [Lin+14] and LVIS [GDG19]. For MS-COCO, we use a model with a ResNet-101 backbone[2] which is trained on COCO-Stuff[3] annotations and can detect 182 concepts. For LVIS, we use a pre-trained model from the Detectron [Gir+18] model zoo[4], which can detect 1230 classes. We only consider a prediction as valid for a specific pixel if the model's predicted probability is at least 0.80 for the COCO-based model and 0.15 for the LVIS-based model (chosen based on manual inspection). Moreover, we treat a concept as present in a specific image if it present in at least 100 pixels (image size is 224×224 for ImageNet and 256×256 for Places).

In order to transform concepts, we utilize the fast style transfer methodology

[2]`https://github.com/kazuto1011/deeplab-pytorch`
[3]`https://github.com/nightrome/cocostuff`
[4]`https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md`

of Ghiasi et al. [Ghi+17] using their pre-trained model[5]. This allows us to quickly apply the same style to a large number of images which is ideal for our use-case. Specifically, we manually choose 14 styles (illustrated in Figure 3-2) and choose 3 images for each. This allows us to perform the concept-level transformation in several ways and evaluate how sensitive our model is to the exact style used.

All the pre-trained models used are open-sourced and freely available for non-commercial research.

B.1.4 Discovering Prediction-rules

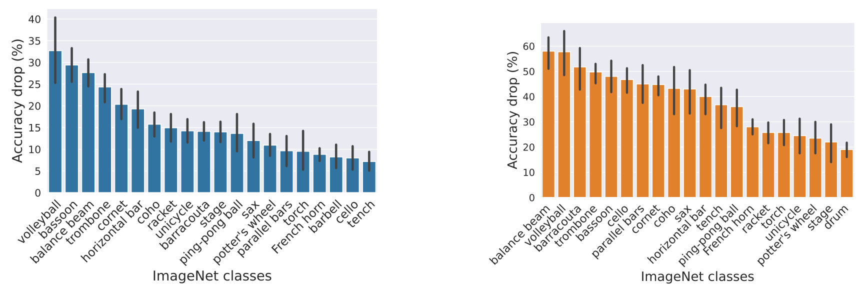
Here, we expand on our analysis in Section 3.2 so as to characterize the effect of concept-level transformations on classifiers.

Per concept. In Figure B-1, we visualize the accuracy drop induced by transformations of a specific concept for classifiers trained on ImageNet and Places-365 (similar to Figure 3-3a). Here, the accuracy drop post-transformation is measured only on images that contain the concept of interest. We then present the average drop across transformations, along with 95% confidence intervals. We find that there is a large variance between: (i) a model's reliance on different concepts, and (ii) different model's reliance on a single concept. For instance, the accuracy of a ResNet-50 ImageNet classifier drops by more than 30% on the class "three-toed sloth" when "tree"s in the image are modified, while the accuracy of a VGG16 model drops by less than 5% under the same setup.

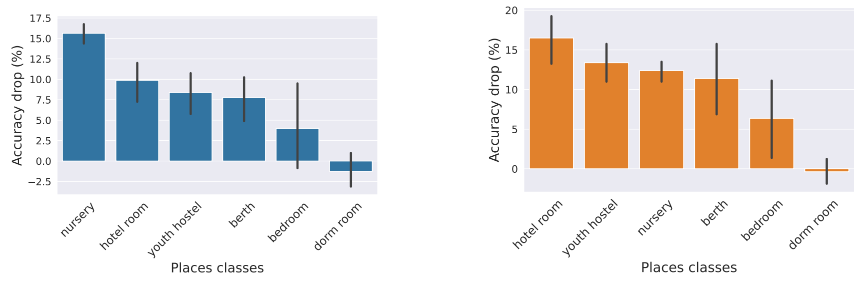
Per transformation. In Figure B-2, we illustrate how the model's sensitivity to specific concepts varies depending on the applied transformation. Across concepts, we find that models are more sensitive to transformations to textures such as "grafitti" and "fall colors" than they are to "wooden" or "metallic".

[5] `https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2`

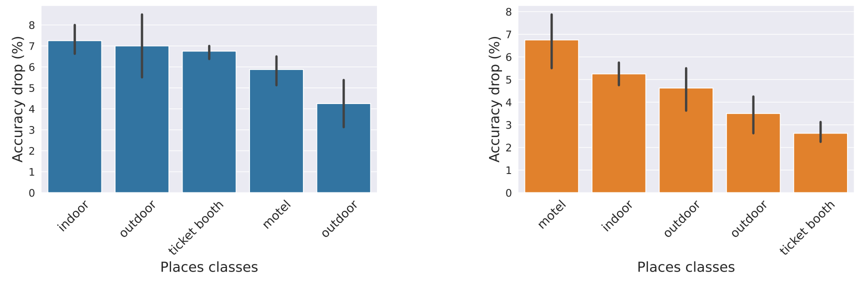
Per class (prediction-rules). In Figure B-3, we provide additional examples of class-level prediction rules identified using our methodology. For each class, the highlighted concepts are those that hurt model accuracy when transformed.



(a) Concept: "person"; Models: VGG16 (*left*) and ResNet-50 (*right*) trained on ImageNet-1k.

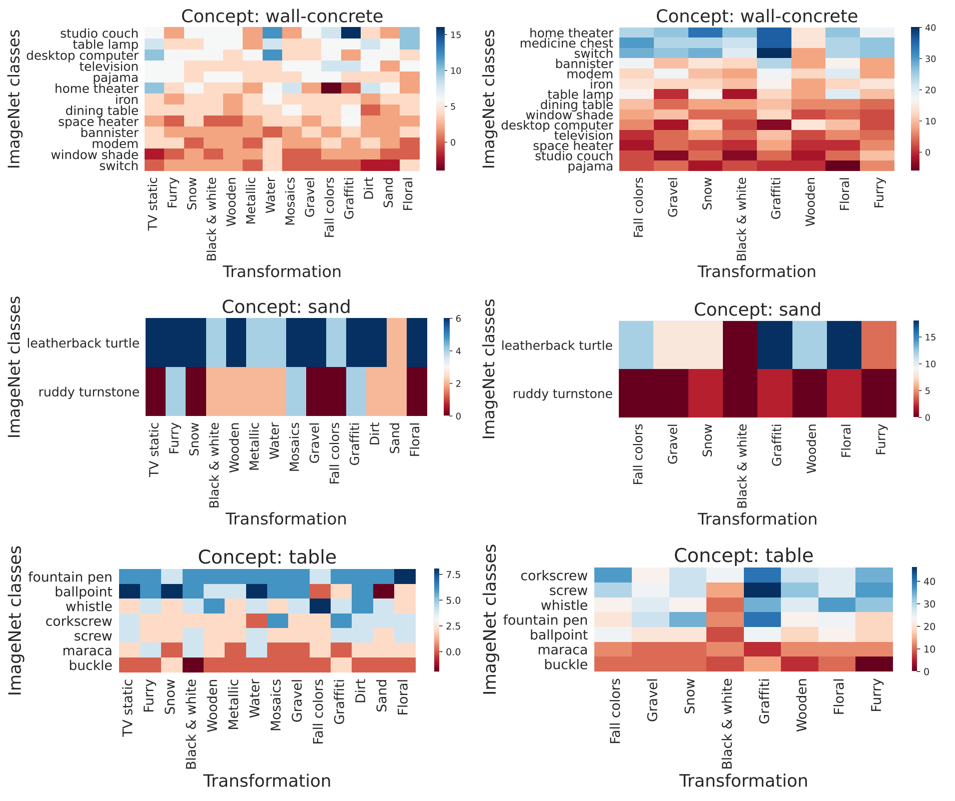


(b) Concept: "bed"; Models: VGG16 (*left*) and ResNet-18 (*right*) trained on Places-365.

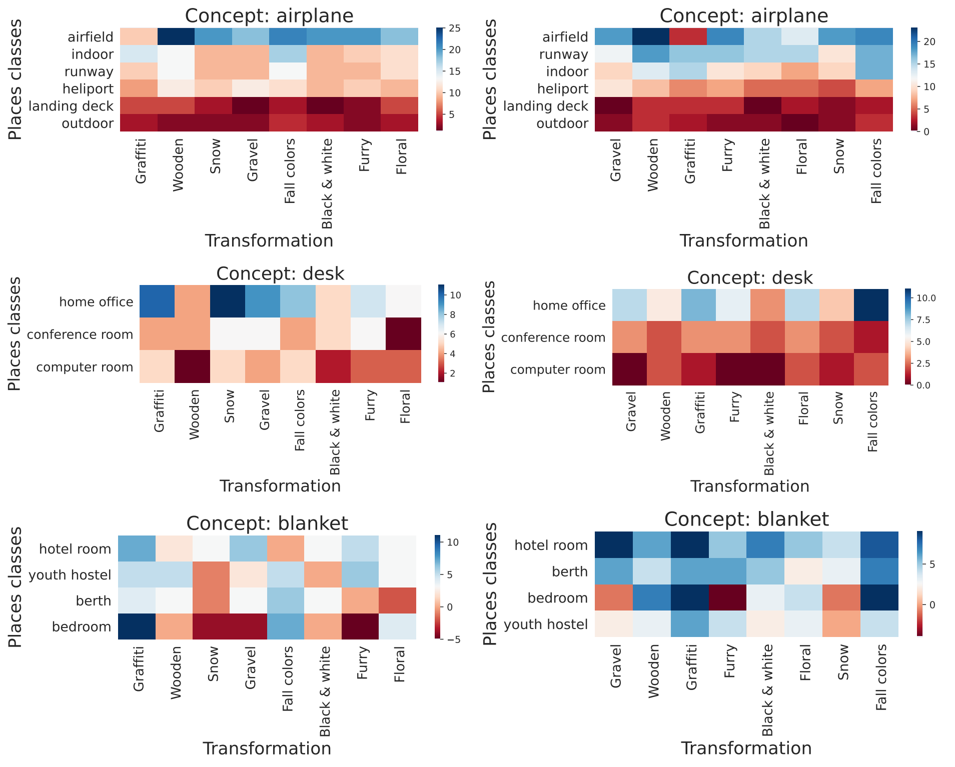


(c) Concept: "signboard"; Models: VGG16 (*left*) and ResNet-18 (*right*) trained on Places-365.

Figure B-1: Dependence of a classifier on a specific high-level concept: average accuracy drop (along with 95% confidence intervals obtained via bootstrapping), over various styles, induced by the transformation of said concept. The classes for which the concept is most often present are shown.

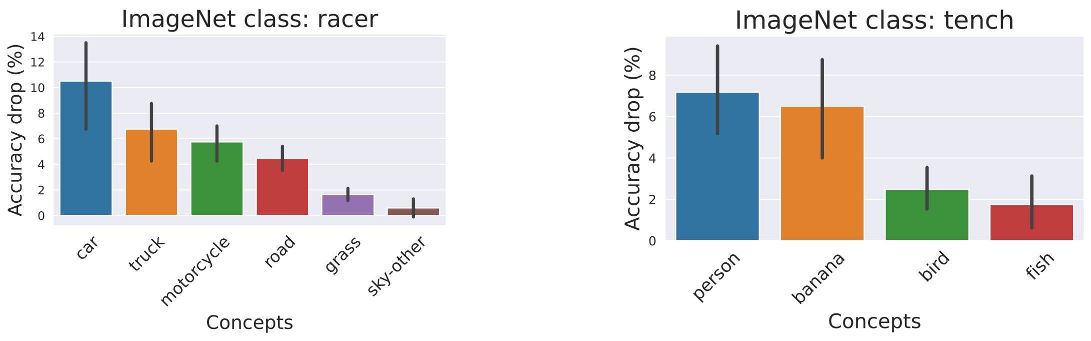


(a) VGG16 (*left*) and ResNet-50 (*right*) models trained on the ImageNet-1k dataset.



(b) VGG16 (*left*) and ResNet-18 (*right*) models trained on the Places-365 dataset.

Figure B-2: Heatmaps illustrating classifier sensitivity to various concept-level transformations. Here, we measure model sensitivity in terms of the per class drop in model accuracy induced by the transformation on images of that class which contain the concept of interest.



(a) VGG16 classifier trained on the ImageNet dataset.

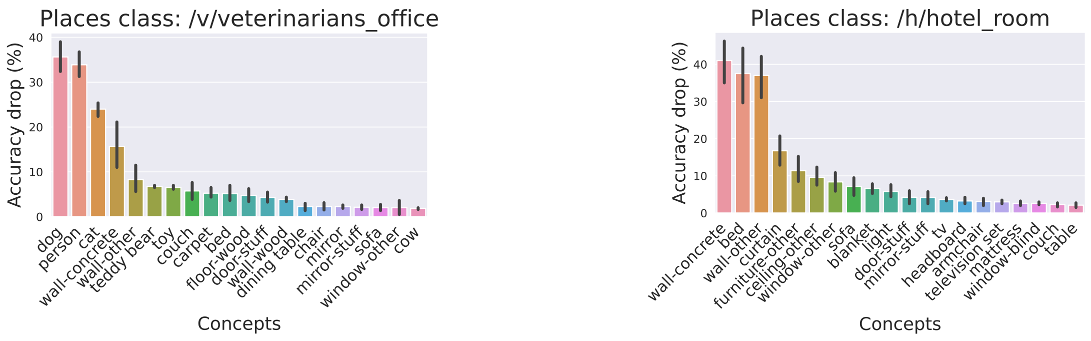


Figure B-3: Per-class prediction rules: high-level concepts, which when transformed, significantly hurt model performance on that class. Here, we visualize average accuracy drop (along with 95% confidence intervals obtained via bootstrapping) for a specific concept, over various styles.

B.2 SAGA-based solver for generalized linear models

In this section, we describe in further detail our solver for learning regularized GLMs in relation to existing work. Note that many of the components underlying our solver have been separately studied in prior work. However, we are the first to effectively combine them in a way that allows for GPU-accelerated fitting of GLMs at ImageNet-scale. The key algorithmic primitives we leverage to this end are variance reduced optimization methods and path algorithms for GLMs.

Specifically, our solver uses a mini-batch derivative of the SAGA algorithm [GGS19], which belongs to a class of a variance reduced proximal gradient methods. These approaches have several benefits: a) they are easily parallelizable via GPU, b) they enjoy faster convergence rates than stochastic gradient methods, and c) they require minimal tuning and can converge with a fixed learning rate.

Algorithm 1 provides a step-by-step description of our solver. Here, the proximal operator for elastic net regularization is

$$\mathrm{Prox}_{\lambda_1,\lambda_2}(\beta) = \begin{cases} \frac{\beta-\lambda_1}{1+\lambda_2} & \text{if } \beta > \lambda_1 \\ \frac{\beta+\lambda_1}{1+\lambda_2} & \text{if } \beta < \lambda_1 \\ 0 & \text{otherwise} \end{cases} \tag{B.1}$$

Table for storing gradients Note that the SAGA algorithm requires saving the gradients of the model for each individual example. For ImageNet-sized problems, this requires a prohibitive amount of memory, as both the number of examples (>1 million) and the size of the gradient (of the linear model) are large.

It turns out that for linear models with k outputs, it is actually possible to store all of the necessary gradient information for a single example in a vector of size k—as demonstrated by Defazio, Bach, and Lacoste-Julien [DBL14]. The key idea behind this approach is that rather than storing the full gradient step $(x_i^T\beta + \beta_0 - y_i)x_i$, we can instead just store the scalar $a_i = (x_i^T\beta + \beta_0 - y_i)$ per output (i.e., a vector of length k in the case of multiple outputs). Thus, for a dataset with n

Algorithm 1 GPU-accelerated solver for the elastic net for a step size γ and regularization parameters λ, α

```
1: Initialize table of scalars $a'_i = 0$ for $i \in [n]$
2: Initialize average gradient of table $g_{avg} = 0$ and $g_{0avg} = 0$
3: for minibatch $B \subset [n]$ do
4:     for $i \in B$ do
5:         $a_i = x_i^T \beta + \beta_0 - y_i$
6:         $g_i = a_i \cdot x_i$ // calculate new gradient information
7:         $g'_i = a'_i \cdot x_i$ // calculate stored gradient information
8:     end for
9:     $g = \frac{1}{|B|} \sum_{i \in B} g_i$
10:    $g' = \frac{1}{|B|} \sum_{i \in B} g'_i$
11:    $g_0 = \frac{1}{|B|} \sum_{i \in B} a_i$
12:    $g'_0 = \frac{1}{|B|} \sum_{i \in B} a'_i$
13:    $\beta = \beta - \gamma(g - g' + g_{avg})$
14:    $\beta_0 = \beta_0 - \gamma(g_0 - g'_0 + g_{0avg})$
15:    $\beta = \text{Prox}_{\gamma\lambda\alpha, \gamma\lambda(1-\alpha)}(\beta)$
16:    for $i \in B$ do
17:        $a'_i = a_i$ // update table
18:        $g_{avg} = g_{avg} + \frac{|B|}{n}(g - g')$ // update average
19:        $g_{0avg} = g_{0avg} + \frac{|B|}{n}(g_0 - g'_0)$
20:    end for
21: end for
```

examples, this reduces the memory requirements of the gradient table to $O(nk)$. For ImageNet, we find that the entire table easily fits within GPU memory limits.

There is one caveat here: in order to use this memory trick, it is necessary to incorporate the ℓ_2 regularization from the elastic net into the proximal operator. This is precisely why we use the proximal operator of the elastic net, rather than of the ℓ_1 regularization. Unfortunately, this means that the smooth part of the objective (i.e. the part not used in the proximal operator) is no longer guaranteed to be strongly convex, and so the theoretical analysis of Gazagnadou, Gower, and Salmon [GGS19] no longer strictly applies. Nonetheless, we find that these variance reduced methods can still provide strong practical convergence rates in this setting without requiring much tuning of batch sizes or learning rates.

Stopping criterion We implement two simple stopping criteria, which both take in a tolerance level ϵ_{tol}. The first is a gradient-based stopping criteria, which terminates when:

$$\sqrt{\|\beta^{i+1} - \beta^{i}\|_2^2 + \|\beta_0^{i+1} - \beta_0^{i}\|_2^2} \leq \epsilon_{\text{tol}}$$

Intuitively, this stops when the change in the estimated coefficients is small. Our second stopping criteria is more conservative and uses a longer search horizon, and stops when the training loss has not improved by more than ϵ_{tol} for more than T epochs for some T, which we call the lookbehind stopping criteria.

In practice, we find that the gradient-based stopping criteria with $\epsilon_{\text{tol}} = 10^{-4}$ is sufficient for most cases (i.e. the solver has converged sufficiently such that the number of non-zero entries will no longer change). For significantly larger problems such as ImageNet, where individual batch sizes can have much larger variability in progressing the training objective, we find that the lookbehind stopping criteria is sufficient with $\epsilon_{\text{tol}} = 10^{-4}$ and $T = 5$.

Relation of the solver to existing work We now discuss how our solver borrows and differs from existing work. First, note that the original SAGA algorithm [DBL14] analyzes the regularized form but updates its gradient estimate with one sample at a time, which is not amenable to GPU parallelism. On the other hand, Gazagnadou, Gower, and Salmon [GGS19] analyze a minibatch variant of SAGA but without regularization. In our solver, we use a straightforward adaptation of minibatch SAGA to its regularized equivalent by including a proximal step for the elastic net regularization after the gradient step.

To compute the regularization paths, we closely follow the framework of Friedman, Hastie, and Tibshirani [FHT10]. Specifically, we compute solutions for a decreasing sequence of regularization, using the solution of the previous regularization as a warm start for the next. The maximum regularization value which fits only the bias term is calculated as the fixed point of the coordinate descent iteration as

$$\lambda_{max} = \max_j \frac{1}{N\alpha} \left| \sum_{i=1}^{n} x_{ij} y_i \right| \tag{B.2}$$

and scheduled down to $\lambda_{min} = \epsilon\lambda_{max}$ over a sequence of K values on a log scale, as done by Friedman, Hastie, and Tibshirani [FHT10]. Typical suggested values are to take $K = 100$ and $\epsilon = 0.001$, which are what we use in all of our experiments. For extensions to logistic and multinomial regression, we refer the reader to Friedman, Hastie, and Tibshirani [FHT10], and note that our approach is the same but substituting our SAGA-based solver in liue of the coordinate descent-based solver.

B.2.1 Timing Experiments

In this section, we discuss how the runtime of our solver scales with the problem size. To be able to compare our solver with existing approaches, the experiments performed here are at a smaller scale than those discussed in Chapter 3.

Problem setting & hyperparameters. The problem we examine is that of fitting a linear decision layer for the CIFAR-10 dataset using the deep feature representation of an ImageNet-trained ResNet-50 (2048-dimensional features). We then vary the number of training examples (from 1k to 50k) and fit an elastic net regularized GLM using various methods. We compare `glmnet` (state-of-the-art, coordinate descent-based solver) on a 9th generation Intel Core i7 with 6 cores clocked at 2.6Ghz, and our approach `glm-saga` using a GeForce GTX 1080ti. We note that in these small-scale experiments, the graphics card remains at around 10-20% utilization, indicating that the problem size is too small to fully utilize the GPU.

We fix $\alpha = 0.99$, $\epsilon = 10^{-4}$, set aside 10% of the training data for validation, and calculate regularization paths for $k = 100$ different values, which are the defaults for `glmnet`. For our approach, we additionally use a mini-batch size of 512, a learning rate of 0.1, and a tolerance level of 10^{-4} for the gradient-based stopping criteria.

Solver	Number of examples					
	1k	2k	3k	4k	5k	50k
`glmnet`	2	7	25	39	58	776
`glm-saga`	9	13	17	19	22	33

Table B.36: Runtime in minutes for `glmnet` and `glm-saga` for fitting a sparse decision layer on the CIFAR-10 dataset using deep representations (2048D) for a pre-trained ResNet-50. Here, we assess how the runtime of different solvers scales as a function of training data points.

Improvements in scalability As expected, on smaller problem instances with a couple thousand examples, `glmnet` is faster than our solver—cf. Table B.36. This is largely due to the increased base running time of our solver—a consequence of gradient based methods requiring some time to converge. However, as the problem size grows, the runtime of `glmnet` increases rapidly, and exceeds the running time of `glm-saga` at 3,000 datapoints. For example, it takes almost 40 minutes to fit 4,000 data points with `glmnet`, an increase of 20x the running time for 4x the data relative to the running time for 1,000 data points. In contrast, our solver only needs 19 minutes to fit 4,000 datapoints, an increase of 2x the running time for 4x the data. Consequently, while `glmnet` takes a considerable amount of time to fit the full CIFAR10 problem size (50,000 datapoints)—nearly 13 hours—our solver can do the same in only 33 minutes. Notably, our solver can fit the regularization paths of the decision layer for the full ImageNet dataset (1 million examples with 2048 features) in approximately 6 hours.

Backpropagation libraries One more alternative to fitting linear models at scale is to use a standard autodifferentiation library such as PyTorch or Tensorflow. However, typical optimizers used in these libraries do not handle non-smooth regularizers well (i.e., the ℓ_1 penalty of the elastic net). In practice, these types of approaches must gradually schedule learning rates down to zero in order to converge, and take too long to compute regularization paths. For example, the fixed-feature transfer experiments from Salman et al. [Sal+20] takes approximately

4 hours to fit the same CIFAR10 timing experiment for a single regularization value. In contrast, the SAGA-based optimizers enables a flexible range of learning rates that can converge rapidly without needing to tune or decay the learning rate over time.

B.2.2 Elastic net, ℓ_1, and ℓ_2 regularization

The elastic net is known to combine the benefits of both ℓ_1 and ℓ_2 regularization for linear models. The ℓ_1 regularization, often seen in the LASSO, primarily provides sparsity in the solution. The ℓ_2 regularization, often seen as ridge regression, brings improved performance, a unique solution via strong convexity, and a grouping effect of similar neurons. Due to this last property of ℓ_2 regularization, highly correlated features will become non-zero at the same time over the regularization path. The elastic net combines all of these strengths, and we refer the reader to Tibshirani and Wasserman [TW17] for further discussion on the interaction between elastic net, ℓ_1, and ℓ_2.

B.3 Feature interpretations

We now discuss in depth our procedure for generating feature interpretations for deep features in the vision and language settings.

B.3.1 Feature visualization

Feature visualization is a popular approach to interpret individual neurons within a deep network. Here, the objective is to synthesize inputs (via optimization in pixel space) that highly activate the neuron of interest. Unfortunately, for standard networks trained via empirical risk minimization, it is well-known that vanilla feature visualization—using just gradient descent in input space—fails to produce semantically-meaningful interpretations. In fact, these visualizations frequently suffer from artifacts and high frequency patterns [OMS17]. One cause for this could be the reliance of standard models on input features that are imperceptible or unintuitive, as has been noted in recent studies [Ily+19].

To mitigate this challenge, there has been a long line of work on defining modified objectives to produce more meaningful feature visualizations [OMS17]. In this work, we use the Tensorflow-based Lucid library[6] to produce feature visualizations for standard models. Therein, the optimization objective contains additional regularizations to penalize high-frequency changes in pixel space and to encourage transformation robustness. Further, gradient descent is performed in the Fourier basis to further discourage high-frequency input patterns. We defer the reader to Olah, Mordvintsev, and Schubert [OMS17] for a more complete presentation.

In contrast, a different line of work [Tsi+19; Eng+19a] has shown that robust (adversarially-trained) models tend to have better feature representations than their standard counterparts. Thus, for robust models, gradient descent in pixel space is already sufficient to find semantically-meaningful feature visualizations.

B.3.2 LIME

Image superpixels. Traditionally, LIME is used to obtain per-instance explanations. That is, to identify the superpixels in a given test image that are most responsible for the model's prediction. However, in our setting, we would like to obtain a global understanding of deep features, independent of specific test examples. Thus, we use the following two step-procedure to obtain LIME-based feature interpretations:

1. Rank test set images based on how strongly they activate the feature of interest. Then select the top-k (or conversely bottom-k) images as the most prototypical examples for positive (negative) activation of the feature.

2. Run LIME on each of these examples to identify relevant superpixels. At a high level, this involves performing linear regression to map image superpixels to the (normalized) activation of the deep feature (rather than the

[6]https://github.com/tensorflow/lucid

Algorithm 2 Word cloud feature visualization for language models for a vocabulary V, a corpus w_{ij} for $i, j \in [m] \times [n]$ of m sentences with n words

```
1: for $i = 1 \dots m$ do
2:     $\beta_i = \texttt{LIME}(w_i)$ // generate LIME explanation for each sentence
3: end for
4: for $w \in V$ do
5:     $K_w = \sum_{ij: w = w_{ij}} 1$ // count number of occurances of word
6:     $\hat{\beta}_w = \frac{1}{K_w} \sum_{ij: w = w_{ij}} \beta_{ij}$ // calculate average LIME explanation of word
7: end for
8: return $\texttt{Wordcloud}(\beta, V)$ // generate word cloud for vocabulary V weighted by $\beta$
```

probability of a specific class as is typical).

Due to space constraints, we use $k = 1$ in all our figures. However, in our analysis, we found the superpixels identified with $k = 1$ to be representative of those obtained with higher values.

Word clouds for language models For language models, off-the-shelf neuron interpretability tools are somewhat more limited than their vision counterparts. Of the tools listed above, only LIME is used in the language domain to produce sentence-specific explanations. Similar to our methodology for vision models, we apply LIME to a given deep feature representation rather than the output neuron. However, rather than selecting prototypical images, we instead aggregate LIME explanations over the entire validation set.

Specifically, for a given feature, we average the LIME weighting for each word over all of the sentences that the word appears in. This allows us to identify words that strongly activate/deactivate the given feature globally over the entire validation set, which we then visualize using word clouds. In practice, since a word cloud has limited space, we provide the top 30 most highly weighted words to the word cloud generator. The exact procedure is shown in Algorithm 2, and we use the word cloud generator from `https://github.com/amueller/word_cloud`.

B.4 Datasets and Models

B.4.1 Datasets

We perform our experiments on the following widely-used vision and language datasets.

- ImageNet-1k [Den+09; Rus+15].
- Places-10: A subset of Places365 [Zho+17] containing the classes "airport terminal", "boat deck", "bridge", "butcher's shop", "church-outdoor", "hotel room", "laundromat", "river", "ski slope" and "volcano".
- Stanford Sentiment Treebank (SST) [Soc+13] with labels for "positive" and "negative" sentiment.
- Toxic Comments [WTD17] with labels for "toxic", "severe toxic", "obscene", "'threat", "insult", and 'identity hate".

Balancing the comment classification task. The toxic comments classification task has a highly unbalanced test set, and is largely skewed towards non-toxic comments. Consequently, the baseline accuracy for simply predicting the non-toxic label is often upwards of 90% on the unbalanced test set. To get a more interpretable and usable performance metric, we instead randomly subsample the test set to be balanced with 50% each of toxic and non-toxic comments from the corresponding toxicity category. Thus, the baseline accuracy for random chance for toxic comment classification in our experiments is 50%.

B.4.2 Models

We consider ResNet-50 [He+16] classifiers and BERT [Dev+18] models for vision and language tasks respectively. In the vision setting, we consider both standard and robust models [Mad+18].

Vision. All the models are trained for 90 epochs, weight decay 1e-4 and momentum 0.9. We used a batch size of 512 for ImageNet and 128 for Places-10. The initial learning rate is 0.1 and is dropped by a factor of 10 every 30 epochs. The robust models were obtained using adversarial training with a ℓ_2 PGD adversary [Mad+18] with $\varepsilon = 3$, 3 attack steps and attack step size of $\frac{2\times\varepsilon}{3}$.

Language. The language models are all pretrained and available from the HuggingFace[7] library, and use the standard BERT base architecture.

B.5 Evaluating sparse decision layers

Selecting a single sparse model

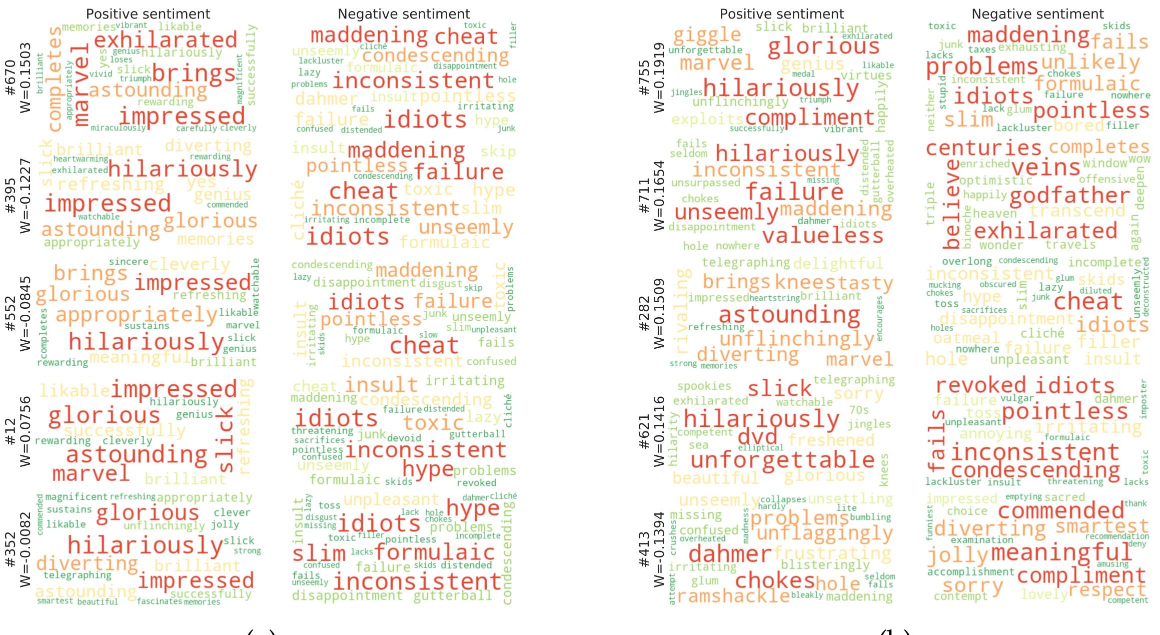
As discussed in Section 3.3.1, the elastic net yields a sequence of linear models—with varying accuracy and sparsity—also known as the regularization path. In practice, performance of these models on a hold-out validation set can be used to guide model selection based on application-specific criteria. In our experiments, we set aside 10% of the train set for this purpose.

Our model selection thresholds. For both vision and NLP tasks, we use the validation set to identify the sparsest decision layer, whose accuracy is no more than 5% lower on the validation set, compared to the best performing decision layer. As discussed in the Chapter 3, these thresholds are meant to be illustrative and can be varied depending on the specific application. We now visualize the per-class distribution of deep features for the sparse decision layers selected in Table 3.1. (We omit the NLP tasks as they entail only two classes.)

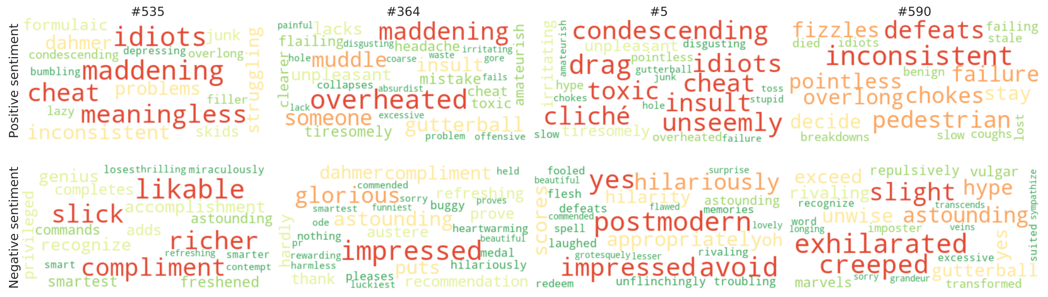
[7]Specifically, the sentiment classification model is from `https://huggingface.co/barissayil/bert-sentiment-analysis-sst` and the toxic comment models (both Toxic-BERT and Debiased-BERT) come from `https://huggingface.co/unitary/toxic-bert`.

B.5.1 Additional comparisons of features

In Figure B-4, we visualize additional deep features used by BERT models with sparse decision layers for the SST sentiment analysis task. Figures B-5- B-7 show feature interpretations of deep features used by ResNet-50 classifiers with sparse decision layers trained on ImageNet and Places-10. Due to space constraints, we limit the feature interpretations for vision models to (at most) five randomly-chosen deep features used by the dense/sparse decision layer in Figure 3-6b and Figures B-5- B-7. To allow for a fair comparison between the two decision layers, we sample these features as follows. Given a target class, we first determine the number of deep features (k) used by the sparse decision layer to recognize objects of that class. Then, for both decision layers, we randomly sample five deep features from the top-k highest weighted ones (for that class).



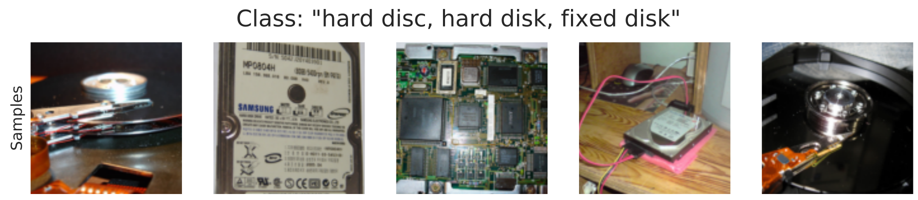
(a) (b)



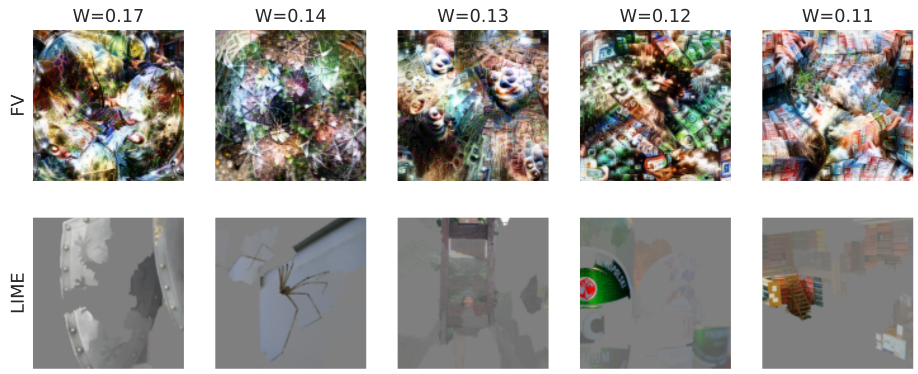
(c)

Figure B-4: Additional SST word clouds visualizing the positive and negative activations for the top 5 features of the (a) sparse decision layer, (b) dense decision layer, and (c) additional randomly-selected features (positive or negative weighting is according to the dense decision layer). While the sparse model focuses on features that have clear positive and negative semantic meaning in their word clouds, the dense model and the other randomly-selected features are noticeably more mixed in sentiment.

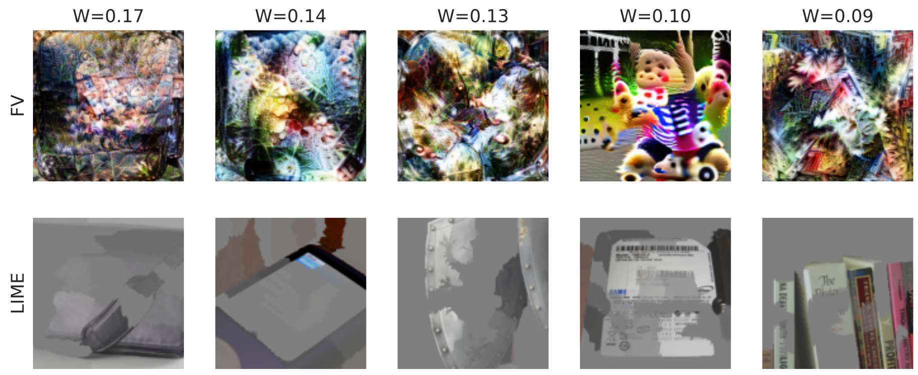
Vision models



(a) Class samples

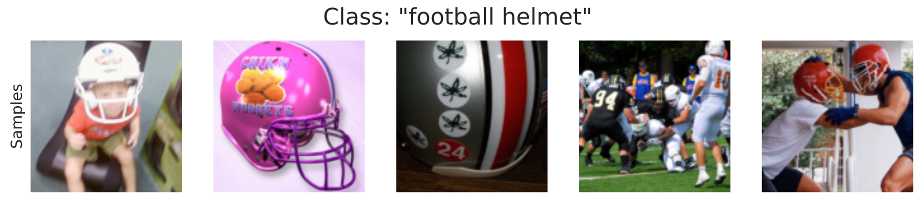


(b) Dense

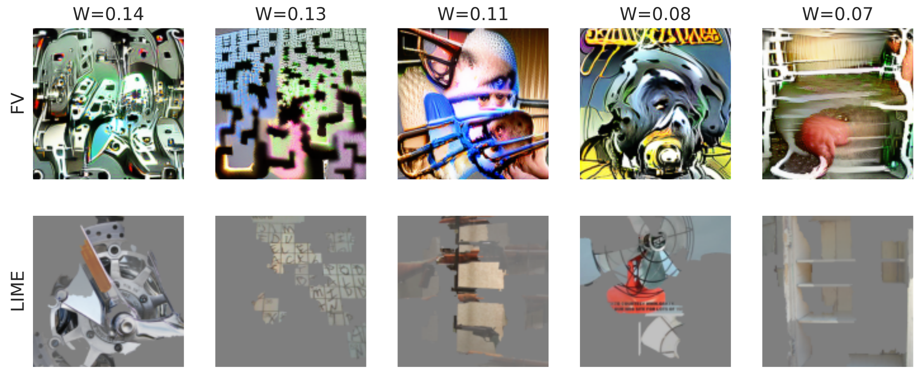


(c) Sparse

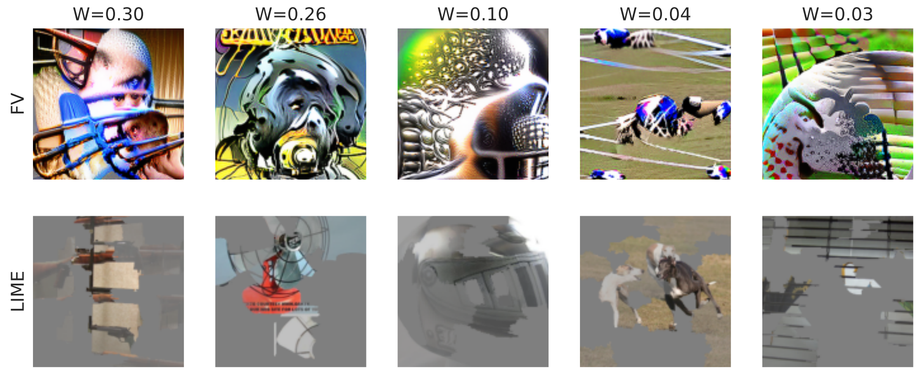
Figure B-5: Deep features used by a standard ($\varepsilon = 0$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen ImageNet class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.



(a) Class samples

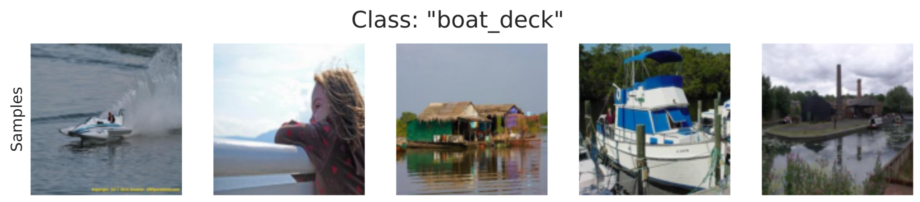


(b) Dense

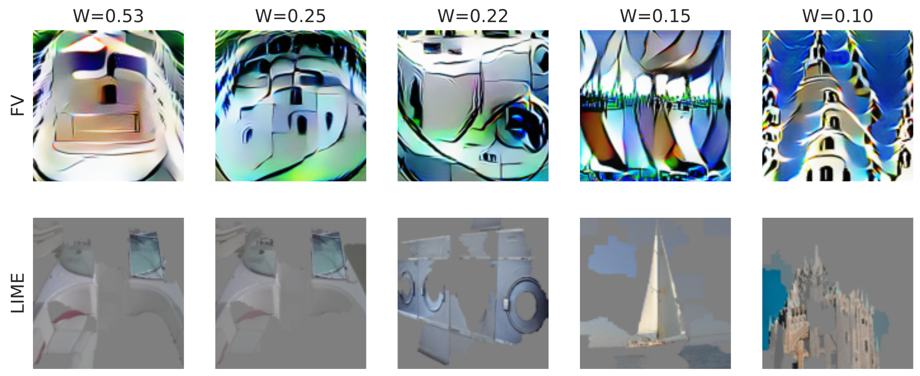


(c) Sparse

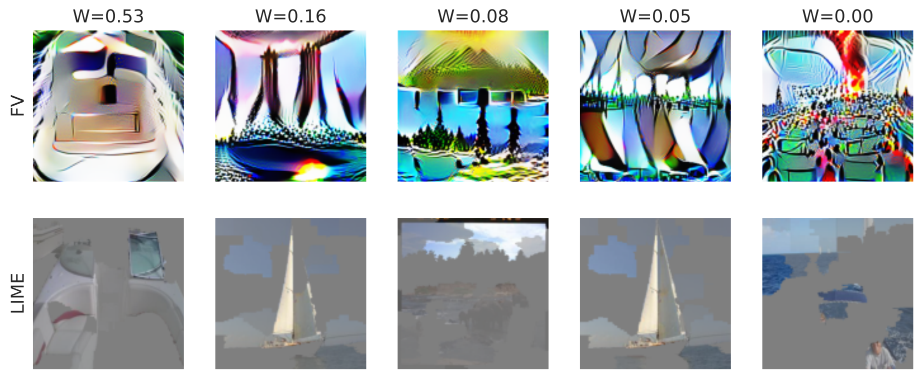
Figure B-6: Deep features used by a adversarially-trained ($\varepsilon = 3$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen ImageNet class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.



(a) Class samples

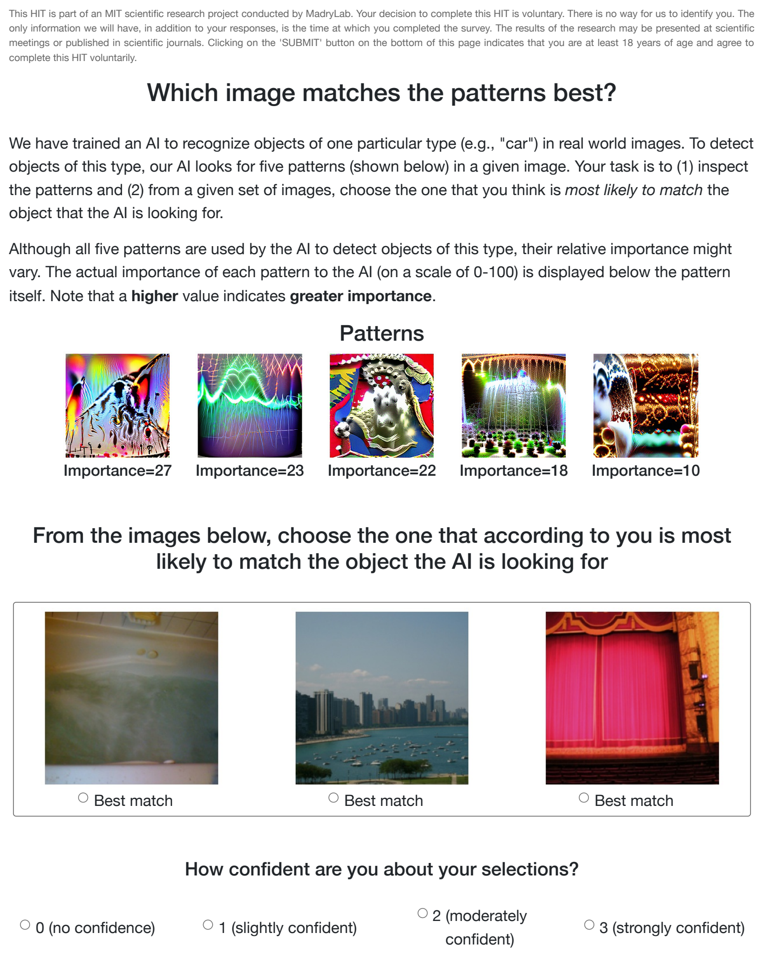


(b) Dense



(c) Sparse

Figure B-7: Deep features used by a adversarially-trained ($\varepsilon = 3$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen Places-10 class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.


This HIT is part of an MIT scientific research project conducted by MadryLab. Your decision to complete this HIT is voluntary. There is no way for us to identify you. The only information we will have, in addition to your responses, is the time at which you completed the survey. The results of the research may be presented at scientific meetings or published in scientific journals. Clicking on the 'SUBMIT' button on the bottom of this page indicates that you are at least 18 years of age and agree to complete this HIT voluntarily.

Which image matches the patterns best?

We have trained an AI to recognize objects of one particular type (e.g., "car") in real world images. To detect objects of this type, our AI looks for five patterns (shown below) in a given image. Your task is to (1) inspect the patterns and (2) from a given set of images, choose the one that you think is *most likely to match* the object that the AI is looking for.

Although all five patterns are used by the AI to detect objects of this type, their relative importance might vary. The actual importance of each pattern to the AI (on a scale of 0-100) is displayed below the pattern itself. Note that a **higher** value indicates **greater importance**.

Patterns

Importance=27 Importance=23 Importance=22 Importance=18 Importance=10

From the images below, choose the one that according to you is most likely to match the object the AI is looking for

○ Best match ○ Best match ○ Best match

How confident are you about your selections?

○ 0 (no confidence) ○ 1 (slightly confident) ○ 2 (moderately confident) ○ 3 (strongly confident)

Figure B-8: Sample MTurk task to assess how amenable models with dense/sparse decision layers are to human understanding.

B.5.2 Human evaluation

We now detail the setup of our MTurk study from Section 3.4.3. For our analysis, we use a ResNet-50 that has been adversarially-trained ($\varepsilon = 3$) on the ImageNet dataset. To obtain a sparse decision layer, we then train a sequence of GLMs via elastic net (cf. Section 3.3.1) on the deep representation of this network. Based on a validation set, we choose a single sparse decision layer—with 57.65% test accuracy and 39.18 deep features/class on average.

Task setup Recall that our objective is to assess how effectively annotators are able to simulate the predictions of a model when they are exposed to its (dense or sparse) decision layer. To this end, we first randomly select 100 ImageNet classes. Then, for each such 'target class' and decision layer (dense/sparse) pair, we created a task by:

1. **Selecting deep features:** We randomly-select *five* deep features utilized by the decision layer to recognize objects of the target class. To make the comparison more fair, we restrict our attention to deep features that are assigned significant weight (>5% of the maximum) by the corresponding model. We then present these deep features to annotators via feature visualizations. Also shown alongside are the (normalized and rescaled) linear coefficients for each deep feature.

2. **Selecting test inputs:** We rank all the test set ImageNet images based on the probability assigned by the corresponding model (i.e., the ResNet-50 with a dense/sparse decision layer) to the target class. We then randomly select three images, such that they lie in the following percentile ranges in terms of target class probability: (90, 95), (98, 99) and (99.99, 100). Note that since ImageNet has 1000 diverse object categories, the target class probability of a randomly sampled image from the dataset is likely to be extremely small. Thus, fixing the percentiles as described above allows us to pick image candidates that are: (i) somewhat relevant to the target class; and (ii) of comparable difficulty for both types of decision layers.

Finally, annotators are presented with the deep features chosen above—describing them as patterns used by an AI model to recognize objects of a certain (unspecified) type. They are then asked to pick one of the image candidates (randomly-permuted) that best matches the patterns. Annotators are also asked to mark their confidence on a likert scale. A sample task is shown in Figure B-8.

For each target label-decision layer pair, we obtain 10 tasks by repeating the random selection process above. This results in a total of 2000 tasks (100 classes x 2 models x 10 tasks/(class, model)). Each task is presented to 5 annotators, compensated at $0.04 per task.

Quality control For each task, we aggregated results over all the annotators. While doing so, we eliminated individual instances where a particular annotator made no selections. We also completely eliminated instances corresponding to annotators who consistently (>80% of the times) left the tasks blank. Finally, while reporting our results, we only keep tasks for which we have selections from at least two (of five) annotators. We determine the final selection based on a majority vote over annotators, weighted by their confidence.

Results In Table B.12, we report annotator accuracy—in terms of their ability to correctly identify the image with the highest target class probability as per the model. We also present a break down of the overall accuracy depending on whether or not the "correct image" is from the target class. We find that sparsity significantly boosts annotators' ability to intuit (simulate) the model—by nearly 30%. In fact, their performance on models with dense decision layers is close to chance (33%). Note also that for models with sparse decision layers, annotators are able to correctly simulate the predictions *even* when the correct image belongs to a different class.

Accuracy (%)	Dense	Sparse
Overall	35.61 ± 3.09	$\mathbf{63.02 \pm 3.02}$
From target class	44.02 ± 5.02	$\mathbf{72.22 \pm 4.74}$
From another class	30.64 ± 3.65	$\mathbf{57.33 \pm 4.00}$

Table B.12: Accuracy of annotators at simulating the model given explanations from the dense and sparse classifiers.

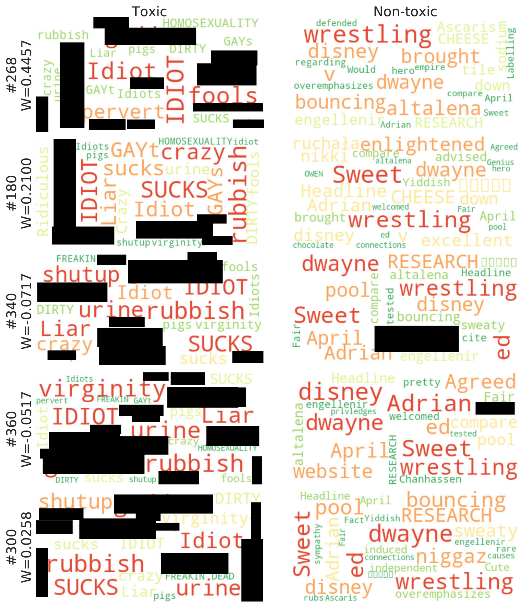
B.6 Model biases and spurious correlations

B.6.1 Toxic comments

In this section, we visualize the word clouds for the toxic comment classifiers which reveal the biases that the model has learned from the data. Note that these figures are heavily redacted due to the nature of these comments.

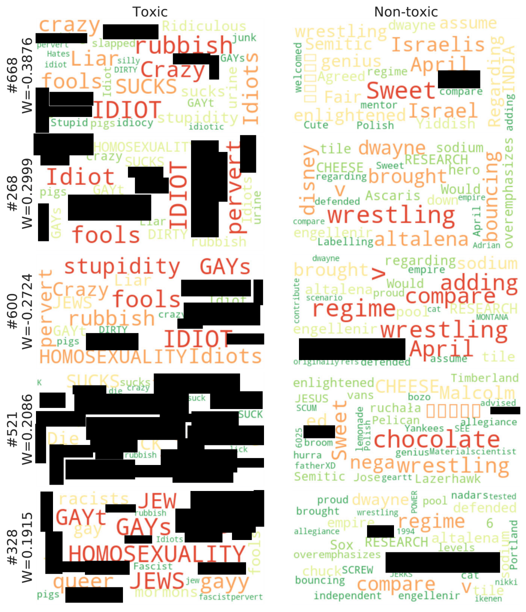
In Figure B-9, we visualize the top five features for the sparse (Figure B-9a) and dense (Figure B-9b) decision layers of Toxic-Bert. We note that more of the words in the sparse decision layer refer to identity groups, whereas this is less clear in the dense decision layer. Even if we expand our interpretation to the top 10 neurons with the largest weight, only 7.5% of the words refer to identity groups for the model with a dense decision layer.

In Figure B-10, we perform a similar visualization as for the Toxic-BERT model, but for the Debiased-BERT model. The word clouds for the sparse decision layer (Figure B-10a) provide evidence that the Debiased-BERT model no longer uses identity words as prevalently for identifying toxic comments. However, it is especially clear from the word clouds for the sparse decision layer that a significant fraction of the non-toxic word clouds contain identity words. This suggests that the model now uses these identity words as strong evidence for non-toxicity, which can be also reflected to a lesser degree in the wordclouds for the dense decision layer (Figure B-10b).



[redacted]

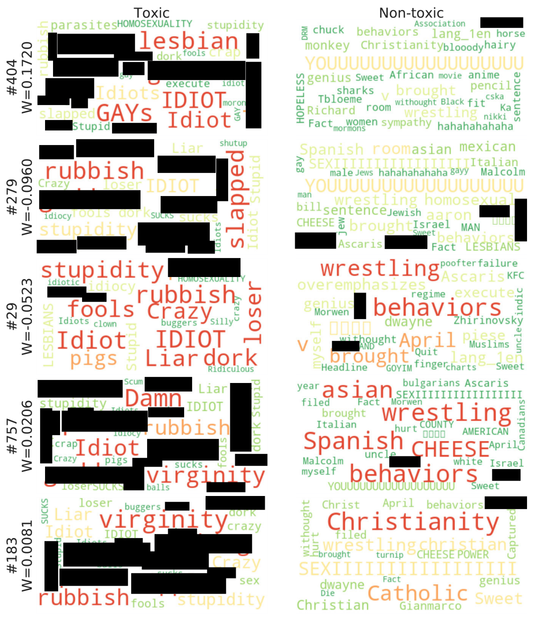
(a)



[redacted]

(b)

Figure B-9: Word cloud visualizations of the top 5 deep features in Toxic-BERT for the (a) sparse decision layer and (b) dense decision layer


(a)

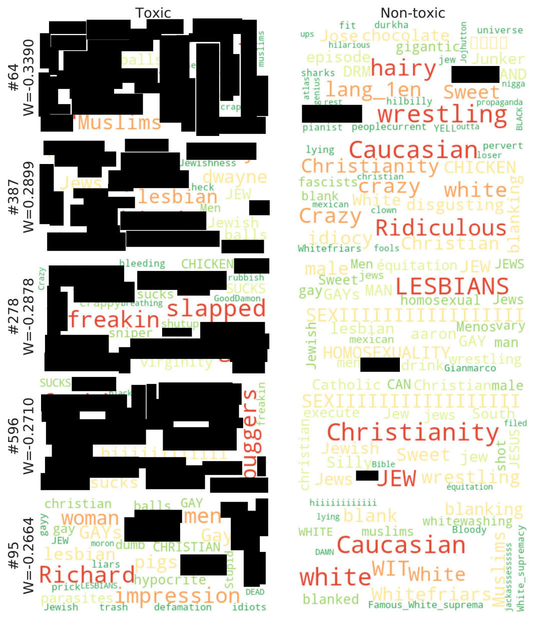

(b)

Figure B-10: Word cloud visualizations of the top 5 deep features in Debiased-BERT for the (a) sparse decision layer and (b) dense decision layer

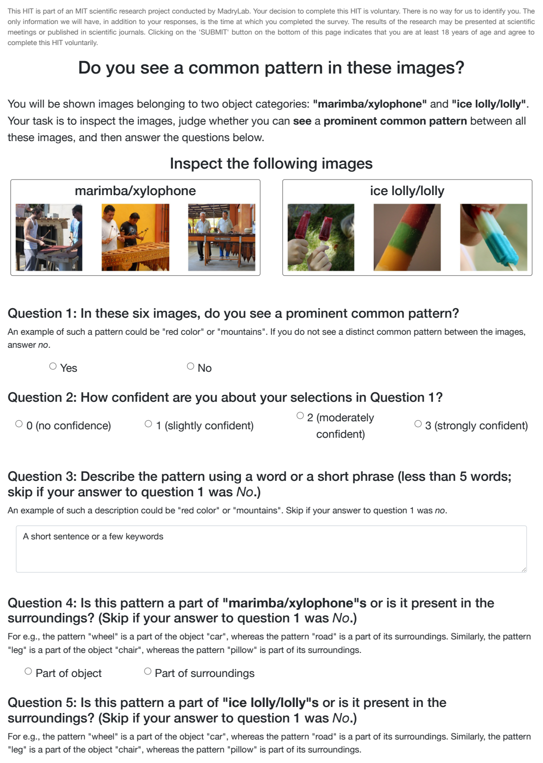
B.6.2 ImageNet

Human study

We now detail the setup of our MTurk study from Section 3.5.1. For our analysis, we use a standard ResNet-50 trained on the ImageNet dataset—with the default (dense) decision layer, as well as its sparse counterpart from Figure 3.1.

Task setup. This task is designed to semi-automatically identify learned correlations in classifiers with dense/sparse decision layers. To this end, we randomly-select 1000 class pairs from each model, such that the classes share a common deep feature in the decision layer. We only consider features to which the model assigns a substantial weight for both classes (>5% maximum weight). Then, for each class (from the pair), we select the three images that maximally activate the deep feature of interest. Doing so allows us to identify the most prototypical images from each class for the given deep feature.

We then present annotators on MTurk with the six chosen images, grouped by


This HIT is part of an MIT scientific research project conducted by MadryLab. Your decision to complete this HIT is voluntary. There is no way for us to identify you. The only information we will have, in addition to your responses, is the time at which you completed the survey. The results of the research may be presented at scientific meetings or published in scientific journals. Clicking on the 'SUBMIT' button on the bottom of this page indicates that you are at least 18 years of age and agree to complete this HIT voluntarily.

Do you see a common pattern in these images?

You will be shown images belonging to two object categories: **"marimba/xylophone"** and **"ice lolly/lolly"**. Your task is to inspect the images, judge whether you can **see** a **prominent common pattern** between all these images, and then answer the questions below.

Inspect the following images

Question 1: In these six images, do you see a prominent common pattern?

An example of such a pattern could be "red color" or "mountains". If you do not see a distinct common pattern between the images, answer *no*.

○ Yes ○ No

Question 2: How confident are you about your selections in Question 1?

○ 0 (no confidence) ○ 1 (slightly confident) ○ 2 (moderately confident) ○ 3 (strongly confident)

Question 3: Describe the pattern using a word or a short phrase (less than 5 words; skip if your answer to question 1 was *No*.)

An example of such a description could be "red color" or "mountains". Skip if your answer to question 1 was *no*.

A short sentence or a few keywords

Question 4: Is this pattern a part of "marimba/xylophone"s or is it present in the surroundings? (Skip if your answer to question 1 was *No*.)

For e.g., the pattern "wheel" is a part of the object "car", whereas the pattern "road" is a part of its surroundings. Similarly, the pattern "leg" is a part of the object "chair", whereas the pattern "pillow" is part of its surroundings.

○ Part of object ○ Part of surroundings

Question 5: Is this pattern a part of "ice lolly/lolly"s or is it present in the surroundings? (Skip if your answer to question 1 was *No*.)

For e.g., the pattern "wheel" is a part of the object "car", whereas the pattern "road" is a part of its surroundings. Similarly, the pattern "leg" is a part of the object "chair", whereas the pattern "pillow" is part of its surroundings.

Figure B-11: Sample MTurk task to diagnose (spurious) correlations in deep networks via their dense/sparse decision layers.

class along with the label. We ask them: (a) whether the images share a common pattern; (b) how confident they are about this selection on a likert scale; (c) to provide a short free text description of the pattern; and (d) for each class, to determine if the pattern is part of the class object or the surrounding. A sample task is shown in Figure B-11. Each task was presented to 5 annotators, compensated at $0.07 per task.

Quality control For each task, we aggregated results over all the annotators. While doing so, we eliminated individual instances where a particular annotator made no selections. We also completely eliminated instances corresponding to annotators who consistently (>80% of the time) left the task blank. Finally, while reporting our results, we only keep tasks for which we have selections from at least three (of five) annotators. We determine the final selection based on a majority vote over annotators, weighted by their confidence.

Algorithm 3 Counterfactual generation for a sentence of n words $x = (x_1, \dots x_n)$, a deep encoder $h : \mathbb{R}^n \to \mathbb{R}^m$, and a linear decision layer with coefficients (w, b).

```
1: $z = h(x)$ // calculate deep features
2: $y = \arg\max_y w_y z + b_y$ // calculate prediction
3: $Z^+, Z^- = \emptyset, \emptyset$ // initialize candidate word substitutions
4: for $i = 1 \dots m$ do
5:     for $j = 1 \dots n$ do
6:         if $x_j \in \text{WordCloud}^+(z_i) \wedge w_{yi} > 0$ then
7:             $Z^+ = Z^+ \cup \{(x_j, z_i)\}$ // candidate word substitution with positive weight and positive activation
8:         else if $x_j \in \text{WordCloud}^-(z_i) \wedge w_{yi} < 0$ then
9:             $Z^- = Z^- \cup \{(x_j, z_i)\}$ // candidate word substitution with negative weight and negative activation
10:        end if
11:    end for
12: end for
13: if $|Z^+ \cup Z^-| = 0$ then
14:     return-1 // No overlapping words found for counterfactual generation
15: end if
16: Randomly select $(x_j, z_i) \in Z^+ \cup Z^-$ // select a random word to substitute and its corresponding feature
17: if $(x_j, z_i) \in Z^+$ then
18:     Randomly select $\hat{x}_j \in \text{WordCloud}^-(z_i)$ // if positive, select a random negative word
19: else if $(x_j, z_i) \in Z^-$ then
20:     Randomly select $\hat{x}_j \in \text{WordCloud}^+(z_i)$ // if negative, select a random positive word
21: end if
22: $\hat{x} = (x_1, \dots, x_{j-1}, \hat{x}_j, x_{j+1}, \dots, x_n)$ // perform word substitution
23: return $\hat{x}$ // return generated counterfactual
```

B.7 Counterfactual experiments

B.7.1 Language counterfactuals

We describe in detail how to generate counterfactuals from the word cloud interpretations and the linear decision layer. The complete algorithm can be found in Algorithm 3, which we describe next.

Let $x = (x_1, \dots, x_n)$ be a sentence with n words, $z = f(s) \in \mathbb{R}^m$ be the deep encoding of x, and $y = \arg\max_y w_y z + b_y \in [k]$ be the model's prediction of x for a given decision layer with coefficients (w, b). Our goal is to generate a counterfactual that can flip the model's prediction y to some other class. Furthermore, let $\text{WordCloud}^+(z_i)$ and $\text{WordCloud}^-(z_i)$ be the LIME-based word clouds representing the positive and negative activations of ith deep feature, z_i. Then, counterfactual generation in the language setting involves the following steps:

1. Find all deep features which use words in x as evidence for the predicted label y (according to the word clouds). Specifically, calculate $Z = Z^- \cup Z^+$ where

$$Z^+ = \{(x_j, z_i) : \exists j \text{ s.t. } x_j \in \text{WordCloud}^+(z_i) \wedge w_{yi} > 0\} \quad \text{(B.3)}$$

$$Z^- = \{(x_j, z_i) : \exists j \text{ s.t. } x_j \in \text{WordCloud}^-(z_i) \wedge w_{yi} < 0\} \quad \text{(B.4)}$$

2. Randomly select a deep feature (and its word) $(x_j, z_i) \in Z$

3. If $z_i \in Z^+$, randomly select a word $\hat{x} \in \text{WordCloud}^-(z_i)$. Otherwise, if $z_i \in Z^-$, randomly select a word $\hat{x}_j \in \text{WordCloud}^+(z_i)$.

4. Perform the word substitution $x_j \to \hat{x}_j$ to get the counterfactual sentence, $\hat{x} = (x_0, \ldots, x_{j-1}, \hat{x}_j, x_{j+1}, \ldots, x_n)$.

Note that it is possible for there to be no features that use words in a given sentence as evidence for its prediction, which results in no candidate word substitutions (i.e. $\|Z\| = 0$). Consequently, it is possible for a sentence to have a counterfactual generated from the dense decision layer but not in the sparse decision layer (or vice versa). For our sentiment counterfactual experiments, we restrict our analysis to sentences which have counterfactuals in both the sparse and dense decision layers. However, we found that similar results hold if one considers all possible counterfactuals for each individual model instead.

B.7.2 ImageNet counterfactuals

In Figure B-12, we illustrate our pipeline for counterfactual image generation. Our starting point is a particular spurious correlation (between a data pattern and a target class) identified via the MTurk study in Section 3.5.1. We then select images from other ImageNet classes to add the spurious pattern to, and annotate the relevant region where it should be added. We obtain the spurious patterns by automatically scraping search engines. Finally, we combine the original images with the retrieved spurious pattern, using the mask as the weighting, to obtain the desired counterfactual images. These images are then supplied to the model, to test whether the addition of the spurious input pattern indeed fools the model into perceiving the counterfactuals as belonging to the target class.

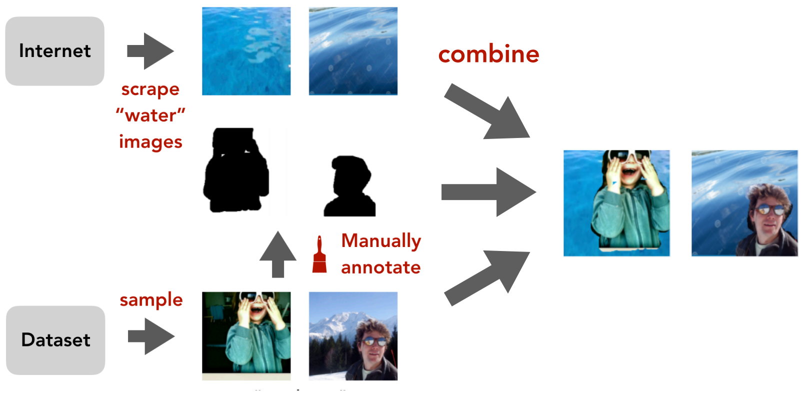


Figure B-12: Image counterfactual generation process. We start with a correlation identified during our MTurk study in Section 3.5.1—for example, the model associates "water" with the class "snorkel". To generate the counterfactuals shown in Figure 3-9a, we first select images from other ImageNet classes. We then manually annotate regions in these images to replaces with "water" bacgrounds obtained via automated image search on the Internet. Finally, we additively combine the "water" backgrounds and the original images, weighted by the mask, to obtain the resulting counterfactual inputs.

B.8 Validating ImageNet misclassifications

B.8.1 Human study

We now detail the setup of our MTurk study from Section 3.4.3. For our analysis, we use a ResNet-50 that has been adversarially-trained ($\varepsilon = 3$) on the ImageNet dataset. To obtain a sparse decision layer, we then train a sequence of GLMs via elastic net (cf. Section 3.3.1) on the deep representation of this network. Based on a validation set, we choose a single sparse decision layer—with 57.65% test accuracy and 39.18 deep features/class on average.

Task setup. In this task, our goal is to understand if annotators can identify data patterns that are responsible for misclassifications. To this end, we start by identifying deep features that are strongly activated for misclassified inputs.

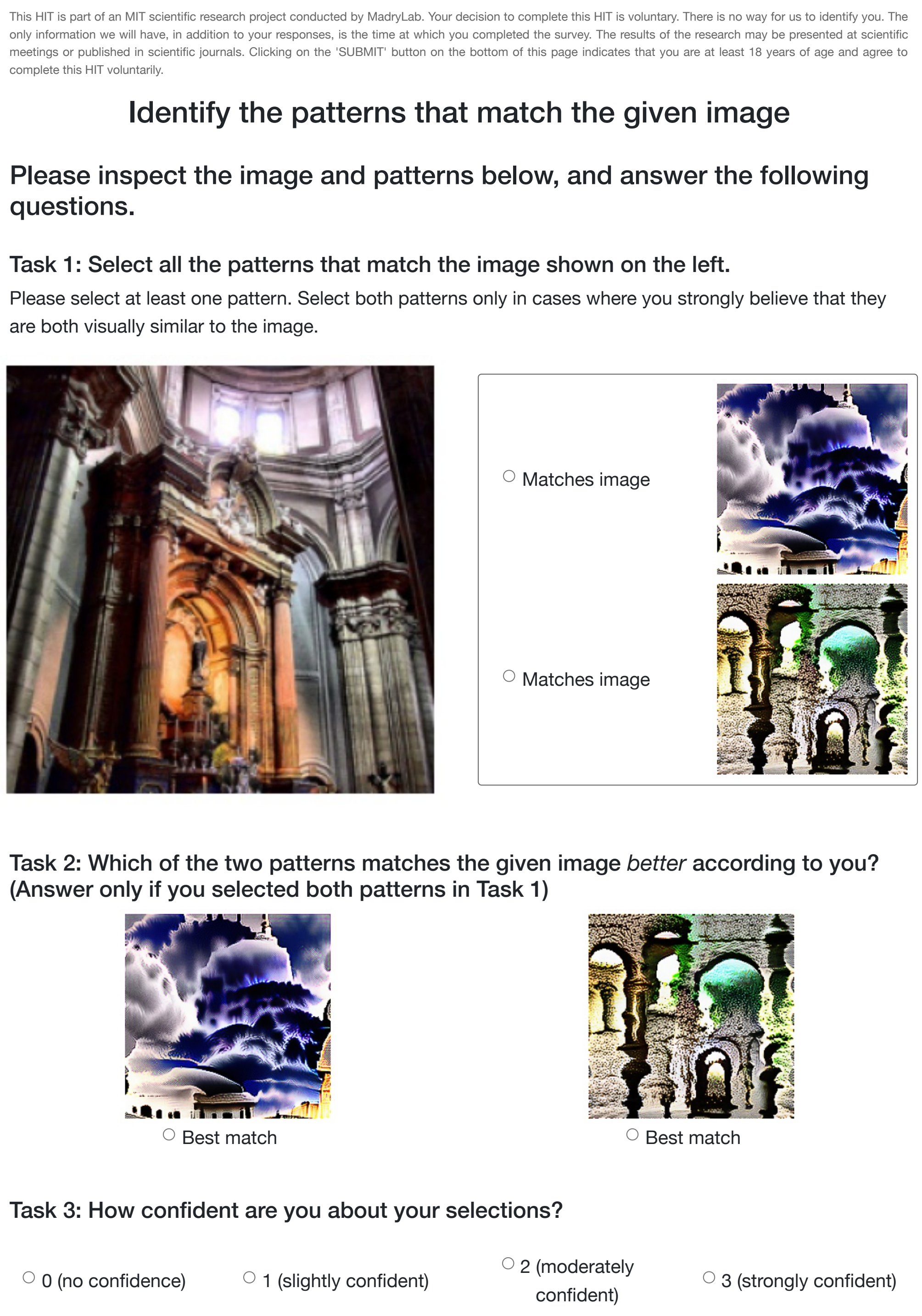
For any misclassified input x with ground truth label l and predicted class p, we can compute for every deep feature $f_i(x)$:

$$\gamma_i = W[p,i] \cdot f_i(x) - W[l,i] \cdot f_i(x) \tag{B.5}$$

where W is the weight matrix of the decision layer. Intuitively, this score measures the extent to which a deep feature contributes to the predicted class, relative to its contribution to the ground truth class. Then, sorting deep features based on decreasing/increasing values of this score, gives us a measure of how important each of them are for the predicted/ground truth label. Let us denote f_p as the deep feature with the highest score γ_i and f_l as the one with the lowest.

We find that for the robust ResNet-50 model with a sparse decision layer, the single top deep feature based on this score (f_p) alone is responsible for 26% of the misclassifications (5673 examples in all). That is, for each of these examples, simply turning $f_p = 0$ flips the model's prediction from p to l. We henceforth refer to these deep features (one per misclassified input) as "problematic" features.

For our task, we randomly subsample 1330 of the aforementioned 5673 misclas-



This HIT is part of an MIT scientific research project conducted by MadryLab. Your decision to complete this HIT is voluntary. There is no way for us to identify you. The only information we will have, in addition to your responses, is the time at which you completed the survey. The results of the research may be presented at scientific meetings or published in scientific journals. Clicking on the 'SUBMIT' button on the bottom of this page indicates that you are at least 18 years of age and agree to complete this HIT voluntarily.

Identify the patterns that match the given image

Please inspect the image and patterns below, and answer the following questions.

Task 1: Select all the patterns that match the image shown on the left.

Please select at least one pattern. Select both patterns only in cases where you strongly believe that they are both visually similar to the image.

○ Matches image

○ Matches image

Task 2: Which of the two patterns matches the given image *better* according to you? (Answer only if you selected both patterns in Task 1)

○ Best match

○ Best match

Task 3: How confident are you about your selections?

○ 0 (no confidence) ○ 1 (slightly confident) ○ 2 (moderately confident) ○ 3 (strongly confident)

Figure B-13: Sample MTurk task to identify input patterns responsible for the misclassifications in deep networks with the help of their (sparse) decision layers.

sified inputs. We then construct MTurk tasks, wherein annotators are presented with one such input (without any information about the ground truth or predicted labels), along with the feature visualizations for two deep features. These two features are either (with equal probability):

- (f_l, f_p): The deep features which (relatively) contribute most to the ground truth and predicted class respectively.
- (f_l, f_r): The deep feature which (relatively) contributes most to the ground truth class, along with a randomly-chosen one (out of the 2048 possible deep features). This is meant to serve as a control.

Annotators are then asked: (a) to select all the patterns (i.e., feature visualization of a deep feature) that match the image; (b) to select the one that best matches the image (if they selected both in (a)); (c) to mark their confidence on a likert scale. A sample task is shown in Figure B-13. Each task was presented to 5 annotators, compensated at $0.03 per task.

Note that, in the case where the ground truth label for each image is actually pertinent to it and that model relies on semantically-meaningful deep features for every class, we would expect annotators to select f_l to match the image 100% of the time. On the other hand, we would expect that annotators rarely select f_r to match the image.

Quality control For each task, we aggregated results over all the annotators. While doing so, we eliminated individual instances where a particular annotator made no selections. We also completely eliminated instances corresponding to annotators who consistently (>80% of the times) left the task blank. Finally, while reporting our results, we only keep tasks for which we have selections from at least two (of five) annotators. We determine the final selection based on a majority vote over annotators, weighted by their confidence.

Appendix C

Details for Chapter 4

C.1 Experimental setup

C.1.1 Datasets

We perform our analysis on the ImageNet dataset [Rus+15]. A full description of the data creation process can be found in Deng et al. [Den+09] and Russakovsky et al. [Rus+15]. For the purposes of our human studies, we use a random subset of the validation set—10,000 images chosen by sampling 10 random images from each of the 1,000 classes. (Model performance on this subset closely mirrors overall test accuracy, as shown in Figure 4-3b.) In our analysis, we refer to the original dataset labels as “ImageNet labels” or “IN labels”.

C.1.2 Models

We perform our evaluation on various standard ImageNet-trained models—see Appendix Table C.1 for a full list. We use open-source pre-trained implementations from `github.com/Cadene/pretrained-models.pytorch` and/or `github.com/rwightman/pytorch-image-models/tree/master/timm` for all architectures.

Model	Top-1	Top-5
alexnet [KSH12]	56.52	79.07
squeezenet1_1 [Ian+16]	57.12	80.13
squeezeNet1_0 [Ian+16]	57.35	79.89
vgg11 [SZ15]	68.72	88.66
vgg13 [SZ15]	69.43	89.03
inception_v3 [Sze+16]	69.54	88.65
googlenet [Sze+15]	69.78	89.53
vgg16 [SZ15]	71.59	90.38
mobilenet_v2 [San+18]	71.88	90.29
vgg19 [SZ15]	72.07	90.74
resnet50 [He+16]	76.13	92.86
efficientnet_b0 [TL19]	76.43	93.05
densenet161 [Hua+17]	77.14	93.56
resnet101 [He+16]	77.37	93.55
efficientnet_b1 [TL19]	78.38	94.04
wide resnet50_2 [ZK16]	78.47	94.09
efficientnet_b2 [TL19]	79.81	94.73
gluon_resnet152_v1d [He+19]	80.49	95.17
inceptionresnetv2 [Sze+17]	80.49	95.27
gluon_resnet152_v1s [He+19]	80.93	95.31
senet154 [HSS18]	81.25	95.30
efficientnet_b3 [TL19]	81.53	95.65
nasnetalarge [Zop+18]	82.54	96.01
pnasnet5large [Liu+18]	82.79	96.16
efficientnet_b4 [TL19]	83.03	96.34
efficientnet_b5 [TL19]	83.78	96.71
efficientnet_b6 [TL19]	84.13	96.96
efficientnet_b7 [TL19]	84.58	97.00

Table C.1: Models used in our analysis with the corresponding ImageNet top-1/5 accuracies.

C.2 Obtaining Image Annotations

Our goal is to use human annotators to obtain labels for each distinct object in ImageNet images (provided it corresponds to a valid ImageNet class). To make this classification task feasible, we first identify a small set of relevant candidate labels per image to present to annotators.

C.2.1 Obtaining candidate labels

As discussed in Section 4.4.1, we narrow down the candidate labels for each image by (1) restricting to the predictions of a set of pre-trained ImageNet models, and then (2) repeating the CONTAINS task on human annotators using the labels from (1) to identify the most reasonable ones.

Pre-filtering using model predictions

We use the top-5 predictions of models with varying ImageNet (validation) accuracies (10 in total): `alexnet`, `resnet101`, `densenet161`, `resnet50`, `googlenet`, `efficientnet_b7` `inception_v3`, `vgg16`, `mobilenet_v2`, `wide_resnet50_2` (cf. Table C.1) to identify a set of *potential labels*. Since model predictions tend to overlap, we end up with ~ 14 potential labels per image on average (see full histogram in Figure C-1). We always include the ImageNet label in the set of potential labels, even if it is absent in all the model predictions.

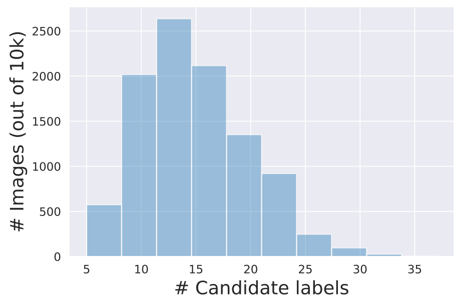


Figure C-1: Distribution of labels per image obtained from the predictions of ImageNet-trained models (plus the ImageNet label). We present these labels (in separate grids) to annotators via the CONTAINS task (cf. Section 4.4.1)to identify a small set of relevant candidate labels for the classification task in Section 4.4.2.

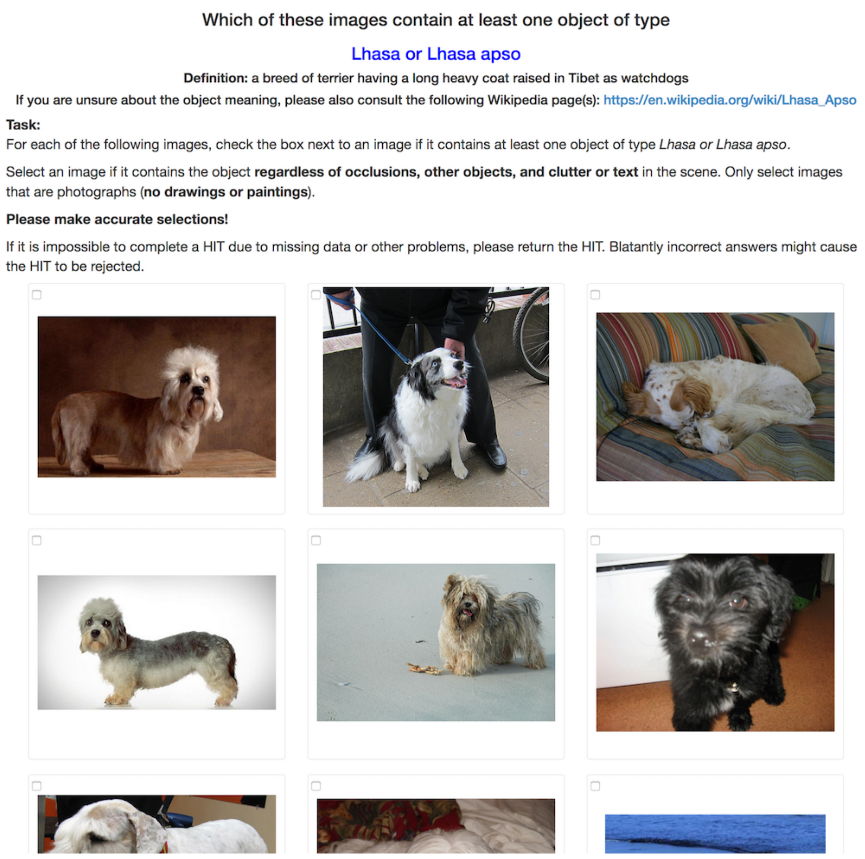
Multi-label validation task

We then use human annotators to go through these potential labels and identify the most reasonable ones via the CONTAINS task. Recall that in CONTAINS task, annotators are shown a grid of images and asked to select the ones that contain an object corresponding to the specified query label. In our case, each image appears in multiple such grids—one for each potential label. By presenting these grids to multiple annotators, we can then obtain a selection frequency for every image-potential label pair, i.e., the number of annotators that perceive the label as being contained in the image (cf. Figure 4-6). Using these selection frequencies, we identify the most relevant *candidate labels* for each image.

Grid setup. The grids used in our study contains 48 images, at least 5 of which are *controls*—obtained by randomly sampling from validation set images labeled as the query class. Along with the images, annotators are provided with a description of the query label in terms of (a) WordNet synsets and (b) the relevant Wikipedia link—see Figure C-2 for an example. (Our MTurk interface is based on a modified version of the code made publicly available by Recht et al. [Rec+19b][1].) We find that a total of 3, 934 grids suffice to obtain selection frequencies for all 10k images used in our analysis (w.r.t. all potential labels). Every grid was shown to 9 annotators, compensated \$0.20 per task.

Quality control. We filtered low-quality responses on a per-annotator and per-task basis. First, we completely omitted results from annotators who selected less than 20% of the control images on half or more of the tasks they completed: a total of 10 annotators and the corresponding 513 tasks. Then we omitted tasks for which less than 40% of the controls were selected: at total of 3,104 tasks. Overall, we omitted 3,617 tasks in total out of the total 35,406. As a result, the selection frequency of some image-label pairs will be computed with fewer than 9 annotators.

[1]`https://github.com/modestyachts/ImageNetV2`


Which of these images contain at least one object of type

Lhasa or Lhasa apso

Definition: a breed of terrier having a long heavy coat raised in Tibet as watchdogs

If you are unsure about the object meaning, please also consult the following Wikipedia page(s): https://en.wikipedia.org/wiki/Lhasa_Apso

Task:
For each of the following images, check the box next to an image if it contains at least one object of type *Lhasa or Lhasa apso*.

Select an image if it contains the object **regardless of occlusions, other objects, and clutter or text** in the scene. Only select images that are photographs (**no drawings or paintings**).

Please make accurate selections!

If it is impossible to complete a HIT due to missing data or other problems, please return the HIT. Blatantly incorrect answers might cause the HIT to be rejected.

Figure C-2: Sample interface of the CONTAINS task we use for label validation: annotators are shown a grid of 48 images and asked to select all images that correspond to a specific label (Section 4.4.1).

Final candidate label selection

We then obtain the most relevant candidate labels by selecting the potential labels with high human selection frequency. To construct this set, we consider (in order):

1. The existing ImageNet label, irrespective of its selection frequency.

2. All the highly selected potential labels: for which annotator selection frequency is at least 0.5.

3. All potential labels with non-zero selection frequency that are semantically very different from the ImageNet label—so as to include labels that may correspond to different objects. Concretely, we select candidate labels that are more than 5 nodes away from the ImageNet label in the WordNet graph.

4. If an image has fewer than 5 candidates, we also consider other potential labels after sorting them based on their selection frequency (if non-zero).

5. To keep the number of candidates relatively small, we truncate the resulting set size to 6 if the excess labels have selection frequencies lower than the ImageNet label, or the ImageNet label itself has selection frequency $\leq 1/8$. During this truncation, we explicitly ensure that the ImageNet label is retained.

In Figure C-3, we visualize the distribution of number of candidate labels per image, over the set of images.

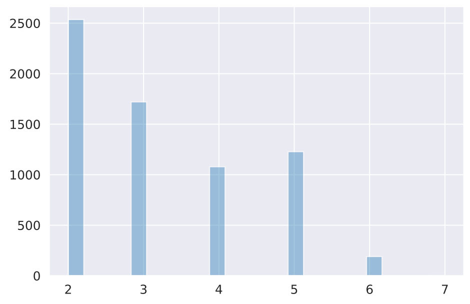
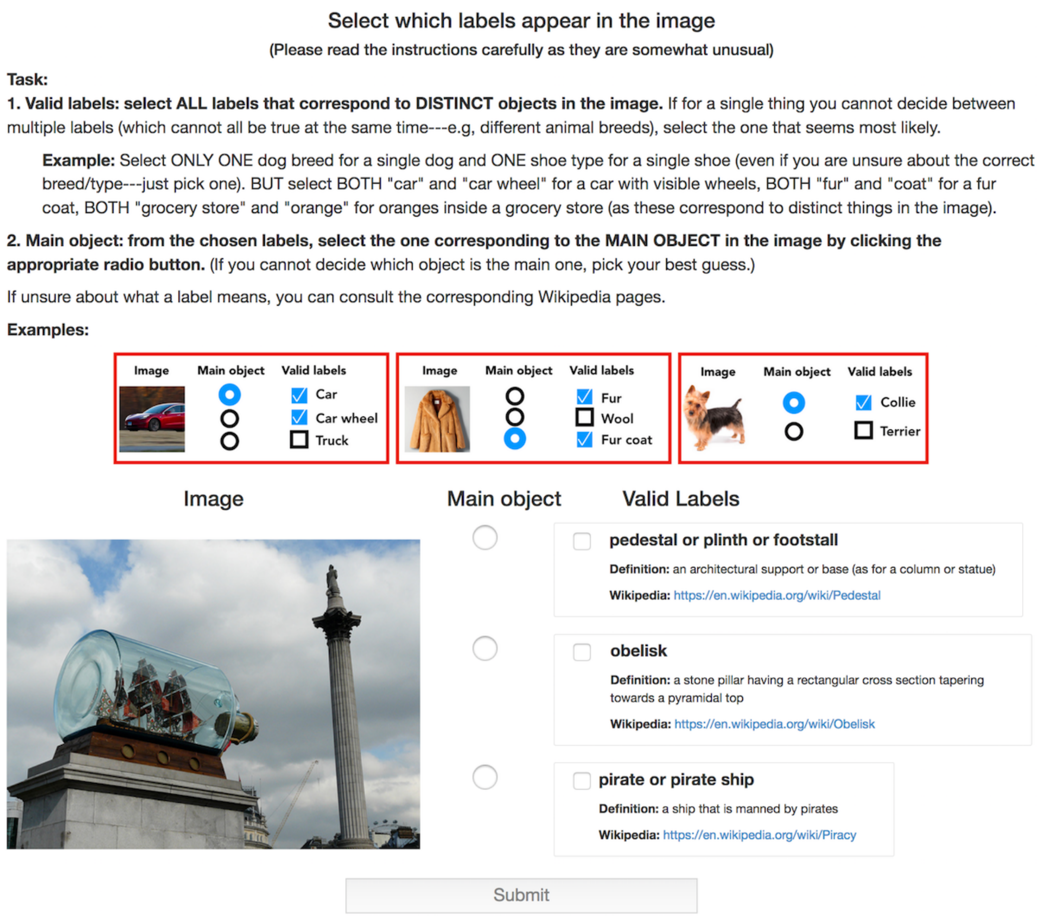


Figure C-3: Distribution of the number of candidate labels used per image presented to annotators during the classification task in Section 4.4.2.

C.2.2 Image classification

The candidate labels are then presented to annotators during the CLASSIFY task (cf. Section 4.4.2). Specifically, annotators are shown images, and their corresponding candidate labels and asked to select: a) all valid labels for that image, b) a label for the main object of the image—see Figure C-4 for a sample task interface. We instruct annotators to pick multiple labels as valid, only if they correspond to different objects in the image and are not mutually exclusive. In particular, in case of confusion about a specific object label, we explicitly ask them to pick a single label making their best guess. Each task was presented to 9 annotators, compensated at $0.08 per task.


Select which labels appear in the image

(Please read the instructions carefully as they are somewhat unusual)

Task:

1. Valid labels: select ALL labels that correspond to DISTINCT objects in the image. If for a single thing you cannot decide between multiple labels (which cannot all be true at the same time---e.g, different animal breeds), select the one that seems most likely.

Example: Select ONLY ONE dog breed for a single dog and ONE shoe type for a single shoe (even if you are unsure about the correct breed/type---just pick one). BUT select BOTH "car" and "car wheel" for a car with visible wheels, BOTH "fur" and "coat" for a fur coat, BOTH "grocery store" and "orange" for oranges inside a grocery store (as these correspond to distinct things in the image).

2. Main object: from the chosen labels, select the one corresponding to the MAIN OBJECT in the image by clicking the appropriate radio button. (If you cannot decide which object is the main one, pick your best guess.)

If unsure about what a label means, you can consult the corresponding Wikipedia pages.

Examples:

Image	Main object	Valid labels
	●	✓ Car
	○	✓ Car wheel
	○	☐ Truck

Image	Main object	Valid labels
	○	✓ Fur
	○	☐ Wool
	●	✓ Fur coat

Image	Main object	Valid labels
	●	✓ Collie
	○	☐ Terrier

Image	Main object	Valid Labels
	○	☐ **pedestal or plinth or footstall** **Definition:** an architectural support or base (as for a column or statue) **Wikipedia:** https://en.wikipedia.org/wiki/Pedestal
	○	☐ **obelisk** **Definition:** a stone pillar having a rectangular cross section tapering towards a pyramidal top **Wikipedia:** https://en.wikipedia.org/wiki/Obelisk
	○	☐ **pirate or pirate ship** **Definition:** a ship that is manned by pirates **Wikipedia:** https://en.wikipedia.org/wiki/Piracy

Figure C-4: Screenshot of a sample image annotation task. (Section 4.4.2). Annotators are presented with an image and multiple candidate labels. They are asked to select all valid labels (selecting only one of mutually exclusive labels in the case of confusion) and indicate the main object of the image.

Images included. We only conduct this experiment on images that annotators identified as having *at least* one candidate label outside the existing ImageNet label (based on experiment in Appendix C.2.1). To this end, we omitted images for which the ImageNet label was clearly the most likely: out of all the labels seen by 6 or more of the 9 annotators, it had more than double the selection frequency of any other class. Note that since we discard some tasks as part of quality control, it is possible that for some image-label pairs, we have the results of fewer than 9 annotators. Furthermore, we also omitted images which were not selected by any annotator as containing their ImageNet label (150 images total)—cf. Appendix Figure C-8 for examples. These likely corresponds to labeling mistakes in the dataset creation process and do not reflect the systemic error we aim to study. The remaining 6,761 images that are part of our follow-up study have at least 1 label, in addition to the ImageNet label, that annotators think could be valid.

Quality control. Performing stringent quality checks for this task is challenging since we do not have ground truth annotations to compare against—which was after all the original task motivation. Thus, we instead perform basic sanity checks for quality control—we ignore tasks where annotators did not select *any* valid labels or selected a main label that they did not indicate as valid. In addition, if the tasks of specific annotators are consistently flagged based on these criteria (more than a third of the tasks), we ignore all their annotations. Overall, we omitted 1,269 out of the total 59,580 tasks.

The responses of multiple annotators are aggregated as described in Section 4.4.2.

C.3 Additional experimental results

C.3.1 Multi-object Images

We observe that annotators tend to agree on the number of objects present—see Figure C-5.

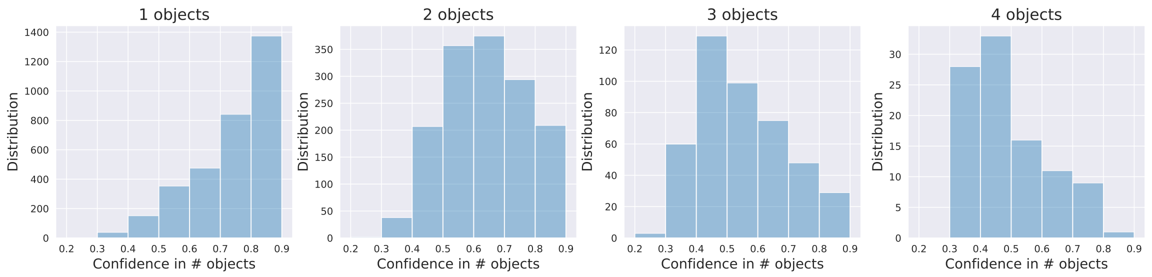


Figure C-5: Annotator agreement for multi-object images. Recall that we determine the number of objects in an image based on a majority vote over annotators. Here, we define "confidence" as the fraction of annotators that make up that majority, relative to the total number of annotators shown the image (cf. Section 4.4.2). We visualize the distribution of annotator confidence, as a function of the number of image objects.

Top-5 accuracy in the multi-label context. The issue of label ambiguity that can arise in multi-object images was noted by the creators of the ILSVRC challenge [Rus+15]. To tackle this issue, they proposed evaluating models based on top-5 accuracy. Essentially, a model is deemed correct if any of the top 5 predicted labels match the ImageNet label. We find that model top-5 accuracy is much higher than top-1 (or even our notion of multi-label) accuracy on multi-object images—see Figure C-6. However, a priori, it is not obvious whether this increase is actually because of adjusting for model confusion between distinct objects. In fact, one would expect that properly accounting for such multi-object confusions should yield numbers similar to (and not markedly higher than) top-1 accuracy on single-object images.

To get a better understanding of this, we visualize the fraction of *top-5 corrections*—images for which the ImageNet label was not the top prediction of the model, but was in the top 5—that correspond to *different objects* in the image. Specifically, we only consider images where the top model prediction and ImageNet label were selected by annotators as: (a) present in the image and (b) corresponding to different objects. We observe that the fraction of top-5 corrections that correspond to multi-object images is relatively small—about 20% for more recent models. This suggests that top-5 accuracy may be overestimating model performance and, in a sense, masking model errors on single objects. Overall, these findings highlight the need for designing better performance metrics that reflect the underlying dataset structure.

C.3.2 Bias in label validation

Potential biases in selection frequency estimates. In the course of obtaining fine-grained image annotations, we collect selection frequencies for several potential image labels (including the ImageNet label) using the CONTAINS task (cf. Section 4.4.1). Recall however, that during the ImageNet creation process, every image was already validated w.r.t. the ImageNet label (also via the CONTAINS task) by a different pool of annotators, and only images with high selection frequency actually made it into the dataset. This fact will result in a *bias* for our new

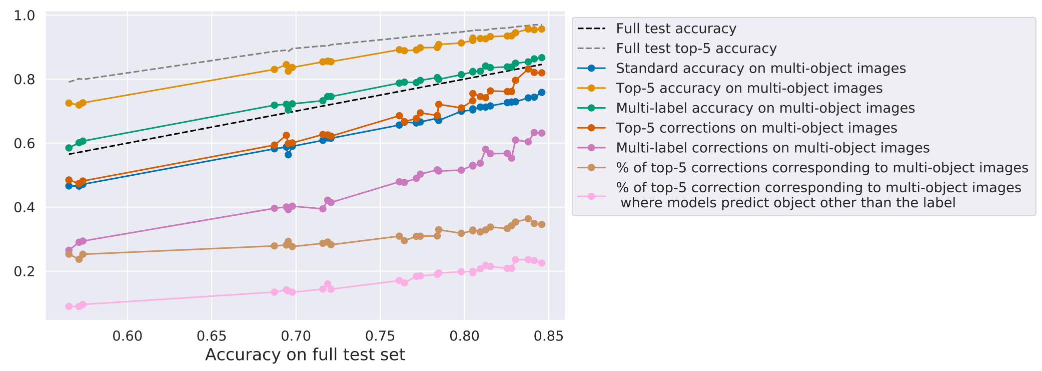


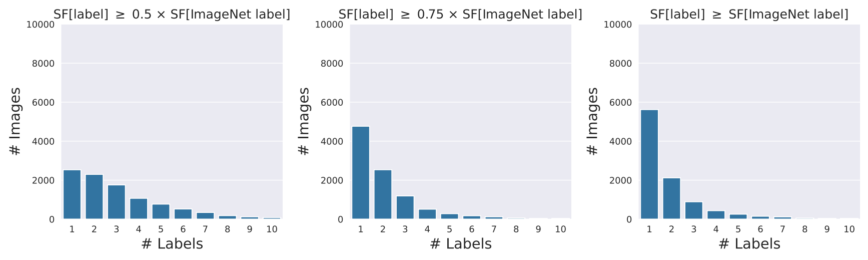
Figure C-6: A closer look at top-5 accuracy: we visualize top-1, top-5 and multi-label (cf. Section 4.5.1) on multi-object images in ImageNet. We also measure the fraction of top-5 corrections (ImageNet label is not top model prediction, but is among top 5) that correspond to multi-object confusions—wherein the ImageNet label and top prediction belong to distinct image objects as per human annotators. We see that although top-5 accuracy is much higher than top-1, even for multi-object images, it may be overestimating model performance. In particular, a relatively small fraction of top-5 corrections actually correspond to the aforementioned multi-object images.

selection frequency measurements [Eng+20]. At a high level, if we measure a low selection frequency for the ImageNet label of an image, it is more likely that we are observing an underestimate, rather than the actual selection frequency being low. In order to understand whether this bias significantly affects our findings, we reproduce the relevant plots in Figure C-7 using only a subset of workers (this should exacerbate the bias allowing us to detect it). We find however, that the difference is quite small, not changing any of the conclusions. Moreover, since most of our analysis is based on the per-image annotation task for which this specific bias does not apply, we can effectively ignore it in our study.

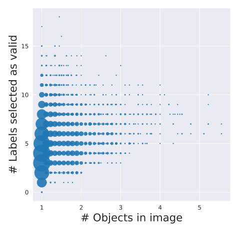
C.3.3 Mislabeled examples

In the course of our human studies in Section 4.4.1, we also identify a set of possibly mislabeled ImageNet images. Specifically, we find images for which:

- Selection frequency for the ImageNet label is 0, i.e., no annotator selected the label to be contained in the image (cf. Section 4.4.1). We identify 150 (of 10k) such images— cf. Appendix Figure C-8 for examples.



(a)



(b)

Figure C-7: Effect of subsampling annotator population (5 instead of 9 annotators): (a) Number of labels annotators consider valid determined based on the selection frequency of a label relative to that of the ImageNet label. Even in this annotator subpopulation, for >70% of images, another label is still selected at least half as often as they select the ImageNet label (*leftmost*). (b) Number of labels that at least one (of five) annotators selected as valid for an image (cf. Section 4.4.1) versus the number of objects in the image (cf. Section 4.4.2). (Dot size is proportional to the number of images in each 2D bin.) Even when annotators consider the image as containing only a single object, they often select multiple labels as valid.

- The ImageNet label was not selected at all (for any object) during the detailed image annotation phase in Section 4.4.2. We identify 119 (of 10k) such images— cf. Appendix Figure C-9 for examples.

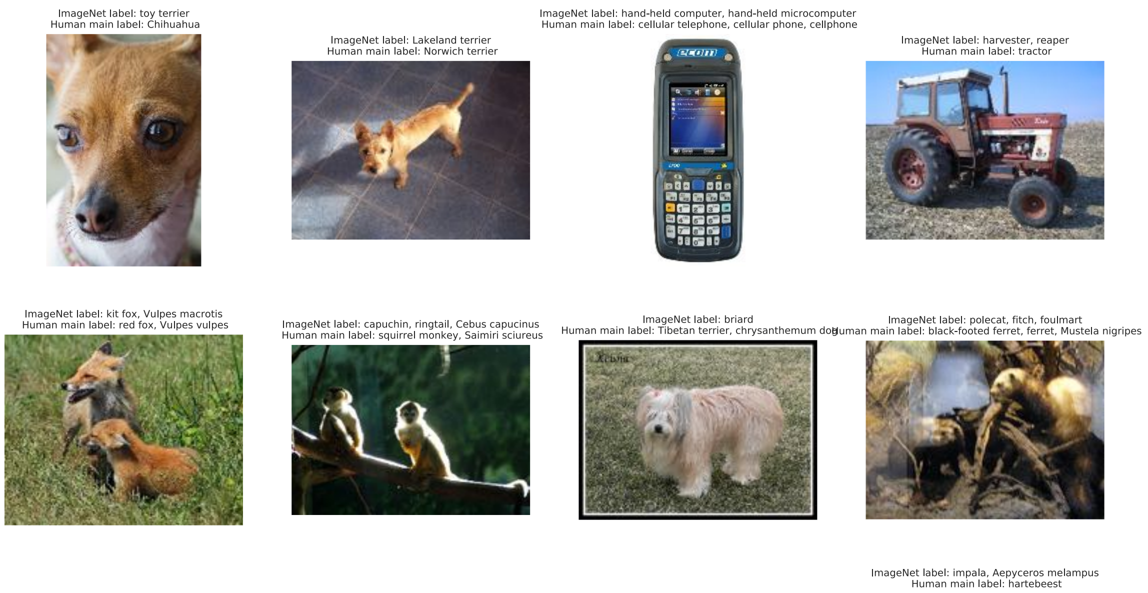

ImageNet label: toy terrier
Human main label: Chihuahua
ImageNet label: Lakeland terrier
Human main label: Norwich terrier
ImageNet label: hand-held computer, hand-held microcomputer
Human main label: cellular telephone, cellular phone, cellphone
ImageNet label: harvester, reaper
Human main label: tractor
ImageNet label: kit fox, Vulpes macrotis
Human main label: red fox, Vulpes vulpes
ImageNet label: capuchin, ringtail, Cebus capucinus
Human main label: squirrel monkey, Saimiri sciureus
ImageNet label: briard
Human main label: Tibetan terrier, chrysanthemum dog
ImageNet label: polecat, fitch, foulmart
Human main label: black-footed ferret, ferret, Mustela nigripes
ImageNet label: impala, Aepyceros melampus
Human main label: hartebeest

Figure C-8: Possibly mislabeled images: human selection frequency for the ImageNet label is 0 (cf. Section 4.4.1). Also depicted is the label most frequently selected by the annotators as contained in the image (*sel*).

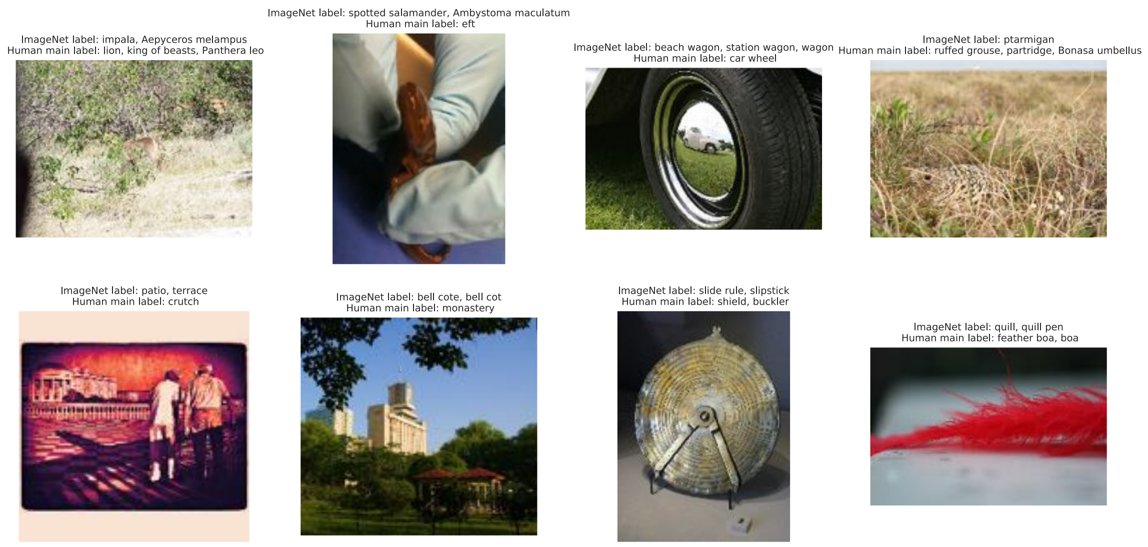

ImageNet label: impala, Aepyceros melampus
Human main label: lion, king of beasts, Panthera leo
ImageNet label: spotted salamander, Ambystoma maculatum
Human main label: eft
ImageNet label: beach wagon, station wagon, wagon
Human main label: car wheel
ImageNet label: ptarmigan
Human main label: ruffed grouse, partridge, Bonasa umbellus
ImageNet label: patio, terrace
Human main label: crutch
ImageNet label: bell cote, bell cot
Human main label: monastery
ImageNet label: slide rule, slipstick
Human main label: shield, buckler
ImageNet label: quill, quill pen
Human main label: feather boa, boa

Figure C-9: Possibly mislabeled images: ImageNet label is not selected by any of the annotators during fine-grained image annotation process described in Section 4.4.2. Also shown in the title is label that was most frequently selected by the annotators as denoting the main object in the image (*sel*).

Appendix D

Details for Chapter 5

D.1 Experimental Setup

D.1.1 Datasets

For our experimental analysis, we use the MNIST [LeC98], CIFAR-10 [Kri09], Restricted ImageNet (cf. Appendix A.1.1), and ImageNet-1k [Rus+15] datasets. For image translation we use the Horse $\leftrightarrow$ Zebra, Apple $\leftrightarrow$ Orange, and Summer $\leftrightarrow$ Winter datasets [Zhu+17].

D.1.2 Models

- MNIST: We use the convolution architecture from the TensorFlow tutorial[1].
- CIFAR-10: We consider a standard ResNet model [He+15a]. It has 4 groups of residual layers with filter sizes (16, 16, 32, 64) and 5 residual units each[2].
- Restricted ImageNet (RIN): We use a ResNet-50 [He+15a] architecture using the code from the `tensorpack` repository [Wu+16], trained with data augmentation, momentum 0.9 and weight decay $5e^{-4}$.
- ImageNet: We use a ResNet-50 [He+15a] architecture. For standard (not adversarially trained) classifiers on the complete 1k-class ImageNet dataset, we

[1] `https://github.com/MadryLab/mnist_challenge/`
[2] `https://github.com/MadryLab/cifar10_challenge/`

use pre-trained models provided in the PyTorch repository[3].

Other hyperparameters are provided in Table D.1 The exact procedure used to train robust models along with the corresponding hyperparameters are described in Section D.1.3.

Dataset	Model	Epochs	LR	Batch Size	LR Schedule
RIN	standard	110	0.1	256	Drop by 10 at epochs $\in [30, 60]$
RIN	robust	110	0.1	256	Drop by 10 at epochs $\in [30, 60]$
ImageNet	robust	110	0.1	256	Drop by 10 at epochs $\in [100]$

Table D.1: Standard hyperparameters for model training.

D.1.3 Adversarial training

To obtain robust classifiers, we employ the adversarial training methodology proposed in [Mad+18]. Specifically, we train against a projected gradient descent (PGD) adversary with a normalized step size, starting from a random initial perturbation of the training data. We consider adversarial perturbations in ℓ_2-norm. Unless otherwise specified, we use the values of ϵ provided in Tables D.2-D.4 to train/evaluate our models (the images themselves lie in the range $[0, 1]$).

Adversary	**Binary MNIST**	**MNIST**	**CIFAR-10**	**Restricted Imagenet**
ℓ_∞	0.2	0.3	$\frac{4}{255}$	0.005
ℓ_2	-	1.5	0.314	1

Table D.2: Value of ϵ used for adversarial training/evaluation in Figure 5-3.

D.1.4 Model Performance

Standard and adversarial test performance for the models used are presented in Table D.5 for the Restricted ImageNet dataset and in Table D.6 for the complete

[3]https://pytorch.org/docs/stable/torchvision/models.html

Dataset	ϵ	# steps	Step size
Restricted ImageNet	3.0	7	0.5
ImageNet	3.0	7	0.5

Table D.3: Hyperparameters used for adversarial training in Section 5.3.

Dataset	ϵ	# steps	Step size
CIFAR-10	0.5	7	0.1
restricted ImageNet	3.5	7	0.1
ImageNet	3	7	0.5
Horse $\leftrightarrow$ Zebra	5	7	0.9
Apple $\leftrightarrow$ Orange	5	7	0.9
Summer $\leftrightarrow$ Winter	5	7	0.9

Table D.4: Hyperparameters used for adversarial training in Section 5.4.

ImageNet dataset. Here, adversarial accuracies are computed against a PGD adversary with 20 steps and step size of 0.375. (We also evaluated against a stronger adversary using more steps (100) of PGD, however this had a marginal effect on the adversarial accuracy of the models.)

Model	Standard	Adversarial (eps=3.0)
Standard ResNet-50	98.01%	4.74%
Robust ResNet-50	92.39%	81.91%

Table D.5: Test accuracy for standard/robust models on the Restricted ImageNet dataset.

Model	Standard	Adversarial (eps=3.0)
Standard ResNet-50	76.13%	0.13%
Robust ResNet-50	57.90%	35.16%

Table D.6: Test accuracy for standard/robust models on the ImageNet dataset.

D.1.5 Hyperparameter setup

Finding representation-feature correspondence

Dataset	ϵ	# steps	Step size
Restricted ImageNet/ImageNet	1000	200	1

Inverting representations and interpolations

Dataset	ϵ	# steps	Step size
Restricted ImageNet/ImageNet	1000	10000	1

Targeted Attacks in Figure 5-12

Dataset	ϵ	# steps	Step size
restricted ImageNet	300	500	1

Generation

In order to compute the class conditional Gaussians for high resolution images ($224\times224\times3$) we downsample the images by a factor of 4 and upsample the resulting seed images with nearest neighbor interpolation.

Dataset	ϵ	# steps	Step size
CIFAR-10	30	60	0.5
restricted ImageNet	40	60	1
ImageNet	40	60	1

Inpainting

To create a corrupted image, we select a patch of a given size at a random location in the image. We reset all pixel values in the patch to be the average pixel value over the entire image (per channel).

Dataset	patch size	ϵ	# steps	Step size
restricted ImageNet	60	21	0.1	720

Image-to-image translation

Dataset	ϵ	# steps	Step size
ImageNet	60	80	1
Horse $\leftrightarrow$ Zebra	60	80	0.5
Apple $\leftrightarrow$ Orange	60	80	0.5
Summer $\leftrightarrow$ Winter	60	80	0.5

Super-resolution

Dataset	$\uparrow$ factor	ϵ	# steps	Step size
CIFAR-10	7	15	1	50
restricted ImageNet	8	8	1	40

D.2 Additional Experimental Results

Interpolations for standard models

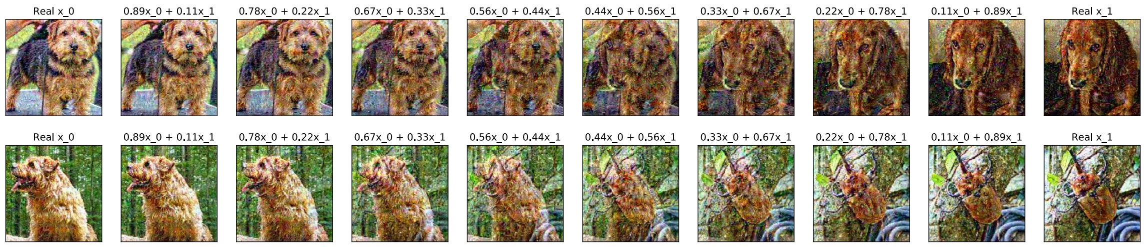


Figure D-1: Image interpolation using standard representations. To find the interpolation in input space, we construct images that map to linear interpolations of the endpoints in standard representation space. Concretely, for randomly selected pairs from the Restricted ImageNet test set, we use (5.3) to find images that match to the linear interpolates in representation space (5.4). Image space interpolations from the standard model appear to be significantly less meaningful than their robust counterparts. They are visibly similar to linear interpolation directly in the input space, which is in fact used to seed the optimization process.

Appendix E

Details for Chapter 6

E.1 Experimental Setup

E.1.1 Datasets

Unless otherwise specified, we use the ImageNet-1k [Den+09; Rus+15] and Places-365 [Zho+17] datasets which contain images from 1,000 and 365 categories respectively for our analysis. In particular, editing is performed on (or using) samples from the standard test sets to avoid overlap with the training data used to develop the models.

Since we manually collected all the data necessary for our analysis in Section 6.3.3, we were able to filter them for offensive content. Moreover, we made sure to only collect images that are available under a Creative Commons license (hence allowing non-commercial use with proper attribution).

For the rest of our analysis, we relied on publicly available datasets that are commonly used for image classification. Unfortunately, due to their scale, these datasets have not been thoroughly filtered for offensive content or identifiable information. In fact, improving these datasets along this axis is an active area of work[1]. Nevertheless, since our research did not involve redistributing these datasets or presenting them to human annotators, we did not perceive any additional risks that would result from our work.

[1] `https://www.image-net.org/update-mar-11-2021.php`

E.1.2 Models

Here, we describe the exact architecture and training process for each model we use. For the bulk of our analysis, we utilize two canonical, yet relatively diverse model architectures for our study: namely, VGG [SZ15] and ResNet [He+16]. We use the standard PyTorch implementation [2] and train the models from scratch on the ImageNet and Places365 datasets. The accuracy of each model on the corresponding test set is provided in Table E.1.

ImageNet classifiers. We study: (i) a VGG16 variant with batch normalization and (ii) a ResNet-50. Both models are trained using standard hyperparameters: SGD for 90 epochs with an initial learning rate of 0.1 that drops by a factor of 10 every 30 epochs. We use a momentum of 0.9, a weight decay of 10^{-4} and a batch size of 256 for the VGG16 and 512 for the ResNet-50.

Places365 classifiers. We study: (i) a VGG16 and (ii) a ResNet-18. Both models are trained for 131072 iterations using SGD with a single-cycle learning rate schedule peaking at 2e-2 and descending to 0 at the end of training. We use a momentum 0.9, a weight decay 5e-4 and a batch size of 256 for both models.

CLIP. We use the ResNet-50 models trained via CLIP [Rad+21], as provided in the original model repository.[3]

	Test Accuracy (%)	
Architecture \ Dataset	ImageNet	Places
VGG	73.70	54.02
ResNet	75.77	54.24
CLIP-ResNet	59.84	-

Table E.1: Model accuracy on the datasets used in our analysis.

[2]`https://pytorch.org/vision/stable/models.html`
[3]`https://github.com/openai/CLIP`

E.1.3 Model rewriting

Here, we describe the training setup of our model editing process, as well as the fine-tuning baseline. Recall that these rewrites are performed with respect to a single concept-style pair.

Layers. We consider a layer to be a block of convolution-BatchNorm-ReLU, similar to Bau et al. [Bau+20a] and rewrite the weights of the convolution. For ResNets (which were not previously studied), we must also account for skip connections. In particular, note that the effect of a rewrite to a layer inside any residual block will be attenuated (or canceled) by the skip connection. To avoid this, we only rewrite the final layer within each residual block—i.e., focus on the convolution-BatchNorm-ReLU right before a skip connection, and include the skip connection in the output of the layer. Unless otherwise specified, we perform rewrites to layers $[8, 10, 11, 12]$ for VGG models, $[4, 6, 7]$ for ResNet-18, and $[8, 10, 14]$ for ResNet-50 models. We tried earlier layers in our initial experiments, but found that both methods perform worse.

Editing

We use the ADAM optimizer with a fixed learning rate to perform the optimization in (6.2). We grid over different learning rate-number of step pairs: $[(10^{-3}, 10000), (10^{-4}, 20000), (10^{-5}, 40000), (10^{-6}, 80000), (10^{-7}, 80000)]$. The second order statistics are computed based on the keys for the entire test set.

Fine-tuning

When fine-tuning a single layer (local fine-tuning), we optimize the weights of the convolution of that particular layer. Instead, when we fine-tune a suffix of the model (global fine-tuning), we optimize all the trainable parameters including and after the chosen layer. In both cases, we use SGD, griding over different learning rate-number of step pairs:

$[(10^{-2}, 500), (10^{-3}, 500), (10^{-4}, 500), (10^{-5}, 800), (10^{-6}, 800)]$.

E.1.4 Evaluation: Synthetic concept-level transformations

We now describe the details of our evaluation in Section 6.3.2, namely, how we chose which concept-style pairs to use for testing and how we chose the hyperparameters for each method.

Selecting concept-style pairs

Concept selection. Recall that our rule-discovery pipeline from Section 3.2 identifies concepts which, when transformed in a certain manner hurts model accuracy on one or more classes. We first filter these concepts (automatically) to identify ones that are particularly salient in the model's prediction-making process. In particular, we focus on concepts which simultaneously: (a) affect at least 3 classes; (b) are present in at least 20% percent of the test images of each class; and (c) cause a drop of at least 15% among these images. This selection results in a test bed where we can meaningfully observe differences in performance between approaches.

At the same time, we need to also ensure that the rewriting task we are solving is meaningful. For instance, if we replace all instances of "dog" with a stylized version, then distinguishing between a "terrier" and a "poodle" can become challenging (or even impossible). Moreover, we cannot expect model performance to improve on other dog breeds if we modify it to treat a stylized dog as a "terrier". To eliminate such test cases, we manually filter the concept-class pairs flagged by our prediction-rule discovery pipeline. In particular, we removed those where the detected concept overlapped significantly with the class object itself. In other words, if the concept detected is essential for correctly recognizing the class of the image, we exclude it from our analysis. Typical examples of excluded concept-class pairs on ImageNet include broad animal categories (e.g., "bird" or "dog") for classes corresponding to specific breeds (e.g., "parrot") or the concept "person" which overlaps with classes corresponding to articles of clothing (e.g., "suit").

Style selection. We consider a subset of 8 styles used in our prediction-rule discovery pipeline (cf. Figure 3-2): "black and white", "floral", "fall colors", "furry", "graffiti", "gravel", "snow" and "wooden". While performing editing with respect to a single concept-style pair—say "wheel"-"wooden"—we randomly select one wooden texture to create train exemplars and hold out the other two for testing (described as held-out styles in the figures).

Hyperparameter selection

As discussed in Appendix E.1.3, for a particular concept-style pair, we grid over different hyperparameters pertaining to the rewrite (via editing or fine-tuning)—in particular the layer that is modified, as well as training parameters such as the learning rate. For our evaluation, we then choose a single set of hyperparameters (per concept-style pair). At a high level, our objective is to find hyperparameters that improve model performance on transformed examples, while also ensuring that the test accuracy of the model does not drop below a certain threshold. To this end, we create a validation set per concept-style pair with 30% of the examples containing this concept (and transformed using the same style as the train exemplars). We then use the performance on that subset (6.3) to choose the best set of hyperparameters. If all of the hyperparameters considered cause accuracy to drop below the specified threshold, we choose to not perform the edit at all. We then report the performance of the method on the test set (the other 70% of samples containing this concept).

E.1.5 Evaluation: Real-world data collection

In Section 6.3.3 we study two real-world applications of our model rewriting methodology. Below, we outline the data-collection process for each case.

Vehicles on snow. We manually chose a subset of Imagenet classes that frequently contain "roads", identified using our prediction-rule discovery pipeline in Section 3.2. In particular, we focus on the classes: "racing car", "army tank", "fire truck", "car wheel", "traffic light", "school bus", and "motor scooter". For each of these classes, we searched Flickr[4] using the query "<class name> on snow" and manually selected the images that clearly depicted the class and actually contained snowy roads. We were able to collect around 20 pictures for each class with the exception of "traffic light" where we only found 9.

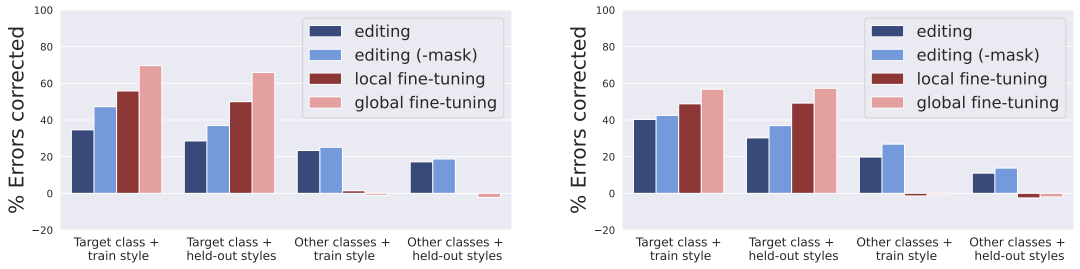
Typographic attacks. We picked six household objects corresponding to ImageNet classes, namely: "teapot", "mug", "flower pot", "toilet tissue", "vase", and "wine bottle". We used a smartphone camera to photograph each of these objects against a plain background. Then, we repeated this process but after affixing a piece of paper with the text "iPod" handwritten on it, as well as when affixing a blank piece of paper—see Figure 6-11.

E.2 Additional Experiments

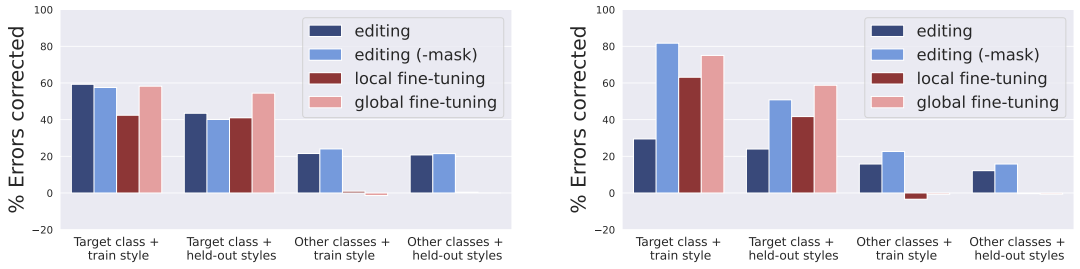
Editing synthetic concept-level transformations

In Figures E-1- E-4, we compare the generalization performance of editing and fine-tuning (and their variants)—for different datasets (ImageNet and Places), architectures (VGG16 and ResNets) and number of exemplars (3 and 10). In performing these evaluations, we only consider hyperparameters (for each concept-style pair) that do not drop the overall (test set) accuracy of the model by over 0.25%. The complete accuracy-performance trade-offs of editing and fine-tuning (and their variants) are illustrated in Appendix Figures E-5-E-7.

[4]`https://www.flickr.com/`

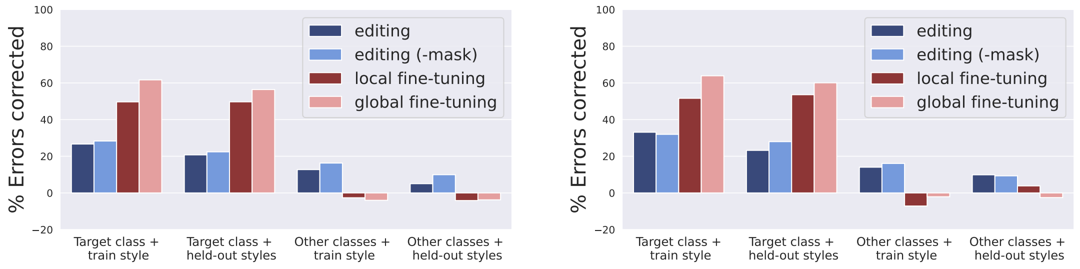


(a) 3 training exemplars

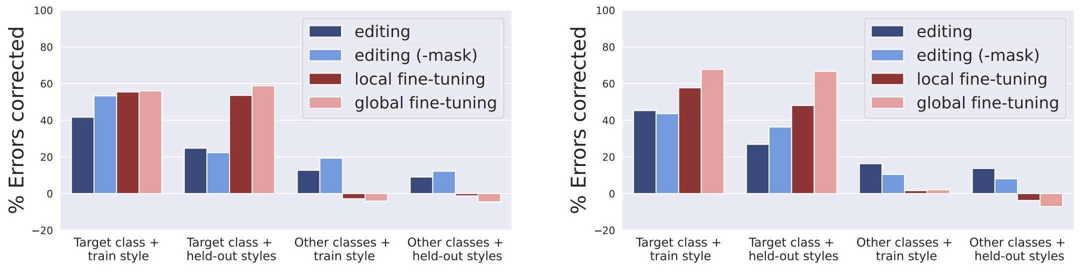


(b) 10 training exemplars

Figure E-1: Editing vs. fine-tuning: average number of misclassifications corrected by the method when applied to an ImageNet-trained VGG-16 classifier. Here, the average is computed over different concept-transformation pairs—with concepts derived from instance segmentation modules trained on MS-COCO (*left*) and LVIS (*right*); and transformations described in Appendix E.1.3. For both editing and fine-tuning, the overall drop in model accuracy is less than 0.25%.

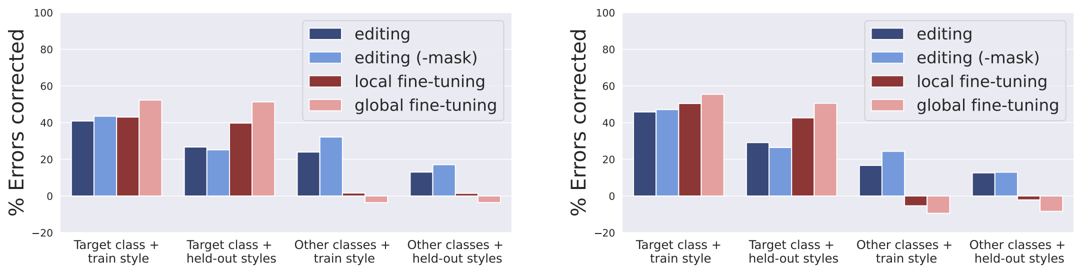


(a) 3 training exemplars

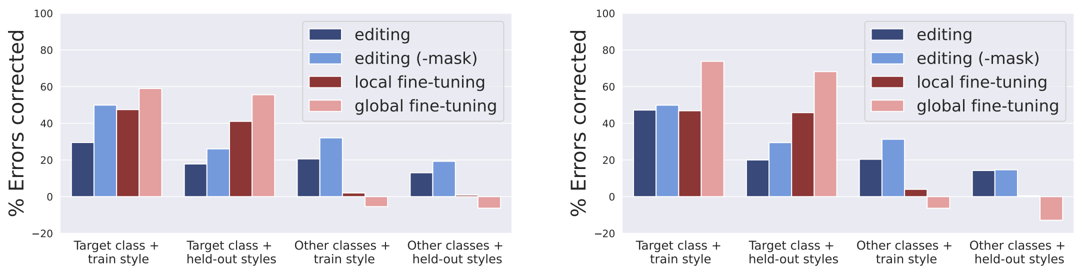


(b) 10 training exemplars

Figure E-2: Appendix Fig. E-1 for an ImageNet-trained ResNet-50 classifier.

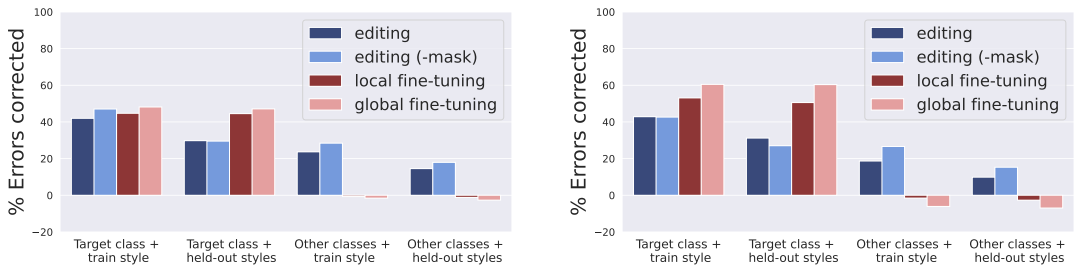


(a) 3 training exemplars

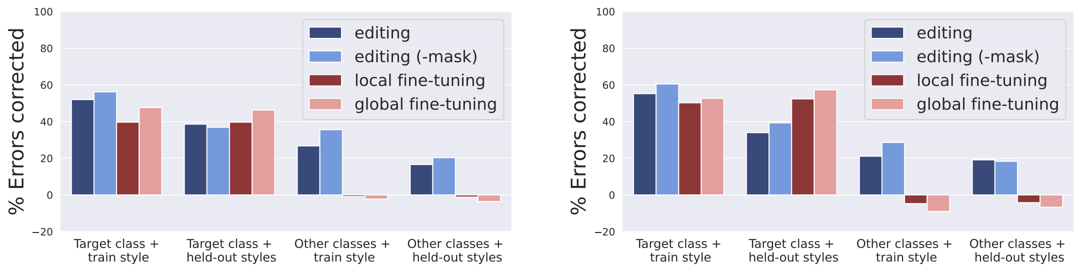


(b) 10 training exemplars

Figure E-3: Appendix Fig. E-1 for a Places365-trained VGG-16 classifier.

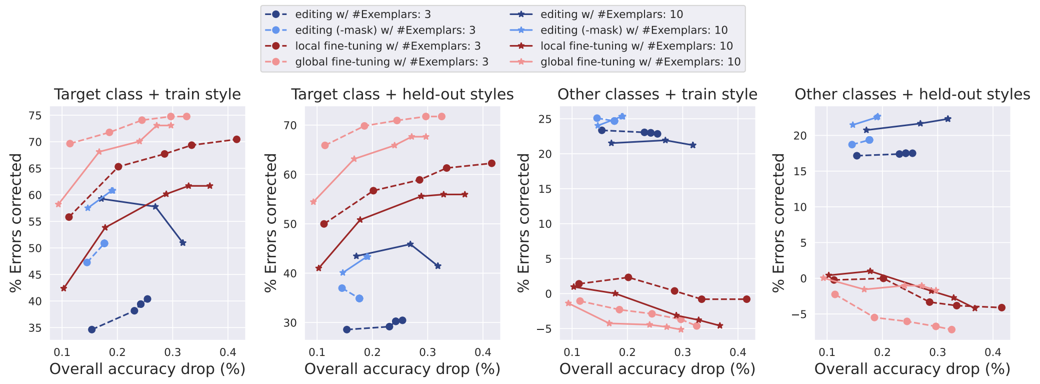


(a) 3 training exemplars

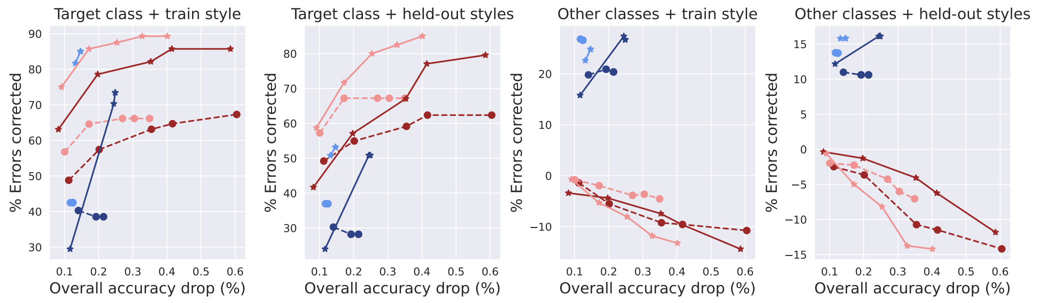


(b) 10 training exemplars

Figure E-4: Appendix Fig. E-1 for a Places365-trained ResNet-18 classifier.

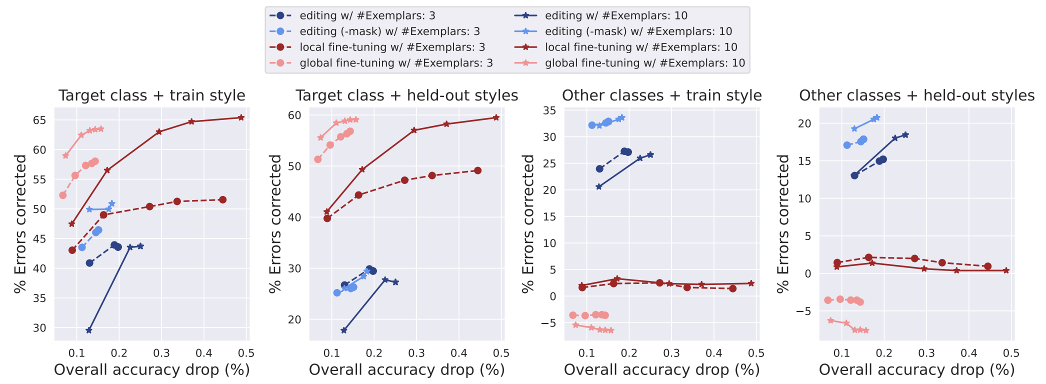


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

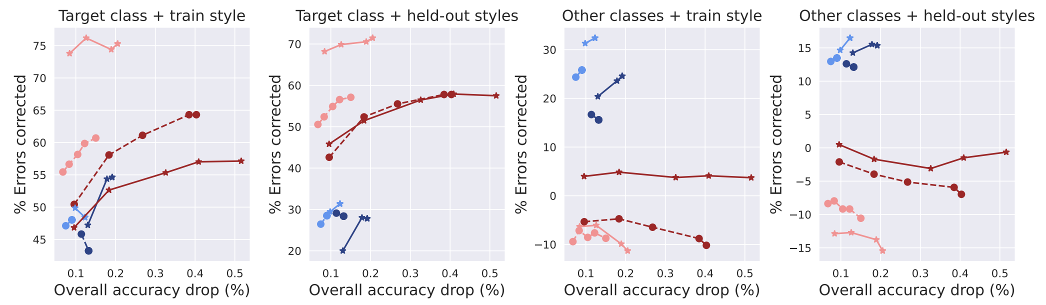


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E-5: Performance vs. drop in overall test set accuracy: Here, we visualize average number of misclassifications corrected by editing and fine-tuning when applied to an ImageNet-trained VGG16 classifier—where the average is computed over different concept-transformation pairs.

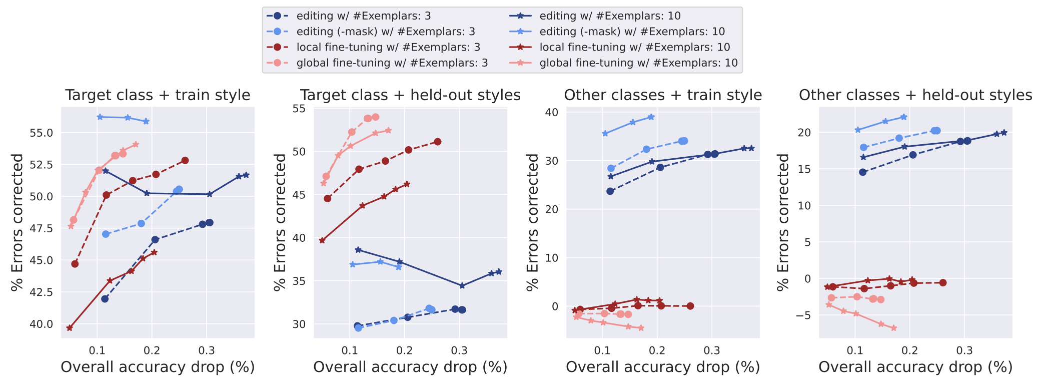


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

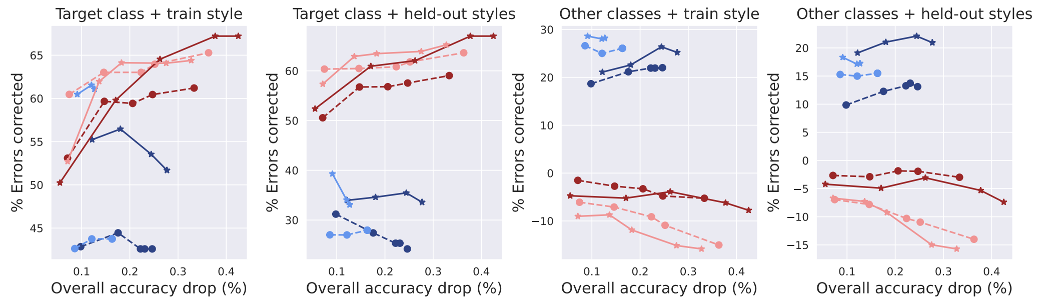


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E-6: Appendix Fig. E-5 for an Places365-trained VGG16 classifier.

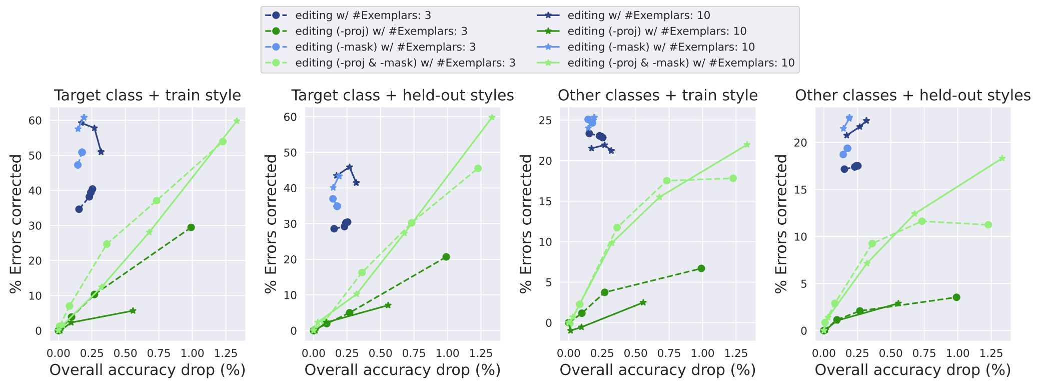


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

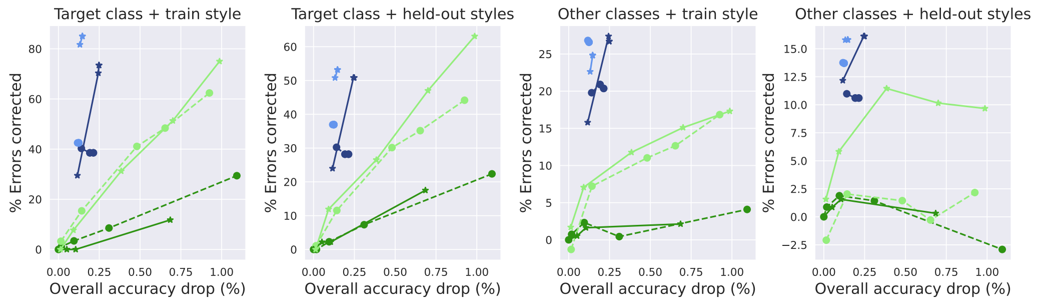


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E-7: Appendix Fig. E-5 for an Places365-trained ResNet-18 classifier.

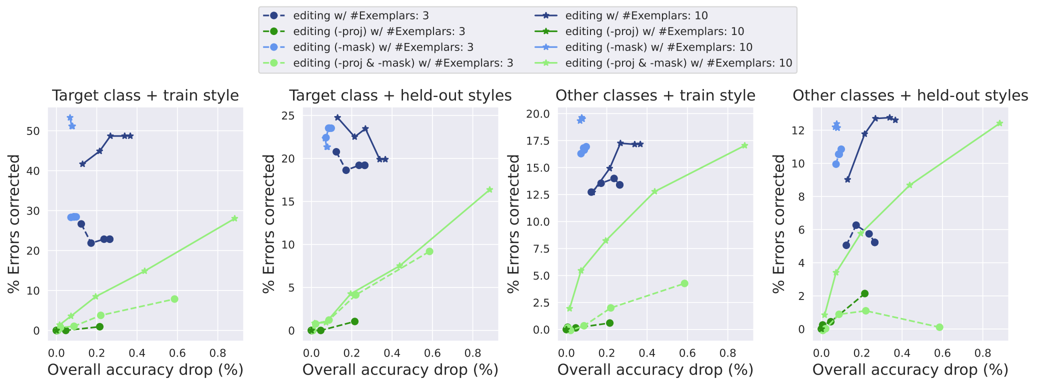


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

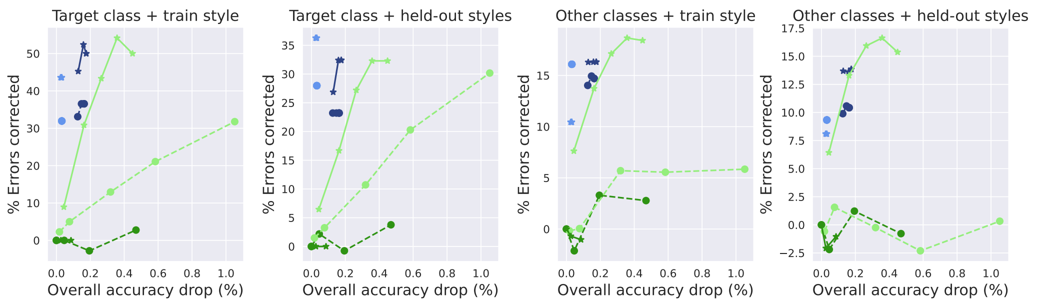


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E-8: Performance vs. drop in overall test set accuracy: Here, we visualize average number of misclassifications corrected by editing variants—based on whether or not we use a mask and perform a rank-one update—when applied to an ImageNet-trained VGG16 classifier.

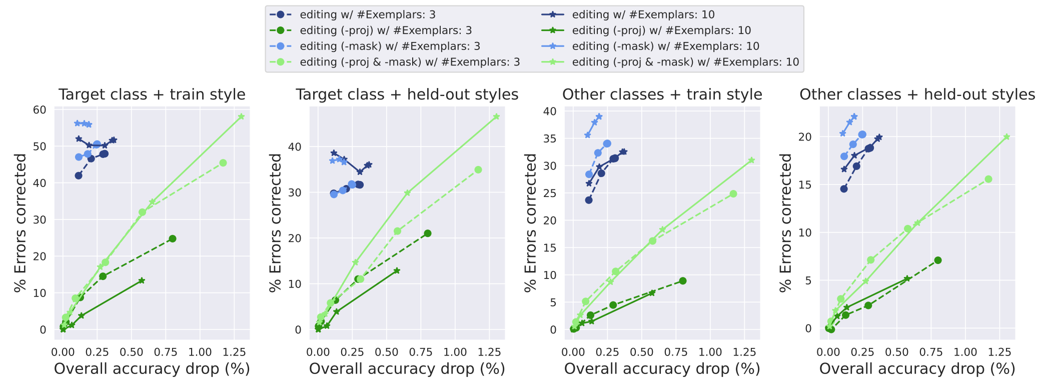


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

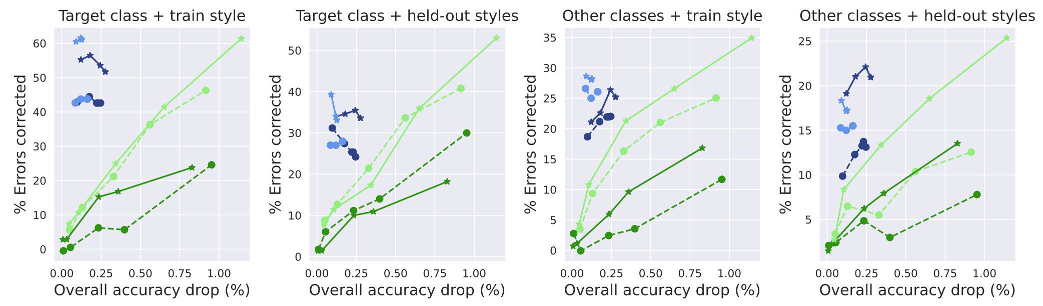


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E-9: Appendix Fig. E-8 for an ImageNet-trained ResNet-50 classifier.



(a) Concepts derived from an instance segmentation model trained on MS-COCO.



(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E-10: Appendix Fig. E-8 for an Places365-trained ResNet-18 classifier.