Learning Through the Lens of Robustness

by

Dimitris Tsipras

Diploma, ECE, National Technical University of Athens (2014)
MSc, CS, 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

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

Learning Through the Lens of Robustness

by

Dimitris Tsipras

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

Despite their impressive performance on large-scale benchmarks, machine learning systems turn out to be quite brittle outside of the exact setting in which they were developed.

How can we build ML models that are robust and reliable enough for real-world deployment?

To answer this question, we first focus on training models that are robust to small, worst-case perturbations of their input. Specifically, we consider the framework of robust optimization and study how these tools can be leveraged in the context of modern ML models. As it turns out, this approach leads us to the first deep learning models that are robust to a wide range of (small) perturbations on realistic datasets.

Next, we explore how such a paradigm of *adversarially robust learning* differs from the standard learning setting. As we will see, robust learning may require training a model that relies on a fundamentally different set of input features. In fact, this requirement can give rise to a trade-off between robustness and accuracy. At the same time, the features that robust models rely on turn out to be more aligned with human perception and, in turn, make these models also useful outside the context of reliability.

Finally, we move beyond the worst-case perturbation setting and investigate other robustness challenges in deploying models in the wild. On one hand, we develop general methodologies for creating benchmarks that gauge model robustness along a variety of axes, such as subpopulation shift and concept transformations. On the other hand, we explore ways to improve the reliability of our models during deployment. To this end, we study how we can bias the features that a model learns towards features that generalize to new environments. Moreover, we develop a methodology that allows us to directly rewrite the prediction rules of a model with virtually no additional data collection.

Thesis Supervisor: Aleksander Mądry

Title: Cadence Design Systems Professor of Computing

Acknowledgements

This thesis would not have been possible without the help and support of many. Unfortunately, I now realize that a short acknowledgements section cannot hope to give them justice, but I'll give it a shot anyway.

First and foremost, I want to thank my advisor, Aleksander. He supported me throughout these years, always providing thoughtful and caring advice on research, work, and life. His endless energy and attention to detail has been one of my main sources of motivation during my PhD. And we actually had fun.

I also want to thank all my collaborators, Andrew, Logan, Shibani, Brandon, Michael, Alex M., Alex T., Adrian, Mahi, David, Saachi, Jacob, Ludwig, Kunal, and Antonio. I learned a lot from each and every one of them. Moreover, I want to single out Andrew, Logan, and Shibani: our lengthy discussions shaped the bulk of my ideas and research agenda through my PhD.

I am also grateful for finding a research home at MIT. Our extended lab was an amazing environment to learn, discuss, and have fun. For that I want to thank everyone involved, Andrew, Logan, Shibani, Brandon, Alex M., Kyriakos, Saachi, Guillaume, Sam, Hadi, Kai, Eric, Jerry, Samanta, Natalia, Kamila, and Debbie. I also want to single out Debbie, for making everything so much easier. Finally, I want to thank Piotr, for being a supportive mentor throughout these years, and Rebecca, for all her help.

I want to also thank Zico and Costis, for serving in my thesis defense committee and their advice throughout, as well as Dimitris Fotakis and Stathis Zachos for believing in me back in the day.

This period of my life would not have been complete without all my friends. This includes my friends from high-school and undergrad, for being there from the start, Adam, Panagiotis, Stratos, Themis, Antones, Kostas, Ilias, and Billy; the amazing family in Boston for making this city feel home, Manolis, Katerina, Konstantinos, Christina, Chara, Lydia P., Terra, Lydia Z., Marinos, Dj Z, Themis, Sophie, Andreas, Andrew, Logan, Ilias, Konstantina M., Konstantina B., Christos, Vasso, Kyriakos, Eva; and the Insti lads, Ishita, Ian (courtesy appointment), Rushina, Gangrade, Priyank, Yamini, and Prithika for all the fun trips.

Finally, I will always be grateful to my parents, Iraklis and Eftychia for their wholehearted support and for raising me to be a person that I can be proud of. Last, but not least, I want to thank Shibs, for making my life better in every way.

Contents

List of Figures and Tables

Introduction

Machine learning (ML) is having a profound impact on our society and lives. Indeed, there is by now a plethora of systems that use ML to streamline a wide range of tasks such as spam detection [Sah+98], image recognition [Kri09; KSH12], language translation [Wu+16], and speech recognition [GMH13]. These developments paint a promising picture. In particular, they give us hope that we may be able to reliably automate decisions in contexts where the cost of mistakes is high—e.g., autonomous vehicles or medical practice.

But, are we there yet? Unfortunately, while the performance of existing ML models is certainly impressive, it also quite brittle. Perhaps the most jarring demonstration of such brittleness is the phenomenon of adversarial examples [Big+13; Sze+14]: state-of-the-art image classifier can be completely fooled by imperceptible input perturbations (cf. Figure 1). In fact, such adversarial examples can be constructed in the physical world. That is, one can 3D-print a turtle that is classified as a rifle [Ath+18], or create innocuous stickers that cause a banana to be recognized as a "toaster" [Bro+18]. Moreover, these failures are not limited to image classification. One can create voice commands that are unintelligible to humans yet home devices can pick up [Car+16] or add irrelevant text to a passage to mislead question answering models [JL17].

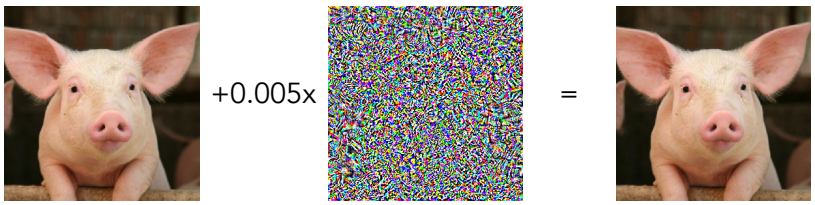


Figure 1: Adding an imperceptible perturbations to a natural image of a "pig" can cause an otherwise highly-accurate classifier to misclassify it into an "airliner".

While these are only a few of the ways in which models can fail outside of their comfort zone, such failures raise questions about how reliable we can expect these models to be in the real world. Crucially, these failures point towards a more fundamental shortcoming of

these models: they may not be solving the underlying problem the way we expect them to. That is, if the model in Figure 1 was really able to recognize the physical object in the image, then a small perturbation should not cause it to classify a pig as an "airplane". Indeed, a key theme throughout this thesis is that robustness is not simply a desirable property of our models but is rather inherently connected to the goal of learning itself.

Motivated by this state of affairs, the overarching goal of this thesis is to lay the foundations for building ML systems that can cope with the real world. Specifically, it revolves around two main thrusts.

First, we will aim to understand the fundamentals of robustness and how the goal of being robust pertains to learning itself. To do so, we will consider small, worst-case perturbations and address the following questions. How can we build models that are robust to such perturbations? Why are models trained in the standard way not robust to them? Is robust learning fundamentally different to standard learning?

In the second thrust, we will move beyond worst-case perturbations and focus on robustness challenges that we expect models to face when deployed in the real world. That is, we ask: how can we simulate such challenges in a precise yet tractable manner? How can we prepare models to face these challenges? Can we develop tools to easily fix our models when they fail?

In the remainder of this introduction, we provide an overview of each of these thrusts, outlining our key ideas and results, and mapping them to the corresponding thesis parts.

Part I: Worst-case perturbations

The existence of adversarial examples represents a profound obstacle to the deployment of ML models in the real world. After all, it shows that the predictions of that model cannot be trusted outside of benign environments. Consequently, robustness to worst-case perturbations has attracted significant research attention as a security objective. That is, if adversarial examples are viewed as attacks, how can we design defenses that ensure models perform as intended?

Building robust models

Previous approaches to preventing adversarial examples focused on mitigating specific types of attacks. For instance, typical adversarial examples are crafted using first-order methods [Sze+14; GSS15] which can be impeded by rendering the optimization landscape of a model hard to navigate—e.g., via distillation [Pap+16] or introducing random-

ness [Dhi+18]. However, the fact that a particular type of attacks is ineffective does not mean that the model is robust in general. In order for a model to be robust in a meaningful way, there needs to be a well-defined *threat model*—a concrete set of perturbations—to which that model is robust to. Indeed, it has been found that essentially *none* of these methods lead to models that are actually robust—it is still possible to arbitrarily fool these models using a slightly modified attack [CW17b; CW17a; He+17; ACW18].

So how can we build models that are actually robust? The first major contribution of this thesis is developing a conceptual framework that allows us to train models that are robust to *all* perturbations within a well-defined threat model.

Formulating the robustness objective. Our starting point will be to formulate a concrete robustness objective that we want our models to satisfy. To this end, recall that models are typically trained via empirical risk minimization (ERM), that is finding the model parameters θ that minimize some loss function $\mathcal{L}$ on the average-case input-label pair (x, y) from some distribution $\mathcal{D}$:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \mathcal{L}(\theta; x, y).$$

However, a small loss value is vacuous when it comes to worst-case guarantees. Instead, if an adversary is able to perturb each input x by a perturbation within a threat model Δ, then the relevant loss function on that input becomes

$$\max_{\delta\in\Delta} \mathcal{L}(\theta; x + \delta, y),$$

as we need to account for the worst-case perturbation of that input. Thus, the key realization here is that we need to directly incorporate robustness into the ERM framework. To do so, we need to focus on the following min-max optimization problem

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \max_{\delta\in\Delta} \mathcal{L}(\theta; x + \delta, y).$$

Indeed, being able to obtain a good solution to this problem would imply that we have a robust model: the loss remains low even under worst-case perturbations within Δ.

Solving the min-max problem. At first glance, this problem appears challenging: we are not minimizing a differentiable function (which is the case in standard training), but rather the maximum of a function over a set of perturbations. In order to bypass this obstacle and train a robust model, we will leverage a theorem for robust optimization, namely Danskin's theorem [Dan67]. As we will describe in more detail in Section 1.2, according

to this theorem, training robust models boils down to training the model on worst-case perturbations of each inputs. This process is also know as adversarial training [GSS15] since it involves effectively training with an adversary in the loop.

Unfortunately, finding worst-case perturbations for complex ML models is, in general, intractable. Nevertheless, our exploration of the optimization landscape of these models in Section 1.3 reveals that approximately worst-case perturbations are actually quite easy to compute. We can thus incorporate such worst-case perturbations into training without a prohibitive computational overhead.

Overall, this process allows us to train the first models that achieve significant robustness against all perturbations within the threat model specified. Indeed, even when evaluating these models against several distinct attacks, including attacks stronger than those used during training, the robustness of these models is not significantly reduced, cf. Section 1.4.

Fundamentals of worst-case robustness

While our exploration so far did lead us to models that are more robust, there is still a lot that we do not understand about adversarial examples as a phenomenon. First and foremost, why are highly-accurate models so brittle to begin with? After all, we now know that robust models for these tasks do exist (cf. Chapter 1). Moreover, why do adversarial examples constructed for one model tend to fool other independent models too (a phenomenon known as transferability [Sze+14; PMG16])? Finally, as it turns out, the robust models that we train in Chapter 1 are less accurate than their standard counterparts. If robustness is a property that we think of as beneficial, why would there be a drop in the model's performance?

The robust features model. The second major contribution of this thesis is the development of a conceptual model through which we can reason about robustness, in general, and these phenomena, in particular. At a high level, our model is based on the notion of the so-called robust and non-robust features. These notions are motivated by the construction of a natural classification task where input features are independent and correlated with the task label at varying degrees (see Section 2.1). Specifically, some of these features will be *non-robust*: while predictive on the average input, they can be manipulated by small perturbation to be predictive of the *wrong* label.

This setting exemplifies a fundamental dichotomy between standard and robust learning. When learning a standard classifier—i.e., a classifier that aims to maximize accuracy—

relying on non-robust features is beneficial, after all they are predictive of the correct label. However, since a small perturbation can render these features misleading, a robust classifier *cannot* depend on them at all.

This dichotomy provides an explanation for many of the aforementioned empirical phenomena that we observe around adversarial examples. First, it hints at their origin: models pick up predictive yet non-robust features which can be easily manipulated via small perturbations—hence their vulnerability to adversarial examples. Moreover, the fact that adversarial examples transfer between distinct model can be directly attributed to the fact that different models are likely to pick up the same non-robust features. Finally, this dichotomy provides an explanation as to why robust models are less accurate than their standard counterparts: they are forced to ignore certain predictive features of the input.

Is this framework predictive? The conceptual framework discussed so far is consistent with existing empirical phenomena related to robustness. But is there a way to test whether it actually reflects reality?

Unfortunately, explicitly manipulating robust and non-robust features of the input is challenging—after all, we still don't have a good way of capturing the individual, high-level features that deep networks rely on. Nevertheless, as we will see in Section 2.2, we can *implicitly* disentangle robust and non-robust features.

Specifically, we will design two experiments that allow us to further scrutinize our conceptual framework. First, in Section 2.2.1, we demonstrate that by training a *standard* model solely on the *robust* features of a dataset, the resulting model ends up being robust to a significant degree. This indicates that the presence of non-robust features is to some extent *necessary* for certain adversarial examples to exist. Second, in Section 2.2.2, we show that by training standard models solely on the non-robust features of a dataset we can obtain non-trivial generalization. This corroborates the hypothesis that adversarial examples arise from signals in the input that are *predictive*, since non-robust features are *sufficient* for generalization.

Overall, these findings demonstrate that real-world datasets do indeed contain predictive non-robust features that our models are sensitive to. This conclusion has implications on model behavior even beyond concerns about security against adversaries. It indicates that ML models can make their predictions quite differently from the way that we, as humans, do, by relying on patterns that we cannot even perceive. Thus, one might wonder if we can ever expect to actually reason about the behavior of such models and trust them in high-stakes applications.

What do robust features look like? Our exploration so far indicates that robust models rely on different features of the input than standard models. Can we visualize and qualitatively examine these robust features?

Indeed, it turns out that even the most simple and natural visualization method sheds light onto these features. That is, constructing large, worst-case perturbations for robust models leads to inputs that contain human-recognizable features of the target class, see Figure 2. This phenomenon indicates that robust models are actually biased towards features that align better with how us, humans, perceive images. Going deeper, in Chapter 2.3 we explore this phenomenon of perceptual alignment in two ways.

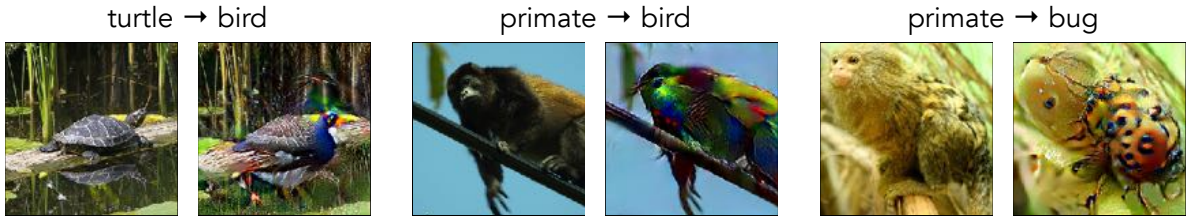


Figure 2: Crafting a large adversarial perturbation for a robust model leads to images that humans recognize as the class that the model predicts.

First, we focus on the latent representations learned by robust models. We find that, in stark contrast with standard models, individual components of these representations can be *directly* visualized to produce human-understandable concepts. That is, in Section 2.3.1 we are able to identify concepts such as "stripes" which are also present in images that strongly activate the corresponding representation component. Moreover, we are able to approximate reconstruct the input of a robust model based solely on the latent representation of that input. This is again in contrast to standard models—where attempting to reconstruct inputs directly results in unintelligible images—and further supports the hypothesis that robust representations capture more human-semantic features of the input.

Second, we further explore that phenomenon that, for robust models, *class maximization*—perturbing an input to maximize the score of a certain class—introduces salient features of the target class. Specifically, in Section 2.3.2, we find that a significant number of traditionally challenging image synthesis tasks such as image generation, image-to-image translation, and image in-painting can be approximately performed through this simple primitive. In fact, all of these tasks can be simultaneously be performed using only a *single* robust classifier without any task-specific training or regularization.

Overall, these findings establish robustness as a property that is desirable even beyond reliability or security concerns and enables a range of new modes of model interactions.

Part II: Real-world robustness

In the first part of this thesis, we focused on model robustness within a concrete, well-defined threat model: small, worst-case input perturbations. This allowed us to rigorously study different facets of robust learning and identify fundamental phenomena such as a potential robustness-accuracy trade-off. However, the perturbations that models will face during deployment are not always small, nor are they always worst-case. Thus, our goal in the second part of the thesis is to expand our study to real-world robustness challenges. Specifically, we will structure our approach around three core directions: creating benchmarks, learning more reliable models, and fixing models during deployment.

Towards capturing real-world deployment

So, what are the robustness challenges that models will face during deployment? The overarching issue here is that, while we train our models on a curated dataset that represents a snapshot of the task we want to solve, these models will be deployed in environments that are likely to be different. For instance, in the context of images, models might encounter objects that are photographed in different poses and against different backgrounds [Pon+06; TE11; Alc+19; Xia+20], under different weather conditions [BVP18; HD19], or using a different camera [Sae+10]. Moreover, even if models were robust to such transformations of a single object, the goal is for them to be able to recognize a *concept* (e.g., "dog") which might actually manifest differently across environments or evolve over time.

It is thus clear that we cannot expect to develop a single benchmark or methodology that will capture all relevant facets of robustness. Instead, we need to decompose the problem into a series of concrete robustness subproblems that we can study separately. In Chapter 3, we take a step in this direction and develop general methodologies for capturing two families of challenges that models are likely to encounter during deployment: subpopulation shift and concept transformations.

Subpopulation shift. Can our models recognize poodles as dogs if, during training, they have only seen dalmatians? This question exemplifies a major challenge in model development: we cannot expect our training set to be perfectly representative of the real world. Indeed, when collecting data for a specific concept of interest (say "dog"), we will inevitable miss some of its subpopulations (say "poodle"). Thus, the ability of the model to generalize is tightly connected to its ability to generalize to such *unseen subpopulations*.

So, how can we create benchmarks for studying robustness to such subpopulation

shifts? At first glance, this appears to require decomposing existing datasets into distinct subpopulations. However, such a process would require a significant data annotation effort which, apart from being quite costly, can also be error-prone, potentially introducing additional human biases into the dataset.

In order to avoid this obstacle, in Section 3.1 we develop a scalable methodology for constructing subpopulation shift benchmarks. The key idea is to group existing, semantically similar classes into super-classes which will then form the subpopulations of interest. By applying this methodology, which we term BREEDS, to the ImageNet dataset [Den+09; Rus+15], we create a suite of benchmarks that comprise subpopulation shifts of varying granularity. We find that while the resulting shifts do indeed pose a significant challenge for existing models, more accurate models are still more robust to these shifts.

Concept transformations. A model can learn to recognize a concept in a variety of ways. For instance, it can recognize the class "car" by associating it with a variety of signals, such as road, car body, or wheels. However, the way these signals manifest across different environments can be quite different. For instance, the model could encounter vehicles on snowy roads or vehicles with old, rusty wheels. How well will the model generalize when such individual input components change?

In order to measure generalization along this axis, in Section 3.2, we develop a pipeline that evaluates a model on input counterfactuals. That is, we study how the prediction of the model on a specific input would change if a component of that input is transformed. For instance, can the model recognize a "poodle" when it is standing on wood instead of grass, or can the model recognize a "bride" when the groom is wearing a colorful jacket? We find that in many of these cases, models are quite sensitive to these transformations, indicating that they indeed often rely on context in an undesirable way.

Improving robustness: Finding the right features

We will now turn our attention to preparing models to tackle robustness challenges such as those described above. To this end, recall that a key takeaway from our discussion on robust and non-robust features was that the model's robustness is inherently tied to the features it uses. While there are many features that are useful for performing the desired task during model training, the way that these features generalize in the real world might be quite different. For instance, both "floppy ears" and "grass" might be useful signals for recognizing beagles—after all, dogs are often photographed outdoors. However, out of

these two features, only the ear shape remains useful when recognizing beagles indoors.

From this perspective, our major focus in Chapter 4 is on biasing the features that the model relies on in order to improve its robustness to a variety of conditions. We explore this approach via two thrusts.

Robustness to synthetic transformations. As we described earlier, training a model to be robust to small, worst-case perturbations can significantly affect the features that it relies on. However, the same principle applies to any method that provides robustness to a set of transformations—e.g., data augmentation or stylized training [Gei+19]. That is, by training a model to be robust to a family of transformations, we can implicitly prevent this model from relying on any feature that is not invariant to these transformations.

Therefore, in Section 4.1 we evaluate how robustness to a range of synthetic transformations affects the ability of models to handle subpopulations shifts (simulated via the BREEDS methodology described above). We find that, indeed, robustness to synthetic transformations has an effect on how well these models generalize. Moreover, this improvement is not always explained by models being more accurate in the absence of subpopulation shift. This indicates that models are actually learning to rely on different sets of features which makes them generalize differently across subpopulations.

Leveraging unlabeled data via feature prior diversity. A promising approach to exposing a model to a more diverse set of inputs is to collect a large amount of additional, unlabeled inputs and incorporate them into training. The canonical approach for doing so is self-learning: train a model on existing labeled data, use it to obtain pseudo-labels for the unlabeled data, and then leverage these for further training. However, this approach suffers from an inherent drawback: models tend to reinforce suboptimal prediction rules learned from the original (labeled) data.

So, how can we bias models away from features that are unlikely to be predictive in the real world? Our key observation in Section 4.2 is that we can treat models which rely on different input features as distinct perspectives on the data. That is, by training multiple models, each with a different feature prior, we can combine them to distill prediction rules that are supported by multiple views of the data. To this end, we employ the classic framework of *co-training* [BM98] where the predictions of each model are added to a common pool of pseudo-labeled examples that are used to further train all models.

We apply this methodology to two setting where the training data is unreliable: one where the labeled dataset is too small and one where it contains a spurious correlation, i.e., a correlation that does not hold on the test set. We find that, in both cases, co-training

using multiple, diverse feature priors results in models that generalize better to the actual task at hand. This demonstrates that, throughout training, these models are able to correct each other's mistakes and steer away from learning misleading correlations.

Adapting models by rewriting their prediction rules

Despite our best efforts to build models that are robust, they will eventually encounter conditions in which their performance degrades significantly. Therefore, reliable deployment requires us to develop tools for adapting our models to such conditions.

Consider for instance a scenario where a model has trouble recognizing vehicles on snowy roads. The canonical method for ameliorating this issue involves collecting additional data and using it for further model training. However, collecting a large number of photographs depicting vehicles on snowy roads might be tricky: we need to ensure that we collect a diverse enough range of vehicles in diverse enough conditions.

Therefore, in Section 5.2, we develop a methodology that allows us to *directly* edit the prediction rules of a model *without the need for additional data collection*. Our approach is based on the method of Bau et al. [Bau+20a] for rewriting generative models and in focused around performing a small, targeted modification to the parameters of the model using a handful of example inputs. Conceptually, the goal of our method is to ensure that a specific concept, say "snowy road", is treated by the model in the same way as another concept, say "road". This allows the model to reuse the original predictions rules—which rely on the presence of "road"—in this new setting without the need for significant retraining. As it turns out, this approach is quite effective. In particular, it enables us to modify the model's behavior on a range of real-world scenarios using a *single, synthetically crafted* example.

Outlook: Towards reliable machine learning

While our progress so far is encouraging, we are still far from building models that are truly reliable. However, going forward, we believe that there are promising avenues for research, focused around the following two directions.

Understanding the space of robustness priors. Our findings so far establish robustness as a powerful tool for influencing the behavior of a model. That is, by enforcing different invariances to a model (through adversarial training or data augmentation), we can bias it

towards relying on different features of the input and thus potentially improve its ability to generalize to new environments.

However, this is only a first step towards exploring and harnessing the space of robustness-based priors. What other invariances can we efficiently enforce when training our models? How do these invariances affect different notions of real-world robustness? In fact, the answer to these questions is likely to be domain- or application-specific. Thus, how can we develop the tools to help practitioners identify the right feature priors for a particular task at hand?

A deployment toolkit. Model deployment is not a one-shot process. Models will inevitably make mistakes which will then need to be analyzed by model designers and used to improve the model. How can we create tools to facilitate this process?

On one hand, we need to enable effective model debugging: how can we distill model failures into concrete, actionable insights? Inevitably, such a process needs to involve domain experts. Thus, when designing debugging tools we need to account for the corresponding costs: what are ways to minimize the need for expert knowledge, potentially by effectively leveraging non-expert human annotators?

On the other hand, after pinpointing the problematic model behavior, model designers need to correct it. Our methodology for directly editing prediction rules is only a first step in this direction of implementing precise modifications to the way models predict. What are other effective interfaces through which humans can interact with models? How can we expose the inner workings of a model to a human designer in a way that facilitates model correction?

Thesis organization

Chapter 1 describes our methodology for training models that are robust to small, worst-case perturbations. It is based on joint work with Aleksander Mądry, Alex Makelov, Ludwig Schmidt, and Adrian Vladu [Mad+18].

Chapter 2 explores the fundamentals of robustness through the robust features model. It is based on joint work with Logan Engstrom, Andrew Ilyas, Aleksander Mądry, Shibani Santurkar, Brandon Tran, and Alex Turner [Tsi+19; Ily+19; Eng+19a; San+19].

Chapter 3 focuses on scalable methods for measuring model robustness to deployment conditions. Section 3.1 focuses on subpopulation shift and is based on joint work with Aleksander Mądry and Shibani Santurkar [STM21], while Section 3.2 focuses on concept

transformations and is based on joint work with David Bau, Mahi Elango, Aleksander Mądry, Shibani Santurkar, and Antonio Torralba [San+21].

Chapter 4 studies how different training methodologies can impact the features that a model learns and, in turn, how this can affect the robustness of the model on downstream tasks. Section 4.1 focuses on robustness to synthetic transformations in the context of subpopulation shift and is based on joint work with Aleksander Mądry and Shibani Santurkar [STM21], while Section 4.2 focus on leveraging unlabeled data through diverse feature priors and is based on joint work with Saachi Jain and Aleksander Mądry [JTM21].

Chapter 5 describes our method for adapting a model by directly editing its prediction rules and is based on joint work with David Bau, Mahi Elango, Aleksander Mądry, Shibani Santurkar, and Antonio Torralba [San+21].

Part I

Robustness to Worst-Case Perturbations

Chapter 1

Building robust models

Imperceptible input perturbations can cause models that are otherwise highly accurate to produce arbitrarily wrong predictions [Big+13; Sze+14]. In fact, such *adversarial perturbations* are quite pervasive. There is by now a large body of work demonstrating that they can be computed quite easily in a number of different domains [JL17; Car+16; SCJ19], even without full access to the model [Che+17; Ily+18]. Moreover, these perturbations can transfer between different model architectures [Sze+14; PMG16] and can also manifest in the physical world [KGB16; Ath+18; Evt+18; Bro+18].

The security implications of this phenomenon are clear, one cannot trust the predictions of the model when it is deployed in an adversarial environment. However, this phenomenon hints at an even more significant reliability concern: if models are sensitive to input changes that we, as humans, cannot even perceive, how can we expect them to be robust to all the challenges that they will face when deployed in the real world? Thus, if we do want to eventually deploy ML models reliably, we need to first understand:

Can we train ML models that are robust to small, worst-case perturbations?

This question has drawn significant research attention with multiple method for training robust models being proposed. Unfortunately, to the best of our knowledge, none of these approaches were successful. The key issue is that these methods are relatively ad hoc approaches that aim to prevent an adversary from easily constructing effective adversarial perturbations. However, they were not tied to any specific robustness objective and can thus be bypassed by adaptive methods for constructing adversarial perturbations [CW17b; He+17; CW17a; ACW18].

Here, we will take an alternative approach. That is, we will study the problem of robustness by first formulating a *concrete robustness objective* that we want our models to satisfy. Specifically, we will study a natural min-max formulation that allows us to

use established tools from robust optimization [BEN09]. As we will see, despite the non-convexity and non-concavity of its constituent parts, the underlying optimization problem is tractable. This will allow us to train the first models that are robust to a wide range of small, worst-case perturbations.

Chapter outline:

- First, we will present our conceptual framework for training robust models based on robust optimization in Section 1.1.
- We will then study the landscape of the corresponding optimization problem. Specifically, we will tackle the outer minimization problem in Section 1.2, the inner maximization problem in Section 1.3, and then demonstrate how this methodology can lead us to robust models in Section 1.4.
- Finally, we will dive deeper into the empirical behavior of robust models by exploring: (a) the impact of model capacity on robustness (Section 1.5); (b) black-box transfer attacks using robust models (Section 1.6).
- We will conclude by discussing subsequent work in this area and outline the current state of worst-case robustness 1.7.

1.1 An optimization view on adversarial robustness

Our discussion will revolve around an optimization view of adversarial robustness. This perspective will allows us to formulate a concrete objective that we want our models to satisfy and, in turn, will allow us to develop a principled training methodology.

Recall that the canonical methodology for training ML learning models is empirical risk minimization (ERM). That is, the goal is to find model parameters that minimize the error of the model on a fixed set of input—the training set. Concretely, consider a standard classification task with an underlying data distribution $\mathcal{D}$ over pairs input label pairs (x, y). Then the goal is to compute model parameters θ that perform best on the average input

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \mathcal{L}(x, y, \theta)$$

where the $\mathcal{L}$ function captures how well the model performs on an input-label pair (x, y).

While this methodology typically leads to models that can predict accurately on new, unseen inputs, these models are not necessarily robust. That is, given an example x with

label y it is possible to construct a similar input x^{adv} such that the model incorrectly assigns x^{adv} a label $y' \neq y$ [Big+13; Sze+14].

In order to *reliably* train models that are robust to adversarial perturbations, it is necessary to augment the ERM paradigm appropriately. The first step towards doing so is to specify a perturbation set Δ that precisely defines that set of perturbations that our models should be resistant to. In the setting of image classification, we choose what is perhaps the simplest set of perturbations: an ℓ_∞-ball around each input. It is worth noting that while a small ℓ_∞ distance between images implies that they are perceptually similar, there are many other natural notions of similarity that one can consider [FF15; Xia+18].

Next, we modify the definition of population risk $\mathbb{E}_{\mathcal{D}}[\mathcal{L}]$ by incorporating the above perturbation model. Instead of feeding samples from the distribution $\mathcal{D}$ directly into the loss $\mathcal{L}$, we allow for a worst-case perturbation of the input first. This gives rise to the following saddle point problem, which is our central object of study:

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\Delta} \mathcal{L}(\theta, x+\delta, y) \right] . \tag{1.1}$$

Formulations of this type (and their finite-sample counterparts) have a long history in robust optimization, going back to Wald [Wal45].

This formulation gives us a unifying perspective that encompasses much of prior work on adversarial robustness. Both the inner maximization and the outer minimization problems have a natural interpretation in our context. The inner maximization problem aims to find an adversarial version of a given data point x that achieves a high loss. This is precisely the problem of finding a worst-case perturbation. On the other hand, the goal of the outer minimization problem is to find model parameters so that the *adversarial loss* given by the inner problem is minimized. This is precisely the problem of training a robust classifier using adversarial training techniques.

Crucially, this saddle point problem presents a clear goal that an ideal robust classifier should achieve. In particular, when the parameters θ yield a (nearly) vanishing risk, the corresponding model is perfectly robust to attacks specified by our attack model. In the rest of this chapter, we will investigates the structure of this saddle point problem in the context of deep neural networks. These investigations then lead us to training techniques that produce models with high resistance to a wide range of adversarial attacks.

1.2 Solving the outer problem: Training robust models

We start our investigation by focusing on solving the outer minimization problem: find model parameters that low adversarial loss. Recall that if we find that parameters, then we know that the resulting model is robust to adversarial perturbations, since, by definition, these cannot increase its loss significantly.

The canonical way of training a model is via stochastic gradient descent (SGD)—iterating updating the parameters to improve the loss on a random set of training examples. However, it is not a priori clear how this methodology can be used to solve our saddle point formulation (1.1). The main challenge is that we are not optimizing a fixed function but rather a *maximum* over function values.

Nevertheless, we can bypass this challenge by leveraging a classic theorem in optimization, namely Danskin's theorem [Dan67]. This theorem states that, under a set of assumptions, if we compute the gradient of the $\mathcal{L}$ with respect to θ on a maximizer δ^* then this correspond to a descent direction for the maximum—i.e., the adversarial loss—(see Appendix A.2 for the exact statement of the theorem). There is thus a very intuitive way of solving the outer problem: use SGD to optimize $\mathcal{L}$ with respect to θ, but *replace* each input-label pair (x, y) with a worst case perturbation (x_{adv}, y). This process is also known as *adversarial training* [GSS15] since we are effectively training against an adversary that perturbs inputs.

Note that the exact assumptions of Danskin's theorem do not hold for the setting of standard deep neural networks. That is, the $\mathcal{L}$ function is not continuously differentiable due to ReLUs and max-pooling units and we are only computing approximate maximizers of the inner problem. Still, as our experiments in Section 1.4 demonstrate, the underlying intuition is still valid and allows to train robust models.

Nevertheless, there is a key takeaway from this discussion: the perturbations used for adversarial training should be approximately worst-case. While this is apparent from the statement of the theorem, it also manifests clearly in empirical experiments. Early attempts at training robust models were using crude methods for computing these perturbations, e.g., by simply linearizing the loss around each data point [GSS15]. As a result, while these methods lead to model that were robust to such crudely-constructed perturbations, they could be fooled completely by slightly more sophisticated perturbations [Tra+17].

1.3 Solving the inner problem: Finding good perturbations

We now turn our attention to the inner maximization problem. That is, for a fixed input-label pair (x, y) and fixed model parameters θ, how can we find that worst ℓ_∞-bound perturbations:

$$\max_{\|\delta\|_\infty \leq \varepsilon} \mathcal{L}(\theta, x + \delta, y)$$

where ε is the maximum allowed perturbation magnitude. Recall, that findings such worst-case perturbations is crucial for our ability to use them for training robust models as discussed in the previous section.

Since the structure of the loss function $\mathcal{L}$ will be quite complex for most modern machine learning models, we will attempt to find an approximate solution to this problem via a first-order method. Specifically, to ensure that δ remains within the allowed set Δ, we need to employ project gradient descent (PGD) where we will project the solution to the allowed set at each step. Moreover, we choose to use a version of PGD that might be better suited for ℓ_∞-based problem wherein at each step we move to the further point within an ℓ_∞-box to the direction of the gradient. Overall, our optimization process is iterative where at each step we perform an update of the form

$$x^{t+1} = \Pi_{x+\Delta}\left(x^t + \alpha \operatorname{sgn}(\nabla_x \mathcal{L}(\theta, x, y))\right),$$

for some small step size α.

In the context of adversarial perturbations, this method is known as the "basic iterative method" [KGB17]. In its simplest form, where only a single step of size ε is taken, this corresponds to the "fast gradient sign method" (FGSM) [GSS15] which equivalent to maximized a first-order approximation of the loss around x.

The landscape of adversarial perturbations

So, can PGD find worst-case perturbations reliably? A priori, this is not clear. After all, in the case of deep neural networks, we are maximizing a highly non-concave functions. Thus, while PGD will eventually lead us to some local maximum, there is not guarantee that this will be a good approximation for the global maximum.

To understand this problem in more detail, we investigate the landscape of local maxima found by PGD for a variety of models trained on the MNIST [LeC98] and CIFAR-10 [Kri09] datasets We will study both standard models and through a small glitch in the timeline, the robust models that we will train in Section 1.4. Specifically, we will pick a

few test examples at random and apply PGD from multiple random starting points in an ℓ_∞-box of size ε around each example.

We find that, over multiple random restarts, PGD behaves quite consistently, cf. Figure 1.1. That is, the loss increases fairly consistently across iterations and plateaus at similar values. Moreover, even over a large number of such random restarts (10^5), the values of the local maxima found are well concentrated without visible outliers, cf. Figure 1.2. We find that while PGD finds multiple distinct local maxima, the loss value of these maxima tends to be well-concentrated.

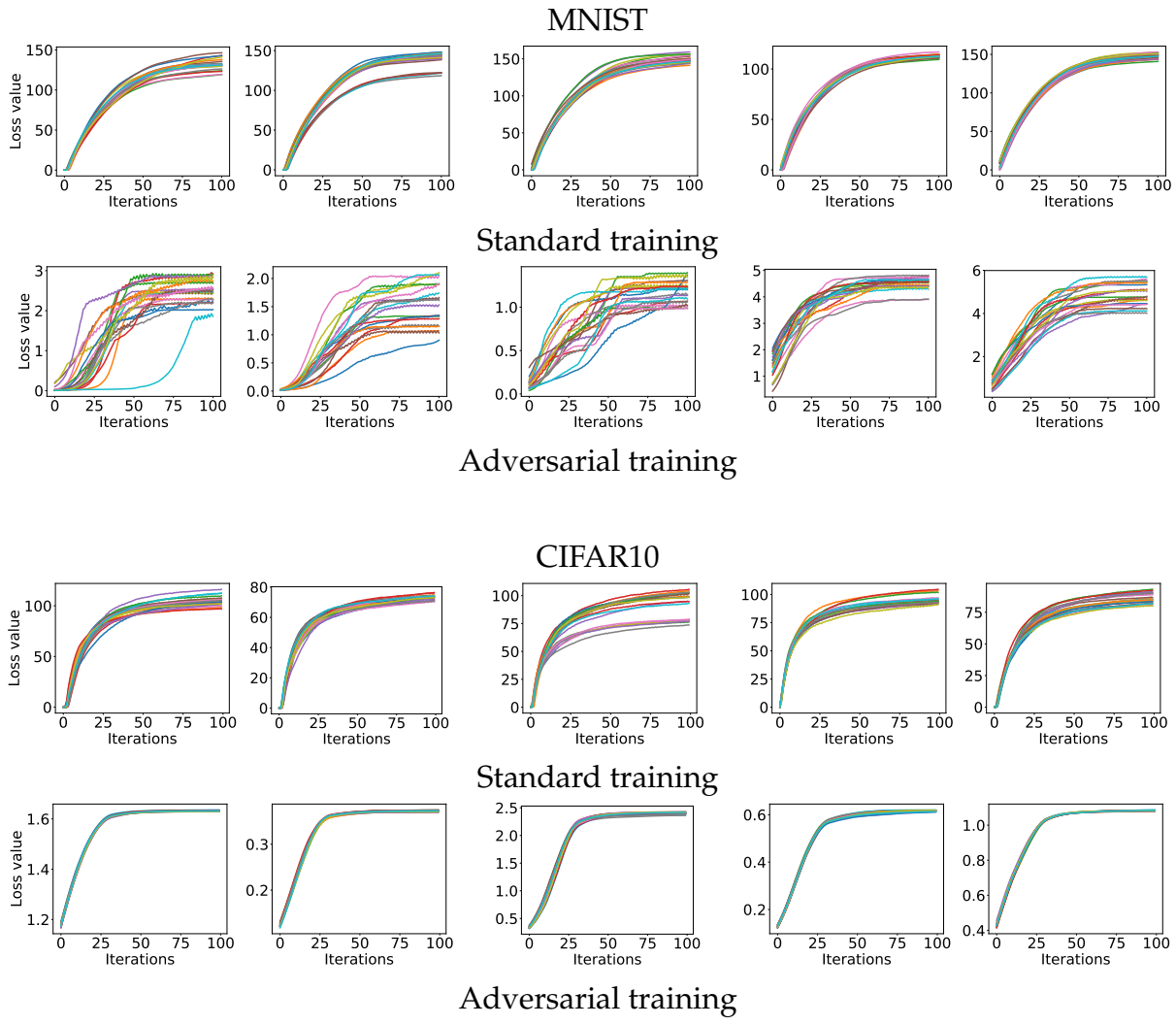


Figure 1.1: Loss value over the course of 20 runs of projected gradient descent (PGD) on random examples from the MNIST and CIFAR-10 dataset. Each run starts at a uniformly random point in the ℓ_∞-ball around the same natural example. Notice how the loss plateaus after a relatively small number of iterations, while the final loss values are fairly clustered.

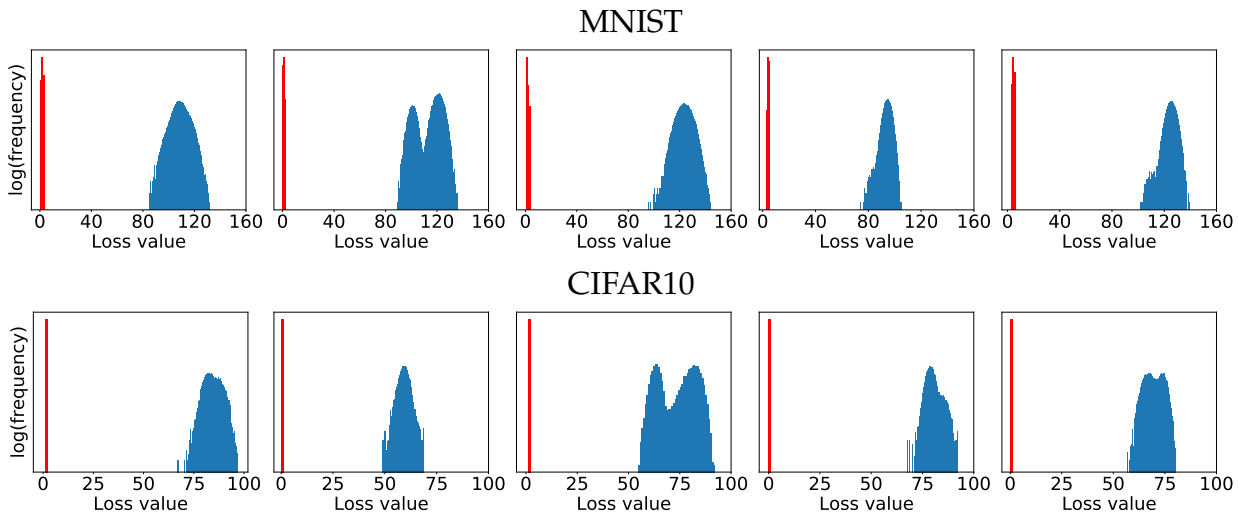


Figure 1.2: Values of the local maxima given by the cross-entropy loss for five examples from the MNIST and CIFAR10 evaluation datasets. For each example, we start projected gradient descent (PGD) from 10^5 uniformly random points in the ℓ_∞-ball around the example and iterate PGD until the loss plateaus. The blue histogram corresponds to the loss on a standard network, while the red histogram corresponds to the adversarially trained counterpart. The loss is significantly smaller for the adversarially trained networks, and the final loss values are very concentrated without any outliers.

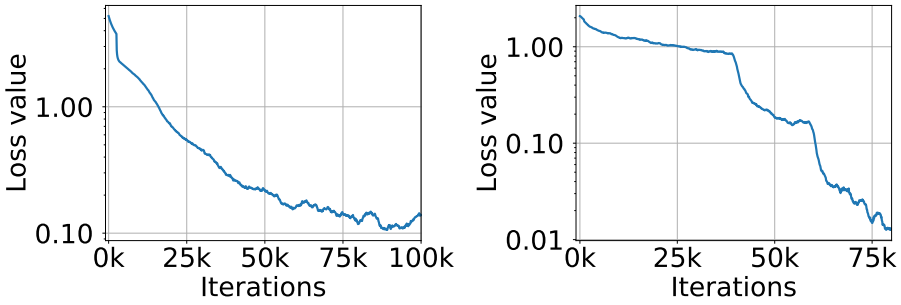
1.4 Adversarially robust deep learning models

We will now apply the methodology described in the last two sections to train models that are robust against small, adversarial perturbations. Specifically, starting from a standard training pipeline for two canonical datasets we will add a worst-case perturbation to each example during training.

Specifically, we will compute these perturbations using a few steps of PGD starting from a random perturbation around each training example. Note that this computation is performed on-the-fly during training with respect to the model parameters at the corresponding training step. Since we are training the model for multiple epochs, there is no benefit from restarting PGD multiple times per batch—a new start will be chosen the next time each example is encountered. See Appendix A.1 for additional details.

We find that over the course of training the adversarial loss decreases consistently, cf. Figure 1.3. This indicates that we are indeed making progress in solving the problem of (1.1). Indeed, the final robust accuracy of these models on the test set is actually significant—93% for MNIST and 50% for CIFAR.

In order to evaluate the robustness of the resulting model, we will compute worst-case



(a) MNIST (b) CIFAR10

Figure 1.3: Cross-entropy loss on adversarial examples during training. The plots show how the adversarial loss on training examples evolves during training the MNIST and CIFAR10 networks against a PGD adversary. The sharp drops in the CIFAR10 plot correspond to decreases in training step size. These plots illustrate that we can consistently reduce the value of the inner problem of the saddle point formulation (1.1), thus producing an increasingly robust classifier.

perturbations using stronger methods that those during training. Specifically, we will consider: (a) PGD with more iterations, including multiple random restarts per examples; (b) PGD based on the Carlini-Wagner (CW) loss [CW17b] thus directly optimizing the difference between correct and incorrect logits; (c) transferring adversarial perturbations from other models (potentially with a different architecture). We present these results in Table 1.4a for MNIST and Table 1.4b for CIFAR10.

We find that none of these perturbations can significantly reduce the performance of the resulting models. This is stark contrast to previous approaches where the robustness of the model was limited to perturbations computed in the same ways as those during training. Instead, our models appear to be robust to *any* perturbation that we can compute within these ℓ_∞-perturbation models.

Third-party evaluation. In order to put the robustness of our models to test, we publicly released them in the form of a challenge[1,2]. Over the four years that these challenges have been public, multiple independent research groups have submitted white-box attacks. Nevertheless, none of these attacks reduced the robust accuracy of our models by a significant amount.

[1] https://github.com/MadryLab/mnist_challenge
[2] https://github.com/MadryLab/cifar10_challenge

Method	Steps	Restarts	Source	Accuracy
Natural	-	-	-	98.8%
FGSM	-	-	A	95.6%
PGD	40	1	A	93.2%
PGD	100	1	A	91.8%
PGD	40	20	A	90.4%
PGD	100	20	A	**89.3%**
Targeted	40	1	A	92.7%
CW	40	1	A	94.0%
CW+	40	1	A	93.9%
FGSM	-	-	A′	96.8%
PGD	40	1	A′	96.0%
PGD	100	20	A′	**95.7%**
CW	40	1	A′	97.0%
CW+	40	1	A′	96.4%
FGSM	-	-	B	**95.4%**
PGD	40	1	B	96.4%
CW+	-	-	B	95.7%

(a) MNIST

Method	Steps	Source	Accuracy
Natural	-	-	87.3%
FGSM	-	A	56.1%
PGD	7	A	50.0%
PGD	20	A	**45.8**%
CW	30	A	46.8%
FGSM	-	A′	67.0%
PGD	7	A′	**64.2%**
CW	30	A′	78.7%
FGSM	-	A_{nat}	85.6%
PGD	7	A_{nat}	86.0%

(b) CIFAR-10

Figure 1.4: Performance of the adversarially trained network against different ℓ_∞-bounded perturbations: (a) MNIST with a bound of 0.3, (b) CIFAR-10 with a bound of 8/255. For each model of attack we show the most effective attack in bold. The source networks considered for the attack are: the network itself (A) (white-box attack), an independtly initialized and trained copy of the network (A′), a copy of the network trained on natural examples (A_{nat}). CW and CW+ denote a PGD attack using the CW loss [CW17b] with confidence parameter (κ) equal to 0 and 50 respectively.

Manually inspecting a robust model. In order to get a better understanding of the internals of a robust model, we manually inspect the resulting MNIST classifier in Appendix A.3. The most striking observation is that the first convolutional layer has collapsed to a few 1×1 convolutions. Combined with the following ReLU unit, this effectively implements a thresholding operation which truncates pixels below or above a certain value. Since in MNIST most pixels are either 0 or 1, this is actually a valid strategy for ignoring small perturbations.

1.5 Network capacity and adversarial robustness

Solving the problem from Equation (1.1) successfully is not sufficient to guarantee robust and accurate classification. We need to also argue that the *value* of the problem (i.e. the final loss we achieve against adversarial examples) is small, thus providing guarantees for the performance of our classifier. In particular, achieving a very small value corresponds to a perfect classifier, which is robust to adversarial inputs.

For a fixed set Δ of possible perturbations, the value of the problem is entirely dependent on the model architecture and training process used. Consequently, the capacity of the model becomes a major factor affecting its overall performance. It is possible that a stronger classifier is necessary for classifying examples robustly (see Figure 1.5 for an illustration).

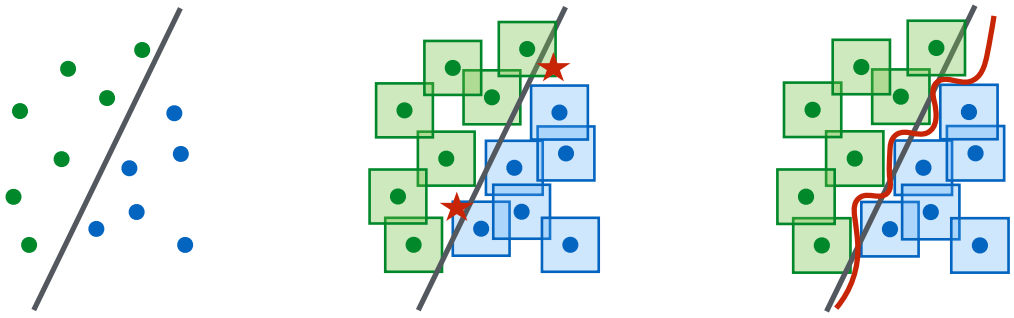


Figure 1.5: A conceptual illustration of standard vs. adversarial decision boundaries. Left: A set of points that can be easily separated with a simple (in this case, linear) decision boundary. Middle: The simple decision boundary does not separate the ℓ_∞-balls (here, squares) around the data points. Hence there are adversarial examples (the red stars) that will be misclassified. Right: Separating the ℓ_∞-balls requires a significantly more complicated decision boundary. The resulting classifier is robust to adversarial examples with bounded ℓ_∞-norm perturbations.

Our experiments verify that capacity is crucial for robustness, as well as for the ability to successfully train against strong adversaries. For the MNIST dataset, we consider a simple convolutional network and study how its behavior changes against different adversaries as we keep doubling the size of network (i.e. double the number of convolutional filters and the size of the fully connected layer). The initial network has a convolutional layer with 2 filters, followed by another convolutional layer with 4 filters, and a fully connected hidden layer with 64 units. Convolutional layers are followed by 2×2 max-pooling layers and adversarial examples are constructed with $\varepsilon = 0.3$. The results are in Figure 1.6.

For the CIFAR10 dataset, we used a ResNet model [He+16]. We performed data augmentation using random crops and flips, as well as per image standarization. To

increase the capacity, we modified the network incorporating wider layers by a factor of 10. This results in a network with 5 residual units with (16, 160, 320, 640) filters each. This network can achieve an accuracy of 95.2% when trained with natural examples. Adversarial examples were constructed with $\varepsilon = 8$. Results on capacity experiments appear in Figure 1.6.

We observe the following phenomena:

Capacity alone helps. We observe that increasing the capacity of the network when training using only natural examples (apart from increasing accuracy on these examples) increases the robustness against one-step perturbations. This effect is greater when considering adversarial examples with smaller ε.

Weak models may fail to learn non-trivial classifiers. In the case of small capacity networks, attempting to train against a strong adversary (PGD) prevents the network from learning anything meaningful. The network converges to always predicting a fixed class, even though it could converge to an accurate classifier through standard training. The small capacity of the network forces the training procedure to sacrifice performance on natural examples in order to provide any kind of robustness against adversarial inputs.

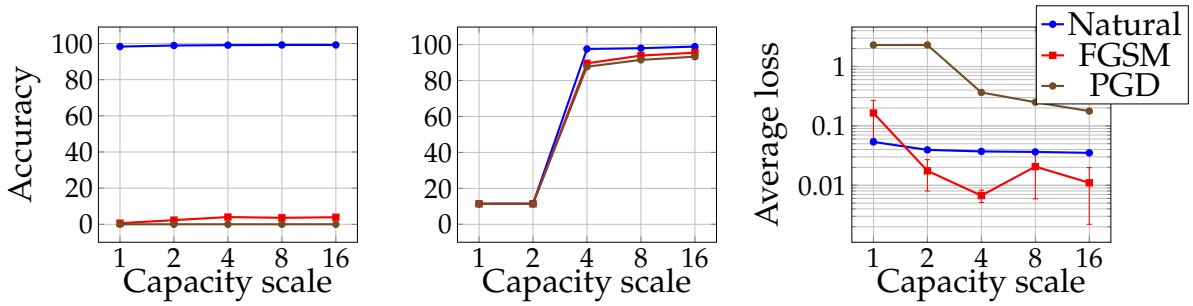
The value of the saddle point problem decreases as we increase the capacity. Fixing an adversary model, and training against it, the value of (1.1) drops as capacity increases, indicating the the model can fit the adversarial examples increasingly well.

1.6 Transferability

We will now turn our attention to transferability—the phenomenon that adversarial perturbations transfer between independently trained models. Specifically, for the CIFAR10 dataset, we investigate the transferability PGD-based perturbations on a standard and wide version of a ResNet, each trained in the standard way (ERM on natural examples) and with PGD adversarial training.

We observe that adversarial perturbations transfer significantly better between models that have been trained in a similar manner—cf. Figure 1.7. That is, transfer attacks from PGD-trained models do not transfer well to standard models and vice-versa. This hints towards the possibility that PGD-trained models have fundamentally different decision boundaries than standard models and are thus sensitive to different perturbations.

MNIST



CIFAR-10

	Simple	Wide
Natural	92.7%	95.2%
FGSM	27.5%	32.7%
PGD	0.8%	3.5%

(a) Standard training

	Simple	Wide
Natural	79.4%	87.3%
FGSM	51.7%	56.1%
PGD	43.7%	45.8%

(b) PGD training

Simple	Wide
0.00357	0.00371
0.0115	0.00557
1.11	0.0218

(c) Training Loss

Figure 1.6: The effect of network capacity on the performance of the network. We trained MNIST and CIFAR10 networks of varying capacity on: (a) natural examples, (b) with FGSM-made adversarial examples, (c) with PGD-made adversarial examples. In the first three plots/tables of each dataset, we show how the standard and adversarial accuracy changes with respect to capacity for each training regime. In the final plot/table, we show the value of the cross-entropy loss on the adversarial examples the networks were trained on. This corresponds to the value of our saddle point formulation (1.1) for different sets of allowed perturbations.

1.7 Subsequent work

Since the publication of this work, there has been significant effort on verifying and improving the robustness of these models.

Formal robustness verification. As we discussed in Section 1.4, despite our best efforts to compute the worst possible perturbations for our models, we still do not have guarantees on the robustness of these models. In order to obtain such guarantees, later work focus on the formal verification of the robustness of PGD-trained models. The earliest such work was that of Carlini et al. [Car+17] who use the Reluplex [Kat+17] framework to verify the robustness of PGD-trained models on a few inputs. Later work developed more efficient methods for model verification [TXT19] including ways of modifying PGD-training to lead to models that are easier to verify [Dvi+18; Xia+19].

A parallel line of work focuses on producing certified lower bounds on the robust

	Source model			
Target model	Standard	PGD-trained	Wide standard	Wide PGD-trained
Standard	6.6%	71.8%	**1.4%**	75.6%
PGD-trained	78.1%	57.7%	77.9%	**65.2%**
Wide standard	**10.9%**	79.1%	0.0%	79.7%
Wide PGD-trained	86.4%	**72.1%**	86.0%	64.2%

Table 1.7: Transferring PGD perturbations between models. We create PGD adversarial examples on CIFAR-10 with $\varepsilon = 8$ for 7 iterations from the evaluation set on the source model, and then evaluate them on an independently initialized target model. The lower the target model accuracy the better these perturbations transfer. The diagonal corresponds to white-box attacks.

accuracy of a model [WK18; RSL18]. While such method do not provide us with an exact robustness guarantee, they are more efficient and can thus scale to larger models.

Improving empirical robustness to ℓ_p-bounded perturbations. There have been a number of methods proposed for improving the empirical robustness of models. These method modify the robust optimization framework by introducing alternative robustness objectives [Zha+19], treating misclassified examples differently [Wan+19], or relying on additional unlabeled data [Car+19; S+19]. However, despite the significant robustness improvements offered by these methods (and combinations of them), the robust accuracy of state-of-the-art models is still far from perfect, even when considering relatively small perturbations [Gow+20].

Other notions of worst-case perturbations. Going beyond ℓ_p-bounded input changes, a number of alternative notions of worst-case robustness have been proposed. These include spatial transformations (e.g., rotations and translations) [FF15; Eng+19b], deformations of the image [WSK19], and combinations of different ℓ_p-based bounds [TB19; MWK20].

Randomized smoothing. Finally, an orthogonal line of work has lead to a methodology know as *randomized smoothing* [Lec+19; CRK19]. The key idea here is to create a smoothed classifier that, starting from some input, evaluates a base classifier on multiple, noisy versions of this input and the predicts the majority vote of these. A byproduct of this methodology is that, if a large fraction of these noisy predictions agree, then this translates into a robustness bound on the overall smoothed classifier.

Chapter 2

Fundamentals of worst-case robustness

In the previous chapter, we focused on the phenomenon of adversarial examples and described a methodology for building more robust models. At the same, this discussion did not really shed any light on why standard models are brittle in the first place. That is, why is it so easy to find adversarial examples and why do they transfer across architectures?

Moreover, the distinction between standard and robust models is unclear. How can we intuitively understand the impact of robust optimization to our models? Also, why are the robust models we trained *less accurate* than their standard counterparts on the standard test set? Finally, are there qualitatively differences in what features these models learn?

Our goal in this chapter is to build a conceptual model of how adversarial examples arise and how robust optimization impacts model behavior.

Chapter outline:

- We will first formulate a simple and natural setting which captures many of the empirical behaviors observed around adversarial examples in Section 2.1.
- Then, in Section 2.2 we will design a set of experiments to better grasp how predictive this conceptual model is in real datasets.
- Finally, we will take a closer look at the features that robust models learn in Section 2.3 and explore what new modes of interaction they allow.

2.1 The robust features model

We will focus out study around a simple binary classification task. Specifically, we will consider inputs which comprise a set of independent, normally-distributed features. Crucially,

the degree to which each feature is correlated with the label of the input varies.

Concretely, the input-label pairs (x, y) are sampled from a distribution $\mathcal{D}$ as follows:

$$y \overset{u.a.r}{\sim} \{-1, +1\}, \qquad x_1 = \begin{cases} +y, & \text{w.p. } p \\ -y, & \text{w.p. } 1-p \end{cases}, \qquad x_2, \ldots, x_{d+1} \overset{i.i.d}{\sim} \mathcal{N}(\eta y, 1), \tag{2.1}$$

where $\mathcal{N}(\mu, \sigma^2)$ is a normal distribution with mean μ and variance σ^2, and $p \geq 0.5$. The parameter p quantifies how correlated the feature x_1 is with the label. For the sake of example, we can think of p as being 0.95. Note that the choice of p is fairly arbitrary; the same qualitative behavior will arise for any $p < 1$.

In the rest of this section, we will demonstrate how adversarial examples arise naturally in this model. Moreover, we will see how robustness requires models to rely on different sets of features, hence giving rise to an inherent robustness-accuracy trade-off.

2.1.1 Standard learning can lead to brittle models

Recall that in the canonical classification setting, the primary focus is on maximizing standard accuracy, i.e., the performance on (yet) unseen samples from the underlying distribution. Specifically, the goal is to train models that have low *expected loss* (also known as population risk):

$$\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} [\mathcal{L}(x, y; \theta)]. \tag{2.2}$$

Each sample from $\mathcal{D}$ consists of a single feature that is *moderately correlated* with the label and d other features that are only *very weakly* correlated with it. Despite the fact that each one of the latter type of features individually is hardly predictive of the correct label, this distribution turns out to be fairly simple to classify from a standard accuracy perspective. Specifically, a natural (linear) classifier

$$f_{\text{avg}}(x) := \text{sign}(w_{\text{unif}}^\top x), \quad \text{where } w_{\text{unif}} := \left[0, \frac{1}{d}, \ldots, \frac{1}{d}\right], \tag{2.3}$$

achieves standard accuracy arbitrarily close to 100%, for d large enough. Indeed, observe that

$$\Pr[f_{\text{avg}}(x) = y] = \Pr[\text{sign}(w_{\text{unif}} x) = y] = \Pr\left[\frac{y}{d}\sum_{i=1}^{d} \mathcal{N}(\eta y, 1) > 0\right] = \Pr\left[\mathcal{N}\left(\eta, \frac{1}{d}\right) > 0\right],$$

which is $> 99\%$ when $\eta \geq 3/\sqrt{d}$.

In general, when learning a standard classifier, any feature that is even slightly correlated with the label is useful. As a result, a standard classifier will take advantage (and thus rely on) the weakly correlated features $x_2, \ldots, x_{d+1}$ (by implicitly pooling information) to achieve almost perfect standard accuracy. However, this is not longer true in the presence of worst-case perturbations. In particular, an ℓ_∞-bounded adversary that is only allowed to perturb each feature by a moderate ε can effectively override the effect of the aforementioned meta-feature. For instance, if $\varepsilon = 2\eta$, an adversary can shift each weakly-correlated feature towards $-y$. The classifier would now see a perturbed input x' such that each of the features $x'_2, \ldots, x'_{d+1}$ are sampled i.i.d. from $\mathcal{N}(-\eta y, 1)$ (i.e., now becoming *anti*-correlated with the correct label). Thus, when $\varepsilon \geq 2\eta$, the adversary can essentially simulate the distribution of the weakly-correlated features as if belonging to the wrong class. In this case, the probability that the classifier of (2.3) is predicting correctly is

$$\min_{\|\delta\|_\infty \leq \varepsilon} \Pr[\operatorname{sign}(x+\delta) = y] \leq \Pr\left[\mathcal{N}\left(\eta, \frac{1}{d}\right) - \varepsilon > 0\right] = \Pr\left[\mathcal{N}\left(-\eta, \frac{1}{d}\right) > 0\right],$$

which in this specific setting is less than 1%.

Transferability. An interesting implication of our analysis is that standard training produces classifiers that rely on features that are weakly correlated with the correct label. This will be true for any classifier trained on the same distribution. Hence, the adversarial examples that are created by perturbing each feature in the direction of $-y$ will transfer across classifiers trained on independent samples from the distribution. This constitutes a manifestation of the phenomenon of transferability [Sze+14] and hints at its origin.

2.1.2 Robustness and accuracy can be at odds

Intriguingly, our discussion so far draws a distinction between *robust* features (x_1) and *non-robust* features ($x_2, \ldots, x_{d+1}$) that arises in the presence of worst-case perturbations. While aggregating the non-robust features is far more predictive of the true label, it can be easily manipulated. Hence, a tension between standard and adversarial accuracy arises. Any classifier that aims for high accuracy (say $> 99\%$) will have to heavily rely on non-robust features (the robust feature provides only, say, 95% accuracy). However, since the non-robust features can be arbitrarily manipulated, this classifier will inevitably have low adversarial accuracy. We make this formal in the following theorem proved in Appendix B.3.1.

Theorem 2.1.1 (Robustness-accuracy trade-off). *Any classifier that attains at least $1-\delta$ accuracy on $\mathcal{D}$ has robust accuracy at most $\frac{p}{1-p}\delta$ against an ℓ_∞-bounded adversary with $\varepsilon \geq 2\eta$.*

This bound implies that if $p < 1$, as accuracy approaches 100% ($\delta \to 0$), adversarial accuracy falls to 0%. As a concrete example, consider $p = 0.95$, for which any classifier with accuracy more than $1-\delta$ will have robust accuracy at most 19δ[1]. Also it is worth noting that the theorem is tight. If $\delta = 1-p$, both the standard and adversarial accuracies are bounded by p which is attained by the classifier that relies solely on the first feature. Additionally, note that compared to the scale of the features ± 1, the value of ε required to manipulate the standard classifier is very small ($\varepsilon = O(\eta)$, where $\eta = O(1/\sqrt{d})$).

Empirical robustness-accuracy trade-off. It turns out this trade-off is fairly pervasive when training robust models. In Figure 2.1 we observe that it holds for a variety of datasets, perturbations, and training set sizes. Interestingly, this trade-off appears to be reversed in the small-data regime especially for MNIST. We hypothesize that in this regime, robust training acts as a form of data augmentation, counteracting the price of robustness.

2.1.3 Learning a robust classifier

When training robust models, the goal is to achieve low *expected adversarial loss*:

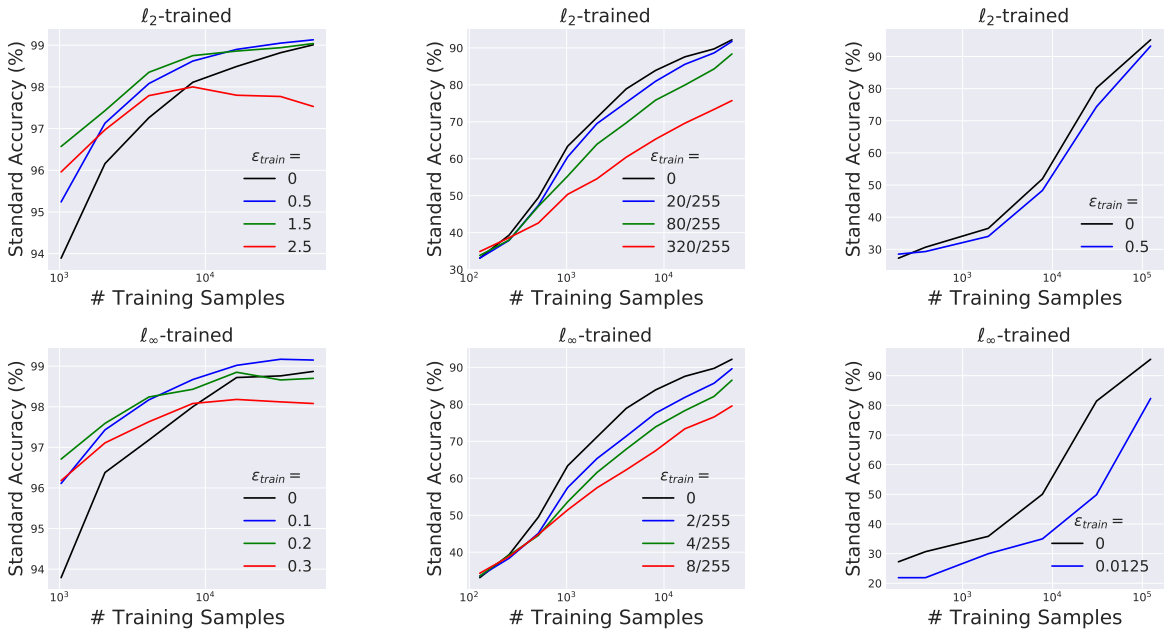
$$\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\Delta} \mathcal{L}(x+\delta, y; \theta) \right]. \tag{2.4}$$

Here, Δ represents the set of perturbations that the adversary can apply to induce misclassification. Similarly to before, we focus on the case when Δ is the set of ℓ_p-bounded perturbations, i.e., $\Delta = \{\delta \in \mathbb{R}^d \mid \|\delta\|_p \leq \varepsilon\}$.

As we have seen in the distributional model $\mathcal{D}$ (2.1), a classifier that achieves very high standard accuracy (2.2) will inevitably have near-zero adversarial accuracy. This is true even when a classifier with reasonable standard and robust accuracy exists. Hence, in an adversarial setting (2.4), where the goal is to achieve high adversarial accuracy, the training procedure needs to be modified. We now make this phenomenon concrete for linear classifiers trained using the soft-margin SVM loss. Specifically, in Appendix B.3.2 we prove the following theorem.

Theorem 2.1.2 (Adversarial training matters). *For $\eta \geq 4/\sqrt{d}$ and $p \leq 0.975$ (the first feature is not perfect), a soft-margin SVM classifier of unit weight norm minimizing the distributional loss*

[1]Hence, any classifier with standard accuracy $\geq 99\%$ has robust accuracy $\leq 19\%$ and any classifier with accuracy $\geq 96\%$ has robust accuracy $\leq 76\%$.



(a) MNIST (b) CIFAR-10 (c) Restricted ImageNet

Figure 2.1: Comparison of the standard accuracy of models trained against ℓ_2- and ℓ_∞-bounded perturbations as a function of size of the training dataset. We observe that when training with few samples, adversarial training has a positive effect on model generalization (especially on MNIST). However, as training data increase, the standard accuracy of robust models drops below that of the standard model ($\varepsilon_{train} = 0$).

achieves a standard accuracy of $> 99\%$ *and adversarial accuracy of* $< 1\%$ *against an* ℓ_∞*-bounded adversary of* $\varepsilon \geq 2\eta$*. Minimizing the distributional adversarial loss instead leads to a robust classifier that has standard and adversarial accuracy of* p *against any* $\varepsilon < 1$*.*

This theorem shows that if our focus is on robust models, adversarial training is *necessary* to achieve non-trivial adversarial accuracy in this setting. Soft-margin SVM classifiers and the constant 0.975 are chosen for mathematical convenience. Our proofs do not depend on them in a crucial way and can be adapted, in a straightforward manner, to other natural settings, e.g. logistic regression.

On the (non-)existence of an accurate and robust classifier. It might be natural to expect that in the regime of infinite data, the Bayes-optimal classifier—the classifier minimizing classification error with full-information about the distribution—is a robust classifier. Note however, that this is not true for the setting we analyze above. Here, the trade-off between standard and adversarial accuracy is an inherent trait of the data distribution itself and not due to having insufficient samples. In this particular classification task, we (implicitly) assumed that there does not exist a classifier that is *both* robust and very accurate (i.e.,

$>$ 99% standard and robust accuracy). Thus, for this task, any classifier that is very accurate (including the Bayes-optimal classifier) will necessarily be non-robust.

This seemingly goes against the common assumption in adversarial ML that such perfectly robust and accurate classifiers for standard datasets exist, e.g., humans. However, this might not necessarily be that case in the standard ML benchmarks. It is possible that given our current architectural and algorithmic choices, combined with the size and diversity of our current datasets, that this trade-off exists. That it, achieving perfect accuracy in these settings might not be possible without relying on brittle features. Thus, while the trade-off we analyzed here is unconditional, it might still arise in practical settings when conditioning on the current state-of-the-art.

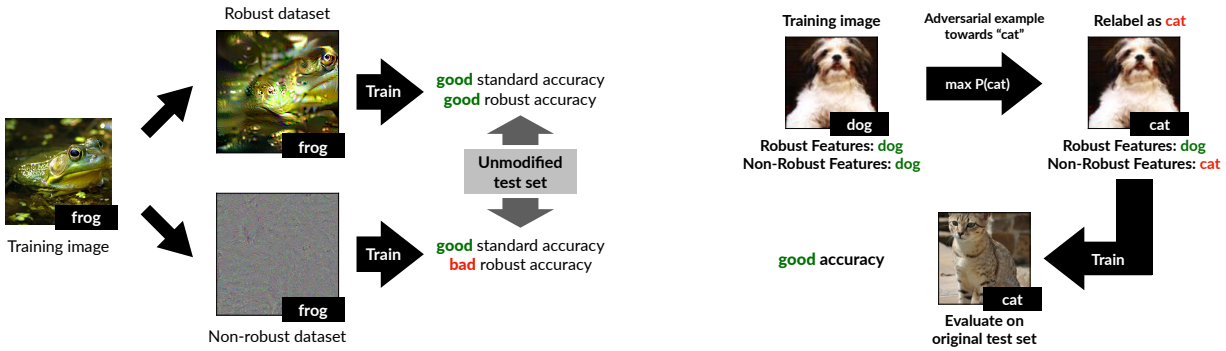
2.2 Non-robust features in real-world datasets

A crucial property of the model we are analyzing so far is that adversarial examples are a direct consequence of the standard learning paradigm. That is, adversarial perturbations arise as *well-generalizing, yet brittle, features*. This is a major departure from previous models for adversarial examples. In these models, adversarial examples tend to be viewed as aberrations arising either from the high dimensional nature of the input space or statistical fluctuations in the training data [Sze+14; GSS15; TG16; Gil+18; Sch+18; MDM18; Sha+19a] (see Section B.1 for a more detailed discussion and comparison).

Our goal in this section will be to take a closer look at real datasets and understand whether adversarial examples actually arise from predictive features. Specifically, we will design ways of modifying a dataset in order to disentangle robust and non-robust features and study how this affects the resulting classifier (see Figure 2.2 for a conceptual illustration). The main challenge is that, for complex domains (e.g., images), analyzing individual features (here pixels) is not particularly meaningful. Nevertheless, we can still conceptually distinguish between robust and non-robust features as follows.

Robust features. Features that are useful for making correct predictions that remain useful even after applying a small, worst-case perturbation. For instance, the presence of a large grass patch cannot be modified by a small perturbation.

Non-robust features. Features that are useful but can be rendered useless or even misleading through a small perturbation. For instance, these might include faint patterns on the texture of an object.



(a) (b)

Figure 2.2: A conceptual diagram of our experiments. In (a) we disentangle features into combinations of robust/non-robust features (Section 2.2.1). In (b) we construct a dataset which appears mislabeled to humans (via adversarial examples) but results in good accuracy on the original test set (Section 2.2.2).

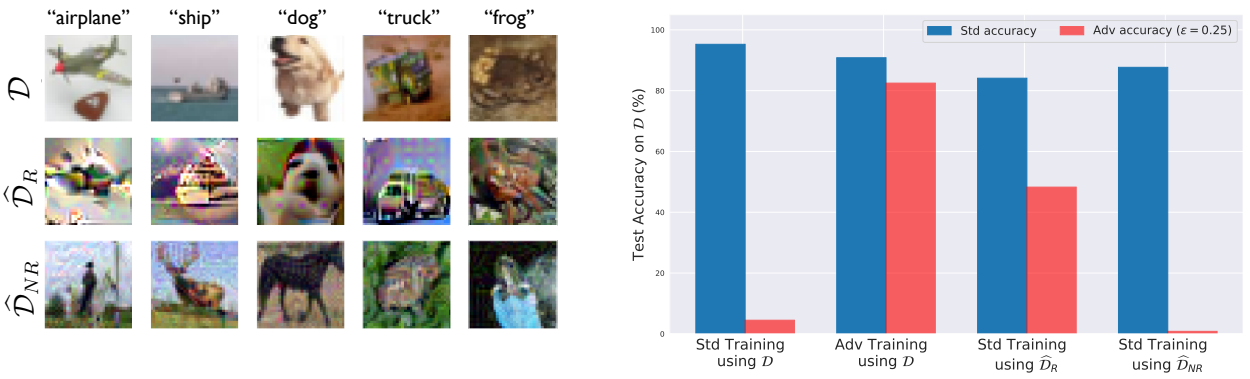
2.2.1 Simply removing non-robust features improves robustness

As a first step, we will explore whether we can effectively remove non-robust features from a dataset. If only the robust features of a dataset are useful then standard training will resulting robust models. Concretely, our goal is to create a training set (semantically similar to the original) on which *standard training* yields *good robust accuracy* on the *original, unmodified* test set (Figure 2.2a).

Unfortunately, we cannot directly manipulate the features of complex, high-dimensional datasets. Our key idea is that we can use a *robust model* (trained using the methodology of Chapter 1) as a proxy for the datasets robust features. We will then manipulate the dataset so that the only useful features are those relevant to that model.

Specifically, given a *robust* model C we will construct a training set $\widehat{\mathcal{D}}_R$ via a one-to-one mapping $x \mapsto x_r$ from the original training set for $\mathcal{D}$. As a proxy for the features that the robust model uses we will consider the representation of a data point in the penultimate layer of the classifier. To construct the desired points we will start from an input consisting of random noise. Since random noise is not associated with the label of the input, this noise-label pair contains no useful features. We will then optimize the input to minimize its distance to the original input in the latent representation space of the robust model. Intuitively, this process aims to preserve the robust features of the original input without introducing any of the other features.

Given the new training set for $\widehat{\mathcal{D}}_R$ (a few random samples are visualized in Figure 2.3a), 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 2.3b) indicate that the classifier learned using



(a) (b)

Figure 2.3: **Left**: Random samples from our variants of the CIFAR-10 [Kri09] training set: 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 relevant to 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 nontrivial robust accuracy.

the new dataset 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 2.3a—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 2.3b). 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.

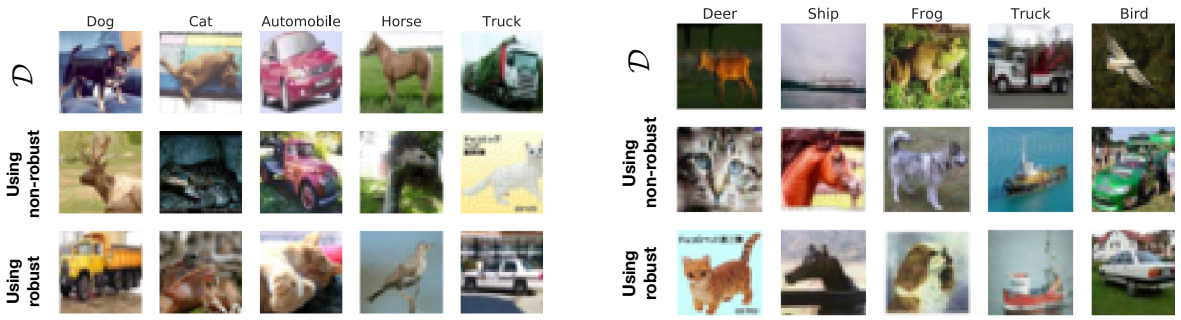
Overall, our findings corroborate the hypothesis that adversarial examples can arise from (non-robust) features of the data itself. By filtering out non-robust features from the dataset (e.g. by restricting the set of available features to those used by a robust model), one can train a significantly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

2.2.2 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 classifiers 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.

To show this, 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. The resulting inputs are nearly indistinguishable from the originals (cf. Figure 2.4)—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 (pseudocode in Appendix B.2 Figure B.6).



(a) $\widehat{\mathcal{D}}_{rand}$ (b) $\widehat{\mathcal{D}}_{det}$

Figure 2.4: 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 $\varepsilon = 0.5$).

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).

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 2.5: 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 (non-robust) 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 $\mathcal{D}$. These datasets are constructed using adversarial perturbations of x towards a class t (random for $\widehat{\mathcal{D}}_{rand}$ and deterministic for $\widehat{\mathcal{D}}_{det}$); the resulting images are relabeled as t.

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) [2].

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.

We find that standard training on these datasets actually generalizes to the *original* test set, as shown in Table 2.5). This indicates that non-robust features are indeed useful for classification in the standard setting. 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. This indicates that non-robust features can be picked up by models during standard training, even in the presence of *robust features* that are predictive

2.2.3 Transferability can arise from non-robust features

We will now turn our attention to the transferability of adversarial examples. Recall that according to our conceptual model, this phenomenon can in fact be viewed as a natural consequence of the existence of non-robust features. Given that such features are inherent

[2]Goh [Goh19] provides an approach to quantifying this "robust feature leakage" and finds that one can obtain a (small) amount of test accuracy by leveraging robust feature leakage on $\widehat{\mathcal{D}}_{rand}$.

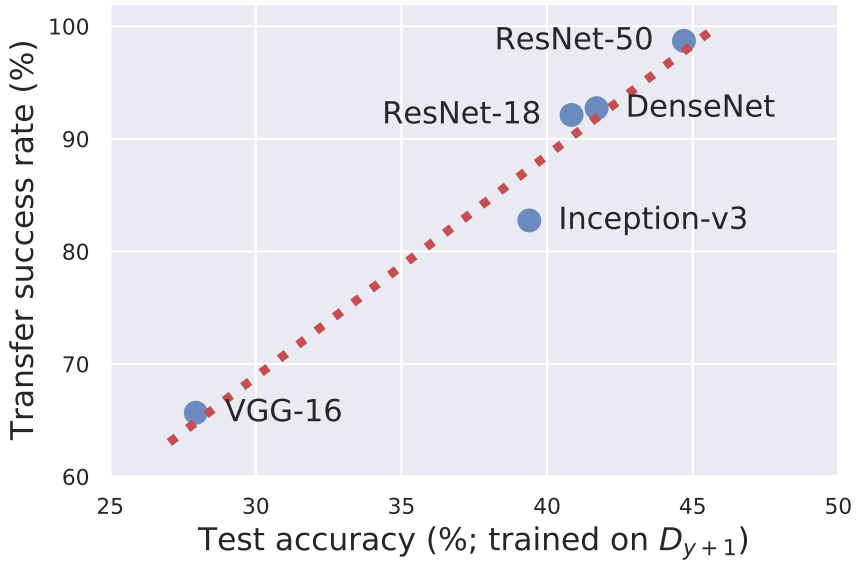


Figure 2.6: Transfer rate of adversarial examples from a ResNet-50 to different architectures alongside test set performance of these architecture when trained on the dataset generated in Section 2.2.2. Architectures more susceptible to transfer attacks also performed better on the standard test set supporting our hypothesis that adversarial transferability arises from utilizing similar *non-robust features*.

to the data distribution, different 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.

In order to illustrate and corroborate this hypothesis, we train five different architectures on the dataset generated in Section 2.2.2 (adversarial examples with deterministic labels) for a standard ResNet-50 [He+16]. Our hypothesis would suggest that architectures which learn better from this training set (in terms of performance on the standard test set) are more likely to learn similar non-robust features to the original classifier. Indeed, we find that the test accuracy of each architecture is predictive of how often adversarial examples transfer from the original model to standard classifiers with that architecture (Figure 2.6). These findings thus corroborate our hypothesis that adversarial transferability arises when models learn similar brittle features of the underlying dataset.

2.3 What do robust features look like?

In the previous section, we provided evidence towards the fact that robust models rely on a different set of features than standard models. Can we explore the features and gain a better understanding of what they are?

Perhaps the most natural approach to visualize the features of a robust model would be to find inputs that are confidently predicted as a specific class. This would correspond to computing large, worst-case perturbations—e.g., via PGD—similarly to how one would find adversarial examples. Intuitively, this process will highlight the features that the model most strongly relies on for recognizing that class.

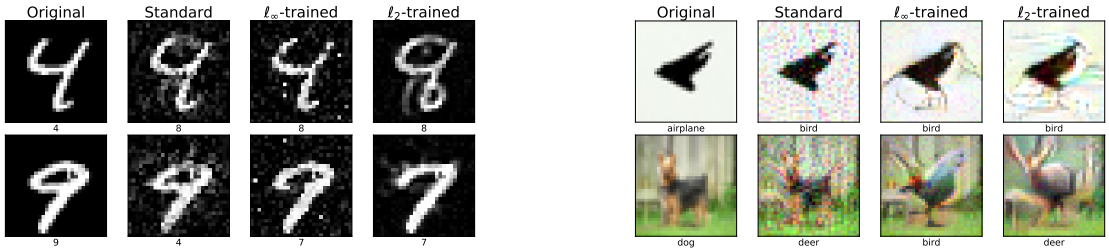
The resulting visualizations are presented in Figure 2.7. Surprisingly, we can observe that adversarial perturbations for robust models tend to produce salient characteristics of another class. In fact, the corresponding adversarial examples for robust models can often be perceived as samples from that class. This behavior is in stark contrast to standard models, for which adversarial examples appear to humans as noisy variants of the input image. Interestingly, Ross and Doshi-Velez [RD18] find a similar behavior when, instead of adversarially training models, they regularize their gradients during training. This indicates that even a very local notion of robustness, can have a large effect on how a model behaves.

In the rest of this section we will further study this representations of robust models in two ways. First, in Section 2.3.1 we will focus on the intermediate representations learned by robust models. Second, in Section 2.3.2 we will study how the phenomena we observe can be leveraged to perform downstream image synthesis tasks.

2.3.1 Latent model representations

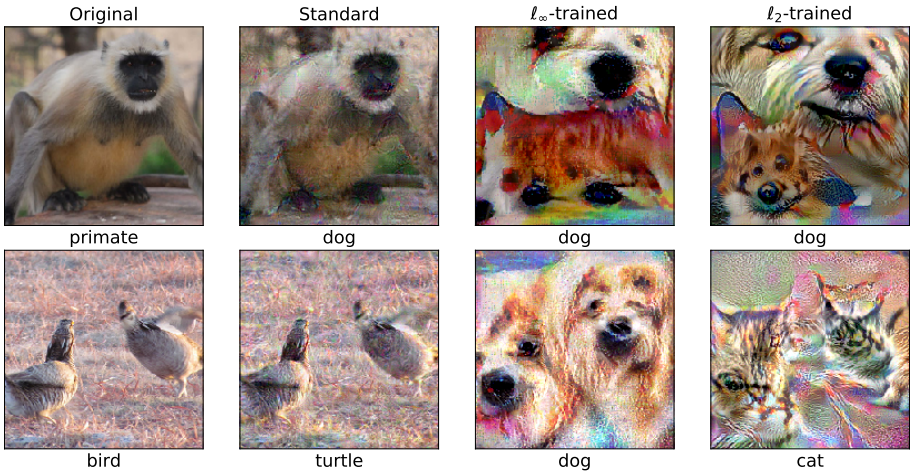
In general, deep neural networks can be thought of as linear classifiers acting on *learned feature representations* (also known as *feature embeddings*). A major goal in representation learning is for these embeddings to encode high-level, interpretable features of any given input [GBC16; BCV13; Ben19]. Indeed, learned representations turn out to be quite versatile—in computer vision, for example, they are the driving force behind transfer learning [Gir+14; Don+14], and image similarity metrics such as VGG distance [DB16a; JAF16; Zha+18].

Here we will see how robustness can significantly improve the utility and versatility of these representations by enabling new modes of interaction. The experiment setup for the rest of this Section is presented in Appendix B.2



(a) MNIST

(b) CIFAR-10



(c) Restricted ImageNet

Figure 2.7: Visualizing large-ε adversarial perturbations for standard and robust (ℓ_2/ℓ_∞-adversarial training) models. We construct these examples by iteratively following the (negative) loss gradient while staying with ℓ_2-distance of ε from the original image. We observe that the images produced for robust models effectively capture salient data characteristics and appear similar to examples of a different class. (The value of ε is equal for all models and much larger than the one used for training.) Additional examples are visualized in Figure B.8 and B.9 of Appendix B.4.

Inverting robust representations

In order to better understand the features captured by robust representations, we will attempt to reconstruct their corresponding inputs. Specifically, starting from some image x_1, we will attempt to find an image x_2 with a similar latent representation. To do so, we will minimize the ℓ_2-distance of these images in representation space by solving the following optimization problem:

$$x_1' = x_1 + \arg\min_{\delta} \|R(x_1 + \delta) - R(x_2)\|_2. \tag{2.5}$$

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]. It turns out that in sharp contrast to what we observe for standard models, the images resulting from minimizing (2.5) for robust models are actually *semantically similar* to the original (target) images whose representation is being matched, and this behavior is consistent across multiple samplings of the starting point (source image) x_1 (cf. Figure 2.8).

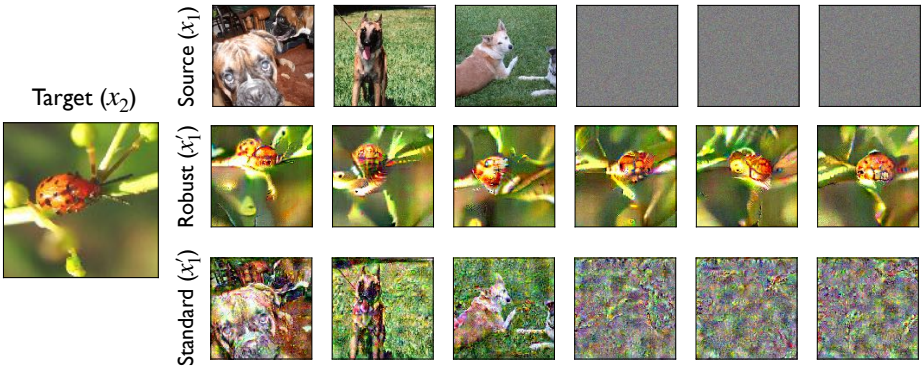


Figure 2.8: Inverting standard and robust models on the Restricted ImageNet dataset. *Target* (x_2) & *Source* (x_1): random examples image from the test set or white noise; *Robust*(x_1') and *Standard*(x_1'): result of minimizing the objective (2.5) to match (in ℓ_2-distance) the representation of the target image starting from the corresponding source image for (*top*): a robust (adversarially trained) and (*bottom*): a standard model respectively. For the robust model, we observe that the resulting images are perceptually similar to the target image in terms of high-level features (even though they do not match it exactly), while for the standard model they often look more similar to the source image which is the seed for the optimization process. Additional results in Appendix B.4.1, and similar results for ImageNet are in Appendix B.4.

Representation proximity seems to entail semantic similarity. In fact, we can see that the similarity between the resulting image and the target image is not simply a result of some implicit bias of the optimization process. 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 (2.5)). 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 2.9. This indicates that for robust models, semantic similarity may in fact be necessary for representation similarity.

We also find that even when δ is highly constrained (i.e., when we are forced to stay close to the source image and thus cannot match the target representation well), the solution to the inversion problem still displays salient features of the target image (c.f.

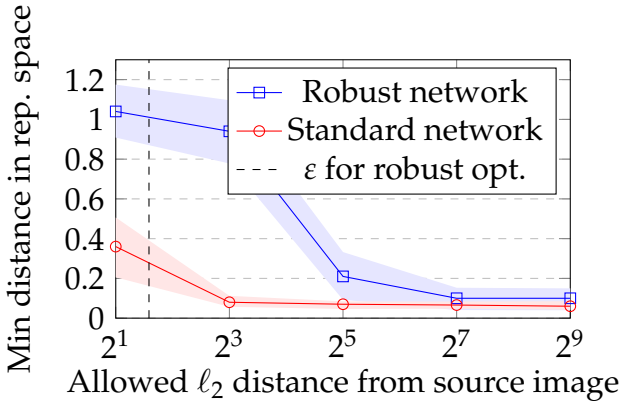


Figure 2.9: Optimizing objective (2.5) with PGD and an ℓ_2-norm constraint around the source image. On the x-axis is the radius of the constraint set, and on the y-axis is the distance between the representation of the minimizer of (2.5) within the constraint set and the representation of the target image, normalized by the norm of the latter: i.e., a point (x_i, y_i) on the graph corresponds to $y_i = \min_{\|\delta\|_2 \leq x_i} \|R(x+\delta) - R(x_{targ})\|_2 / \|R(x_{targ})\|_2$. Notably, we are unable to closely match the representation of the target image for the robust network until the norm constraint grows very large, and in particular much larger than the norm of the perturbation that the model is trained to be robust against. Shown are 95% confidence intervals over random choice of source and target images.

Figure 2.10). Both of these observations suggest that the representations of robust networks function closer to how we would expect high-level feature representations to behave.

Inversion of out-of-distribution inputs. 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 (minimization of distance in representation space) using images from classes not present in the original training dataset (Figure 2.11 right) and structured random patterns (Figure B.11 in Appendix B.4.1): the reconstructed images consistently resemble the targets.

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, we use gradient descent to find

Figure 2.10: Visualizing the final solutions to the optimizing objective (2.5) when constraining the solution to lie in an ℓ_2 ball around the source image for an adversarially robust neural network. Even when the radius of the constraint set is small and we cannot match the representation very well, salient features of the target image still arise.

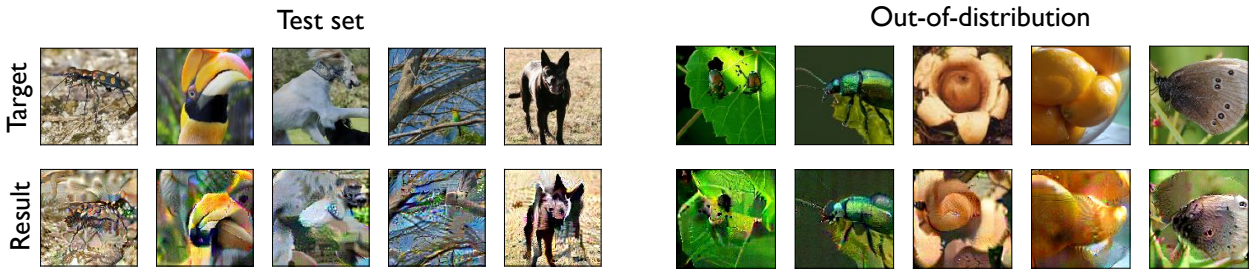


Figure 2.11: Robust representations yield semantically meaningful embeddings. *Target*: random images from the test set (col. 1-5) and from outside of the training distribution (6-10); *Result*: images obtained from optimizing inputs (using Gaussian noise as the source image) to minimize ℓ_2-distance to the representations of the corresponding image in the top row. (More examples appear in Appendix B.4.1.)

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

$$x' = x_0 + \arg\max_{\delta} R(x_0 + \delta)_i \tag{2.6}$$

for various starting points x_0 which might be random images from $\mathcal{D}$ or random noise.

For robust representations, we find that easily recognizable high-level features emerge from optimizing objective (2.6) directly, *without any regularization or post-processing*. We present the results of this maximization in Figure 2.12 (top): coordinates consistently represent the same concepts across different choice of starting input x_0 (both in and out of distribution). 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 2.13).

This is again a significant departure from standard models where direct feature visualization does not produce unintelligible results. Thus prior work on visualization usually tries to regularize objective (2.6) through a variety of methods. These methods include applying random transformations during the optimization process [MOT15; OMS17],

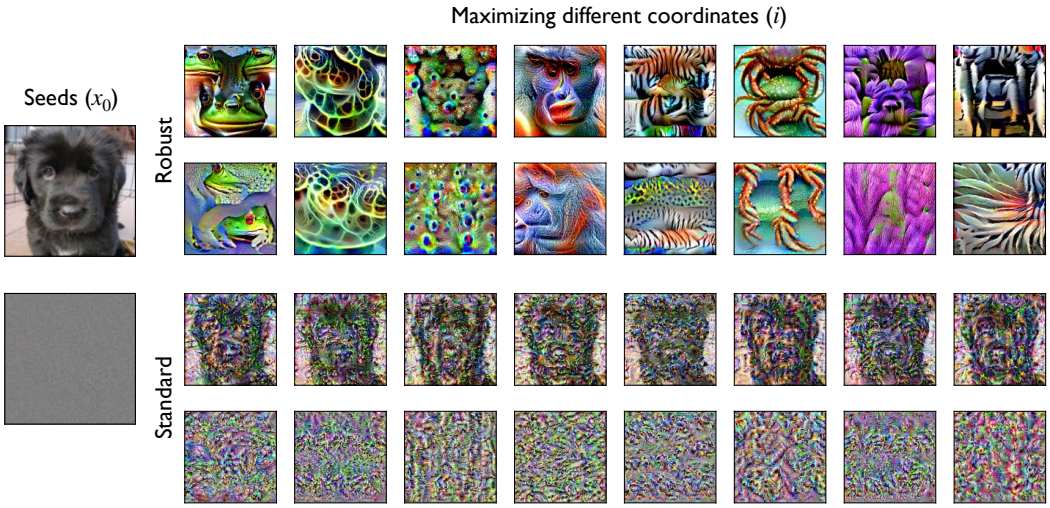


Figure 2.12: Correspondence between image-level patterns and activations learned by standard and robust models on the Restricted ImageNet dataset. Starting from randomly chosen seed inputs (noise/images), we use PGD to find inputs that (locally) maximally activate a given component of the representation 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 (2.6), 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. Additional visualizations in Appendix B.4.2, and results for ImageNet in B.4.2.

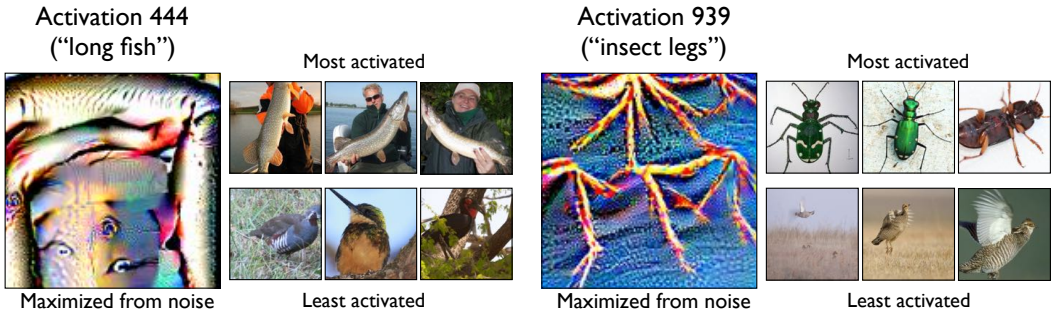


Figure 2.13: Maximizing inputs x' found by solving (2.6) with x_0 being a gray image for two *random* activations of a robust model trained on the Restricted ImageNet dataset. For each activation, we plot the three images from the validation set that had the highest or lowest activation value sorted by the magnitude of the selected activation.

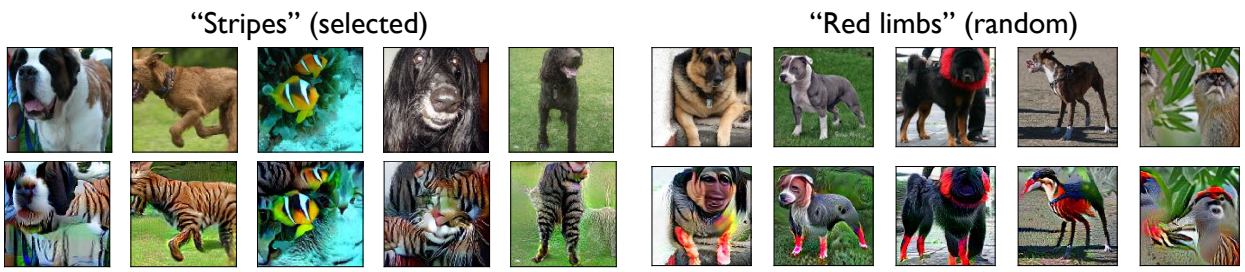


Figure 2.14: Visualization of the results from maximizing a chosen (left) and a *random* (right) representation coordinate starting from *random* images for the Restricted ImageNet dataset. In each figure, the top row has the initial images, and the bottom row has a feature added. Additional examples in Appendix B.4.3.

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. 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. Indeed, Olah, Mordvintsev, and Schubert [OMS17] note that in the representation layer of a standard GoogLeNet, "Neurons do not seem to correspond to particularly meaningful semantic ideas."

Feature manipulation. The ability to directly visualize high-level, recognizable features reveals another application of robust representations, which we refer to as *feature manipulation*. Consider the visualization objective (2.6) 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 2.14, where *"stripes"* or *"red limbs"* are introduced seamlessly into images without any processing or regularization. We repeat this process with many additional random images and random features in Appendix B.4.3. Note that this property is reminiscent of how the latent space of generative adversarial networks (GANs) [Goo+14] tends to allow for "semantic feature arithmetic" [RMC16; Lar+16].

2.3.2 Using robust features in downstream tasks

Recall that in Figure 2.7, we demonstrated how simple PGD in the input space of robust models is sufficient to produce salient class characteristics. This process, which we will call *class maximization*, effectively utilizes the features learned by the model to manipulate inputs semantically. In this section, we will see how this primitive is sufficient to perform some challenging image synthesis tasks. This toolkit we develop is rather minimal: it uses a *single, off-the-shelf* robust classifier for *all* these tasks.

Realistic Image Generation

Synthesizing realistic samples for natural data domains (such as images) has been a long standing challenge in computer vision [HS97; Gra13; KW15; Goo+14; VKK16; DSB17; KD18; Kar+18; BDS19]. Many of these methods, however, can be tricky to train and properly tune, often requiring fine-grained performance optimizations.

In contrast, we will see 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 the process of class score maximization is deterministic, generating a diverse set of samples requires a random seed as the starting point of the maximization process. That is, 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 projected gradient descent (PGD). Ideally, samples from $\mathcal{G}_y$ should be diverse and statistically similar to the data distribution. Here, we use a simple choice for $\mathcal{G}_y$—a multivariate normal distribution fit to the empirical class-conditional distribution

$$\mathcal{G}_y := \mathcal{N}(\mu_y, \Sigma_y), \quad \text{where } \mu_y = \mathbb{E}_{x\sim\mathcal{D}_y}[x],\ \Sigma = \mathbb{E}_{x\sim\mathcal{D}_y}[(x-\mu_y)^\top (x-\mu_y)],$$

and $\mathcal{D}_y$ is the distribution of natural inputs conditioned on the label y. We visualize example seeds from these multivariate Gaussians in Figure B.21.

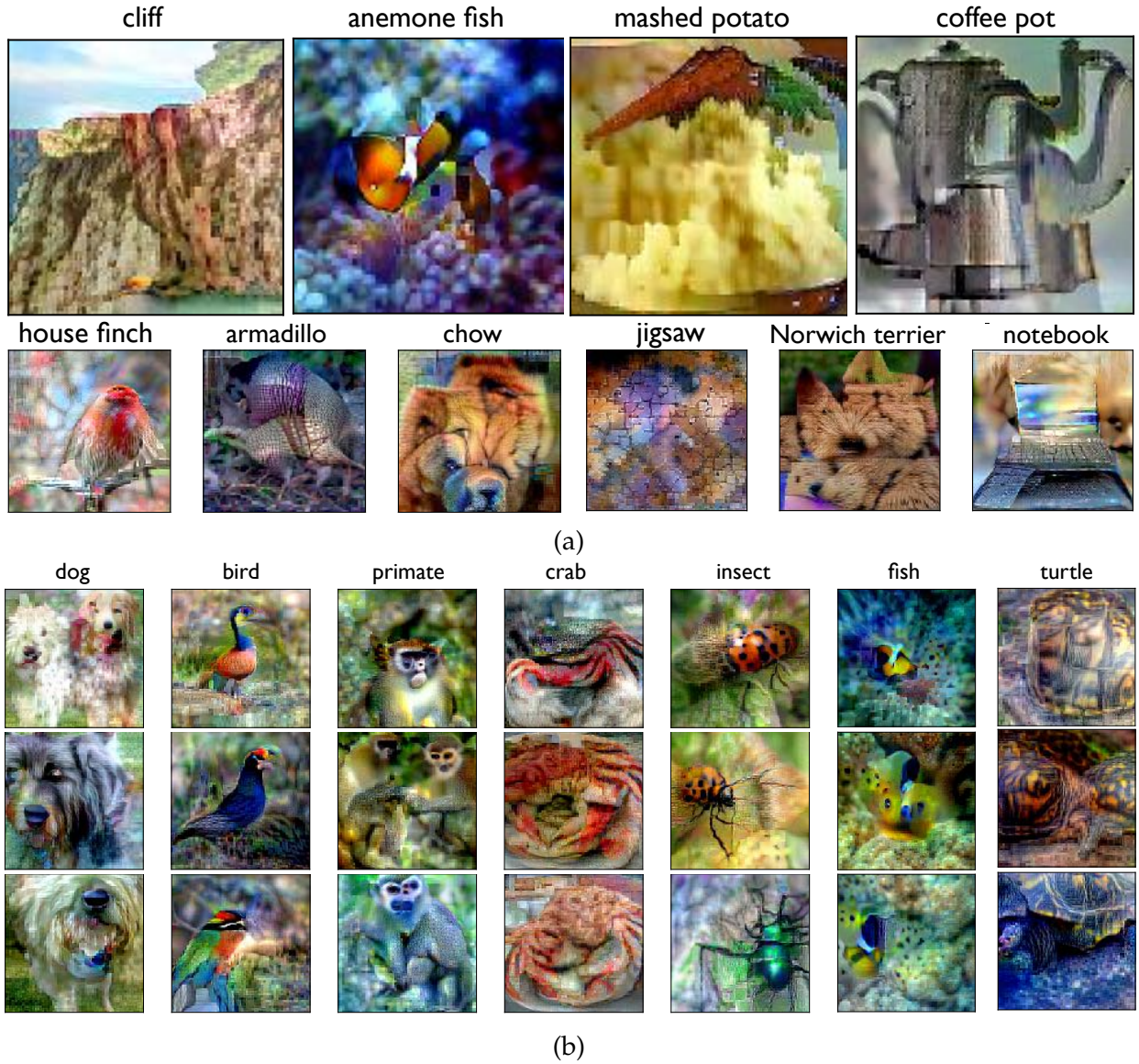


Figure 2.15: *Random* samples (of resolution 224×224) produced using a robustly trained classifier. We show: (a) samples from several (random) classes of the ImageNet dataset and (b) multiple samples from a few random classes of the restricted ImageNet dataset (to illustrate diversity). See Figures B.17, B.18, B.19, and B.20 of Appendix B.4 for additional samples.

This approach enables us to perform *conditional* image synthesis given any target class. Samples (at resolution 224×224) produced by our method are shown in Figure 2.15 (also see Appendix B.4). The resulting images are diverse and realistic, despite the fact that they are generated using targeted PGD on off-the-shelf robust models.

Inpainting

Since class maximization can introduce salient class features, it has potential to restore the features of corrected images. This task is know as inpainting [EL99; Ber+00; HE07], where, given an image x, corrupted in a region corresponding to a binary mask $m \in \{0,1\}^d$, the goal is to recover the missing pixels in a manner that is perceptually plausible with respect to the rest of the image. To do so using a robust classifier, given a corrupted image x with label y, we solve

$$x_I = \arg\min_{x'} \mathcal{L}(x', y) + \lambda ||(x - x') \odot (1 - m)||_2 \tag{2.7}$$

where $\mathcal{L}$ is the cross-entropy loss, $\odot$ denotes element-wise multiplication, and λ is an appropriately chosen constant.

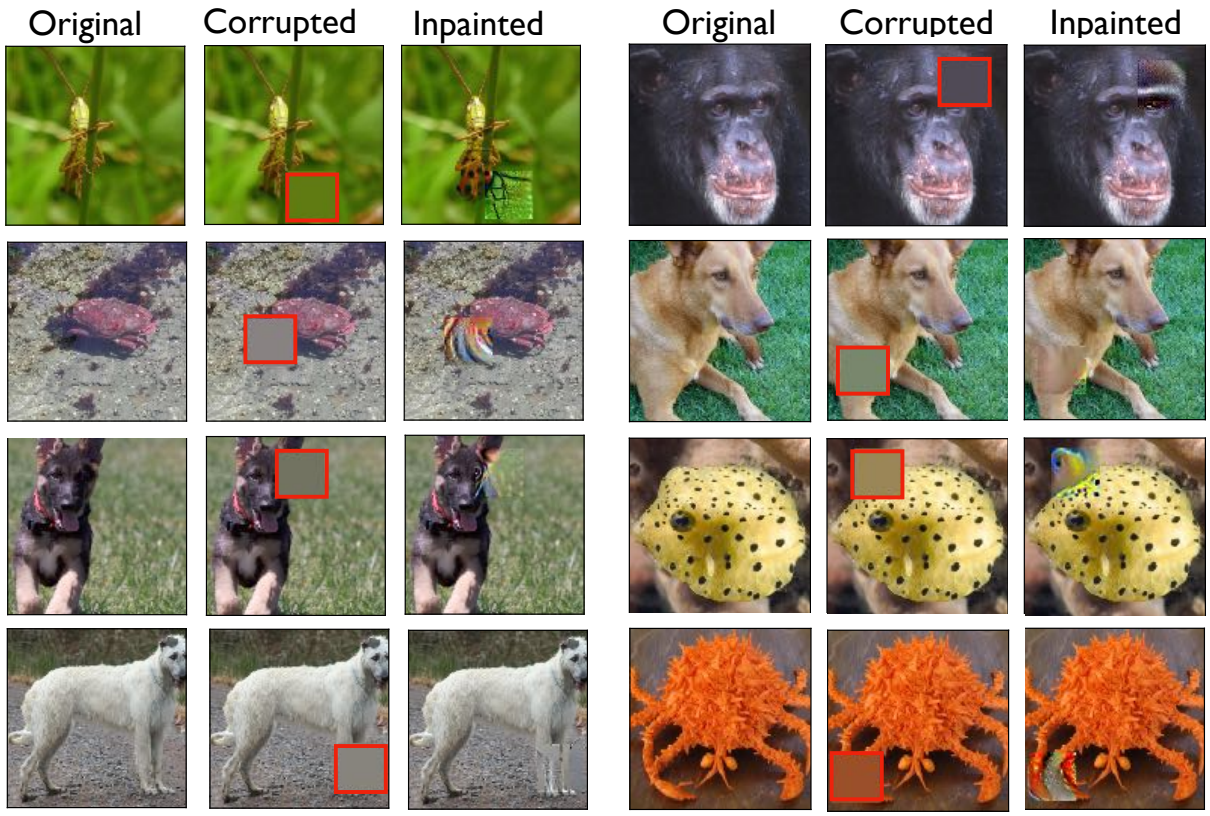
In Figure 2.16, we show sample reconstructions obtained by optimizing (2.7) using PGD (cf. Appendix B.2 for details). We can observe that these reconstructions look remarkably similar to the uncorrupted images in terms of semantic content. Interestingly, even when this approach fails (reconstructions differ from the original), the resulting images do tend to be perceptually plausible to a human, as shown in Appendix Figure B.25.

Image-to-Image Translation

Another task which can be cast as transforming images between class is *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].

In this section, we demonstrate that robust classifiers give rise to a new methodology for performing such image-to-image translations. 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 2.17, 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. (For completeness, we present in Appendix B.4 Figure B.23 results corresponding to using a classifier trained on the complete ImageNet dataset.) 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).



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

Figure 2.16: Image inpainting using robust models – *left:* original, *middle:* corrupted and *right:* inpainted samples. To recover missing regions, we use PGD to maximize the class score predicted for the image while penalizing changes to the uncorrupted regions.

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 of objects[3]) 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.

[3]In fact, we encountered such an issue with ℓ_∞-robust classifiers for horses and zebras (Figure B.24). Note that generative approaches also face similar issues, where the background is transformed instead of the objects [Zhu+17].

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

Figure 2.17: 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. See Appendix B.2 for experimental details and Figure B.22 for additional input-output pairs.

Super-Resolution

Accentuating the salient characteristics of an image is also an instance of super-resolution, the task of recovering high-resolution images given their low resolution version [DFE07; BSH12]. Within our framework, 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

$$\hat{x}_H = \underset{||x' - \uparrow(x_L)|| < \varepsilon}{\arg\min} \quad \mathcal{L}(x', y) \tag{2.8}$$

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

We use this approach to upsample *random* 32×32 CIFAR-10 images to full ImageNet size (224×224)—cf. Figure 2.18a. For comparison, we also show upsampled images obtained from bicubic interpolation. In Figure 2.18b, we visualize the results for super-resolution on *random* 8-fold down-sampled images from the restricted ImageNet dataset.

(a) 7x super-resolution on CIFAR-10 (b) 8x super-resolution on restricted ImageNet

Figure 2.18: 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 obtain semantically meaningful reconstructions that are especially sharp in regions that contain class-relevant information.

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.

Interactive Image Manipulation

Finally, we take a look at two interactive applications based on the primitives that robust classifier enable. Such applications have been recently shown to be possible through the use of generative models [CH18; Par+19; Bau+19]. In this section, we show how our framework can be used to enable similar artistic applications.

Sketch-to-image. By performing PGD to maximize the probability of a chosen target class, we can use robust models to convert hand-drawn sketches to natural images. The resulting images (Figure 2.19) appear realistic and contain fine-grained characteristics of the corresponding class.

[4]Sketches were produced by a graduate student without any training in arts.

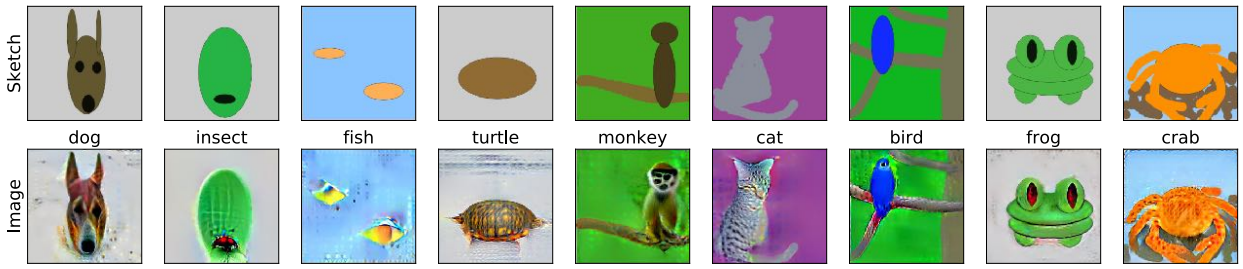


Figure 2.19: Sketch-to-image using robust model gradients. *Top:* manually drawn sketches of animals; and *bottom:* result of performing PGD towards a chosen class. The resulting images appear realistic while preserving key characteristics of the original sketches[4].

Feature Painting. Generative model–based paint applications often allow the user to control more fine-grained features, as opposed to just the overall class. We now show that we can perform similar feature manipulation through a minor modification to our basic primitive of class score maximization. Our methodology is based on the findings of the previous section wherein manipulating individual activations within representations of a robust model actually results in consistent and meaningful changes to high-level image features (e.g., adding stripes to objects). We can thus build a tool to paint specific features onto images by maximizing individual activations directly, instead of just the class scores.

Concretely, given an image x, if we want to add a single feature corresponding to component f of the representation vector $R(x)$ in the region corresponding to a binary mask m, we simply apply PGD to solve

$$x_I = \arg\max_{x'} R(x')_f - \lambda_P ||(x - x') \odot (1 - m)||. \tag{2.9}$$

In Figure 2.20, we demonstrate progressive addition of features at various levels of granularity (e.g., grass or sky) to selected regions of the input image. We can observe that such direct maximization of individual activations gives rise to a versatile paint tool.

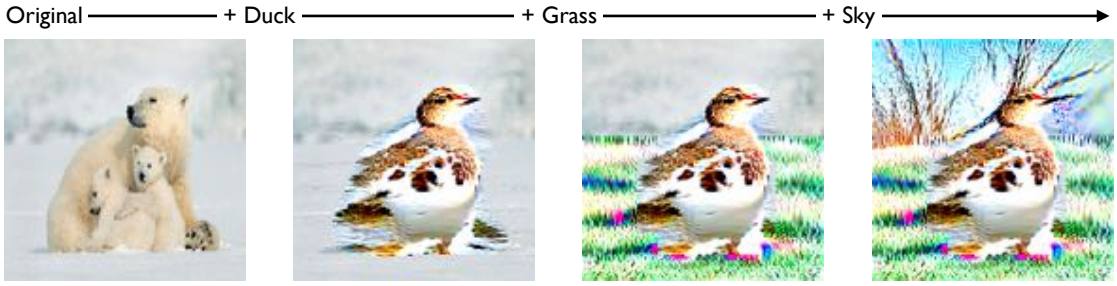


Figure 2.20: Paint-with-features using a robust model—we present a sequence of images obtained by successively adding specific features to regions of the image by solving (2.9).

Part II

Real-World Robustness

Chapter 3

Towards capturing real-world deployment

Our discussion has so far been restricted on a concrete and well-defined notion of robustness: small, worst-case perturbations. While this restriction was necessary to rigorously explore different facets of robustness, it is not representative of the conditions that model will encounter during deployment. Thus, we will now shift our focus and attempt to capture the robustness challenges of the real world.

The gap between the training and deployment environment of a model has been the focus of a long line of work in machine learning [SG86; WK93; KHA99; Shi00; SKM07; Qui+09; Mor+12; SK12]. At a high-level, the goal is to ensure that models perform well not only on unseen samples from the datasets they are trained on, but also on the diverse set of inputs they are likely to encounter in the real world. However, building benchmarks for evaluating such robustness is challenging—it requires modeling realistic data variations in a way that is well-defined, controllable, and easy to simulate.

Prior work in this context has focused on building benchmarks that capture distribution shifts caused by natural or adversarial input corruptions [Sze+14; FF15; FMF16; Eng+19b; For+19; HD19; Kan+19], differences in data sources [Sae+10; TE11; Kho+12; TT14; Rec+19], and changes in the frequencies of data subpopulations [Ore+19; Sag+20]. While each of these approaches captures a different source of real-world distribution shift, we cannot expect any single benchmark to be comprehensive. Thus, to obtain a holistic understanding of model robustness, we need to keep expanding our testbed to encompass more natural modes of variation.

Our goal in this chapter will be to design benchmark for two general families of challenges that models will face during deployment.

Chapter structure

- In Section 3.1, we develop a methodology for constructing subpopulation shift benchmarks without the need for significant data collection or annotation.
- In Section 3.2, we focus on measuring model robustness to perturbations that affect individual concepts within an input.

3.1 Simulating subpopulation shift

In this section, we will explore the question: How well do models generalize to data subpopulations they have not seen during training? The notion of *subpopulation shift* this question refers to is quite pervasive. After all, our training datasets will inevitably fail to perfectly capture the diversity of the real word. Hence, during deployment, our models are bound to encounter unseen subpopulations—for instance, unexpected weather conditions in the self-driving car context or different diagnostic setups in medical applications.

Specifically, we focus on modeling a pertinent, yet relatively little studied, form of subpopulation shift: one wherein the target distribution (used for testing) contains subpopulations that are *entirely* absent from the source distribution that the model was trained on. Intuitively, we will aim to understand whether models can recognize Dalmatians as "dogs" even when their training data for "dogs" comprises only Poodles and Terriers.

3.1.1 The BREEDS methodology

The crux of our approach is to leverage existing dataset labels and use them to identify *superclasses*—i.e., groups of semantically similar classes. This allows us to construct classification tasks over such superclasses, and repurpose the original dataset classes to be the subpopulations of interest. Our procedure for doing this comprises two stages that are outlined below—see Figure 3.1 for an illustration and Appendix C.1.2 for pseudocode.

Devising subpopulation structure. Typical datasets do not contain annotations for individual subpopulations. Since collecting such annotations would be challenging, we take an alternative approach: we bootstrap the existing dataset labels to simulate subpopulations. That is, we group semantically similar classes into broader superclasses which, in turn, allows us to re-purpose existing class labels as the desired subpopulation annotations. Moreover, we can group classes in a hierarchical manner, obtaining superclasses of differ-

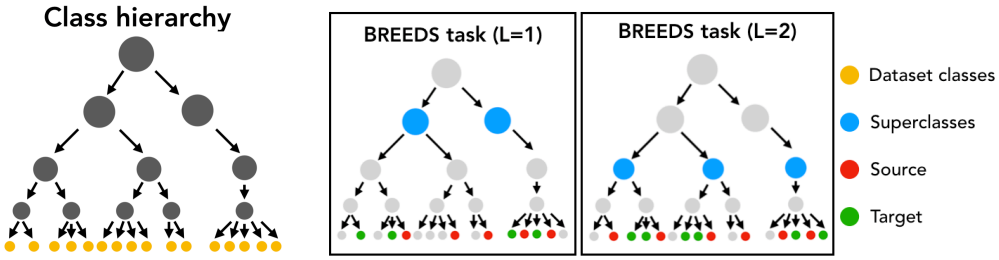


Figure 3.1: Illustration of our pipeline to create subpopulation shift benchmarks. Given a dataset, we define superclasses based on the semantic hierarchy of dataset classes. This allows us to treat the dataset labels as subpopulation annotations. Then, we construct a BREEDS task of specified granularity (i.e., depth in the hierarchy) by posing the classification task in terms of superclasses at that depth and then partitioning their respective subpopulations into the source and target domains.

ent specificity. As we will see in Section 3.1.2, such class hierarchies are already present in large-scale benchmarks [Den+09; Kuz+18].

Simulating subpopulation shifts. Given a set of superclasses, we can define a classification task over them: the inputs of each superclass correspond to pooling together the inputs of its subclasses (i.e., the original dataset classes). Within this setup, we can simulate subpopulation shift in a relatively straightforward manner. Specifically, for each superclass, we split its subclasses into two *random* and *disjoint* sets, and assign one of them to the source and the other to the target domain. Then, we can evaluate model robustness under subpopulation shift by simply training on the source domain and testing on the target domain. Note that the classification task remains identical between domains—both domains contain the same (super)classes but the subpopulations that comprise each (super)class differ. [1] Intuitively, this corresponds to using different dog breeds to represent the class "dog" during training and testing—hence the name of our toolkit.

This methodology is quite general and can be applied to a variety of setting to simulate realistic distribution shifts. Moreover, it has a number of additional benefits:

- **Flexibility:** Different semantic groupings of a fixed set of classes lead to BREEDS tasks of varying granularity. For instance, by only grouping together classes that are quite similar one can reduce the severity of the subpopulation shift. Alternatively, one can consider broad superclasses, each having multiple subclasses, resulting in a

[1] Note that this approach can be extended to simulate milder subpopulation shifts where the source and target distributions overlap but the relative subpopulation frequencies vary, similar to the setting of Oren et al. [Ore+19].

more challenging benchmark.

- **Precise characterization:** The exact subpopulation shift between the source and target domains is known. Since both domains are constructed from the same dataset, the impact of any external factors (e.g., differences in data collection pipelines) is minimized. Note that such external factors can significantly impact the difficulty of the task [Pon+06; TE11; Tsi+20]. In fact, minimizing these effects and ensuring that the shift between the source and target domain is caused solely by the intended input variations is one of the major challenges in building distribution shift benchmarks. For instance, recent work [Eng+20] demonstrates that statistical biases during data collection can significantly skew the intended target distribution.

- **Symmetry:** Since subpopulations are split into the source and test domains randomly, we expect the resulting tasks to have comparable difficulty.

- **Reuse of existing datasets:** No additional data collection or annotation is required other than choosing the class grouping. This approach can thus be used to also repurpose other existing large-scale datasets—even beyond image recognition—with minimal effort.

Note that this methodology can be viewed as a way of creating benchmarks for *domain generalization*. However, we focus on generalizing between different distributions of real-world images (photographs). This is in contrast to typical domain generalization benchmarks that focus on generalizing between different stylistic representations, e.g., from cartoons to drawings. Hence, the only comparable benchmark would be VLCS [Ghi+15], which is however significantly smaller in scale and granularity than our benchmarks. In a similar vein, datasets used in federated learning [Cal+18] can be viewed as subpopulation shift benchmarks since the users present during training and testing might differ. However, to the best of our knowledge, there has been no large-scale vision benchmark in this setting.

Hendrycks and Dietterich [HD19], in Appendix G, also (manually) construct a classification task over superclasses and use ImageNet classes outside of ILSVRC2012 (ImageNet-1k) to measure "subtype robustness". (Unfortunately, these classes are no longer publicly available [Yan+19].) Compared to their work, we use a general methodology to create a broader suite of benchmarks. Also, our analysis of architectures and robustness interventions is significantly more extensive.

3.1.2 Utilizing the ImageNet class hierarchy

We now describe how our methodology can be applied to ImageNet [Den+09]—specifically, the ILSVRC2012 subset [Rus+15]—to create a suite of BREEDS benchmarks. ImageNet contains a large number of classes, making it particularly well-suited for our purpose.

Recall that creating BREEDS tasks requires grouping together similar classes. For ImageNet, such a semantic grouping already exists—ImageNet classes are a part of the WordNet hierarchy [Mil95]. However, WordNet is not a hierarchy of objects but rather one of word meanings. Thus, intermediate hierarchy nodes are not always well-suited for object recognition due to:

- **Abstract groupings:** WordNet nodes often correspond to abstract concepts, e.g., related to the functionality of an object. Children of such nodes might thus share little visual similarity—e.g., "umbrella" and "roof" are visually different, despite both being "coverings."
- **Non-uniform categorization:** The granularity of object categorization is vastly different across the WordNet hierarchy—e.g., the subtree rooted at "dog" is 25-times larger than the one rooted at "cat." Hence, the depth of a node in this hierarchy does not always reflect the specificity of the corresponding object category.
- **Lack of tree structure:** Nodes in WordNet can have multiple parents and thus the resulting classification task would contain overlapping classes, making it inherently ambiguous.

Due to these issues, we cannot directly use WordNet to identify superclasses that correspond to a well-calibrated classification task. To illustrate this, we present some of the superclasses that Huh, Agrawal, and Efros [HAE16] constructed by applying clustering algorithms directly to the WordNet hierarchy in Appendix Table C.1. Even putting the issue of overlapping classes aside, a BREEDS task based on these superclasses would induce a very skewed subpopulation shift across classes—e.g., varying the types of "bread" is very different that doing the same for different "mammal" species.

To better align the WordNet hierarchy with the task of object recognition in general, and BREEDS benchmarks in particular, we manually modify it according to the following two principles: (i) nodes should be grouped together based on their visual characteristics rather than abstract relationships like functionality, and (ii) nodes of similar specificity should be at the same distance from the root, irrespective of how detailed their categorization within WordNet is. Details of this procedure along with the resulting hierarchy are presented in Appendix C.1.4.

3.1.3 ImageNet-based BREEDS tasks

Once the modified version of the WordNet hierarchy is in place, BREEDS tasks can be created in an automated manner. Specifically, we first choose the desired granularity of the task by specifying the distance from the root ("entity") and retrieving all superclasses at that distance in a top-down manner. Each resulting superclass corresponds to a subtree of our hierarchy, with ImageNet classes as its leaves. Note that these superclasses are roughly of the same specificity, due to our hierarchy restructuring process. Then, we randomly sample a fixed number of subclasses for each superclass to produce a balanced dataset (omitting superclasses with an insufficient number of subclasses). Finally, as described in Section 3.1, we randomly split these subclasses into the source and target domain. [2]

For our analysis, we create four tasks (cf. Table 3.2) based on different levels/parts of the hierarchy. To illustrate what the corresponding subpopulation shifts look like, we present (random) image samples for a subset of the tasks in Figure 3.3. Note that while we focus on the tasks in Table 3.2 in our study, our methodology readily enables us to create other variants of these tasks in an automated manner.

Name	Subtree	Level	Subpops	Examples
ENTITY-13	"entity" (root)	3	20	"mammal", "appliance"
ENTITY-30	"entity" (root)	4	8	"fruit", "carnivore"
LIVING-17	"living thing"	5	4	"ape", "bear"
NON-LIVING-26	"non-living thing"	5	4	"fence", "ball"

Table 3.2: BREEDS benchmarks constructed using ImageNet. Here, "level" indicates the depth of the superclasses in the class hierarchy (task granularity), and the number of "subpopulations" (per superclass) is fixed to create balanced datasets. We also construct specialized tasks by focusing on subtrees in the hierarchy, e.g., only living (LIVING-17) or non-living (NON-LIVING-26) objects. Datasets naming reflects the root of the subtree and the number of superclasses they contain.

BREEDS benchmarks beyond ImageNet. It is worth nothing that the methodology we described is not restricted to ImageNet and can be readily applied to other datasets as well. The only requirement is that we have access to a semantic grouping of the dataset classes, which is the case for many popular vision datasets—e.g., CIFAR-100 [Kri09], Pascal-VOC [Eve+10], OpenImages [Kuz+18], COCO-Stuff [CUF18]. Moreover, even when a class hierarchy is entirely absent, the needed semantic class grouping can be manually

[2] We also consider more benign or adversarial subpopulation splits for these tasks in Appendix C.2.2.

Figure 3.3: Sample images from random object categories for the ENTITY-13 and LIVING-17 tasks. For each task, the top and bottom row correspond to the source and target distributions respectively.

constructed with relatively little effort (proportional to the number of classes, not the number of datapoints).

More broadly, the methodology of utilizing existing dataset annotations to construct data subpopulations goes beyond image classification tasks. In particular, by splitting inputs into a source and target domain based on some attribute, we can measure how well models generalize along this axis. Examples would include grouping by brand in Amazon reviews [McA+15], by location in Berkeley DeepDrive [Yu+20], and by facial attributes in CelebA [Liu+15].

3.1.4 Calibrating BREEDS benchmarks via human studies

For a distribution shift benchmark to be meaningful, it is essential that the source and target domains capture the same high-level task—otherwise generalizing from one domain to the other would be impossible. To ensure that this is the case for the BREEDS task, we assess how significant the resulting distribution shifts are for human annotators (crowd-sourced via MTurk).

Annotator task. To obtain meaningful performance estimates, it is crucial that annotators perform the task based only *on the visual content of the images*, without leveraging prior knowledge. To achieve this, we design the following annotation task. First, annotators are shown images from the source domain, grouped by superclass, without being aware of the superclass name (i.e., the grouping it corresponds to). Then, they are presented with images from the target domain and are asked to assign each of them to one of the groups. For simplicity, we present two random superclasses at a time, effectively simulating binary classification. Annotator accuracy can be measured directly as the fraction of images that they assign to the superclass to which they belong. We perform this experiment for each of the BREEDS tasks constructed in Section 3.1.3. For comparison, we repeat this experiment

without subpopulation shift (test images are sampled from the source domain) and for the superclasses constructed by Huh, Agrawal, and Efros [HAE16] using the WordNet hierarchy directly (cf. Appendix C.1.6).

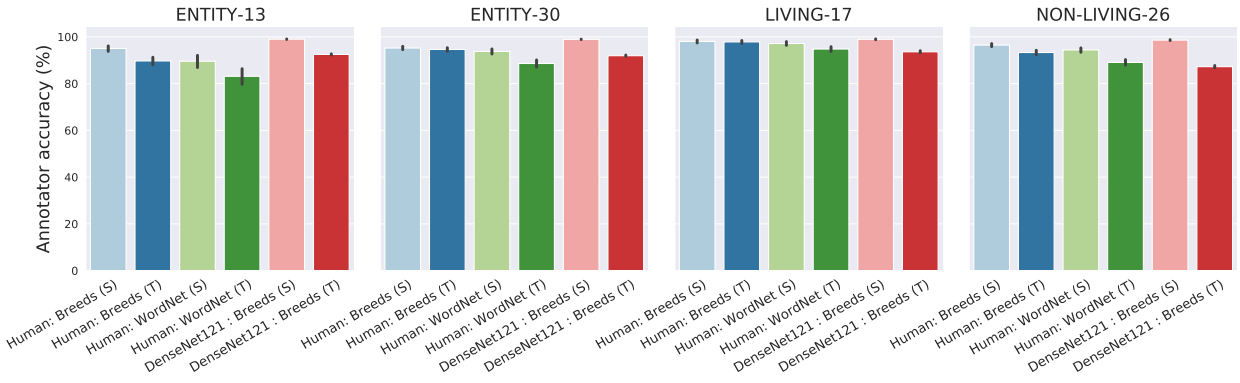


Figure 3.4: Human performance on (binary) BREEDS tasks. Annotators are provided with labeled images from the source distribution for a *pair of (undisclosed) superclasses*, and asked to classify samples from the target domain ('T') into one of the two groups. As a baseline we also measure annotator performance without subpopulation shift (i.e., on test images from the source domain, 'S') and tasks created via the WordNet hierarchy (cf. Appendix C.1.6). We observe that annotators are fairly robust to subpopulation shift. Further, they consistently perform better on BREEDS task compared to those based on WordNet directly—indicating that our modified class hierarchy is indeed better calibrated for object recognition. (We discuss model performance in Section 3.1.5.)

Human performance. We find that, across all tasks, annotators perform well on unseen data from the source domain, as expected. More importantly, annotators also appear to be quite robust to subpopulation shift, experiencing only a small accuracy drop between the source and target domains (cf. Figure 3.5). This indicates that the source and target domains are indeed perceptually similar for humans, making these benchmarks suitable for studying model robustness. Finally, across all benchmarks, annotators perform better on BREEDS tasks, compared to their WordNet equivalents—even on source domain samples. This indicates that our modified class hierarchy is indeed better aligned with the underlying visual recognition task.

3.1.5 Model performance under subpopulation shift

We can now use our suite of BREEDS tasks as a testbed for assessing model robustness to subpopulation shift Specifics of the evaluation setup and additional experimental results are provided in Appendices C.1.7 and C.2.2.

We start by evaluating the performance of various model architectures trained in the standard fashion: empirical risk minimization (ERM) on the source distribution (cf. Appendix C.1.7). While models perform well on unseen inputs from the domain they are trained on, i.e., they achieve high *source accuracy*, their accuracy considerably drops under subpopulation shift—more than 30% in most cases (cf. Figure 3.5). At the same time, models that are more *accurate* on the source domain also appear to be more *robust* to subpopulation shift. Specifically, the fraction of source accuracy that is preserved in the target domain typically increases with source accuracy. (If this were not the case, i.e., the model accuracy dropped by a constant fraction under distribution shift, the target accuracy would match the baseline in Figure 3.5.) This indicates that, improvements in source accuracy *do* correlate with models generalizing better to variations in testing conditions.

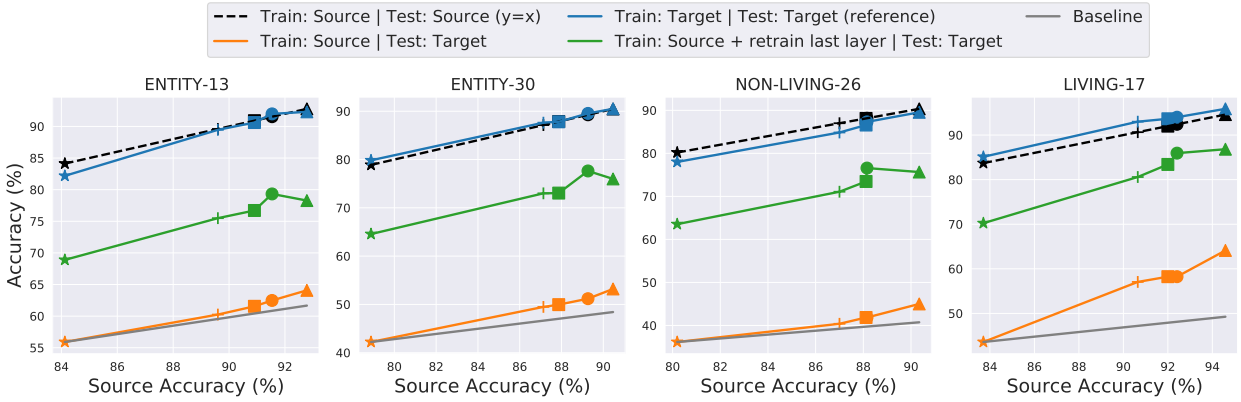


Figure 3.5: Robustness of standard models to subpopulation shifts. For each task, we plot the accuracy of various model architectures (denoted by different symbols) on the target domain as a function of their source accuracy. We find that model accuracy drops significantly between domains (*orange* vs. *dashed* line). Still, models that are more accurate on the source domain seem to also be more robust (the improvements exceed the baseline (*grey*) which would correspond to a constant accuracy drop relative to AlexNet). Moreover, the drop in model performance can be significantly (but not fully) reduced by retraining the final model layer with data from the target domain (*green*).

Models vs. humans. We compare the best performing model (DenseNet-121 in this case) to our previously obtained human baselines in Figure 3.4. To allow for a fair comparison, model accuracy is measured on pairwise superclass classification tasks (cf. Appendix C.1.7). We observe that models do exceedingly well on unseen samples from the source domain—significantly outperforming annotators under our task setup. At the same time, models also appear to be more brittle, performing worse than humans on the target domain of these binary BREEDS tasks, despite their higher source accuracy.

Adapting models to the target domain. Finally, we focus on the intermediate data representations learned by these models, to assess how suitable they are for distinguishing classes in the target domain. To evaluate this, we retrain the last (fully-connected) layer of models trained on the source domain with data from the target domain. We find that the target accuracy of these models increases significantly after retraining, indicating that the learned representations indeed generalize to the target domain. However, we cannot match the accuracy of models trained directly (end-to-end) on the target domain—see Figure 3.5—demonstrating that there is significant room for improvement.

3.2 Concept-level transformations

Our goal in this section is to evaluate model performance when individual, high-level concept are transformed. Intuitively, we want to understand questions such as: how is the prediction of the model affected if there is snow on the road?

Specifically, we will build a pipeline that 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 robust the model's predictions are when high-level concepts in the image are transformed. For example, to understand whether the model is robust to changes in the "wheel" when recognizing a "car" in an image, we will evaluate it on the same image with a different (transformed) "wheel". We now discuss our approach to generate such counterfactuals (Section 3.2.1) and then describe how they can be used to evaluate the models robustness (Section 3.2.2). Our complete pipeline is illustrated in Figure 3.6 (see Appendix C.3 for additional details).

3.2.1 Synthesizing concept-level counterfactuals

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. So, instead, we leverage pre-trained instance segmentation

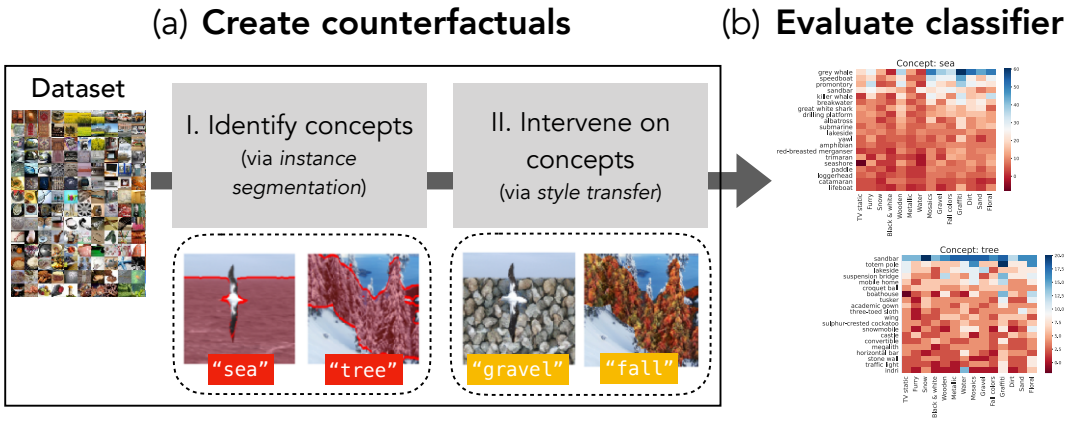


Figure 3.6: Concept-based robustness pipeline. To pinpoint the robustness of a model to transformations of high-level concepts, we (a) synthesize counterfactual images in which a given concept in the image (detected via instance segmentation) is modified (via style transfer). Then (b) the impact of this 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".

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 toautomatically synthesize counterfactuals with realistic *concept-level transformations*, such as "snowy road" or "wooden wheels". Note that this process does not require *any* annotation effort and can thus be directly applied to new datasets.

3.2.2 Probing model robustness 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 3.7a, 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; RSG16; 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 C.12), and across architectures for a single concept (Appendix C.4). Finally, we can also examine the effect of the style used to transform a concept—cf. Figure 3.7b.

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 3.7c and Appendix Figure C.14). We can also examine which concept-level transformations hurt model performance the most—e.g., we find that making "plants" "floral" hurts accuracy on the class "damselfly" 15% more than making the "plants" "snowy".

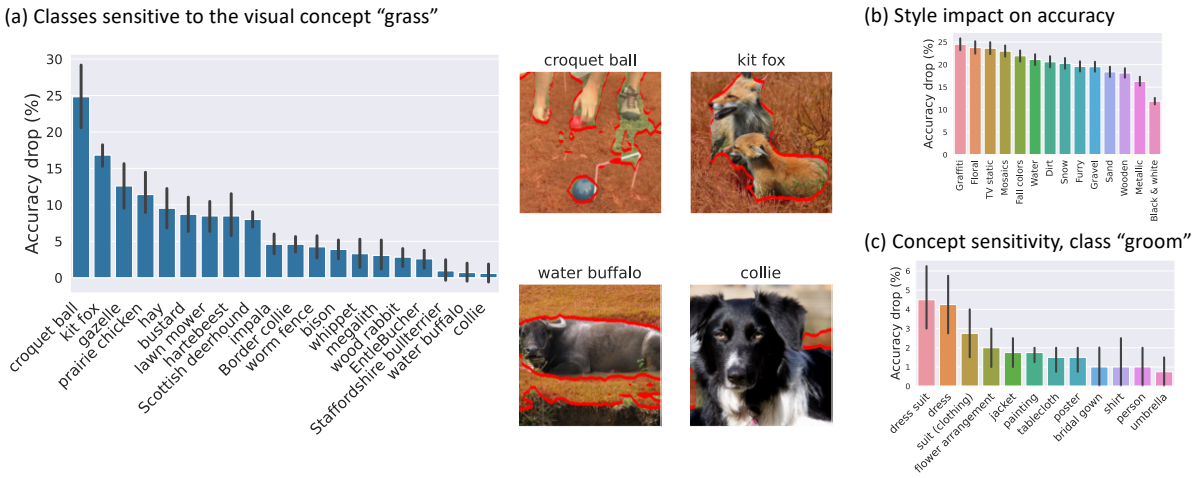


Figure 3.7: Model sensitivities diagnosed using our pipeline in a VGG16 classifier trained on ImageNet. (a) The accuracy drop induced by transformations of the concept “grass” highlights classes for which the model relies on this concept: e.g., a “croquet ball” is not accurately recognized if “grass” 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.”

Chapter 4

Improving robustness: Finding the right features

Our goal in this chapter will be to explore ways in which we can train our models to be more reliable in the face of real-world deployment challenges. Unfortunately, as we discussed in Chapter 3, these challenges are impossible to capture within a concrete mathematical model. Thus, we cannot rely on methods such as robust optimization (as we did in Chapter 1) since we cannot precisely formulate a desired robustness guarantee.

Instead, we will focus on biasing our models towards features that generalize better. The key intuition here is that, while many different input signals can be equally predictive of the correct output during training, only some of these signals might be predictive during deployment. As an example, under typical conditions, a model can recognize a car based on the wheels, the body, or the presence of a road. However, if, during deployment, a car is encountered in a showroom, a model relying on the presence of a road might have trouble recognizing it.

Chapter structure

- In Section 4.1, we will study how different synthetic perturbations used during training can affect the ability of a model to generalize across subpopulations.

- In Section 4.2, we will leverage a pool of unlabeled data and explore how we can use models with diverse feature priors to extract prediction rules that generalize better.

4.1 Robustness to synthetic transformations

In Chapter 2, we saw how training models to be robust to small, worst-case perturbations had a significant effect on the input features they utilized. In this section, we will examine whether such robust features are more suitable for generalizing across subpopulation while also expanding our study to new types of robustness interventions. Concretely, we consider the following families of interventions in the context of the BREEDS benchmarks developed in Chapter 3 (cf. Appendix D.1 for details):

- **Adversarial training**: Enhances robustness to worst-case ℓ_p-bounded perturbations (in our case ℓ_2) by training models against a projected gradient descent (PGD) adversary (cf. Chapter 1).
- **Stylized Training**: Encourages models to rely more on shape rather than texture by training them on a stylized version of ImageNet [Gei+19].
- **Random noise**: Improves model robustness to data corruptions by incorporating them as data augmentations during training—we focus on Gaussian noise and Erase noise [Zho+20], i.e., randomly obfuscating a block of the image.

Relative accuracy. To measure the impact of these interventions, we will focus on the models' *relative accuracy*—the ratio of target accuracy to source accuracy. This metric accounts for the fact that train-time interventions can impact model accuracy on the source domain itself. By measuring relative performance, we are able to compare different training methods on an equal footing.

We find that robustness interventions *do* have a small, yet non-trivial, impact on the robustness of a particular model architecture to subpopulation shift—see Figure 4.1. Specifically, for the case of adversarial training and erase noise, models often retain a larger fraction of their accuracy to the target domain compared to standard training, hence lying on the Pareto frontier of a robustness-accuracy trade-off. In fact, for some of the models trained with these interventions, the target accuracy is slightly higher than models obtained via standard training, even without adjusting for their lower source accuracy (raw accuracies for all methods are in Appendix D.1.2). Nonetheless, it is important to note that none of these method offer significant subpopulation robustness—relative accuracy is not improved by more than a few percentage points.

Adapting models to the target domain. The impact of these interventions is more pronounced if we consider the target accuracy of these models after their last layer has been

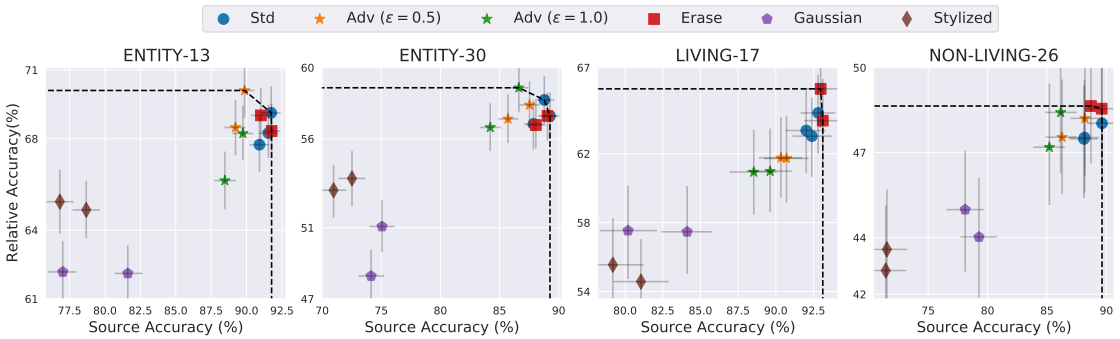


Figure 4.1: Effect of train-time interventions on model robustness to subpopulation shift. We measure model performance in terms of *relative accuracy*–i.e., the ratio between its target and source accuracies. This allows us to visualize the accuracy-robustness trade-off along with the corresponding Pareto frontier (*dashed*). (Also shown are 95% confidence intervals computed via bootstrapping.) We observe that some of these interventions do improve model robustness to subpopulation shift by a small amount—specifically, erase noise and adversarial training—albeit sometimes at the cost of source accuracy.

retrained on data from the target domain—see Figure 4.2. In particular, we observe that for adversarially robust models, retraining significantly boosts accuracy on the target domain—e.g., in the case of LIVING-17 it is almost comparable to the initial accuracy on the source domain. This indicates that the feature priors imposed by these interventions incentivize models to learn representations that generalize better to similar domains—in line with recent results of Utrera et al. [Utr+20] and Salman et al. [Sal+20]. Moreover, we observe that models trained on the stylized version of these datasets perform consistently worse, suggesting that texture might be an important feature for these tasks, especially in the presence of subpopulation shift. Finally, note that we did not perform an exhaustive exploration of the hyper-parameters used for these interventions (e.g., ℓ_2-norm)—it is possible that these results can be improved by additional tuning. For instance, we would expect that we can tune the magnitude of the Gaussian noise to achieve performance that is comparable to that of ℓ_2-bounded adversarial training [For+19].

4.2 Combining distinct feature priors

In this section, we will explore whether models that depend on different sets of features can improve each other. Specifically, we will treat models trained with different feature priors as diverse perspectives on the data and use them to extract more reliable prediction rules from a pool of unlabeled data.

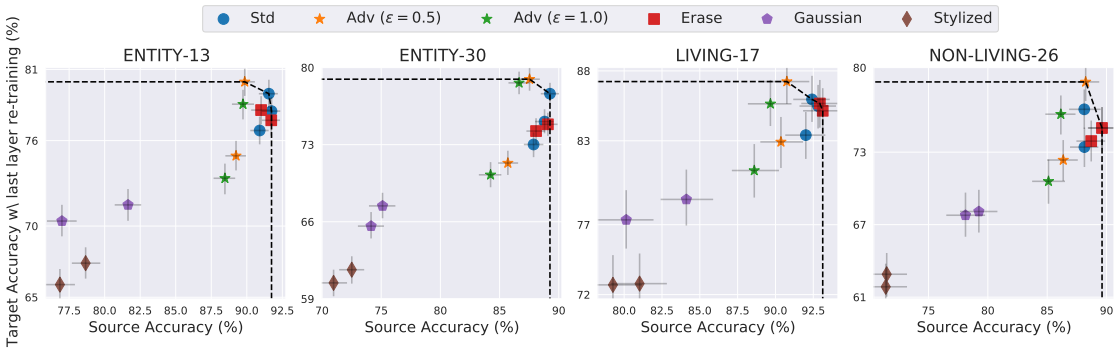


Figure 4.2: Target accuracy of models after they have been retrained (only the final linear layer) on data from the target domain (with 95% bootstrap confidence intervals). Models trained with robustness interventions often have higher target accuracy than standard models post retraining.

4.2.1 Feature priors as different perspectives

We will start by studying to what extent models with different feature priors can actually provide distinct perspectives on the data. To do so we will focus on a pair of feature priors that arise naturally in the context of image classification: shape and texture.

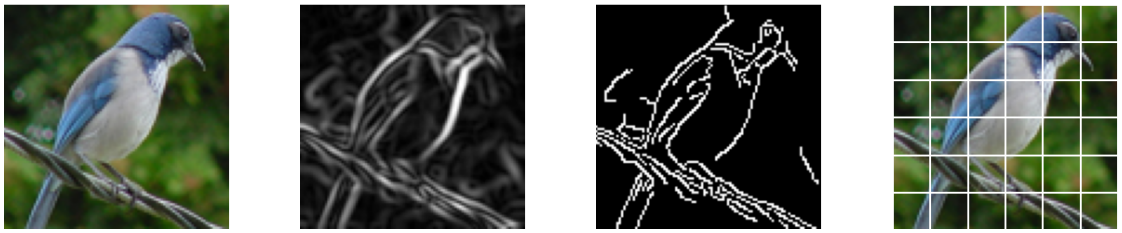
Training shape- and texture-biased models

In order to train shape- and texture-biased models, we either pre-process the model input or modify the model architecture. Specifically:

Shape-biased models. To suppress texture information in the images, we pre-process our inputs by applying an edge detection algorithm. We consider two canonical edge detection algorithms from the computer vision literature: the *Canny* edge detection algorithm [DG01] which produces a binary edge mask, and the *Sobel* edge detection algorithm [SF68] which provide a softer edge detection, hence retaining some texture information (see Figures 4.3b and 4.3c).

Texture-biased models. To prevent the model from relying on the global structure of the image we utilize a variant of the *BagNet* architecture [BB19]. This architecture deliberately limits the receptive field of the model, thus forcing it to make predictions based on local features (see Figure 4.3d).

We visualize all of these priors in Figure 4.3 and provide further implementation details in Appendix D.2.



(a) Original (b) Sobel (c) Canny (d) BagNet

Figure 4.3: Visualizing different feature priors: (a) an image from the STL-10 dataset; (b) Sobel edge detection; (c) Canny edge detection; (d) the limited receptive field of a BagNet.

	CIFAR-10				STL-10			
	Standard	Canny	Sobel	BagNet	Standard	Canny	Sobel	BagNet
Standard	0.598	0.237	0.259	0.38	0.554	0.305	0.385	0.357
Canny		0.545	0.324	**0.143**		0.523	0.392	**0.212**
Sobel			0.594	0.173			0.649	0.262
BagNet				0.655				0.486

Table 4.1: Correlation (Pearson coefficient) of correct predictions on the test set between different pairs of models. The diagonal entries correspond to models trained with the same prior but from different random initializations. While the two shape-biased models (Sobel and Canny) are more aligned with each other, they are both quite different from the texture-biased model (BagNet).

Diversity of feature-biased models

After training models with shape and texture biases as described above, we evaluate whether these models capture complementary information about the input. Specifically, we study models trained on a small subset (100 examples per class) of the CIFAR-10 [Kri09] and STL-10 [CNL11] datasets, and measure the correlation between which test examples they correctly classify.

We find that pairs consisting of a shape-biased model and a texture-biased model (i.e Canny/BagNet or Sobel/BagNet) indeed have the least correlated predictions—cf. Table 4.1. In other words, the mistakes that these models make are significantly more diverse than those made by identical models trained from different random initializations. At the same time, different shape-biased models (Sobel and Canny) are relatively well-correlated with each other, which corroborates that models trained on similar features of the input are likely to make similar mistakes.

Model ensembles. Having shown that training models with these feature priors results in diverse prediction rules, we can now combine them to improve our generalization. The canonical approach for doing so is to incorporate these models into an ensemble.

We find that the diversity of models trained with different feature priors directly translates into an improved performance when combining them into an ensemble—cf. Table 4.4. In fact, we find that the performance of the ensemble is tightly connected to prediction similarity of its constituents (as measured in Table 4.1), i.e., more diverse ensembles tend to perform better. For instance, the best ensemble for the STL-10 dataset is the one combining a shape-biased (Canny) and a texture-biased model (BagNet) which were the models with the least aligned predictions.

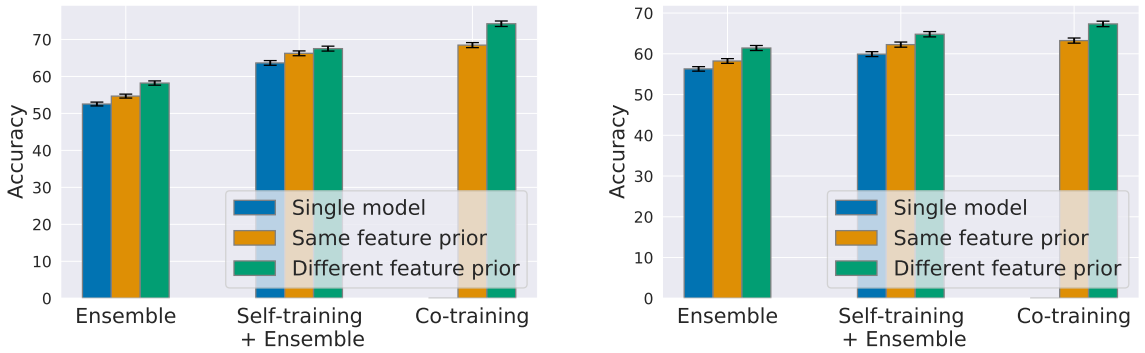
	Feature Priors	Model 1	Model 2	Ensemble
Same	Standard + Standard	52.54 ± 0.86	51.82 ± 0.86	54.02 ± 0.80
	Sobel + Sobel	51.94 ± 0.84	53.69 ± 0.82	54.68 ± 0.83
	BagNet + BagNet	42.22 ± 0.88	42.56 ± 0.80	43.49 ± 0.83
Different	Standard + Sobel	52.54 ± 0.83	51.94 ± 0.83	**58.21 ± 0.82**
	Standard + BagNet	52.54 ± 0.84	42.22 ± 0.84	53.03 ± 0.81
	Sobel + BagNet	51.94 ± 0.90	42.22 ± 0.84	55.14 ± 0.81

(a) CIFAR-10

	Feature Priors	Model 1	Model 2	Ensemble
Same	Standard + Standard	53.73 ± 0.91	55.38 ± 0.88	57.06 ± 0.91
	Canny + Canny	56.29 ± 0.96	54.99 ± 0.96	58.23 ± 0.93
	BagNet + BagNet	52.04 ± 0.98	50.34 ± 0.94	53.42 ± 0.93
Different	Standard + Canny	53.73 ± 0.95	56.29 ± 0.91	**60.96 ± 0.96**
	Standard + BagNet	53.73 ± 0.98	52.04 ± 0.90	57.17 ± 0.90
	Canny + BagNet	56.29 ± 0.91	52.04 ± 0.95	**61.42 ± 0.92**

(b) STL-10

Table 4.4: Ensemble accuracy when combining models trained with a diverse set of feature priors (models with the same prior are trained from different random initialization). Notice how models trained with different priors lead to ensembles with better performance. Moreover, when the accuracy of the two base models is comparable, models that are more diverse (as measured in Table 4.1) result in better ensembles. We describe the different methods of combining models in Appendix D.2.4 and provide the full results in Appendix D.2.7.



(a) CIFAR-10 (b) STL-10

Figure 4.5: Test accuracy of pre-trained, self-trained, and co-trained models selecting the best feature prior for each (full results in Table 4.4, Appendix Table D.9, and Table 4.2 respectively). Notice how combinations of models with different feature priors consistently outperform combinations of models with the same feature prior.

4.2.2 Combining diverse priors on unlabeled data

In the previous section, we saw that training models with different feature priors (e.g., shape- and texture-biased models) can lead to prediction rules with less overlapping failure modes—which, in turn, can lead to more effective model ensembles. However, ensembles only combine model predictions post hoc and thus cannot take advantage of diversity during the training process.

In this section, we instead focus on utilizing diversity *during* training. Specifically, we will leverage the diversity introduced through these priors in the context of self-training [Lee+13]: a framework commonly used when the labeled data is insufficient to learn a well-generalizing model. This framework utilizes unlabeled data, which are then pseudo-labeled using an existing model and used for further training. While such methods can often improve the overall model performance, they suffer from a significant drawback: models tend to reinforce suboptimal prediction rules even when these do not generalize to the underlying distribution [Ara+20].

Our goal here is to leverage diverse feature priors to address this exact shortcoming. Specifically, we will *jointly* train models with different priors on the unlabeled data through the framework of co-training [BM98]. Since these models capture complementary information about the input (cf. Table 4.1), we expect them to correct each other's mistakes and improve their prediction rules. As we will see in this section, this approach can indeed have a significant impact on the performance of the resulting model, outperforming ensembles that combine such models only at evaluation time—see summary in Figure 4.5.

Setup. We base our analysis on the CIFAR-10 and STL-10 datasets. Specifically, we treat a small fraction of the training set as *labeled* examples (100 examples per class), another fraction as our validation set for tuning hyperparameters (10% of the total training examples), and the rest of the training set as our *unlabeled* data. We report our results on the standard test set of each dataset. (See Appendix D.2 for a full description of our setup and training process.)

Self-training and ensembles

Before outlining our method for jointly training models with multiple priors on unlabeled data, we first describe the standard approach for self-training a single model. At a high level, the predictions of the model on the unlabeled data are treated as correct labels and are then used to re-train the same model [Lee+13; Isc+19; Zou+19; Xie+20]. The underlying intuition is that the classifier will predict the correct labels for that data better than chance, and thus these *pseudo-labels* can be used to expand the training set.

In practice, however, these pseudo-labels tend to be noisy. Thus, a common approach is to only use the labels to which the model assigns the highest probability [Lee+13]. This process is repeated, self-training on increasingly larger fractions of the unlabeled data until all of it is used. We refer to each such training phase as an *era*.

Ensembles of diverse self-trained models. Similarly to our results in Table 4.4, we find that ensembles comprised of self-trained models with diverse feature priors outperform those that use the same prior from different random initializations (see Figure 4.5 for a summary and Appendix D.2.8 for the full results). This demonstrates that, after self-training, these models continue to capture complementary information about the input that can be leveraged to improve performance.

Co-training models with different feature priors

Moving beyond self-training with a single feature prior, our goal in this section is to leverage multiple feature priors by jointly training them on the unlabeled data. This idea naturally fits into the framework of *co-training*: a method used to learn from unlabeled data when the inputs correspond to multiple independent sets of features [BM98]. In this framework, each set of features is used to train a different model. Then, these models are used to pseudo-label the unlabeled data, from which the most confident examples are added to a common pool to be used for further training.

Concretely, we first train a model for each feature prior. Then, we combine the pseudo-labels on the unlabeled data that were assigned the highest probability for each model—including duplicates with potentially different labels—to form a new training set which we use for further training. Similarly to the self-training case, we repeat this process over several eras, increasing the fraction of the unlabeled dataset used with each era. Intuitively, this iterative process allows the models to bootstrap off of each other's predictions, learning correlations that they were unable to learn from the labeled data alone. At the end of this process, we are left with two models, one for each prior used during co-training, which we combine into a single classifier by training a standard model from scratch on the combined pseudo-labels. We provide a more detailed explanation of the methodology in Appendix D.2.5.

Co-training performance. We find that co-training with shape- and texture-based priors can significantly improve the test accuracy of the final model compared to self-training with any of the priors alone (Table 4.2). This is despite the fact that, when using self-training alone, the standard model outperforms all other models (Column 4, Table 4.2). Moreover, co-training models with diverse priors improves upon simply combining them in an ensemble (Appendix D.2.8).

In Appendix D.2.9, we report the performance of co-training with every pair of priors. We find that co-training with shape- and texture-based priors together (Canny + BagNet for STL-10 and Sobel + BagNet for CIFAR-10) outperform every other prior combination. Note that this is the case even though, when only ensembling models with different priors (c.f Table 4.4 and Appendix D.2.8), Standard + Sobel is consistently the best performing pair for CIFAR-10. Overall, these results indicate that the diversity of shape- and texture-biased models allows them to improve each other over training.

Additionally, we find that, even when training a single model on the pseudo-labels of another model, prior diversity can help. Specifically, we compare the performance of a standard model trained from scratch using pseudo-labels from various self-trained models (Column 5, Table 4.2). In this setting, using a self-trained shape- or texture-biased model for pseudo-labeling outperforms using a self-trained standard model. This is despite the fact that, in isolation, the standard model has higher accuracy than the shape- or texture-biased ones (Column 4, Table 4.2).

Model alignment over co-training. To further explore the dynamics of co-training, we evaluate how the correlation between the predictions of the two models evolves as the eras progress in Figure 4.6 (using the same measure of prediction alignment as in Table 4.1).

Methods	Prior(s)	Labeled Only	+Unlabeled Self/Co-Training	+ Standard model with Pseudo-labels
Self-training	Standard	52.54 ± 0.81	63.65 ± 0.78	64.02 ± 0.79
	Sobel	51.94 ± 0.90	63.05 ± 0.85	64.77 ± 0.81
	BagNet	42.22 ± 0.81	53.92 ± 0.84	54.21 ± 0.81
Co-training	Standard	52.54 ± 0.83	65.06 ± 0.78	65.10 ± 0.79
	+Standard	51.82 ± 0.79	64.93 ± 0.83	
	Sobel	51.94 ± 0.82	$\mathbf{71.88 \pm 0.76}$	$\mathbf{74.25 \pm 0.75}$
	+BagNet	42.22 ± 0.80	$\mathbf{73.91 \pm 0.73}$	

(a) CIFAR-10

Methods	Prior(s)	Labeled Only	+Unlabeled Self/Co-Training	+ Standard model with Pseudo-labels
Self-training	Standard	53.73 ± 0.94	59.92 ± 0.93	60.52 ± 0.91
	Canny	56.29 ± 0.92	58.40 ± 0.89	62.19 ± 0.91
	BagNet	52.04 ± 0.92	57.80 ± 0.99	61.69 ± 0.96
Co-training	Standard	53.73 ± 0.94	58.05 ± 0.95	61.16 ± 0.94
	+Standard	55.38 ± 0.92	60.44 ± 0.92	
	Canny	56.29 ± 0.94	$\mathbf{62.21 \pm 0.93}$	$\mathbf{67.33 \pm 0.89}$
	+BagNet	52.04 ± 1.00	$\mathbf{66.74 \pm 0.94}$	

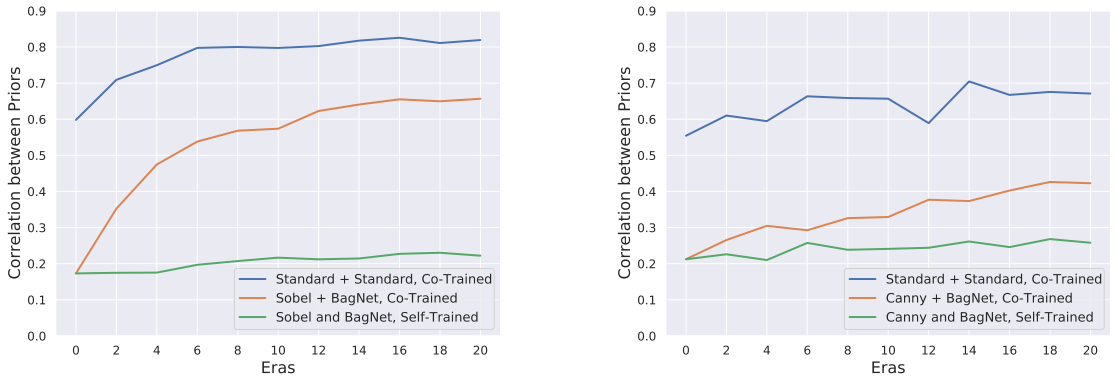
(b) STL-10

Table 4.2: Test accuracy of self-training and co-training methods on STL-10 and CIFAR-10. For each model, we report the original accuracy when trained only labeled data (Column 3) as well as the accuracy after being trained on pseudo-labeled data (Column 4). (Recall that, for the case of co-training pseudo-labeling is performed by combining the predictions of both models.) Finally, we report the performance of a standard model trained from scratch on the resulting pseudo-labels (Column 5). We provide 95% confidence intervals computed via bootstrap with 5000 iterations.

We find that the shape- and texture-biased models exhibit low correlation at the start of co-training, but this correlation increases as co-training progresses. This is in contrast to the case of self-training each model on its own, where the correlation remains relatively low. It is also worth noting that the correlation appears to plateau at a lower value when co-training models with distinct feature priors as opposed to co-training two standard models.

Finally, we find that a standard model trained on the pseudo-labels of other models correlates well with the models themselves (see Appendix D.2.10). Overall, these findings

indicate that models trained on each other's pseudo-labels end up behaving more similarly.



(a) CIFAR-10 (b) STL-10

Figure 4.6: Correlation between the correct predictions of shape- and texture-biased models over the course of co-training for STL-10 and CIFAR-10. For comparison, we also plot the correlation between the predictions when the models induced by these priors are individually self-trained, as well as the correlation of two standard models when co-trained together.

4.2.3 Using co-training to avoid spurious correlations

A major challenge when training models for real-world deployment is avoiding spurious correlations: associations which are predictive on the training data but not valid for the actual task. Since models are typically trained to minimize loss on the training set, they are quite likely to rely on such spurious correlations [Gur+18; BVP18; Gei+20; Xia+20].

In this section, our goal is to leverage diverse feature priors to control the sensitivity of the training process to such spurious correlations. Specifically, we will assume that the spurious correlation does not hold on the unlabeled data (which is likely since unlabeled data is often collected through a more diverse process). As we will see, if the problematic correlation is not easily captured by one of the priors, the corresponding model generates pseudo-labels that are inconsistent with this association, thus steering other models away from this correlation during co-training.

Setup. We study spurious correlations in two settings. First, we create a synthetic dataset by tinting each image of the STL-10 labeled dataset in a class-specific way. This encourages models to predict using the tint, as it is highly predictive on the training set. At the same time, such a prediction rule does not generalize to the test set where

this correlation is absent. Second, we consider a real-world gender classification task based on CelebA [Liu+15] where hair color ("blond" vs. "non-blond") is predictive on the labeled data but not on the unlabeled and test data. Specifically, gender and hair color are independent attributes on the unlabeled dataset, while the labeled dataset consists only of blond females and non-blond males. Similarly to the synthetic case, the labeled data encourages a prediction rule based only on hair color. See Appendix D.2.1 for details.

Methods	Prior(s)	Labeled Only	+Unlabeled Self/Co-Training	+ Standard model with Pseudo-labels
Self-training	Standard	13.99 ± 0.66	17.56 ± 0.70	17.81 ± 0.74
	Canny	55.95 ± 0.92	57.31 ± 0.89	57.81 ± 0.92
	Sobel	55.11 ± 0.91	56.12 ± 0.92	57.16 ± 0.91
	BagNet	13.10 ± 0.64	13.53 ± 0.62	14.65 ± 0.66
Co-training	Canny	55.95 ± 0.90	57.74 ± 0.90	57.85 ± 0.95
	+BagNet	13.10 ± 0.65	**55.33 ± 0.92**	
	Sobel	55.11 ± 0.95	57.71 ± 0.90	57.60 ± 0.94
	+BagNet	13.10 ± 0.62	**54.61 ± 0.94**	

(a) Tinted STL-10

Methods	Prior(s)	Labeled Only	+Unlabeled Self/Co-Training	+ Standard model with Pseudo-labels
Self-training	Standard	67.07 ± 0.58	71.57 ± 0.53	71.89 ± 0.53
	Canny	80.90 ± 0.47	85.73 ± 0.40	86.55 ± 0.42
	Sobel	82.94 ± 0.45	85.42 ± 0.43	84.96 ± 0.43
	BagNet	69.35 ± 0.55	64.89 ± 0.59	66.15 ± 0.58
Co-training	Canny	80.90 ± 0.46	**89.64 ± 0.36**	**91.99 ± 0.31**
	+BagNet	69.35 ± 0.55	**91.44 ± 0.33**	
	Sobel	82.94 ± 0.44	**90.64 ± 0.35**	**90.99 ± 0.34**
	+BagNet	69.35 ± 0.57	**88.72 ± 0.39**	

(b) CelebA

Table 4.3: Test accuracy of self-training and co-training on tinted STL-10 and CelebA, two datasets with spurious features (table structure is identical Table 4.2). In both datasets, the spurious correlation is more easily captured by the BagNet and Standard models over the shape-based ones. Nevertheless, when co-trained with a shaped-biased model, BagNets are able to significantly improve their performance, indicating that they rely less on this spurious correlation. CI: 95% bootstrap.

Performance on datasets with spurious features. We find that, when trained only on the labeled data (where the correlation is fully predictive), both the standard and BagNet models generalize poorly in comparison to the shape-biased models (see Table 4.3). This behavior is expected: the spurious attribute in both datasets is color-related and hence mostly suppressed by the edge detection algorithms used to train shape-based models. Moreover, even after self-training on the unlabeled data (where the correlation is absent), the performance of the standard and BagNet models does not improve significantly. Finally, simply ensembling self-trained models post hoc does not improve their performance. Since the texture-biased and standard models are significantly less accurate than the shape-biased one, they end up lowering the overall accuracy of the ensemble (see Appendix D.2.11).

In contrast, when we co-train a texture-biased model with a shape-biased one, the texture-biased model improves substantially. For instance, when co-trained with a Canny model, the resulting BagNet model outperforms its self-training performance by 42% on the tinted STL-10 dataset and 27% on the CelebA dataset. This improvement can be attributed to the fact that the predictions of the shape-biased model are not consistent with the spurious correlation on the unlabeled data. Hence, by being trained on pseudo-labels from that model, the BagNet model is forced to rely on alternative, non-spurious features for its predictions.

Moreover, particularly on the CelebA dataset, the shape-biased model also improves when co-trained with a texture-biased model. This indicates that even though the texture-biased model relies heavily on the spurious correlation, it also captures non-spurious features that, through pseudo-labeling, improve the performance of the shape-based model. In Appendix D.2.12, we find that these improvements are concentrated on input samples where the spurious correlation does not hold.

Chapter 5

Adapting models by rewriting their prediction rules

Unfortunately, despite our best efforts to build robust models, these models will eventually face deployment conditions that cause their performance to degrade. Such situations can arise when some of the correlations that a model learns are spurious, i.e., they are predictive on the training data but not in the real world. For instance, cows may not always stand on pastures, even if standard ML datasets indicate so. In fact, there is by now plenty of evidence that not all of the correlations that an ML model relies on are actually meaningful [TE11; BVP18; SSF19; ASF20; Xia+20; BVA20; Gei+20]. Consequently, in order to improve the reliability of the model, model designers might want their models to avoid such correlations. This raises the question:

How can we modify the way in which a given model makes its predictions?

The canonical approach for modifying a model post-hoc 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 rules from it [Pon+06; TE11; Tsi+20; Bey+20].

Our goal in this section our work is to develop a toolkit that enables users to *directly modify* the prediction rules learned by an (image) classifier (see Figure 5.1 for examples).

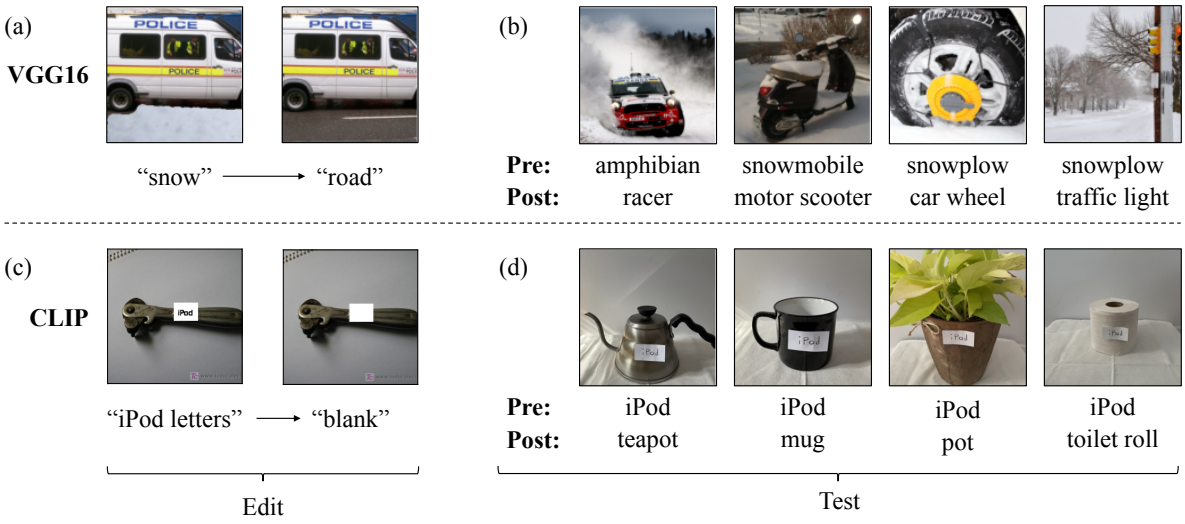


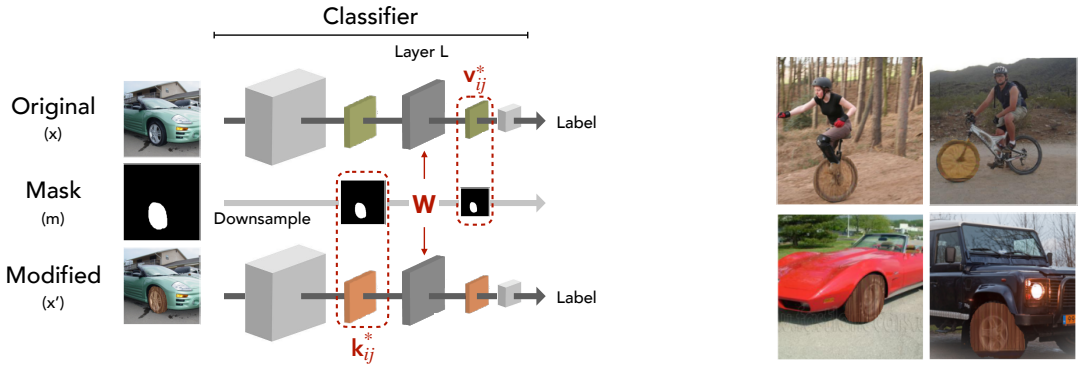
Figure 5.1: Editing prediction rules in pre-trained classifiers using a *single* exemplar. (a) We edit a VGG-16 ImageNet classifier to map the representation of the concept "snow" to that of "asphalt road". (b) This edit corrects systematic classification errors on snowy scenes for various ImageNet classes. (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].

Chapter structure

- In Section 5.1, we outline the approach of Bau et al. [Bau+20a] for editing generative models that will serve as the basis for our approach.
- In Section 5.2, we describe our methodology for directly rewriting the prediction rules of a classifier.
- In Section 5.3, we evaluate the performance of the method on the concept transformation benchmarks of Section 3.2.
- In Section 5.4, we construct two new benchmarks based on new, real-world images which we use to further evaluate our method.

5.1 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



(a) Edit (b) Test

Figure 5.2: 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 (5.1). (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".

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:

$$\min_{\Lambda} \quad \sum_{(i,j)\in S} \left\| v_{ij}^* - f(k_{ij}^*; W') \right\| \tag{5.1}$$

$$\text{s.t.} \quad W' = W + \Lambda (C^{-1} d)^\top. \tag{5.2}$$

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 (5.1) 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 Appendix E.1 and Bau et al. [Bau+20a] for additional details.

5.2 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 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 downsam-

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

pling the mask to the appropriate dimensions.) Finally, the actual edit is performed by feeding the resulting key-value pairs into the optimization problem (5.2) to determine the updated layer weights W'—see Figure 5.2 for an illustration of the overall process.

5.3 Does editing generalize?

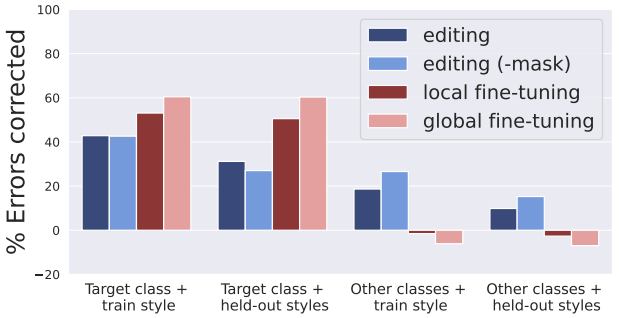
We now demonstrate how the methodology described above can be applied to edit vision classifiers—specifically, VGG [SZ15] and ResNet [He+15] models trained on the ImageNet [Den+09; Rus+15] and Places-365 [Zho+17] datasets (cf. Appendix E.1.2). Our focus will be on improving model generalization to the concept-level transformations from Section 3.2. Recall that our analysis in that section 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]

5.3.1 Evaluation setup

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 5.2) 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 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). Our baseline is 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-tuning (cf. Appendix E.1.3): (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. In each case, we select the best hyperparameters—including the choice of the layer to modify—based on the validation set performance (cf. Appendix E.1.4).

[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 5.3: Editing vs. fine-tuning, averaged over concept-style pairs. We find that 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.3-E.6 for other experimental settings.

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 5.2b). 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:

$$\frac{N_{pre}(D) - N_{post}(D)}{N_{pre}(D)} \tag{5.3}$$

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%). We found that the exact accuracy threshold did not have significant impact on the results—see Appendix Figures E.8-E.11 for the full accuracy-effectiveness trade-off.

5.3.2 The effectiveness of editing

Recall that we want our prediction-rule rewrites to generalize. That is, if we modify the way that our model treats a specific concept, we want 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 performance in Figure 5.3, we treat inputs belonging to the (target) class used to perform the modification separately.

Editing. We find that editing is able to consistently correct mistakes in a manner that *generalizes across classes*. 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. In Appendix E.2.3, we conduct ablation studies to get a better sense of the key algorithmic factors driving performance. Notably, we find imposing the editing constraints (5.1) on the entirety of the image—as opposed to only focusing on key-value pairs that correspond to the concept of interest—leads to even better performance (cf. '-mask' in Figure 5.3). We hypothesize that this has a regularizing effect as it constrains the weights to preserve the original mapping between keys and values in regions that do not contain the concept.

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 modification, 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—cf. Appendix Figures E.8-E.11. 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 5.2b). We present examples of errors (not) corrected by editing and fine-tuning in Appendix Figure E.7, and provide a per-concept/style break down in Appendix Figures E.12 and E.13.

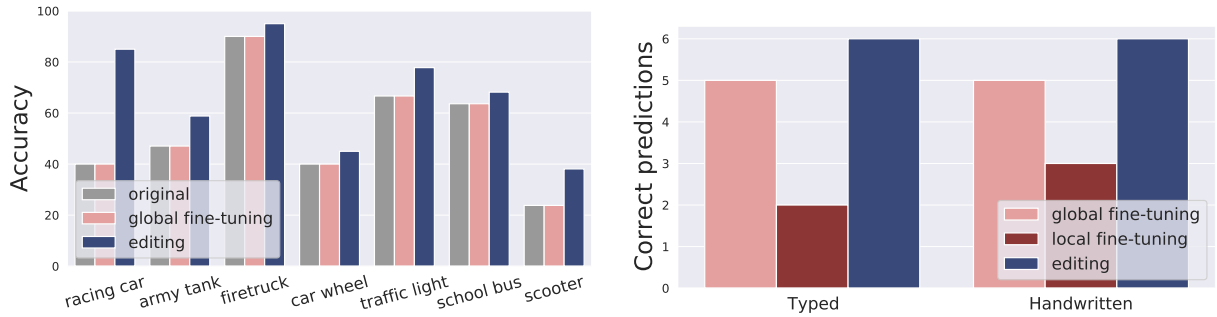
5.4 Real-world demonstrations

So far, we have seen that our rule rewriting methodology can significantly improve model generalization to concept-level transformations synthesized using our pipeline from 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 5.2), using this single snow-to-road exemplar—see Figure 5.1.

In Figure 5.4a, 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.

Ignoring a spurious feature: Typographic attacks. Our second use-case is modifying a model to ignore a spurious feature. We focus on the 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 Appendix Figure E.2 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 (Appendix Figure E.2); or an ImageNet image from of a "can opener" (randomly-chosen) with the typed text "iPod" programatically pasted on it (Figure 5.1). The original image x for our approach is then obtained by replacing the handwritten/typed text with a white mask—cf. Figure 5.1. We then use this single training exemplar to perform the



(a) Vehicles in snowy weather

(b) Typographic attacks

Figure 5.4: (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 hand-written 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.

network modification.

In both cases, we find that editing (Section 5.2) is able to fix *all* the errors caused by the typographic attacks, see Figure 5.4b. 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 (Appendix Figure E.20).

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 (cit. on pp. 14, 25, 120).

[Alc+19] Michael A Alcorn et al. "Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on p. 18).

[Ara+20] Eric Arazo et al. "Pseudo-labeling and confirmation bias in deep semi-supervised learning". In: *International Joint Conference on Neural Networks (IJCNN)* (2020) (cit. on p. 83).

[ASF20] Vedika Agarwal, Rakshith Shetty, and Mario Fritz. "Towards causal vqa: Revealing and reducing spurious correlations by invariant and covariant semantic editing". In: *Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 90).

[Ath+18] Anish Athalye et al. "Synthesizing Robust Adversarial Examples". In: *International Conference on Machine Learning (ICML)*. 2018 (cit. on pp. 12, 25).

[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 (cit. on p. 75).

[Bau+19] David Bau et al. "GAN Dissection: Visualizing and Understanding Generative Adversarial Networks". In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on p. 61).

[Bau+20a] David Bau et al. "Rewriting a deep generative model". In: *European Conference on Computer Vision (ECCV)*. 2020 (cit. on pp. 21, 91–93, 190, 203).

[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) (cit. on pp. 73, 203).

[BB19] Wieland Brendel and Matthias Bethge. "Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet". In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on pp. 80, 175).

[BCV13] Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: (2013) (cit. on p. 49).

[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 (cit. on p. 56).

[BEN09] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Princeton University Press, 2009 (cit. on p. 26).

[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 (cit. on p. 49).

[Ber+00] Marcelo Bertalmio et al. "Image inpainting". In: *Computer graphics and interactive techniques*. 2000 (cit. on p. 58).

[Bey+20] Lucas Beyer et al. "Are we done with ImageNet?" In: *arXiv preprint arXiv:2006.07159*. 2020 (cit. on pp. 75, 90).

[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 (cit. on pp. 12, 25, 27).

[BM98] Avrim Blum and Tom Mitchell. "Combining labeled and unlabeled data with co-training". In: *Proceedings of the eleventh annual conference on Computational learning theory*. 1998 (cit. on pp. 20, 83, 84).

[BPR19] Sébastien Bubeck, Eric Price, and Ilya Razenshteyn. "Adversarial examples from computational constraints". In: *International Conference on Machine Learning*. 2019 (cit. on p. 120).

[Bro+18] Tom B. Brown et al. *Adversarial Patch*. 2018. arXiv: 1712.09665 [cs.CV] (cit. on pp. 12, 25).

[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 (cit. on p. 60).

[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 (cit. on p. 90).

[BVP18] Sara Beery, Grant Van Horn, and Pietro Perona. "Recognition in terra incognita". In: *European Conference on Computer Vision (ECCV)*. 2018 (cit. on pp. 18, 87, 90).

[Cal+18] Sebastian Caldas et al. "Leaf: A benchmark for federated settings". In: *arXiv preprint arXiv:1812.01097* (2018) (cit. on p. 67).

[Can86] John Canny. "A computational approach to edge detection". In: 1986 (cit. on p. 175).

[Car+16] Nicholas Carlini et al. "Hidden Voice Commands". In: *USENIX Security Symposium*. 2016 (cit. on pp. 12, 25).

[Car+17] Nicholas Carlini et al. “Ground-Truth Adversarial Examples”. In: *ArXiv preprint arXiv:1709.10207*. 2017 (cit. on p. 36).

[Car+19] Yair Carmon et al. “Unlabeled data improves adversarial robustness”. In: *Neural Information Processing Systems (NeurIPS)*. 2019 (cit. on p. 37).

[CH18] Wengling Chen and James Hays. “Sketchygan: Towards diverse and realistic sketch to image synthesis”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018 (cit. on p. 61).

[Che+17] Pin-Yu Chen et al. “Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models”. In: *Workshop on Artificial Intelligence and Security*. 2017 (cit. on p. 25).

[CNL11] Adam Coates, Andrew Ng, and Honglak Lee. “An analysis of single-layer networks in unsupervised feature learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011 (cit. on pp. 81, 174).

[CRK19] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. “Certified adversarial robustness via randomized smoothing”. In: *International Conference on Machine Learning (ICML)*. 2019 (cit. on pp. 37, 119, 120).

[CUF18] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. “COCO-Stuff: Thing and stuff classes in context”. In: *Computer vision and pattern recognition (CVPR), 2018*. 2018 (cit. on p. 69).

[CW17a] Nicholas Carlini and David Wagner. “Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods”. In: *Workshop on Artificial Intelligence and Security (AISec)*. 2017 (cit. on pp. 14, 25).

[CW17b] Nicholas Carlini and David Wagner. “Towards evaluating the robustness of neural networks”. In: *Symposium on Security and Privacy (SP)*. 2017 (cit. on pp. 14, 25, 32, 33).

[Dan67] John M. Danskin. *The Theory of Max-Min and its Application to Weapons Allocation Problems*. 1967 (cit. on pp. 14, 28).

[DB16a] Alexey Dosovitskiy and Thomas Brox. “Generating images with perceptual similarity metrics based on deep networks”. In: *neural information processing systems (NeurIPS)*. 2016 (cit. on p. 49).

[DB16b] Alexey Dosovitskiy and Thomas Brox. “Inverting visual representations with convolutional networks”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 50).

[Den+09] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2009 (cit. on pp. 19, 66, 68, 94, 145, 165, 189).

[DFE07] Kostadin Dabov, Alessandro Foi, and Karen Egiazarian. “Video denoising by sparse 3D transform-domain collaborative filtering”. In: *European Signal Processing Conference*. 2007 (cit. on p. 60).

[DG01] Lijun Ding and Ardeshir Goshtasby. "On the Canny edge detector". In: *Pattern Recognition*. 2001 (cit. on p. 80).

[Dhi+18] Guneet S Dhillon et al. "Stochastic activation pruning for robust adversarial defense". In: *International Conference on Learning Representations (ICLR)* (2018) (cit. on p. 14).

[Don+14] Jeff Donahue et al. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International conference on machine learning (ICML)*. 2014 (cit. on p. 49).

[DSB17] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using real NVP". In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on p. 56).

[Dvi+18] Krishnamurthy Dvijotham et al. "Training verified learners with learned verifiers". In: *ArXiv preprint arXiv:1805.10265*. 2018 (cit. on p. 36).

[EL99] Alexei A Efros and Thomas K Leung. "Texture synthesis by non-parametric sampling". In: *conference on computer vision (CVPR)*. 1999 (cit. on p. 58).

[Eng+19a] Logan Engstrom et al. "Adversarial Robustness as a Prior for Learned Representations". In: *ArXiv preprint arXiv:1906.00945*. 2019 (cit. on p. 22).

[Eng+19b] Logan Engstrom et al. "Exploring the Landscape of Spatial Robustness". In: *International Conference on Machine Learning (ICML)*. 2019 (cit. on pp. 37, 64).

[Eng+19c] Logan Engstrom et al. *Robustness (Python Library)*. 2019. URL: https://github.com/MadryLab/robustness (cit. on p. 121).

[Eng+20] Logan Engstrom et al. "Identifying Statistical Bias in Dataset Replication". In: *International Conference on Machine Learning (ICML)*. 2020 (cit. on p. 67).

[Eve+10] M. Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision*. 2010 (cit. on p. 69).

[Evt+18] Ivan Evtimov et al. "Robust Physical-World Attacks on Machine Learning Models". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (cit. on p. 25).

[FF15] Alhussein Fawzi and Pascal Frossard. "Manitest: Are classifiers really invariant?" In: *British Machine Vision Conference (BMVC)*. 2015 (cit. on pp. 27, 37, 64).

[FFF18a] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. "Adversarial vulnerability for any classifier". In: *Advances in Neural Information Processing Systems (NeuRIPS)*. 2018 (cit. on p. 119).

[FFF18b] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. "Analysis of classifiers' robustness to adversarial perturbations". In: *Machine Learning* 107.3 (2018), pp. 481–508 (cit. on p. 120).

[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 (cit. on pp. 64, 120).

[For+19] Nic Ford et al. "Adversarial Examples Are a Natural Consequence of Test Error in Noise". In: *arXiv preprint arXiv:1901.10513*. 2019 (cit. on pp. 64, 79, 120).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. on p. 49).

[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 (cit. on pp. 74, 165).

[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 (cit. on p. 74).

[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 (cit. on pp. 20, 78).

[Gei+20] Robert Geirhos et al. "Shortcut learning in deep neural networks". In: *Nature Machine Intelligence*. 2020 (cit. on pp. 87, 90).

[Ghi+15] Muhammad Ghifary et al. "Domain generalization for object recognition with multi-task autoencoders". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2551–2559 (cit. on p. 67).

[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 (cit. on pp. 74, 166).

[Gil+18] Justin Gilmer et al. "Adversarial spheres". In: *Workshop of International Conference on Learning Representations (ICLR)*. 2018 (cit. on pp. 43, 119).

[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 (cit. on p. 49).

[Gir+18] Ross Girshick et al. *Detectron*. https://github.com/facebookresearch/detectron. 2018 (cit. on p. 165).

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech Recognition with Deep Recurrent Neural Networks". In: *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 2013 (cit. on p. 12).

[Goh+21] Gabriel Goh et al. "Multimodal neurons in artificial neural networks". In: *Distill* (2021) (cit. on pp. 91, 97, 98, 192).

[Goh19] Gabriel Goh. "A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Robust Feature Leakage". In: *Distill* (2019). https://distill.pub/2019/advex-bugs-discussion/response-2. DOI: 10.23915/distill.00019.2 (cit. on p. 47).

[Goo+14] Ian Goodfellow et al. "Generative adversarial nets". In: *neural information processing systems (NeurIPS)*. 2014 (cit. on pp. 55, 56).

[Gow+20] Sven Gowal et al. “Uncovering the limits of adversarial training against norm-bounded adversarial examples”. In: *arXiv preprint arXiv:2010.03593* (2020) (cit. on p. 37).

[Goy+19a] Yash Goyal et al. “Counterfactual visual explanations”. In: *arXiv preprint arXiv:1904.07451* (2019) (cit. on p. 73).

[Goy+19b] Yash Goyal et al. “Explaining classifiers with causal concept effect (cace)”. In: *arXiv preprint arXiv:1907.07165* (2019) (cit. on p. 73).

[Gra13] Alex Graves. “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850*. 2013 (cit. on p. 56).

[GSS15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *International Conference on Learning Representations (ICLR)*. 2015 (cit. on pp. 13, 15, 28, 29, 43, 120).

[Gur+18] Suchin Gururangan et al. “Annotation artifacts in natural language inference data”. In: *North American Chapter of the Association for Computational Linguistics (NAACL)*. 2018 (cit. on p. 87).

[HAE16] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. “What makes ImageNet good for transfer learning?” In: *arXiv preprint arXiv:1608.08614* (2016) (cit. on pp. 68, 71, 146, 147, 155).

[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 (cit. on pp. 18, 64, 67).

[He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015 (cit. on pp. 94, 114).

[He+16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 34, 48, 121, 165, 189).

[He+17] Warren He et al. “Adversarial example defense: Ensembles of weak defenses are not strong”. In: *USENIX Workshop on Offensive Technologies (WOOT)*. 2017 (cit. on pp. 14, 25).

[HE07] James Hays and Alexei A Efros. “Scene completion using millions of photographs”. In: *ACM Transactions on Graphics (TOG)*. 2007 (cit. on p. 58).

[Her+01] Aaron Hertzmann et al. “Image analogies”. In: *Computer graphics and interactive techniques*. 2001 (cit. on p. 58).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation*. 1997 (cit. on p. 56).

[Ily+18] Andrew Ilyas et al. “Black-box Adversarial Attacks with Limited Queries and Information”. In: *International Conference on Machine Learning (ICML)*. 2018 (cit. on p. 25).

[Ily+19] Andrew Ilyas et al. “Adversarial Examples Are Not Bugs, They Are Features”. In: *Neural Information Processing Systems (NeurIPS)*. 2019 (cit. on p. 22).

[IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning (ICML)*. 2015 (cit. on p. 175).

[Isc+19] Ahmet Iscen et al. “Label propagation for deep semi-supervised learning”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on p. 84).

[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 (cit. on p. 49).

[JL17] Robin Jia and Percy Liang. “Adversarial Examples for Evaluating Reading Comprehension Systems”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2017 (cit. on pp. 12, 25).

[JTM21] Saachi Jain, Dimitris Tsipras, and Aleksander Madry. “Co-Priors: Combining Biases on Learned Features”. In: *Preprint*. 2021 (cit. on p. 23).

[Kan+19] Daniel Kang et al. “Testing Robustness Against Unforeseen Adversaries”. In: *ArXiv preprint arxiv:1908.08016*. 2019 (cit. on p. 64).

[Kar+18] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: *International Conference on Learning Representations*. 2018 (cit. on p. 56).

[Kat+17] Guy Katz et al. “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks”. In: *International Conference on Computer Aided Verification*. 2017 (cit. on p. 36).

[KD18] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative flow with invertible 1x1 convolutions”. In: *Neural Information Processing Systems (NeurIPS)*. 2018 (cit. on p. 56).

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. “Adversarial examples in the physical world”. In: *arXiv preprint arXiv:1607.02533* (2016) (cit. on p. 25).

[KGB17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. “Adversarial Machine Learning at Scale”. In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on p. 29).

[KHA99] Mark G Kelly, David J Hand, and Niall M Adams. “The impact of changing populations on classifier performance”. In: *international conference on Knowledge discovery and data mining (SIGKDD)*. 1999 (cit. on p. 64).

[Kho+12] Aditya Khosla et al. “Undoing the damage of dataset bias”. In: *European Conference on Computer Vision (ECCV)*. 2012 (cit. on p. 64).

[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 (cit. on p. 92).

[Kri09] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: *Technical report*. 2009 (cit. on pp. 12, 29, 45, 69, 81, 114, 121, 174).

[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 (cit. on p. 12).

[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 (cit. on p. 120).

[Kuz+18] Alina Kuznetsova et al. "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale". In: *arXiv preprint arXiv:1811.00982* (2018) (cit. on pp. 66, 69).

[KW15] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2015 (cit. on p. 56).

[Lar+16] Anders Boesen Lindbo Larsen et al. "Autoencoding beyond pixels using a learned similarity metric". In: *International Conference on Machine Learning (ICML)*. 2016 (cit. on p. 55).

[Lec+19] Mathias Lecuyer et al. "Certified robustness to adversarial examples with differential privacy". In: *Symposium on Security and Privacy (SP)*. 2019 (cit. on pp. 37, 120).

[LeC98] Yann LeCun. "The MNIST database of handwritten digits". In: *Technical report*. 1998 (cit. on pp. 29, 114, 121).

[Lee+13] Dong-Hyun Lee et al. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks". In: *Workshop on challenges in representation learning, ICML*. 2013 (cit. on pp. 83, 84).

[Lin+14] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision (ECCV)*. 2014 (cit. on pp. 74, 165).

[Liu+15] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *International Conference on Computer Vision (ICCV)*. 2015 (cit. on pp. 70, 88, 174).

[Mad+18] Aleksander Madry et al. "Towards deep learning models resistant to adversarial attacks". In: *International Conference on Learning Representations (ICLR)*. 2018 (cit. on pp. 22, 119–121, 124, 171).

[McA+15] Julian McAuley et al. "Image-based recommendations on styles and substitutes". In: *Research and development in Information Retrieval (SIGIR)*. 2015 (cit. on p. 70).

[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 (cit. on pp. 43, 119).

[Mil95] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* (1995) (cit. on pp. 68, 145).

[Mor+12] Jose G Moreno-Torres et al. “A unifying view on dataset shift in classification”. In: *Pattern recognition* (2012) (cit. on p. 64).

[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 (cit. on pp. 53, 56).

[MV15] Aravindh Mahendran and Andrea Vedaldi. “Understanding deep image representations by inverting them”. In: *computer vision and pattern recognition (CVPR)*. 2015 (cit. on p. 50).

[MWK20] Pratyush Maini, Eric Wong, and Zico Kolter. “Adversarial robustness against the union of multiple perturbation models”. In: *International Conference on Machine Learning (ICML)*. 2020 (cit. on p. 37).

[Nak19] Preetum Nakkiran. “Adversarial robustness may be at odds with simplicity”. In: *arXiv preprint arXiv:1901.00532*. 2019 (cit. on p. 120).

[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 (cit. on pp. 55, 56).

[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 (cit. on pp. 55, 56).

[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 (cit. on pp. 55, 56).

[OMS17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. “Feature Visualization”. In: *Distill*. 2017 (cit. on pp. 52, 53, 55).

[Ore+19] Yonatan Oren et al. “Distributionally Robust Language Modeling”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2019 (cit. on pp. 64, 66).

[Oyg15] Audun Oygard. *Visualizing GoogLeNet Classes*. 2015. URL: https://www.auduno.com/2015/07/29/visualizing-googlenet-classes/ (cit. on pp. 55, 56).

[Pap+16] Nicolas Papernot et al. “Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks”. In: *Symposium on Security and Privacy (SP)*. 2016 (cit. on p. 13).

[Par+19] Taesung Park et al. “Semantic Image Synthesis with Spatially-Adaptive Normalization”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on p. 61).

[Pea10] Judea Pearl. “Causal inference”. In: *Causality: Objectives and Assessment*. 2010 (cit. on p. 73).

[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 (cit. on pp. 15, 25).

[Pon+06] Jean Ponce et al. "Dataset issues in object recognition". In: *Toward category-level object recognition*. 2006 (cit. on pp. 18, 67, 90).

[Qui+09] Joaquin Quionero-Candela et al. *Dataset shift in machine learning*. The MIT Press, 2009 (cit. on p. 64).

[Rad+21] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *arXiv preprint arXiv:2103.00020*. 2021 (cit. on pp. 97, 98, 189).

[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 (cit. on p. 49).

[Rec+19] Benjamin Recht et al. "Do ImageNet Classifiers Generalize to ImageNet?" In: *International Conference on Machine Learning (ICML)*. 2019 (cit. on p. 64).

[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 (cit. on p. 55).

[RSG16] 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 (cit. on p. 75).

[RSL18] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. "Certified defenses against adversarial examples". In: *International Conference on Learning Representations (ICLR)*. 2018 (cit. on pp. 37, 119).

[Rus+15] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)*. 2015 (cit. on pp. 19, 68, 94, 121, 145, 165, 189).

[RZT18] Amir Rosenfeld, Richard Zemel, and John K. Tsotsos. "The Elephant in the Room". In: *arXiv preprint arXiv:1808.03305*. 2018 (cit. on p. 75).

[S+19] Robert Stanforth, Alhussein Fawzi, Pushmeet Kohli, et al. "Are Labels Required for Improving Adversarial Robustness?" In: *Neural Information Processing Systems (NeurIPS)*. 2019 (cit. on p. 37).

[Sae+10] Kate Saenko et al. "Adapting visual category models to new domains". In: *European conference on computer vision (ECCV)*. 2010 (cit. on pp. 18, 64).

[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 (cit. on p. 64).

[Sah+98] Mehran Sahami et al. "A Bayesian approach to filtering junk e-mail". In: *Learning for Text Categorization*. 1998 (cit. on p. 12).

[Sal+20] Hadi Salman et al. “Do Adversarially Robust ImageNet Models Transfer Better?” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020 (cit. on p. 79).

[San+19] Shibani Santurkar et al. “Image Synthesis with a Single (Robust) Classifier”. In: *Neural Information Processing Systems (NeurIPS)*. 2019 (cit. on p. 22).

[San+21] Shibani Santurkar et al. “Editing a classifier by rewriting its prediction rules”. In: *Preprint*. 2021 (cit. on p. 23).

[SC18] Pierre Stock and Moustapha Cisse. “Convnets and imagenet beyond accuracy: Understanding mistakes and uncovering biases”. In: *European Conference on Computer Vision (ECCV)*. 2018 (cit. on p. 75).

[Sch+18] Ludwig Schmidt et al. “Adversarially Robust Generalization Requires More Data”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018 (cit. on pp. 43, 119).

[SCJ19] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. “Exploring Adversarial Examples in Malware Detection”. In: *IEEE Security and Privacy Workshops (SPW)*. 2019 (cit. on p. 25).

[SF68] Irwin Sobel and Gary Feldman. “A 3x3 isotropic gradient operator for image processing”. In: 1968 (cit. on pp. 80, 175).

[SG86] Jeffrey C Schlimmer and Richard H Granger. “Beyond Incremental Processing: Tracking Concept Drift.” In: *AAAI*. 1986 (cit. on p. 64).

[Sha+19a] Ali Shafahi et al. “Are adversarial examples inevitable?” In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on pp. 43, 119).

[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 (cit. on p. 120).

[Shi00] Hidetoshi Shimodaira. “Improving predictive inference under covariate shift by weighting the log-likelihood function”. In: *Journal of statistical planning and inference* (2000) (cit. on p. 64).

[SK12] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press, 2012 (cit. on p. 64).

[SKM07] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert MÃžller. “Covariate shift adaptation by importance weighted cross validation”. In: *Journal of Machine Learning Research (JMLR)* (2007) (cit. on p. 64).

[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: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on p. 90).

[STM21] Shibani Santurkar, Dimitris Tsipras, and Aleksander Madry. “Breeds: Benchmarks for subpopulation shift”. In: *International Conference on Learning Representations (ICLR)*. 2021 (cit. on pp. 22, 23).

[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) (cit. on p. 56).

[SZ15] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations (ICLR)*. 2015 (cit. on pp. 94, 165, 175, 189).

[Sze+14] Christian Szegedy et al. "Intriguing properties of neural networks". In: *International Conference on Learning Representations (ICLR)*. 2014 (cit. on pp. 12, 13, 15, 25, 27, 40, 43, 64).

[TB19] Florian Tramèr and Dan Boneh. "Adversarial Training and Robustness for Multiple Perturbations". In: *Neural Information Processing Systems (NeurIPS)*. 2019 (cit. on p. 37).

[TE11] Antonio Torralba and Alexei A Efros. "Unbiased look at dataset bias". In: *CVPR 2011*. 2011 (cit. on pp. 18, 64, 67, 90).

[TG16] Thomas Tanay and Lewis Griffin. "A Boundary Tilting Perspective on the Phenomenon of Adversarial Examples". In: *ArXiv preprint arXiv:1608.07690*. 2016 (cit. on pp. 43, 119).

[TM98] Carlo Tomasi and Roberto Manduchi. "Bilateral filtering for gray and color images". In: *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*. 1998 (cit. on p. 175).

[Tra+17] Florian Tramer et al. "The Space of Transferable Adversarial Examples". In: *ArXiv preprint arXiv:1704.03453*. 2017 (cit. on p. 28).

[Tsi+19] Dimitris Tsipras et al. "Robustness May Be at Odds with Accuracy". In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on p. 22).

[Tsi+20] Dimitris Tsipras et al. "From ImageNet to Image Classification: Contextualizing Progress on Benchmarks". In: *International Conference on Machine Learning (ICML)*. 2020 (cit. on pp. 67, 75, 90).

[TT14] Tatiana Tommasi and Tinne Tuytelaars. "A testbed for cross-dataset analysis". In: *European Conference on Computer Vision (ECCV)*. 2014 (cit. on p. 64).

[TXT19] Vincent Tjeng, Kai Xiao, and Russ Tedrake. "Evaluating Robustness of Neural Networks with Mixed Integer Programming". In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on p. 36).

[Tyk16] Mike Tyka. *Class visualization with bilateral filters*. 2016. URL: https://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html (cit. on pp. 55, 56).

[Utr+20] Francisco Utrera et al. "Adversarially-Trained Deep Nets Transfer Better". In: *ArXiv preprint arXiv:2007.05869*. 2020 (cit. on p. 79).

[UVL17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Deep Image Prior". In: *ArXiv preprint arXiv:1711.10925*. 2017 (cit. on p. 51).

[VKK16] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *International Conference on Machine Learning (ICML)*. 2016 (cit. on p. 56).

[Wal45] Abraham Wald. "Statistical Decision Functions Which Minimize the Maximum Risk". In: *Annals of Mathematics*. 1945 (cit. on p. 27).

[Wan+19] Yisen Wang et al. "Improving adversarial robustness requires revisiting misclassified examples". In: *International Conference on Learning Representations*. 2019 (cit. on p. 37).

[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 (cit. on pp. 37, 119).

[WK93] Gerhard Widmer and Miroslav Kubat. "Effective learning in dynamic environments by explicit context tracking". In: *European Conference on Machine Learning*. 1993 (cit. on p. 64).

[WSK19] Eric Wong, Frank Schmidt, and Zico Kolter. "Wasserstein adversarial examples via projected sinkhorn iterations". In: *International Conference on Machine Learning (ICML)*. 2019 (cit. on p. 37).

[Wu+16] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144*. 2016 (cit. on p. 12).

[Xia+18] Chaowei Xiao et al. "Spatially Transformed Adversarial Examples". In: *International Conference on Learning Representations (ICLR)*. 2018 (cit. on p. 27).

[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 (cit. on pp. 36, 119).

[Xia+20] Kai Xiao et al. "Noise or signal: The role of image backgrounds in object recognition". In: *arXiv preprint arXiv:2006.09994* (2020) (cit. on pp. 18, 75, 87, 90).

[Xie+20] Qizhe Xie et al. "Self-training with noisy student improves imagenet classification". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020 (cit. on p. 84).

[Yan+19] Kaiyu Yang et al. *Towards Fairer Datasets: Filtering and Balancing the Distribution of the People Subtree in the ImageNet Hierarchy*. http://image-net.org/update-sep-17-2019. Accessed: 2020-10-01. 2019 (cit. on p. 67).

[Yu+20] Fisher Yu et al. "BDD100K: A diverse driving dataset for heterogeneous multitask learning". In: *Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 70).

[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 (cit. on p. 75).

[Zha+18] Richard Zhang et al. “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2018 (cit. on p. 49).

[Zha+19] Hongyang Zhang et al. “Theoretically Principled Trade-off between Robustness and Accuracy”. In: *International Conference on Machine Learning (CIML)*. 2019 (cit. on p. 37).

[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) (cit. on pp. 94, 165, 189).

[Zho+20] Zhun Zhong et al. “Random Erasing Data Augmentation.” In: *AAAI*. 2020 (cit. on p. 78).

[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 (cit. on pp. 58–60, 142, 143).

[Zou+19] Yang Zou et al. “Confidence regularized self-training”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019 (cit. on p. 84).

Appendix

Appendix A

Additional details for Chapter 1

A.1 Experimental setup.

Our first dataset will be MNIST [LeC98] which contains black-and-white images of handwritten digits. We will use a since convolutional network consisting of two convolutional layers with 32 and 64 filters respectively, each followed by 2×2 max-pooling, and a fully connected layer of size 1024. Since most pixels are either 0 (black) or 1 (white) we restrict perturbations to an ℓ_∞ norm of 0.3. We will craft these perturbations using 40 steps of PGD with a step size of 0.01.

Our second dataset will be CIFAR-10 [Kri09] which contains RGB photographs from 10 categories. Here we will restrict perturbations to an ℓ_∞ norm of $8/255$ computed using 7 steps of PGD with step size $2/255$. Our architecture will be a wide variant of a ResNet [He+15].

A.2 Statement and application of Danskin's theorem

Recall that our goal is to minimize the value of the saddle point problem

$$\min_\theta \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\Delta} \mathcal{L}(\theta, x+\delta, y) \right] .$$

In practice, we don't have access to the distribution $\mathcal{D}$ so both the gradients and the value of $\rho(\theta)$ will be computed using sampled input points. Therefore we can consider –without loss of generality– the case of a single random example x with label y, in which case the problem becomes

$$\min_\theta \max_{\delta\in\Delta} g(\theta, \delta), \quad \text{where} \quad g(\theta, \delta) = \mathcal{L}(\theta, x+\delta, y) .$$

If we assume that the loss $\mathcal{L}$ is continuously differentiable in θ, we can compute a descent direction for θ by utilizing the classical theorem of Danskin.

Theorem A.2.1 (Danskin). *Let Δ be nonempty compact topological space and $g : \mathbb{R}^n \times \Delta \to \mathbb{R}$ be such that $g(\cdot, \delta)$ is differentiable for every $\delta \in \Delta$ and $\nabla_\theta g(\theta, \delta)$ is continuous on $\mathbb{R}^n \times \Delta$. Also, let $\delta^*(\theta) = \{\delta \in \arg\max_{\delta\in\Delta} g(\theta, \delta)\}$.*

Then the corresponding max-function

$$\phi(\theta) = \max_{\delta\in\Delta} g(\theta, \delta)$$

is locally Lipschitz continuous, directionally differentiable, and its directional derivatives satisfy

$$\phi'(\theta, h) = \sup_{\delta \in \delta^*(\theta)} h^\top \nabla_\theta g(\theta, \delta)\,.$$

In particular, if for some $\theta \in \mathbb{R}^n$ the set $\delta^*(\theta) = \{\delta^*_\theta\}$ is a singleton, the the max-function is differentiable at θ and

$$\nabla \phi(\theta) = \nabla_\theta g(\theta, \delta^*_\theta)$$

The intuition behind the theorem is that since gradients are local objects, and the function $\phi(\theta)$ is locally the same as $g(\theta, \delta^*_\theta)$ their gradients will be the same. The theorem immediately gives us the following corollary, stating the we can indeed compute gradients for the saddle point by computing gradients at the inner optimizers.

Corollary A.2.2. *Let $\overline{\delta}$ be such that $\overline{\delta} \in \Delta$ and is a maximizer for $\max_\delta L(\theta, x + \delta, y)$. Then, as long as it is nonzero, $-\nabla_\theta \mathcal{L}(\theta, x + \overline{\delta}, y)$ is a descent direction for $\phi(\theta) = \max_{\delta \in \Delta} \mathcal{L}(\theta, x + \delta, y)$.*

Proof of Corollary A.2.2. We apply Theorem A.2.1 to $g(\theta, \delta) := \mathcal{L}(\theta, x + \delta, y)$ and $\Delta = B_{\|\cdot\|}(\varepsilon)$. We see that the directional derivative in the direction of $h = \nabla_\theta \mathcal{L}(\theta, x + \overline{\delta}, y)$ satisfies

$$\phi'(\theta, h) = \sup_{\delta \in \delta^*(\theta)} h^\top \nabla_\theta \mathcal{L}(\theta, x + \delta, y) \geq h^\top h = \|\nabla_\theta \mathcal{L}(\theta, x + \overline{\delta}, y)\|_2^2 \geq 0\,.$$

If this gradient is nonzero, then the inequality above is strict. Therefore it gives a descent direction. □

A technical issue is that, since we use ReLU and max-pooling units in our neural network architecture, the loss function is not continuously differentiable. Nevertheless, since the set of discontinuities has measure zero, we can assume that this will not be an issue in practice, as we will never encounter the problematic points.

Another technical issue is that, due to the not concavity of the inner problem, we are not able to compute global maximizers, since PGD will converge to local maxima. In such cases, we can consider a subset Δ' of Δ such that the local maximum is a global maximum in the region Δ'. Applying the theorem for Δ' gives us that the gradient corresponds to a descent direction for the saddle point problem when the adversary is constrained in Δ'. Therefore if the inner maximum is a true adversarial example for the network, then SGD using the gradient at that point will decrease the loss value at this particular adversarial examples, thus making progress towards a robust model.

These arguments suggest that the conclusions of the theorem are still valid in our saddle point problem, and—as our experiments confirm—we can solve it reliably.

A.3 Inspecting a robust model

The robust MNIST model described so far is small enough that we can visually inspect most of its parameters. Doing so will allow us to understand how it is different from a standard network and what are the general characteristics of a network that is robust against ℓ_∞ adversaries. We will compare three different networks: a standard model, and two adversarially trained ones. The latter two models are identical, modulo the random weight initialization, and were used as the public and secret models used for our robustness challenge.

Initially, we examine the first convolutional layer of each network. We observe that the robust models only utilize 3 out of the total 32 filters, and for each of these filters only one weight is non-zero. By doing so, the convolution degrades into a scaling of the original image. Combined with the bias and the ReLU that follows, this results in a *thresholding filter*, or equivalently $\text{ReLU}(\alpha x - \beta)$ for some constants α, β. From the perspective of adversarial robustness, thresholding filters are immune to any perturbations on pixels with value less than $\beta - \varepsilon$. We visualize a sample of the filters in Figure A.1 (plots a, c, and e).

Having observed that the first layer of the network essentially maps the original image to three copies thresholded at different values, we examine the second convolutional layer of the classifier. Again, the filter weights are relatively sparse and have a significantly wider value range than the standard version. Since only three channels coming out of the first layer matter, is follows (and is verified) that the only relevant convolutional filters are those that interact with these three channels. We visualize a sample of the filters in Figure A.1 (plots b, d, and f).

Finally, we examine the softmax/output layer of the network. While the weights seem to be roughly similar between all three version of the network, we notice a significant difference in the class biases. The adversarially trained networks heavily utilize class biases (far from uniform), and do so in a way very similar to each other. A plausible explanation is that certain classes tend to be very vulnerable to adversarial perturbations, and the network learns to be more conservative in predicting them. The plots can be found in Figure A.2.

All of the "tricks" described so far seem intuitive to a human and would seem reasonable directions when trying to increase the adversarial robustness of a classifier. We emphasize the none of these modifications were hard-coded in any way and they were all *learned solely through adversarial training*. We attempted to manually introduce these modifications ourselves, aiming to achieve adversarial robustness without adversarial training, but with no success. A simple PGD adversary could fool the resulting models on all the test set examples.

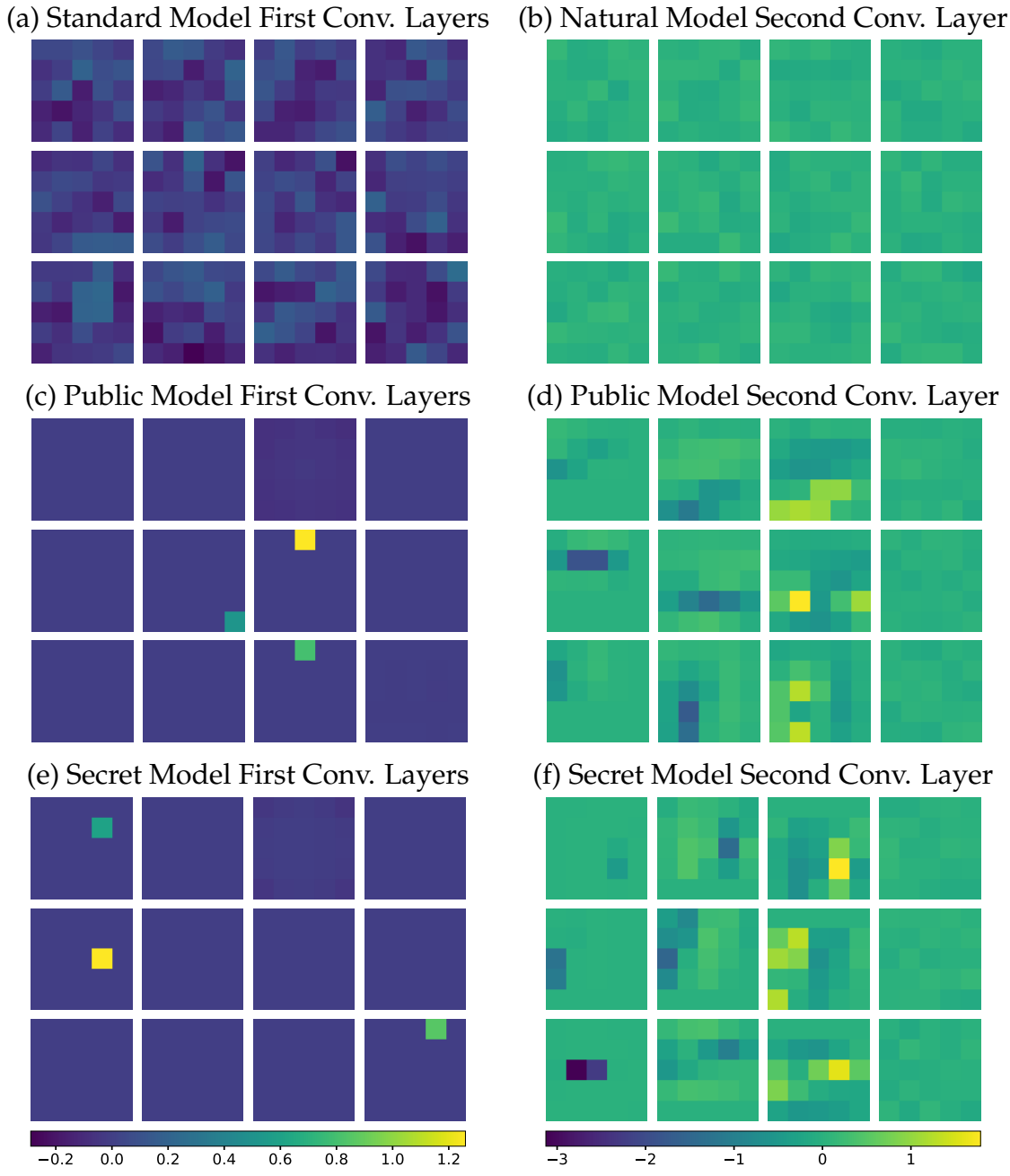
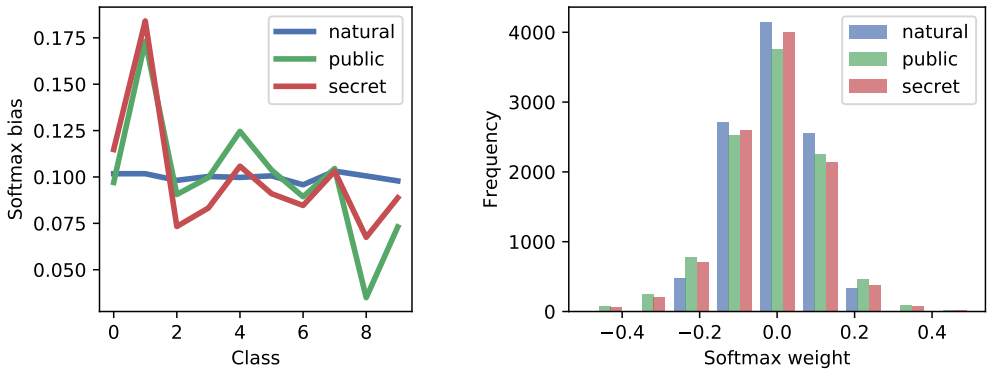


Figure A.1: Visualizing a sample of the convolutional filters. For the standard model (a,b) we visualize random filters, since there is no observable difference in any of them. For the first layer of robust networks we make sure to include the 3 non-zero filters. For the second layer, the first three columns represent convolutional filters that utilize the 3 non-zero channels, and we choose the most interesting ones (larger range of values). We observe that adversarially trained networks have significantly more concentrated weights. Moreover, the first convolutional layer degrades into a few thresholding filters.



(a) Softmax biases for each class (b) Distribution of softmax weights

Figure A.2: Softmax layer examination. For each network we create a histogram of the layer's weights and plot the per-class bias. We observe that while weights are similar (slightly more concentrated for the standard one) the biases are far from uniform and with a similar pattern for the two adversarially trained networks.

Appendix B

Additional details for Chapter 2

B.1 Alternative models for adversarial examples

Here, we describe other models for adversarial examples and how they relate to the model presented in Chapter 2.

Concentration of measure in high-dimensions. An orthogonal line of work [Gil+18; FFF18a; MDM18; Sha+19a], argues that the high dimensionality of the input space can present fundamental barriers on classifier robustness. At a high level, one can show that, for certain data distributions, any decision boundary will be close to a large fraction of inputs and hence no classifier can be robust against small perturbations. While there might exist such fundamental barriers to robustly classifying standard datasets, this model cannot fully explain the situation observed in practice, where one can train (reasonably) robust classifiers on standard datasets [Mad+18; RSL18; WK18; Xia+19; CRK19].

Insufficient data. Schmidt et al. [Sch+18] propose a theoretical model under which a single sample is sufficient to learn a good, yet non-robust classifier, whereas learning a good robust classifier requires $O(\sqrt{d})$ samples. Under this model, adversarial examples arise due to insufficient information about the true data distribution. However, unless the adversary is strong enough (in which case no robust classifier exists), adversarial inputs cannot be utilized as inputs of the opposite class (as done in our experiments in Section 2.2.2). We note that our model does not explicitly contradict the main thesis of Schmidt et al. [Sch+18]. In fact, this thesis can be viewed as a natural consequence of our conceptual framework. In particular, since training models robustly reduces the effective amount of information in the training data (as non-robust features are discarded), more samples should be required to generalize robustly.

Boundary Tilting. Tanay and Griffin [TG16] introduce the "boundary tilting" model for adversarial examples, and suggest that adversarial examples are a product of over-fitting. In particular, the model conjectures that "adversarial examples are possible because the class boundary extends beyond the submanifold of sample data and can be—under certain circumstances—lying close to it." Consequently, the authors suggest that mitigating adversarial examples may be a matter of regularization and preventing finite-sample overfitting. In contrast, our empirical results in Section 2.2.2 suggest that adversarial inputs consist of features inherent to the data distribution, since they can encode generalizing information about the target class.

Inspired by this hypothesis 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 2.1.1. (Note that this exact statement is not true beyond two dimensions, as discussed in Section 2.1.1.)

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.

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.

Other theoretical models utilizing non-robust features. Bubeck, Price, and Razenshteyn [BPR19] and Nakkiran [Nak19] 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. [FFF18b] prove upper bounds on the robustness of classifiers and exhibit a standard vs. robust accuracy trade-off for specific classifier families on a synthetic task. Their setting also (implicitly) utilizes the notion of robust and non-robust features, however these features have small magnitude rather than weak correlation.

B.2 Experimental setup

B.2.1 Datasets

We perform our analysis on the MNIST [LeC98], CIFAR-10 [Kri09], and ImageNet [Rus+15] datasets. We also consider a smaller variant of the ImageNet dataset by grouping together classes that are semantically similar into superclasses, as shown in Table B.1. This leads to a dataset that has significantly more samples per class compared to ImageNet.

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

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

B.2.2 Models

- MNIST: We use a simple convolutional architecture [Mad+18][1].
- CIFAR-10: We consider a standard ResNet model [He+16]. It has 4 groups of residual layers with filter sizes (16, 16, 32, 64) and 5 residual units each[2].
- ImageNet and Restricted ImageNet: We use the ResNet-50 architecture trained with a learning rate of 0.1 which drops by a factor of 10 every 30 epochs. We trained for a total of 110 epochs with a batch size of 256.

B.2.3 Adversarial training

We perform adversarial training to train robust classifiers following [Mad+18]. Specifically, we train against a projected gradient descent (PGD) adversary, starting from a random initial perturbation of the training data. We consider adversarial perturbations in ℓ_p norm where $p = \{2, \infty\}$. Unless otherwise specified, we use the values of ε provided in Table B.2 to train/evaluate our models (pixel values in $[0, 1]$). The implementation of this process has been released under the `robustness` library [Eng+19c].

[1] https://github.com/MadryLab/mnist_challenge/
[2] https://github.com/MadryLab/cifar10_challenge/

Table B.2: Value of ε used for adversarial training/evaluation of each dataset and ℓ_p-norm.

Constraint	MNIST	CIFAR-10	(Restricted) Imagenet
ℓ_∞	0.3	$4/255$	0.005
ℓ_2	1.5	0.314	1

B.2.4 Adversarial examples for large ε

The images we generated for Figure 2.7 were allowed a much larger perturbation from the original sample in order to produce visible changes to the images. These values are listed in Table B.3. Since these levels of perturbations would allow to truly change the class of the image, training against such strong adversaries would be impossible. Still, we observe that smaller values of ε suffice to ensure that the models rely on the most robust features.

Table B.3: Value of ε used for large-ε adversarial examples of Figure 2.7.

Adversary	MNIST	CIFAR-10	Restricted Imagenet
ℓ_∞	0.3	0.125	0.25
ℓ_2	4	4.7	40

B.2.5 Constructing a Robust Dataset

In Section 2.2.1, we describe a procedure to construct a dataset that contains features relevant only to a given (standard/robust) model. We initialize x_r as a different randomly chosen sample from the training set. 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 B.5. We provide the pseudocode for the construction in Figure B.4.

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 ℓ_2-PGD starting from x'
   $D_R \leftarrow D_R \bigcup \{(x_R, y)\}$
4. Return D_R

Figure B.4: Algorithm to construct a "robust" dataset, by restricting to features used by a robust model.

Table B.5: Parameters used for optimization procedure to construct dataset in Section 2.2.1.

	CIFAR-10	**Restricted Imagenet**
step size	0.1	1
iterations	1000	2000

B.2.6 Non-robust features suffice for standard classification

To construct the dataset as described in Section 2.2.2, 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. 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 B.7. We provide pseudocode in Figure B.6.

GETNONROBUSTDATASET(D, ε)

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 ℓ_2 PGD
   $D_{NR} \leftarrow D_{NR} \bigcup \{(x_{NR}, t)\}$
4. Return D_{NR}

Figure B.6: Algorithm to construct a dataset where input-label association is based entirely on non-robust features.

Table B.7: Projected gradient descent parameters used to construct constrained adversarial examples in Section 2.2.2.

Attack Parameters	CIFAR-10	Restricted Imagenet
ε	0.5	3
step size	0.1	0.1
iterations	100	100

B.2.7 Image-to-image translation

The parameters used for PGD are as follows.

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

B.2.8 Generation

In order to compute the class conditional Gaussians for high resolution images ($224 \times 224 \times 3$) 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

B.2.9 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

B.2.10 Super-resolution

We directly perform PGD using the following parameters for each datasets.

Dataset	$\uparrow$ factor	ε	# steps	Step size
CIFAR-10	7	15	1	50
restricted ImageNet	8	8	1	40

B.3 Proofs for Section 2.1

B.3.1 Proof of Theorem 2.1.1

The main idea of the proof is that an adversary with $\varepsilon = 2\eta$ is able to change the distribution of features $x_2, \ldots, x_{d+1}$ to reflect a label of $-y$ instead of y by subtracting εy from each variable. Hence any information that is used from these features to achieve better standard accuracy can be used by the adversary to reduce adversarial accuracy. We define G_+ to be the distribution of $x_2, \ldots, x_{d+1}$ when $y = +1$ and G_- to be that distribution when $y = -1$. We will consider the setting where $\varepsilon = 2\eta$ and fix the adversary that replaces x_i by $x_i - y\varepsilon$ for each $i \geq 2$. This adversary is able to change G_+ to G_- in the adversarial setting and vice-versa.

Consider any classifier $f(x)$ that maps an input x to a class in $\{-1, +1\}$. Let us fix the probability that this classifier predicts class $+1$ for some fixed value of x_1 and distribution of $x_2, \ldots, x_{d+1}$. Concretely, we define p_{ij} to be the probability of predicting $+1$ given that the first feature has sign i and the rest of the features are distributed according to G_j. Formally,

$$
\begin{aligned}
p_{++} &= \Pr_{x_{2,\ldots,d+1} \sim G_+}(f(x) = +1 \mid x_1 = +1), \\
p_{+-} &= \Pr_{x_{2,\ldots,d+1} \sim G_-}(f(x) = +1 \mid x_1 = +1), \\
p_{-+} &= \Pr_{x_{2,\ldots,d+1} \sim G_+}(f(x) = +1 \mid x_1 = -1), \\
p_{--} &= \Pr_{x_{2,\ldots,d+1} \sim G_-}(f(x) = +1 \mid x_1 = -1).
\end{aligned}
$$

Using these definitions, we can express the standard accuracy of the classifier as

$$
\begin{aligned}
\Pr(f(x) = y) &= \Pr(y = +1)\,(p \cdot p_{++} + (1-p) \cdot p_{-+}) \\
&\quad + \Pr(y = -1)\,(p \cdot (1 - p_{--}) + (1-p) \cdot (1 - p_{+-})) \\
&= \frac{1}{2}\,(p \cdot p_{++} + (1-p) \cdot p_{-+} + p \cdot (1 - p_{--}) + (1-p) \cdot (1 - p_{+-})) \\
&= \frac{1}{2}\,(p \cdot (1 + p_{++} - p_{--}) + (1-p) \cdot (1 + p_{-+} - p_{+-}))\,.
\end{aligned}
$$

Similarly, we can express the accuracy of this classifier against the adversary that replaces G_+ with G_- (and vice-versa) as

$$
\begin{aligned}
\Pr(f(x_{\text{adv}}) = y) &= \Pr(y = +1)\,(p \cdot p_{+-} + (1-p) \cdot p_{--}) \\
&\quad + \Pr(y = -1)\,(p \cdot (1 - p_{-+}) + (1-p) \cdot (1 - p_{++})) \\
&= \frac{1}{2}\,(p \cdot p_{+-} + (1-p) \cdot p_{--} + p \cdot (1 - p_{-+}) + (1-p) \cdot (1 - p_{++})) \\
&= \frac{1}{2}\,(p \cdot (1 + p_{+-} - p_{-+}) + (1-p) \cdot (1 + p_{--} - p_{++}))\,.
\end{aligned}
$$

For convenience we will define $a = 1 - p_{++} + p_{--}$ and $b = 1 - p_{-+} + p_{+-}$. Then we can rewrite

$$\begin{aligned}\text{standard accuracy}: \quad & \frac{1}{2}(p(2-a) + (1-p)(2-b)) \\ & = 1 - \frac{1}{2}(pa + (1-p)b), \\ \text{adversarial accuracy}: \quad & \frac{1}{2}((1-p)a + pb).\end{aligned}$$

We are assuming that the standard accuracy of the classifier is at least $1 - \delta$ for some small δ. This implies that

$$1 - \frac{1}{2}(pa + (1-p)b) \geq 1 - \delta \implies pa + (1-p)b \leq 2\delta.$$

Since p_{ij} are probabilities, we can guarantee that $a \geq 0$. Moreover, since $p \geq 0.5$, we have $p/(1-p) \geq 1$. We use these to upper bound the adversarial accuracy by

$$\begin{aligned}\frac{1}{2}((1-p)a + pb) &\leq \frac{1}{2}\left((1-p)\frac{p^2}{(1-p)^2}a + pb\right) \\ &= \frac{p}{2(1-p)}(pa + (1-p)b) \\ &\leq \frac{p}{1-p}\delta.\end{aligned}$$

□

B.3.2 Proof of Theorem 2.1.2

We consider the problem of fitting the distribution $\mathcal{D}$ of (2.1) by using a standard soft-margin SVM classifier. Specifically, this can be formulated as:

$$\min_w \mathbb{E}\left[\max(0, 1 - yw^\top x)\right] + \frac{1}{2}\lambda\|w\|_2^2 \tag{B.1}$$

for some value of λ. We will assume that we tune λ such that the optimal solution w^* has ℓ_2-norm of 1. This is without much loss of generality since our proofs can be adapted to the general case. We will refer to the first term of (B.1) as the *margin* term and the second term as the *regularization* term.

First we will argue that, due to symmetry, the optimal solution will assign equal weight to all the features x_i for $i = 2, \ldots, d+1$.

Lemma B.3.1. *Consider an optimal solution w^* to the optimization problem (B.1). Then,*

$$w_i^* = w_j^* \quad \forall\, i, j \in \{2, ..., d+1\}.$$

Proof. Assume that $\exists\, i, j \in \{2, ..., d+1\}$ such that $w_i^* \neq w_j^*$. Since the distribution of x_i and x_j are identical, we can swap the value of w_i and w_j, to get an alternative set of parameters $\hat{w}$ that has the same loss function value ($\hat{w}_j = w_i$, $\hat{w}_i = w_j$, $\hat{w}_k = w_k$ for $k \neq i, j$).

Moreover, since the margin term of the loss is convex in w, using Jensen's inequality, we get that averaging w^* and $\hat{w}$ will not increase the value of that margin term. Note, however, that

$\|\frac{w^*+\hat{w}}{2}\|_2 < \|w^*\|_2$, hence the regularization loss is strictly smaller for the average point. This contradicts the optimality of w^*. □

Since every optimal solution will assign equal weight to all x_i for $k \geq 2$, we can replace these features by their sum (and divide by $\sqrt{d}$ for convenience). We will define

$$z = \frac{1}{\sqrt{d}} \sum_{i=2}^{d+1} x_i,$$

which, by the properties of the normal distribution, is distributed as

$$z \sim \mathcal{N}(y\eta\sqrt{d}, 1).$$

By assigning a weight of v to that combined feature the optimal solutions can be parametrized as

$$w^\top x = w_1 x_1 + vz,$$

where the regularization term of the loss is $\lambda(w_1^2 + v^2)/2$.

Recall that our chosen value of η is $4/\sqrt{d}$, which implies that the contribution of vz is distributed normally with mean $4yv$ and variance v^2. By the concentration of the normal distribution, the probability of vz being larger than v is large. We will use this fact to show that the optimal classifier will assign on v at least as much weight as it assigns on w_1.

Lemma B.3.2. *Consider the optimal solution (w_1^*, v^*) of the problem (B.1). Then*

$$v^* \geq \frac{1}{\sqrt{2}}.$$

Proof. Assume for the sake of contradiction that $v^* < 1/\sqrt{2}$. Then, with probability at least $1 - p$, the first feature predicts the wrong label and without enough weight, the remaining features cannot compensate for it. Concretely,

$$\begin{aligned}
\mathbb{E}[\max(0, 1 - yw^\top x)] &\geq (1-p)\ \mathbb{E}\left[\max\left(0, 1 + w_1 - \mathcal{N}\left(4v, v^2\right)\right)\right] \\
&\geq (1-p)\ \mathbb{E}\left[\max\left(0, 1 + \frac{1}{\sqrt{2}} - \mathcal{N}\left(\frac{4}{\sqrt{2}}, \frac{1}{2}\right)\right)\right] \\
&> (1-p) \cdot 0.016.
\end{aligned}$$

We will now show that a solution that assigns zero weight on the first feature ($v = 1$ and $w_1 = 0$), achieves a better margin loss.

$$\begin{aligned}
\mathbb{E}[\max(0, 1 - yw^\top x)] &= \mathbb{E}\left[\max\left(0, 1 - \mathcal{N}\left(4, 1\right)\right)\right] \\
&< 0.0004.
\end{aligned}$$

Hence, as long as $p \leq 0.975$, this solution has a smaller margin loss than the original solution. Since both solutions have the same norm, the solution that assigns weight only on v is better than the original solution (w_1^*, v^*), contradicting its optimality. □

We have established that the learned classifier will assign more weight to v than w_1. Since z will be at least y with large probability, we will show that the behavior of the classifier depends entirely on z.

Lemma B.3.3. *The standard accuracy of the soft-margin SVM learned for problem* (B.1) *is at least* 99%.

Proof. By Lemma B.3.2, the classifier predicts the sign of $w_1 x_1 + vz$ where $vz \sim \mathcal{N}(4yv, v^2)$ and $v \geq 1/\sqrt{2}$. Hence with probability at least 99%, $vzy > 1/\sqrt{2} \geq w_1$ and thus the predicted class is y (the correct class) independent of x_1. □

We can utilize the same argument to show that an adversary that changes the distribution of z has essentially full control over the classifier prediction.

Lemma B.3.4. *The adversarial accuracy of the soft-margin SVM learned for* (B.1) *is at most 1% against an ℓ_∞-bounded adversary of $\varepsilon = 2\eta$.*

Proof. Observe that the adversary can shift each feature x_i towards y by 2η. This will cause z to be distributed as

$$z_{\text{adv}} \sim \mathcal{N}(-y\eta\sqrt{d}, 1).$$

Therefore with probability at least 99%, $vyz < -y \leq -w_1$ and the predicted class will be $-y$ (wrong class) independent of x_1. □

It remains to show that adversarial training for this classification task with $\varepsilon > 2\eta$ will results in a classifier that has relies solely on the first feature.

Lemma B.3.5. *Minimizing the adversarial variant of the loss* (B.1) *results in a classifier that assigns* 0 *weight to features x_i for $i \geq 2$.*

Proof. The optimization problem that adversarial training solves is

$$\min_w \max_{\|\delta\|_\infty \leq \varepsilon} \mathbb{E}\Big[\max(0, 1 - yw^\top(x + \delta))\Big] + \frac{1}{2}\lambda\|w\|_2^2,$$

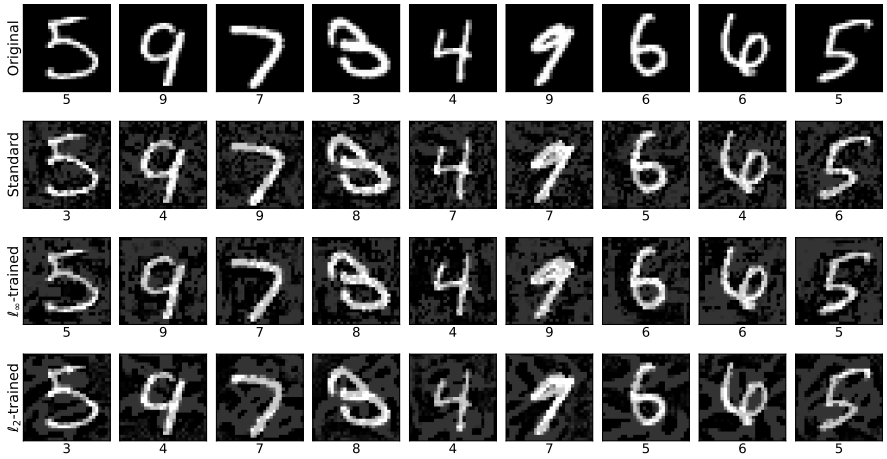
which is equivalent to

$$\min_w \mathbb{E}\Big[\max(0, 1 - yw^\top x + \varepsilon\|w\|_1)\Big] + \frac{1}{2}\lambda\|w\|_2^2.$$

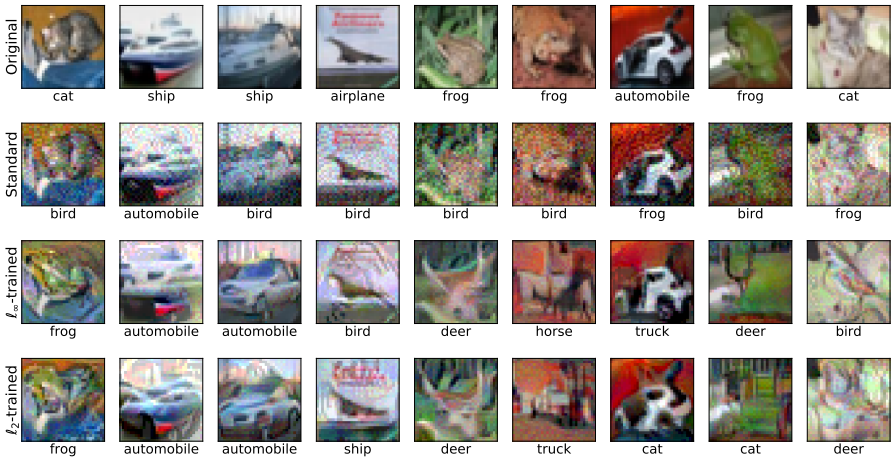
Consider any optimal solution w for which $w_i > 0$ for some $i > 2$. The contribution of terms depending on w_i to $1 - yw^\top x + \varepsilon\|w\|_1$ is a normally-distributed random variable with mean $2\eta - \varepsilon \leq 0$. Since the mean is non-positive, setting w_i to zero can only decrease the margin term of the loss. At the same time, setting w_i to zero *strictly* decreases the regularization term, contradicting the optimality of w. □

Clearly, such a classifier will have standard and adversarial accuracy of p against any $\varepsilon < 1$ since such a value of ε is not sufficient to change the sign of the first feature. This concludes the proof of the theorem.

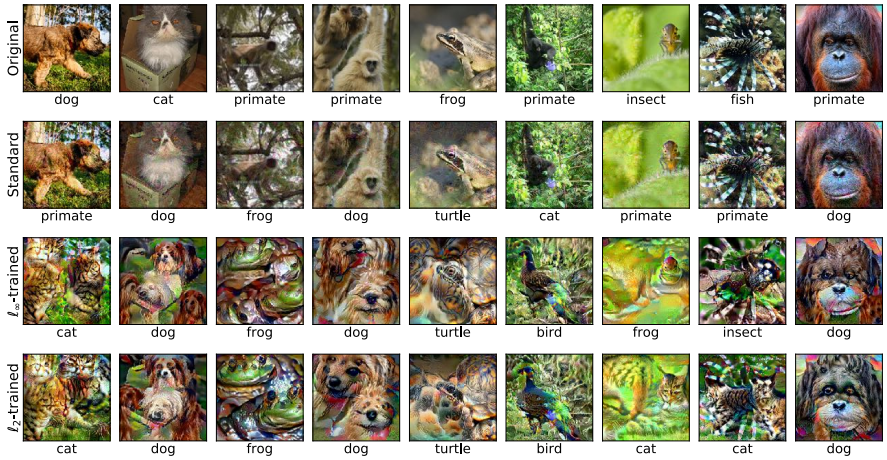
B.4 Additional figures



(a) MNIST

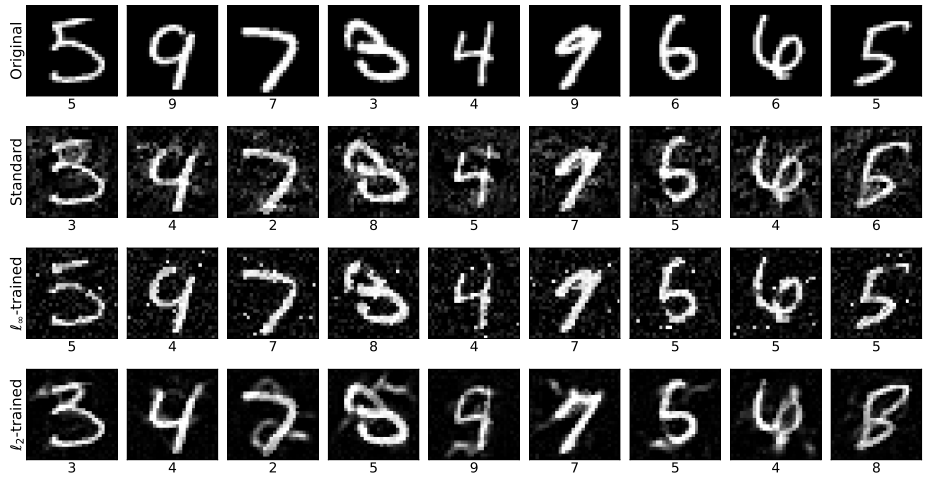


(b) CIFAR-10

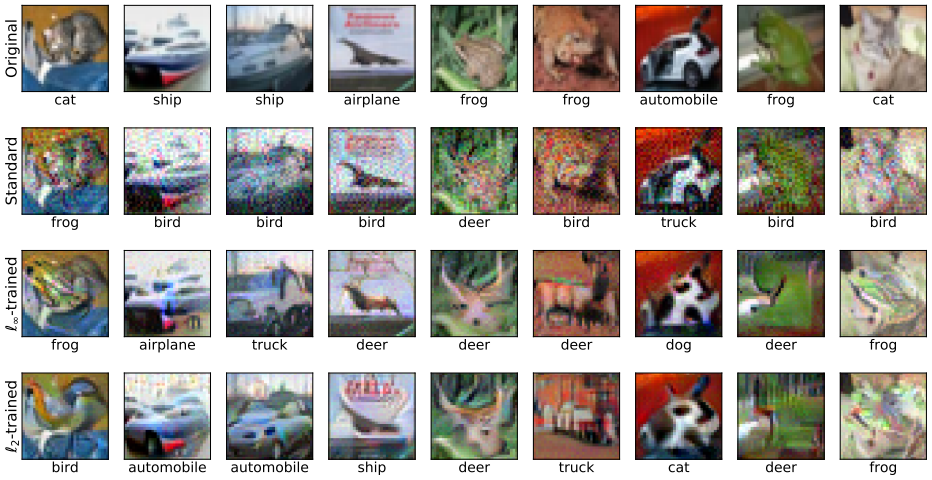


(c) Restricted ImageNet

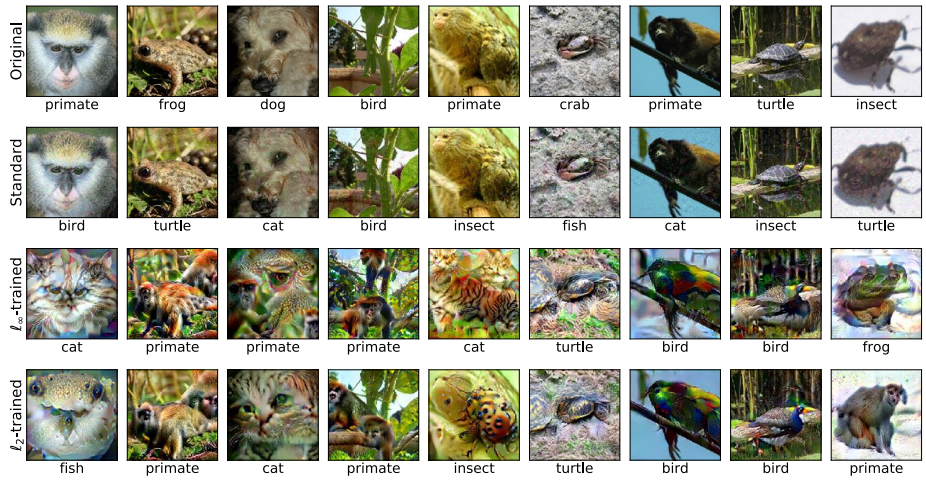
Figure B.8: Large-ε perturbations, bounded in ℓ_∞-norm, similar to those in Figure 2.7.



(a) MNIST



(b) CIFAR-10



(c) Restricted ImageNet

Figure B.9: Large-ε adversarial examples, bounded in ℓ_2-norm, similar to those in Figure 2.7.

B.4.1 Inverting representations

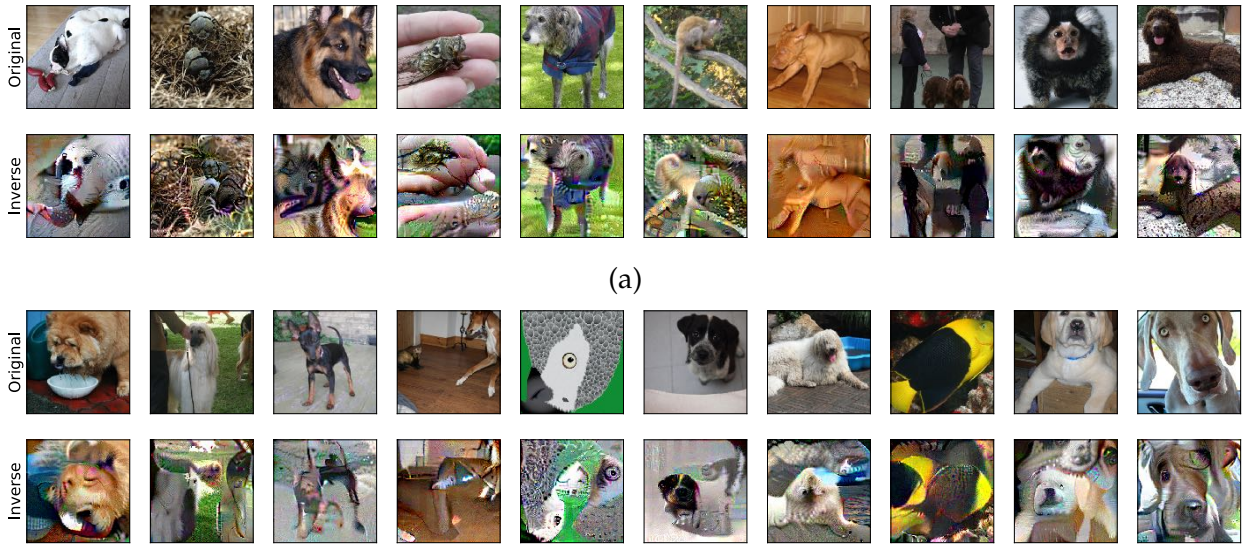
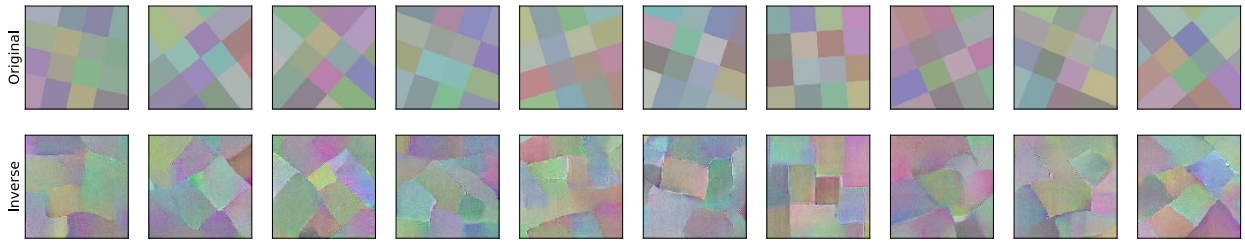
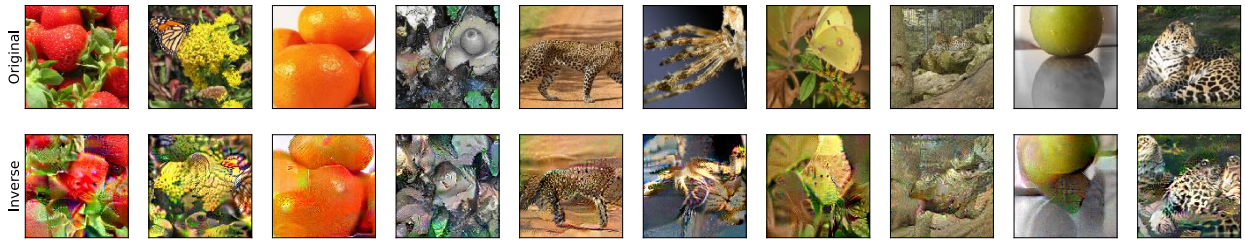


Figure B.10: Robust representations yield semantically meaningful inverses: *Original*: randomly chosen test set images from the Restricted ImageNet dataset; *Inverse*: images obtained by inverting the representation of the corresponding image in the top row starting from: (a) different test images and (b) Gaussian noise.

Recovering out-of-distribution inputs using robust representations



(a) Random kaleidoscope patterns.



(b) Samples from other ImageNet classes outside what the model is trained on.

Figure B.11: Robust representations yield semantically meaningful inverses: (*Original*): randomly chosen out-of-distribution inputs; (*Inverse*): images obtained by inverting the representation of the corresponding image in the top row starting from Gaussian noise.

Inverting standard representations

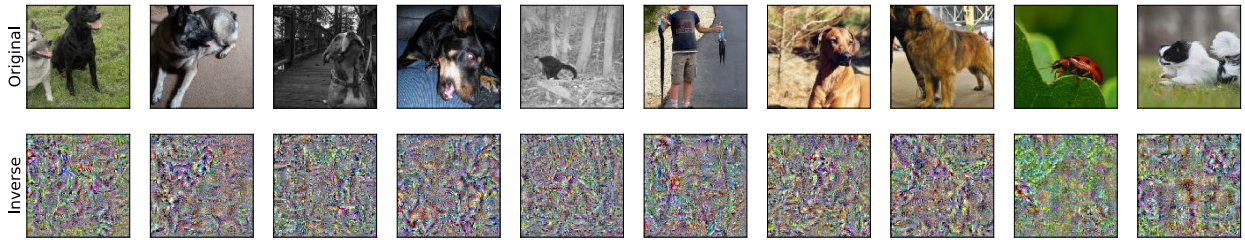


Figure B.12: Standard representations *do not* yield semantically meaningful inverses: (*Original*): randomly chosen test set images from the Restricted ImageNet dataset; (*Inverse*): images obtained by inverting the representation of the corresponding image in the top row starting from Gaussian noise.

B.4.2 Direct feature visualizations for standard and robust models

Additional feature visualizations for the Restricted ImageNet dataset

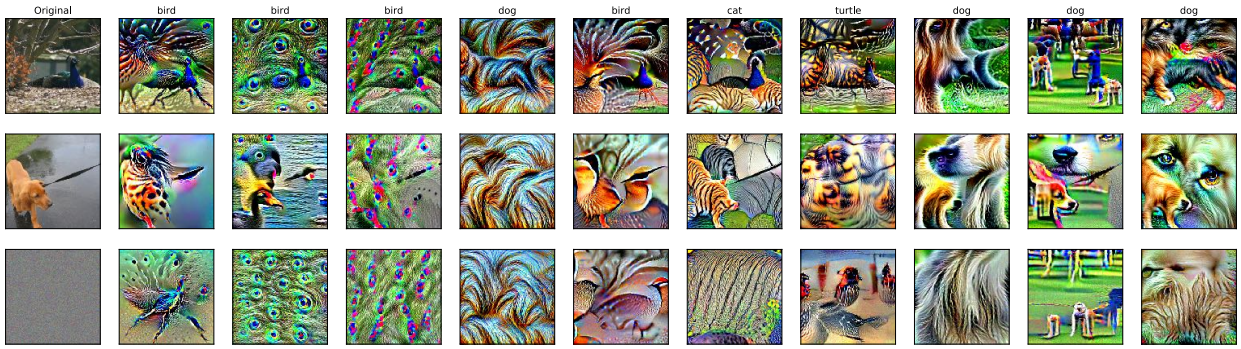


Figure B.13: Correspondence between image-level features and representations learned by a robust model on the Restricted ImageNet dataset. Starting from randomly chosen seed inputs (noise/images), we use a constrained optimization process to identify input features that maximally activate a given component of the representation vector. Specifically, (*left column*): inputs to the optimization process, and (*subsequent columns*): features that activate randomly chosen representation components, along with the predicted class of the feature.

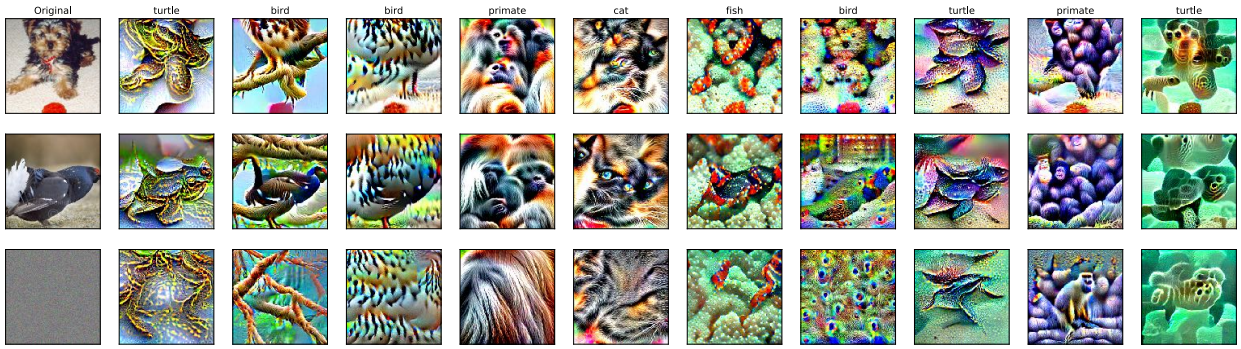


Figure B.14: Correspondence between image-level features and representations learned by a robust model on the Restricted ImageNet dataset. Starting from randomly chosen seed inputs (noise/images), we use a constrained optimization process to identify input features that maximally activate a given component of the representation vector. Specifically, (*left column*): inputs to the optimization process, and (*subsequent columns*): features that activate select representation components, along with the predicted class of the feature.

Feature visualizations for the ImageNet dataset

Figure B.15: Correspondence between image-level patterns and activations learned by standard and robust models on the complete 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 original inputs (selected *randomly*), and in subsequent columns we visualize the result of the optimization (2.6) for different activations, with each row starting from the same (far left) input for (*top*): a robust (adversarially trained) ResNet-50 model, (*middle*): a standard ResNet-50 model and (*bottom*): a standard VGG16 model.

B.4.3 Additional examples of feature manipulation

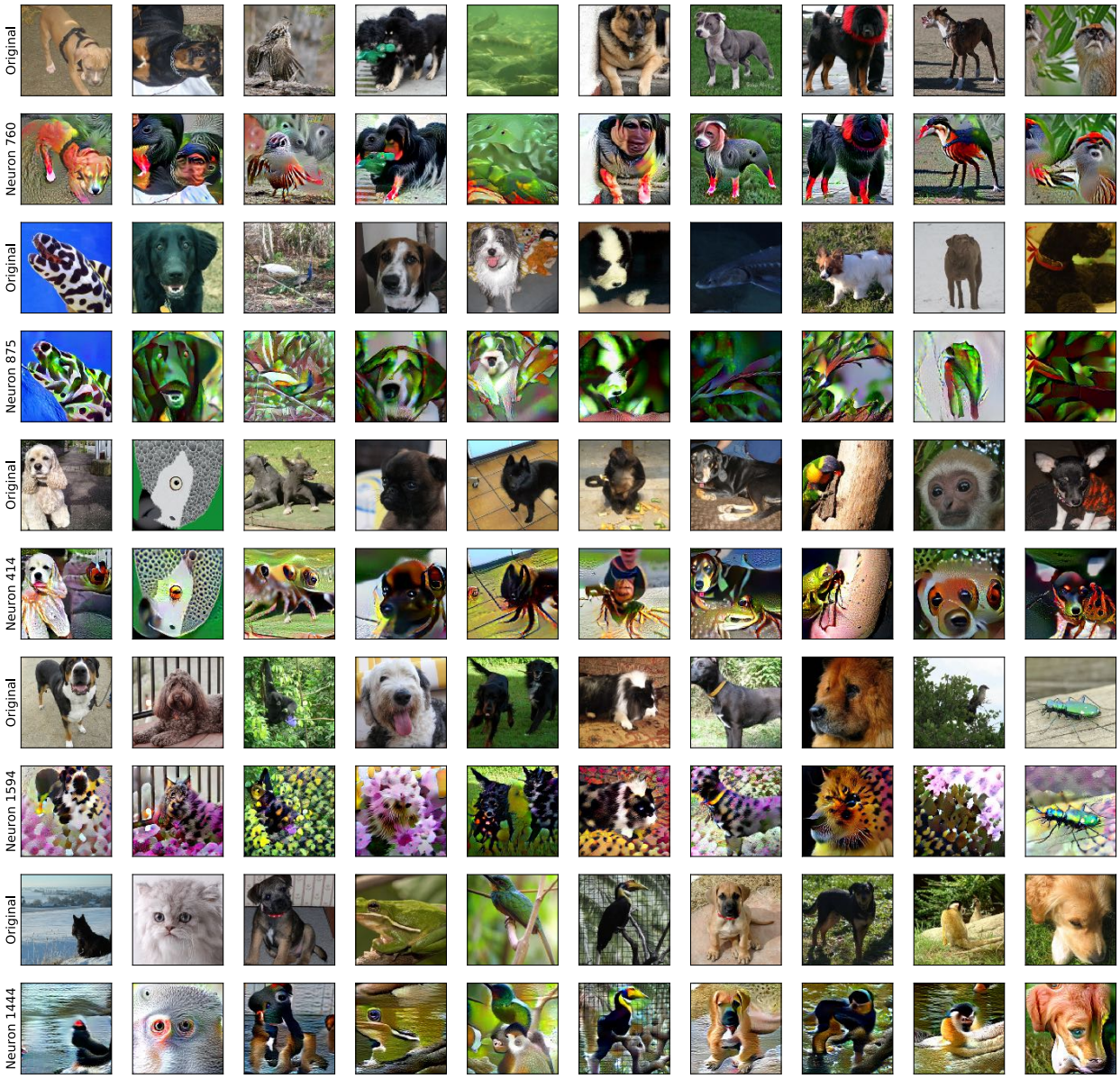


Figure B.16: Visualization of the results adding various neurons, labelled on the left, to randomly chosen test images. The rows alternate between the original test images, and those same images with an additional feature arising from maximizing the corresponding neuron.

B.4.4 Image generation

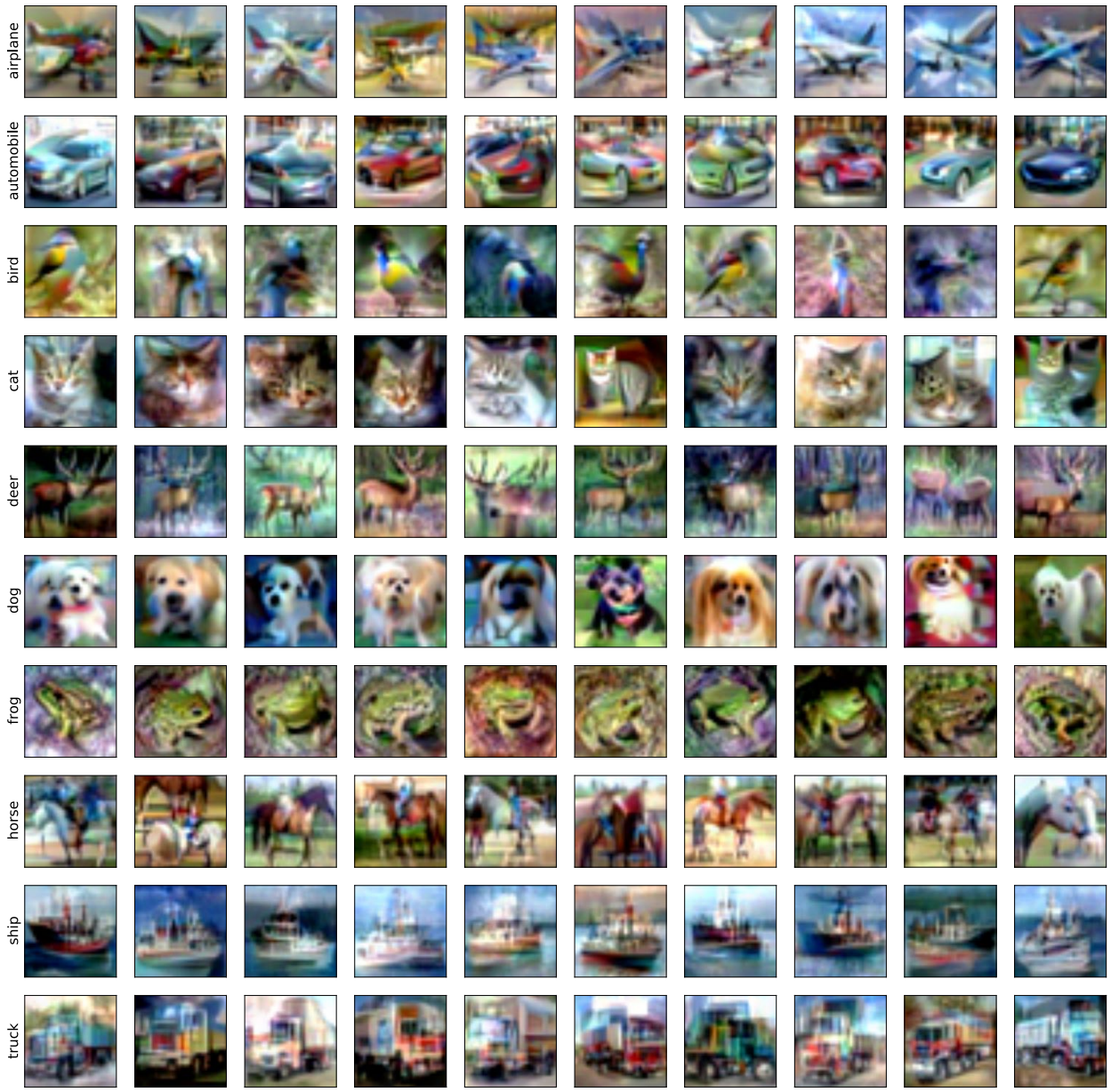


Figure B.17: Random samples generated for the CIFAR dataset.

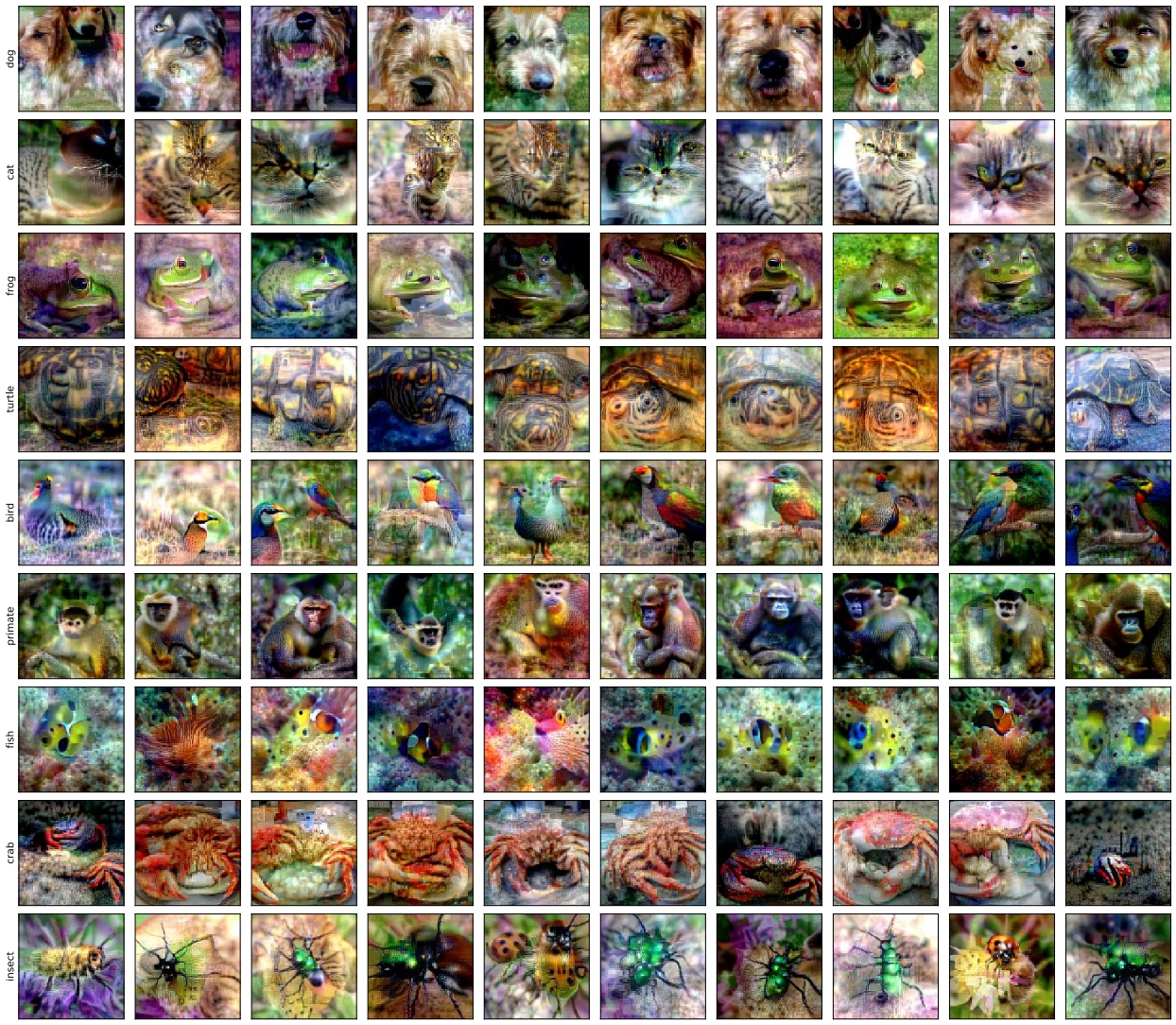


Figure B.18: Random samples generated for the Restricted ImageNet dataset.

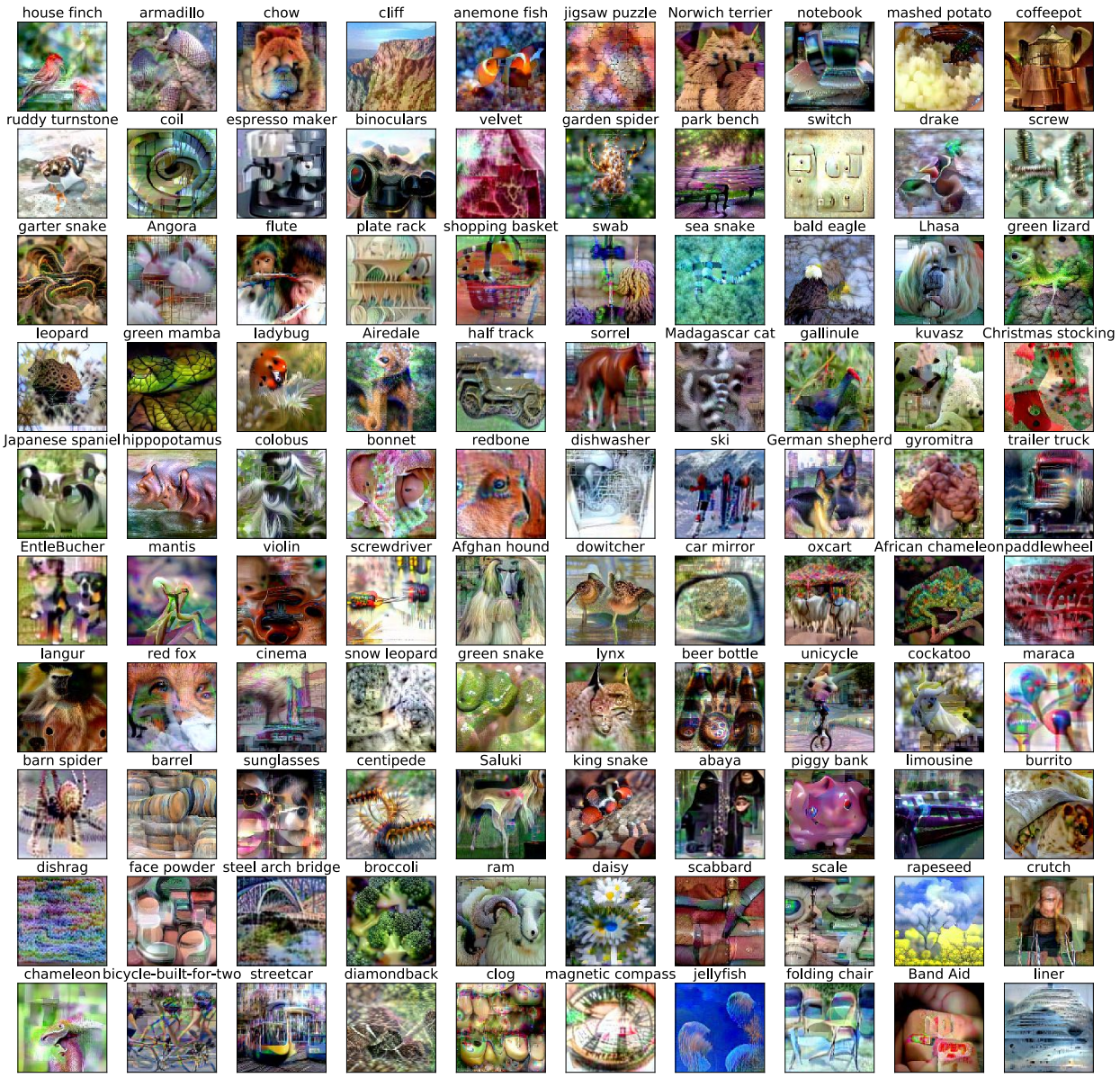


Figure B.19: Random samples generated for the ImageNet dataset.

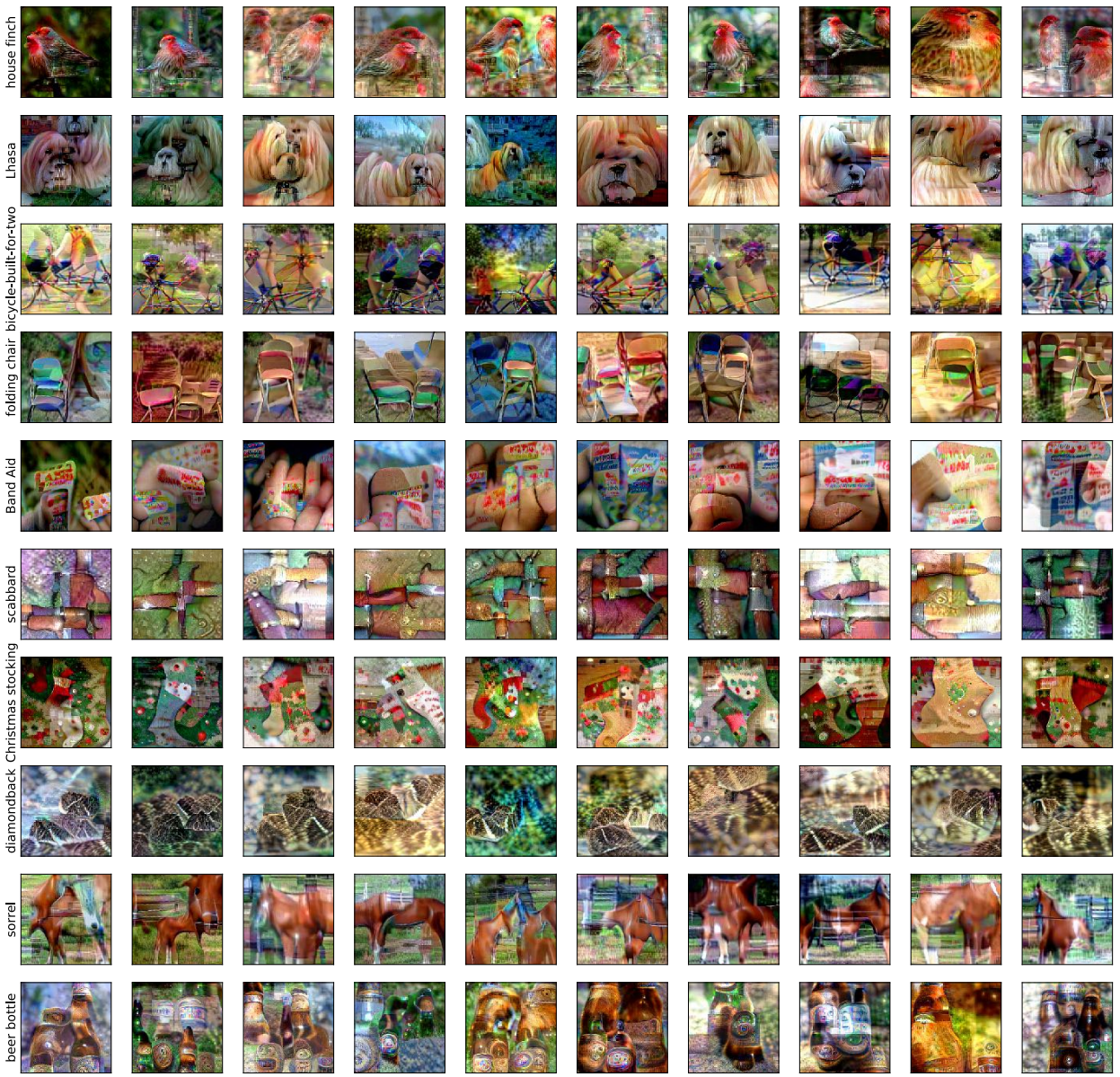


Figure B.20: Random samples from a random class subset.

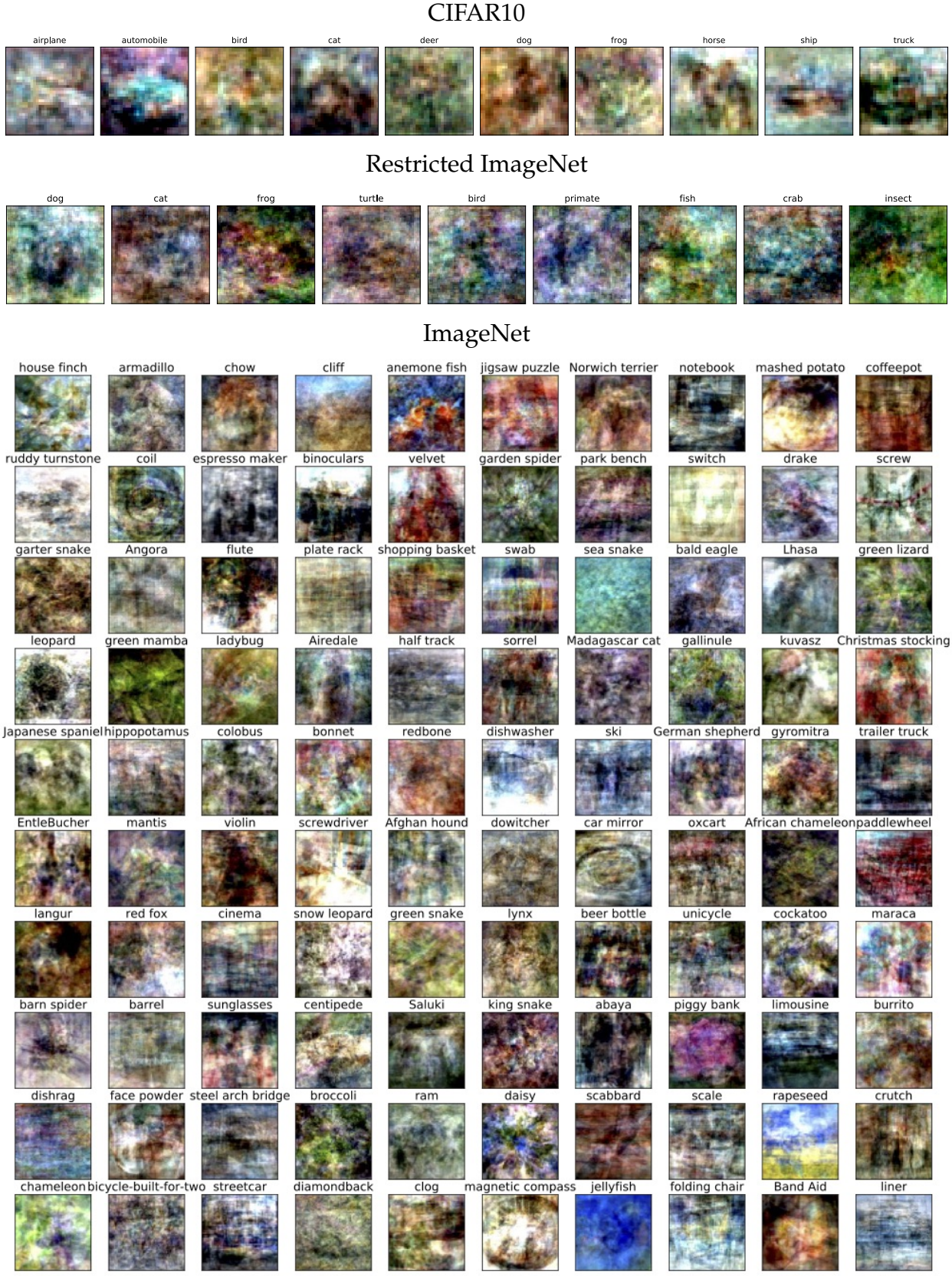


Figure B.21: Samples from class-conditional multivariate normal distributions used as a seed for the generation process.

B.4.5 Image-to-image translation

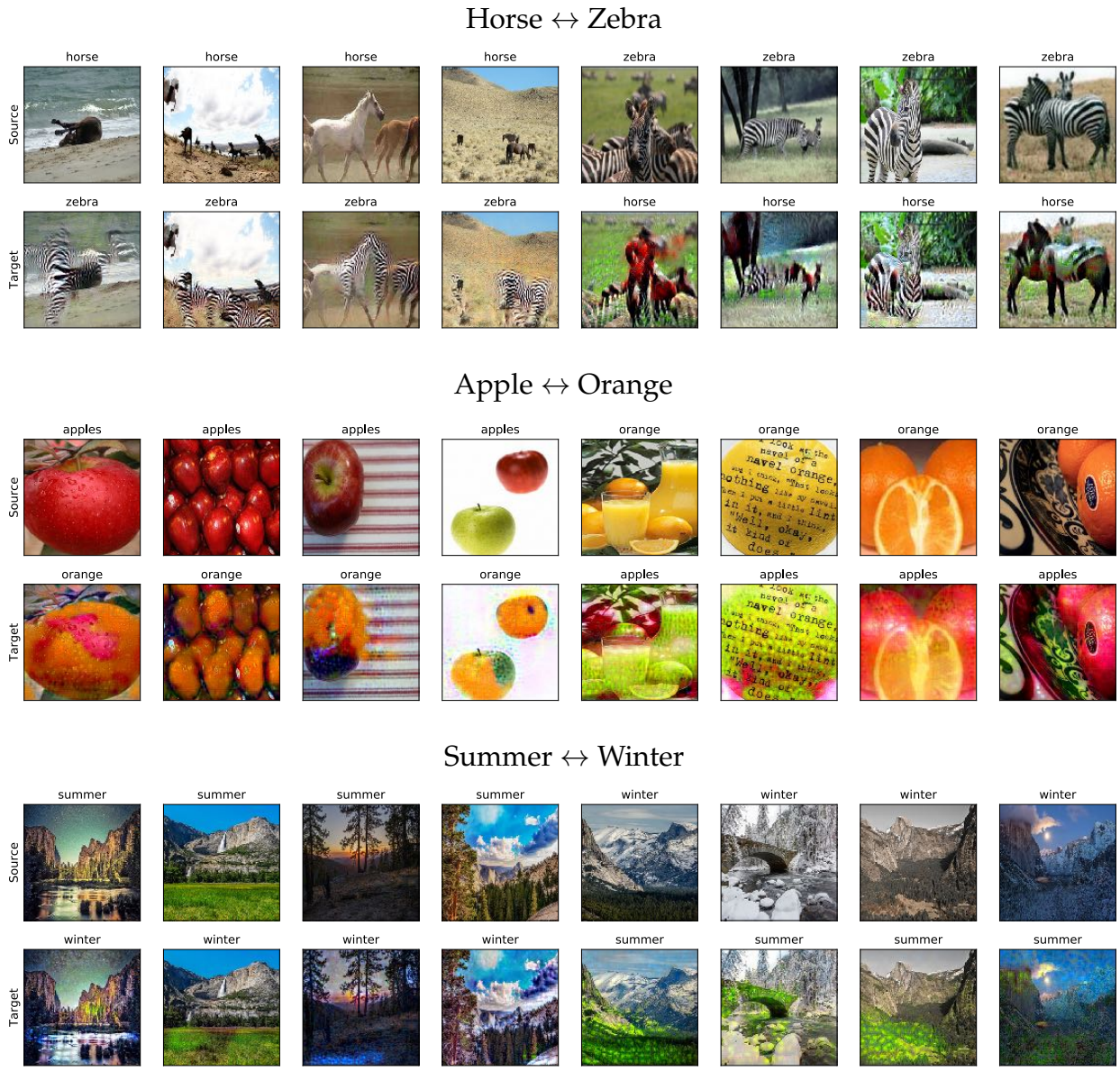


Figure B.22: Random samples for image-to-image translation on the Horse ↔ Zebra, Apple ↔ Orange, and Summer ↔ Winter datasets [Zhu+17]. Details in Appendix B.2.

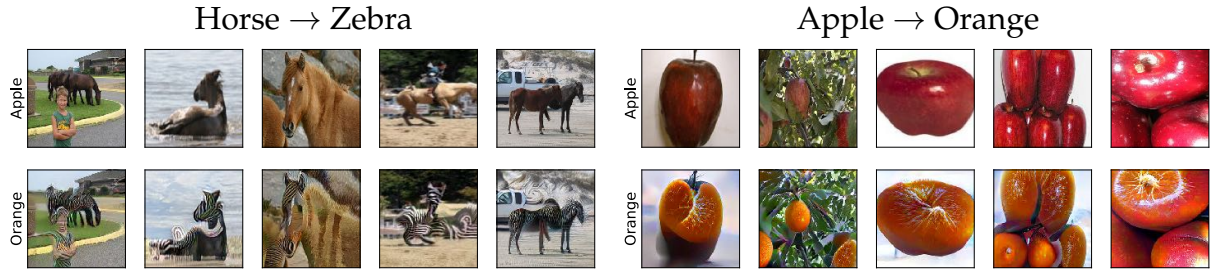


Figure B.23: Random samples for image-to-image translation on the Horse ↔ Zebra and Apple ↔ Orange datasets [Zhu+17] using the *same* robust model trained on the *entire ImageNet* dataset. Here we use ImageNet classes "zebra" (340) and "orange" (950).

Figure B.24: Training an ℓ_∞-robust model on the Horse ↔ Zebra dataset does not lead to plausible image-to-image translation. The model appears to associate "horse" with "blue sky" in which case the zebra to horse translation does not behave as expected.

B.4.6 Inpainting

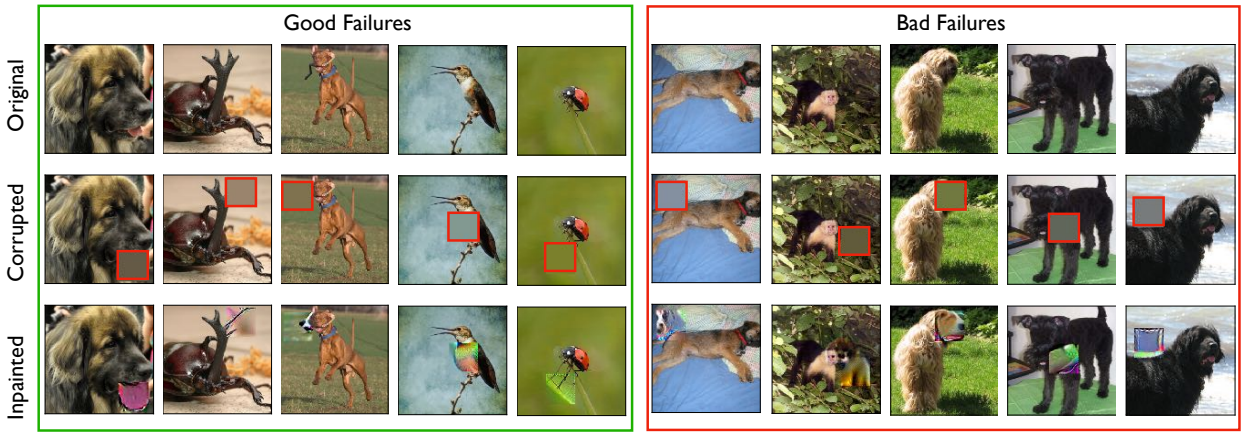


Figure B.25: Failure cases for image inpainting using robust models – *top:* original, *middle:* corrupted and *bottom:* inpainted samples. To recover missing regions, we use PGD to maximise the class score of the image under a robust model while penalizing changes to the uncorrupted regions. The failure 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.

Appendix C

Additional details for Chapter 3

C.1 Experimental setup for Section 3.1

C.1.1 Dataset

We perform our analysis on the ILSVRC2012 dataset [Rus+15]. This dataset contains a thousand classes from the ImageNet dataset [Den+09] with an independently collected validation set. The classes are part of the broader hierarchy, WordNet [Mil95], through which words are organized based on their semantic meaning. We use this hierarchy as a starting point of our investigation but modify it as described in Appendix C.1.5.

For all the BREEDS superclass classification tasks, the train and validation sets are obtained by aggregating the train and validation sets of the descendant ImageNet classes (i.e., subpopulations). Specifically, for a given subpopulation, the training and test splits from the original ImageNet dataset are used as is.

C.1.2 Pipeline formalization

Recall that our process for evaluating model robustness under subpopulation shift (cf. Section 3.1) is as follows. We present the pseudocode for this process in Algorithm 1.

1. Choose a level in the hierarchy and use it to define a set of superclasses by grouping the corresponding dataset classes together. Note that the original dataset classes form the subpopulations of the superclasses.

2. For every superclass, select a (random) set of subpopulations (i.e., classes in the original dataset) and use them to train the model to distinguish between superclasses (we call this the source domain).

3. For every superclass, use the remaining unseen subpopulations (i.e., classes in the original dataset) to test how well the model can distinguish between the superclasses (target domain).

Algorithm 1 The BREEDS methodology. Evaluating the training method *train* on level L of the hierarchy H—restricted to the subtree under *root*—using N_{sub} subpopulations per superclass.

```
function createDatasets(H, L, N_sub, root):
    source, target ← [], []
    for node ∈ H do
        if node.depth = L and root ∈ node.ancestors and len(node.leaves) ≥ N_sub then
            y ← node.label
            subclasses ← random.choice(node.leaves, N_sub)
            for (i, c) ∈ enumerate(subclasses) do
                if i ≤ N_sub / 2 then
                    domain ← source
                else
                    domain ← target
                for x ∈ c.inputs do
                    domain.append((x, y))
    return (source, target)

function evaluateMethod(train, H, L, N_sub, root):
    source, target ← createDatasets (H, L, N_sub, root)
    model ← train(source)
    correct, total ← 0, 0
    for (x, y) ∈ target do
        correct += (model(x) = y)
        total += 1
    targetAccuracy ← correct / total
    return targetAccuracy
```

C.1.3 WordNet issues

As discussed in Section 3.1.2, WordNet is a semantic rather than a visual hierarchy. That is, object classes are arranged based on their meaning rather than their visual appearance. Thus, using intermediate nodes for a visual object recognition task is not straightforward. To illustrate this, we examine a sample superclass grouping created by Huh, Agrawal, and Efros [HAE16] via automated bottom-up clustering in Table C.1.

First, we can notice that these superclasses have vastly different granularities. For instance, "organism" contains the entire animal kingdom, hence being much broader than "produce". Moreover, "covering" is rather abstract class, and hence its subclasses often share little visual similarity (e.g., "window shade", "pajama"). Finally, due to the abstract nature of these superclasses, a large number of subclasses overlap—"covering" and "commodity" share 49 ImageNet descendants.

Superclass	Random ImageNet classes
instrumentality	fire engine, basketball, electric fan, wok, thresher, horse cart, harvester, balloon, racket, can opener, carton, gong, unicycle, toilet seat, carousel, hard disc, cello, mousetrap, neck brace, barrel
man-made structure	beacon, yurt, picket fence, barbershop, fountain, steel arch bridge, library, cinema, stone wall, worm fence, palace, suspension bridge, planetarium, monastery, mountain tent, sliding door, dam, bakery, megalith, pedestal
covering	window shade, vestment, running shoe, diaper, sweatshirt, breastplate, shower curtain, shoji, miniskirt, knee pad, apron, pajama, military uniform, theater curtain, jersey, football helmet, book jacket, bow tie, suit, cloak
commodity	espresso maker, maillot, iron, bath towel, lab coat, bow tie, washer, jersey, mask, waffle iron, mortarboard, diaper, bolo tie, seat belt, cowboy hat, wig, knee pad, vacuum, microwave, abaya
organism	thunder snake, stingray, grasshopper, barracouta, Newfoundland, Mexican hairless, Welsh springer spaniel, bluetick, golden retriever, keeshond, African chameleon, jacamar, water snake, Staffordshire bullterrier, Old English sheepdog, pelican, sea lion, wire-haired fox terrier, flamingo, green mamba
produce	spaghetti squash, fig, cardoon, mashed potato, pineapple, zucchini, broccoli, cauliflower, butternut squash, custard apple, pomegranate, strawberry, Granny Smith, lemon, head cabbage, artichoke, cucumber, banana, bell pepper, acorn squash

Table C.1: Superclasses constructed by Huh, Agrawal, and Efros [HAE16] via bottom-up clustering of WordNet to obtain 36 superclasses—for brevity, we only show superclasses with at least 20 ImageNet classes each.

C.1.4 Manual calibration

We manually modify the WordNet hierarchy according to the following two principles so as to make it better aligned for visual object recognition.

1. Nodes should be grouped together based on their visual characteristics, rather than abstract relationships like functionality—e.g., we eliminate nodes that do not convey visual information such as "covering".

2. Nodes of similar specificity should be at the same distance from the root, irrespective of how detailed their categorization within WordNet is—for instance, we placed "dog" at the same level as "cat" and "flower", even though the "dog" sub-tree in WordNet is much larger.

Finally, we removed a number of ImageNet classes that did not naturally fit into the hierarchy. Concretely, we modified the WordNet hierarchy by applying the following operations:

- *Collapse node*: Delete a node from the hierarchy and add edges from each parent to each child. Allows us to remove redundant or overly specific categorization while preserving the overall structure.

- *Insert node above*: Add a dummy parent to push a node further down the hierarchy. Allows us to ensure that nodes of similar granularity are at the same level.

- *Delete node*: Remove a node and all of its edges. Used to remove abstract nodes that do not reveal visual characteristics.

- *Add edge*: Connect a node to a parent. Used to reassign the children of nodes deleted by the operation above.

We manually examined the hierarchy and implemented these actions in order to produce superclasses that are calibrated for classification. The resulting hierarchy contains nodes of comparable granularity at the same level. Moreover, as a result of this process, each node ends up having a single parent and thus the resulting hierarchy is a tree. The full hierarchy can be explored using the notebooks provided with the hierarchy in the Supplementary Material.

C.1.5 Resulting hierarchy

The parameters for constructing the BREEDS benchmarks (hierarchy level, number of subclasses, and tree root) are given in Table 3.2. The resulting tasks—obtained by sampling disjoint ImageNet classes (i.e., subpopulations) for the source and target domain—are shown in Tables C.1, C.2, C.3, and C.4. Recall that for each superclass we randomly sample a fixed number of subclasses per superclass to ensure that the dataset is approximately balanced.

Superclass	Source	Target
garment	trench coat, abaya, gown, poncho, military uniform, jersey, cloak, bikini, miniskirt, swimming trunks	lab coat, brassiere, hoopskirt, cardigan, pajama, academic gown, apron, diaper, sweatshirt, sarong
bird	African grey, bee eater, coucal, American coot, indigo bunting, king penguin, spoonbill, limpkin, quail, kite	prairie chicken, red-breasted merganser, albatross, water ouzel, goose, oystercatcher, American egret, hen, lorikeet, ruffed grouse
reptile	Gila monster, agama, triceratops, African chameleon, thunder snake, Indian cobra, green snake, mud turtle, water snake, loggerhead	sidewinder, leatherback turtle, boa constrictor, garter snake, terrapin, box turtle, ringneck snake, rock python, American chameleon, green lizard
arthropod	rock crab, black and gold garden spider, tiger beetle, black widow, barn spider, leafhopper, ground beetle, fiddler crab, bee, walking stick	cabbage butterfly, admiral, lacewing, trilobite, sulphur butterfly, cicada, garden spider, leaf beetle, longhorned beetle, fly
mammal	Siamese cat, ibex, tiger, hippopotamus, Norwegian elkhound, dugong, colobus, Samoyed, Persian cat, Irish wolfhound	English setter, llama, lesser panda, armadillo, indri, giant schnauzer, pug, Doberman, American Staffordshire terrier, beagle
accessory	bib, feather boa, stole, plastic bag, bathing cap, cowboy boot, necklace, crash helmet, gasmask, maillot	hair slide, umbrella, pickelhaube, mitten, sombrero, shower cap, sock, running shoe, mortarboard, handkerchief
craft	catamaran, speedboat, fireboat, yawl, airliner, container ship, liner, trimaran, space shuttle, aircraft carrier	schooner, gondola, canoe, wreck, warplane, balloon, submarine, pirate, lifeboat, airship
equipment	volleyball, notebook, basketball, hand-held computer, tripod, projector, barbell, monitor, croquet ball, balance beam	cassette player, snorkel, horizontal bar, soccer ball, racket, baseball, joystick, microphone, tape player, reflex camera
furniture	wardrobe, toilet seat, file, mosquito net, four-poster, bassinet, chiffonier, folding chair, fire screen, shoji	studio couch, throne, crib, rocking chair, dining table, park bench, chest, window screen, medicine chest, barber chair
instrument	upright, padlock, lighter, steel drum, parking meter, cleaver, syringe, abacus, scale, corkscrew	maraca, saltshaker, magnetic compass, accordion, digital clock, screw, can opener, odometer, organ, screwdriver

man-made structure	castle, bell cote, fountain, planetarium, traffic light, breakwater, cliff dwelling, monastery, prison, water tower	suspension bridge, worm fence, turnstile, tile roof, beacon, street sign, maze, chainlink fence, bakery, drilling platform
wheeled vehicle	snowplow, trailer truck, racer, shopping cart, unicycle, motor scooter, passenger car, minibus, jeep, recreational vehicle	jinrikisha, golfcart, tow truck, ambulance, bullet train, fire engine, horse cart, streetcar, tank, Model T
produce	broccoli, corn, orange, cucumber, spaghetti squash, butternut squash, acorn squash, cauliflower, bell pepper, fig	pomegranate, mushroom, strawberry, lemon, head cabbage, Granny Smith, hip, ear, banana, artichoke

Table C.1: Superclasses used for the ENTITY-13 task, along with the corresponding subpopulations that comprise the source and target domains.

Superclass	Source	Target
serpentes	green mamba, king snake, garter snake, thunder snake	boa constrictor, green snake, ringneck snake, rock python
passerine	goldfinch, brambling, water ouzel, chickadee	magpie, house finch, indigo bunting, bulbul
saurian	alligator lizard, Gila monster, American chameleon, green lizard	Komodo dragon, African chameleon, agama, banded gecko
arachnid	harvestman, barn spider, scorpion, black widow	wolf spider, black and gold garden spider, tick, tarantula
aquatic bird	albatross, red-backed sandpiper, crane, white stork	goose, dowitcher, limpkin, drake
crustacean	crayfish, spiny lobster, hermit crab, Dungeness crab	king crab, rock crab, American lobster, fiddler crab
carnivore	Italian greyhound, black-footed ferret, Bedlington terrier, basenji	flat-coated retriever, otterhound, Shih-Tzu, Boston bull
insect	lacewing, fly, grasshopper, sulphur butterfly	long-horned beetle, leafhopper, dung beetle, admiral
ungulate	llama, gazelle, zebra, ox	hog, hippopotamus, hartebeest, warthog
primate	baboon, howler monkey, Madagascar cat, chimpanzee	siamang, indri, capuchin, patas
bony fish	coho, tench, lionfish, rock beauty	sturgeon, puffer, eel, gar
barrier	breakwater, picket fence, turnstile, bannister	chainlink fence, stone wall, dam, worm fence
building	bookshop, castle, mosque, butcher shop	grocery store, toyshop, palace, beacon
electronic equipment	printer, pay-phone, microphone, computer keyboard	modem, cassette player, monitor, dial telephone
footwear	clog, Loafer, maillot, running shoe	sandal, knee pad, cowboy boot, Christmas stocking
garment	academic gown, apron, miniskirt, fur coat	jean, vestment, sarong, swimming trunks
headdress	pickelhaube, hair slide, shower cap, bonnet	bathing cap, cowboy hat, bearskin, crash helmet
home appliance	washer, microwave, Crock Pot, vacuum	toaster, espresso maker, space heater, dishwasher
kitchen utensil	measuring cup, cleaver, coffeepot, spatula	frying pan, cocktail shaker, tray, caldron

measuring instrument	digital watch, analog clock, parking meter, magnetic compass	barometer, wall clock, hourglass, digital clock
motor vehicle	limousine, school bus, moped, convertible	trailer truck, beach wagon, police van, garbage truck
musical instrument	French horn, maraca, grand piano, upright	acoustic guitar, organ, electric guitar, violin
neckwear	feather boa, neck brace, bib, Windsor tie	necklace, stole, bow tie, bolo tie
sports equipment	ski, dumbbell, croquet ball, racket	rugby ball, balance beam, horizontal bar, tennis ball
tableware	mixing bowl, water jug, beer glass, water bottle	goblet, wine bottle, coffee mug, plate
tool	quill, combination lock, padlock, screw	fountain pen, screwdriver, shovel, torch
vessel	container ship, lifeboat, aircraft carrier, trimaran	liner, wreck, catamaran, yawl
dish	potpie, mashed potato, pizza, cheeseburger	burrito, hot pot, meat loaf, hotdog
vegetable	zucchini, cucumber, butternut squash, artichoke	cauliflower, spaghetti squash, acorn squash, cardoon
fruit	strawberry, pineapple, jackfruit, Granny Smith	buckeye, corn, ear, acorn

Table C.2: Superclasses used for the ENTITY-30 task, along with the corresponding subpopulations that comprise the source and target domains.

Superclass	Source	Target
salamander	eft, axolotl	common newt, spotted salamander
turtle	box turtle, leatherback turtle	loggerhead, mud turtle
lizard	whiptail, alligator lizard	African chameleon, banded gecko
snake	night snake, garter snake	sea snake, boa constrictor
spider	tarantula, black and gold garden spider	garden spider, wolf spider
grouse	ptarmigan, prairie chicken	ruffed grouse, black grouse
parrot	macaw, lorikeet	African grey, sulphur-crested cockatoo
crab	Dungeness crab, fiddler crab	rock crab, king crab
dog	bloodhound, Pekinese	Great Pyrenees, papillon
wolf	coyote, red wolf	white wolf, timber wolf
fox	grey fox, Arctic fox	red fox, kit fox
domestic cat	tiger cat, Egyptian cat	Persian cat, Siamese cat
bear	sloth bear, American black bear	ice bear, brown bear
beetle	dung beetle, rhinoceros beetle	ground beetle, long-horned beetle
butterfly	sulphur butterfly, admiral	cabbage butterfly, ringlet
ape	gibbon, orangutan	gorilla, chimpanzee
monkey	marmoset, titi	spider monkey, howler monkey

Table C.3: Superclasses used for the LIVING-17 task, along with the corresponding subpopulations that comprise the source and target domains.

Superclass	Source	Target
bag	plastic bag, purse	mailbag, backpack
ball	volleyball, punching bag	ping-pong ball, soccer ball
boat	gondola, trimaran	catamaran, canoe
body armor	bulletproof vest, breastplate	chain mail, cuirass
bottle	pop bottle, beer bottle	wine bottle, water bottle
bus	trolleybus, minibus	school bus, recreational vehicle
car	racer, Model T	police van, ambulance
chair	folding chair, throne	rocking chair, barber chair
coat	lab coat, fur coat	kimono, vestment
digital computer	laptop, desktop computer	notebook, hand-held computer
dwelling	palace, monastery	mobile home, yurt
fence	worm fence, chainlink fence	stone wall, picket fence
hat	bearskin, bonnet	sombrero, cowboy hat
keyboard instrument	grand piano, organ	upright, accordion
mercantile establishment	butcher shop, barbershop	shoe shop, grocery store
outbuilding	greenhouse, apiary	barn, boathouse
percussion instrument	steel drum, marimba	drum, gong
pot	teapot, Dutch oven	coffeepot, caldron
roof	dome, vault	thatch, tile roof
ship	schooner, pirate	aircraft carrier, liner
skirt	hoopskirt, miniskirt	overskirt, sarong
stringed instrument	electric guitar, banjo	violin, acoustic guitar
timepiece	digital watch, stopwatch	parking meter, digital clock
truck	fire engine, pickup	tractor, forklift
wind instrument	oboe, sax	flute, bassoon
squash	spaghetti squash, acorn squash	zucchini, butternut squash

Table C.4: Superclasses used for the NON-LIVING-26 task, along with the corresponding subpopulations that comprise the source and target domains.

C.1.6 Annotator task

As described in Section 3.1.4, the goal of our human studies is to understand whether humans can classify images into superclasses even without knowing the semantic grouping. Thus, the task involved showing annotators two groups of images, each sampled from the source domain of a random superclass. Then, annotators were shown a new set of images from the target domain (or the source domain in the case of control) and were asked to assign each of them into one of the two groups. A screenshot of an (random) instance of our annotator task is shown in Figure C.2.

Each task contained 20 images from the source domain of each superclass and 12 images for annotators to classify (the images where rescaled and center-cropped to size 224×224 to match the input size use for model predictions). The two superclasses were randomly permuted at load time. To ensure good concentration of our accuracy estimates, for every superclass, we performed binary classification tasks w.r.t. 3 other (randomly chosen) superclasses. Further, we used 3 annotators per task and annotators were compensated \$0.15 per task.

Comparing with the original hierarchy. In order to compare our superclasses with those obtained by Huh, Agrawal, and Efros [HAE16] via WordNet clustering,[1] we need to define a correspondence between them. To do so, for each of our tasks, we selected the clustering (either top-down or bottom-up) that had the closest number of superclasses. Following the terminology from that work, this mapping is: ENTITY-13 $\rightarrow$ DOWNUP-36, ENTITY-30 $\rightarrow$ UPDOWN-127, LIVING-17 $\rightarrow$ DOWNUP-753 (restricted to "living" nodes), and NON-LIVING-26 $\rightarrow$ DOWNUP-345 (restricted to "non-living" nodes).

[1] https://github.com/minyoungg/wmigftl/tree/master/label_sets/hierarchy

Figure C.2: Sample MTurk annotation task to obtain human baselines for BREEDS benchmarks.

C.1.7 Evaluating model performance

Model architectures and training

The model architectures used in our analysis are in Table C.3 for which we used standard implementations from the PyTorch library (https://pytorch.org/docs/stable/torchvision/models.html). For training, we use a batch size of 128, weight decay of 10^{-4}, and learning rates listed in Table C.3. Models were trained until convergence. On ENTITY-13 and ENTITY-30, this required a total of 300 epochs, with 10-fold drops in learning rate every 100 epochs, while on LIVING-17and NON-LIVING-26, models a total of 450 epochs, with 10-fold learning rate drops every 150 epochs. For adapting models, we retrained the last (fully-connected) layer on the train split of the target domain, starting from the parameters of the source-trained model. We trained that layer using SGD with a batch size of 128 for 40,000 steps and chose the best learning rate out of $[0.01, 0.1, 0.25, 0.5, 1.0, 2.0, 3.0, 5.0, 7.0, 8.0, 10.0, 11.0, 12.0]$, based on test accuracy.

Model	Learning Rate
`alexnet`	0.01
`vgg11`	0.01
`resnet18`	0.1
`resnet34`	0.1
`resnet50`	0.1
`densenet121`	0.1

Table C.3: Models used in our analysis.

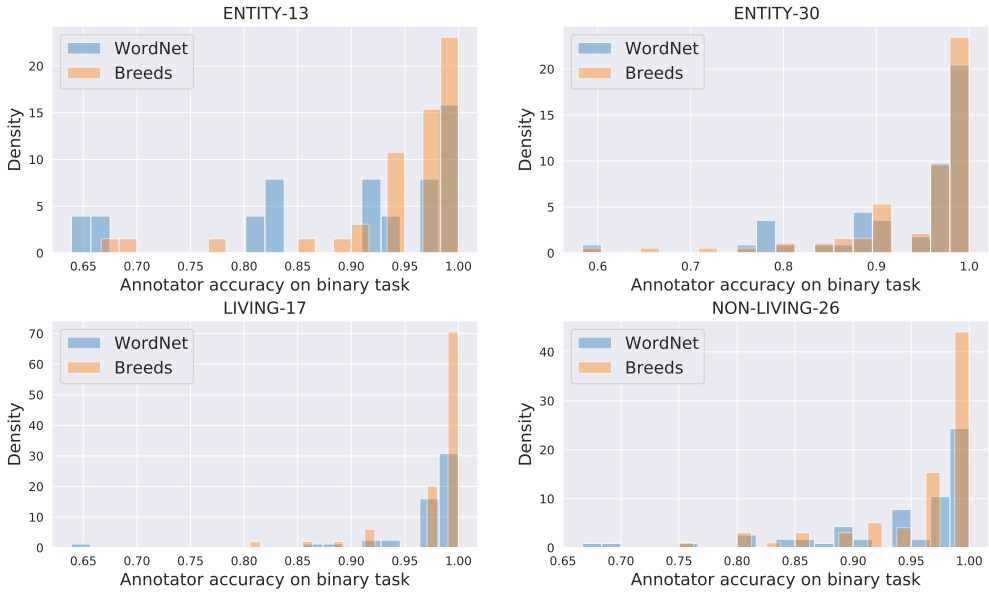
Model pairwise accuracy

In order to make a fair comparison between the performance of models and human annotators on the BREEDS tasks, we evaluate model accuracy on pairs of superclasses. On images from that pair, we determine the model prediction to be the superclass for which the model's predicted probability is higher. A prediction is deemed correct if it matches the superclass label for the image. Repeating this process over random pairs of superclasses allows us to estimate model accuracy on the average-case binary classification task.

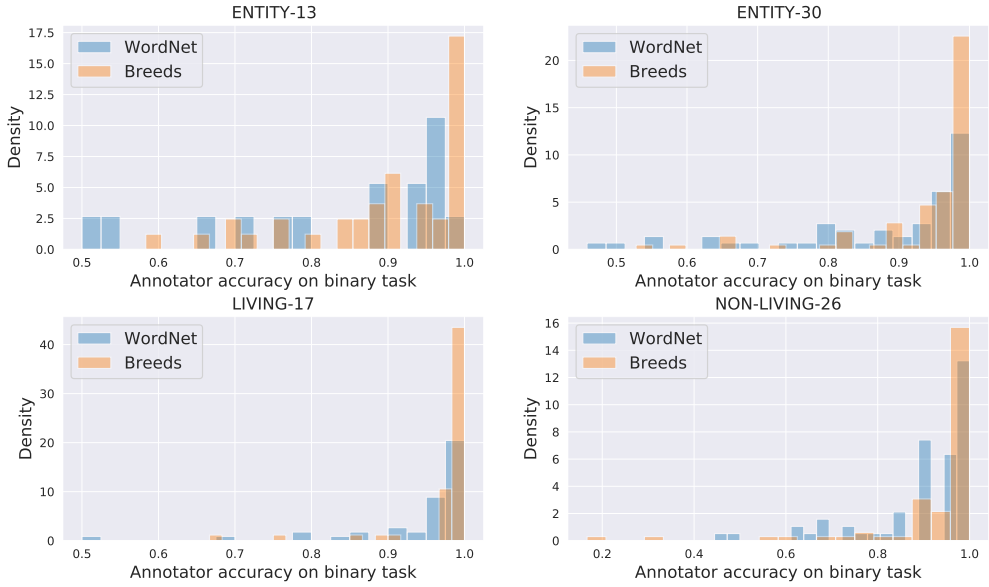
C.2 Additional experimental results

C.2.1 Human baselines for BREEDS tasks

In Section 3.1.4, we evaluate human performance on binary versions of our BREEDS tasks. Appendix Figures C.4a and C.4b show the distribution of annotator accuracy over different pairs of superclasses for test data sampled from the source and target domains respectively.



(a) Source domain (no subpopulation shift)



(b) Target domain (with subpopulation shift)

Figure C.4: Distribution of annotator accuracy over pairwise superclass classification tasks. We observe that human annotators consistently perform better on tasks constructed using our modified ImageNet class hierarchy (i.e., BREEDS) as opposed to those using WordNet.

C.2.2 Model evaluation

In Figures C.5- C.7, we visualize model performance over BREEDS superclasses for different model architectures. We observe in general that models perform fairly uniformly over classes when the test data is drawn from the source domain. This indicates that the tasks are well-calibrated—the various superclasses are of comparable difficulty. At the same time, we see that model robustness to subpopulation shift, i.e., drop in accuracy on the target domain, varies widely over superclasses. This could be either due to some superclasses being broader by construction or due to models being more sensitive to subpopulation shift for some classes.

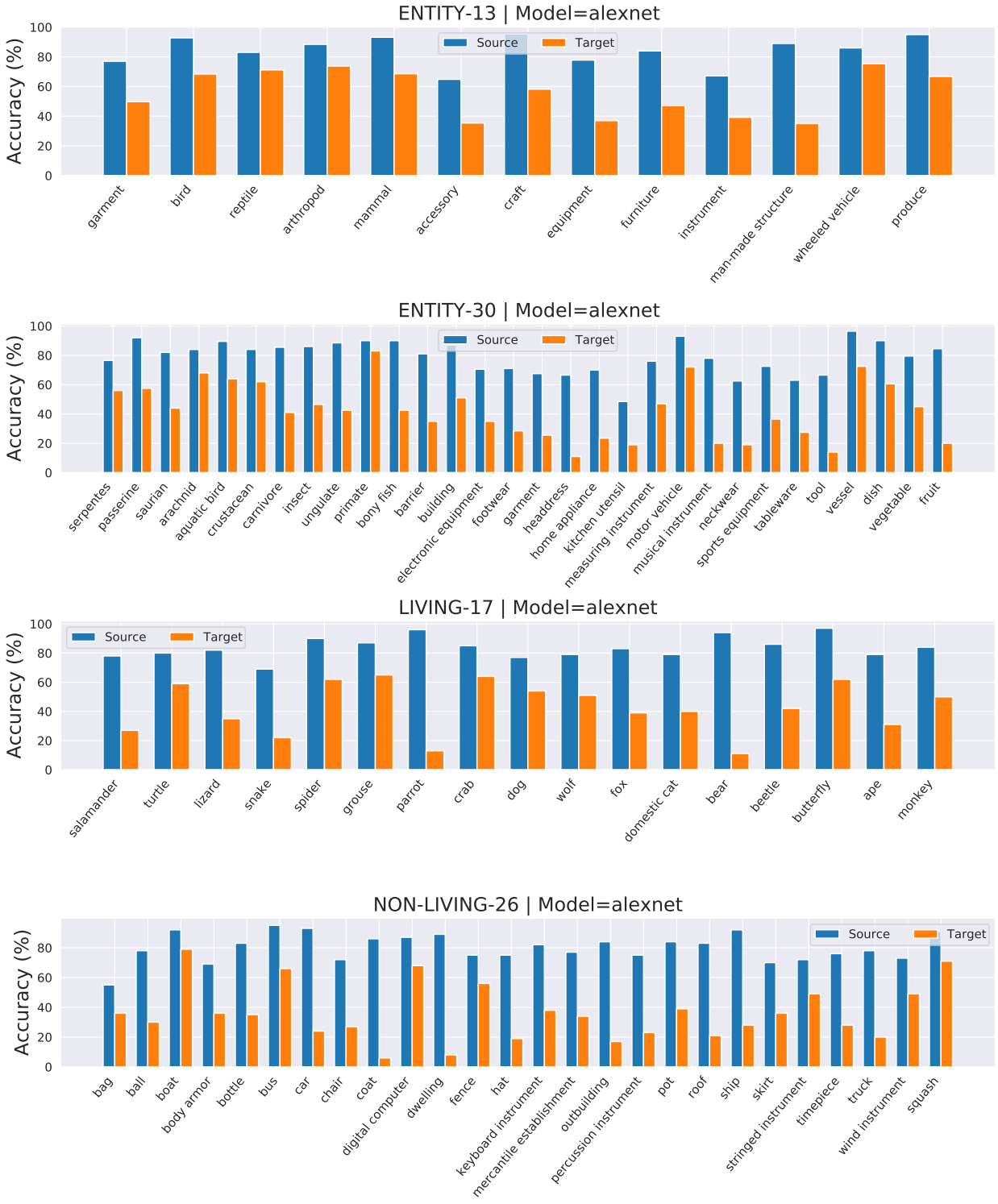


Figure C.5: Per-class source and target accuracies for AlexNet on BREEDS tasks.

Figure C.6: Per-class source and target accuracies for ResNet-50 on BREEDS tasks.

Figure C.7: Per-class source and target accuracies for DenseNet-121 on BREEDS tasks.

Effect of different splits

As described in Section 3.1, to create BREEDS tasks, we first identify a set of relevant superclasses (at the chosen depth in the hierarchy), and then partition their subpopulations between the source and target domains. For all the tasks listed in Table 3.2, the superclasses are balanced—each of them comprise the same number of subpopulations. To ensure this is the case, the desired number of subpopulations is chosen among all superclass subpopulations at random. These subpopulations are then randomly split between the source and target domains.

Instead of randomly partitioning subpopultions (of a given superclass) between the two domains, we could instead craft partitions to be more/less adversarial as illustrated in Figure C.8. Specifically, we could control how similar the subpopulations in the target domain are to those in the source domain. For instance, a split would be less adversarial (*good*) if subpopulations in the source and target domain share a common parent. On the other hand, we could make a split more adversarial (*bad*) by ensuring a greater degree of separation (in terms of distance in the hierarchy) between the source and target domain subpopulations.

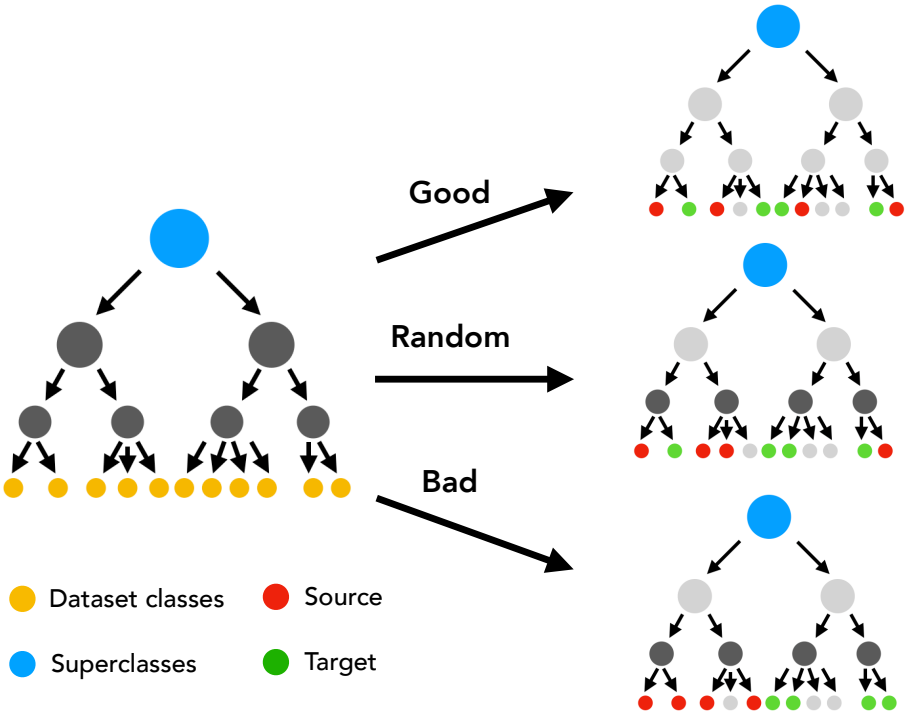
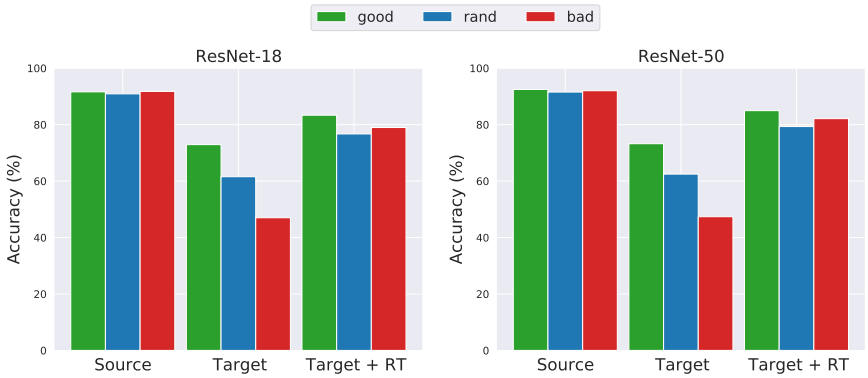


Figure C.8: Different ways to partition the subpopulations of a given superclass into the source and target domains. Depending on how closely related the subpopulations in the two domain are, we can construct splits that are more/less adversarial.

We now evaluate model performance under such variations in the nature of the splits themselves—see Figure C.9. As expected, models perform comparably well on test data from the source domain, independent of the how the subpopulations are partitioned into the two domains. However, model robustness to subpopulation shift varies considerably based on the nature of the split—it is lowest for the most adversarially chosen split. Finally, we observe that retraining the linear layer on data from the target domain recovers a considerable fraction of the accuracy drop in all cases—indicating

that even for the more adversarial splits, models do learn features that transfer well to unknown subpopulations.



(a) ENTITY-13 task

(b) ENTITY-30 task

Figure C.9: Model robustness as a function of the nature of subpopulation shift within specific BREEDS tasks. We vary how the underlying subpopulations of each superclass are split between the source and target domain—we compare random splits (used in the majority of our analysis), to ones that are more (*bad*) or less adversarial (*good*). When models are tested on samples from the source domain, they perform equally well across different splits, as one might expect. However, under subpopulation shift (i.e., on samples from the target domain), model robustness varies drastically, and is considerably worse when the split is more adversarial. Yet, for all the splits, models have comparable target accuracy after retraining their final layer.

C.3 Experimental details for Section 3.2

C.3.1 Experimental setup

For our experimental analysis 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 and editing are performed on (or using) samples from the standard test sets to avoid overlap with the training data used to develop the models.

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 C.10.

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.

	Test Accuracy (%)	
Architecture \ Dataset	ImageNet	Places
VGG	73.70	54.02
ResNet	75.77	54.24
CLIP-ResNet	59.84	-

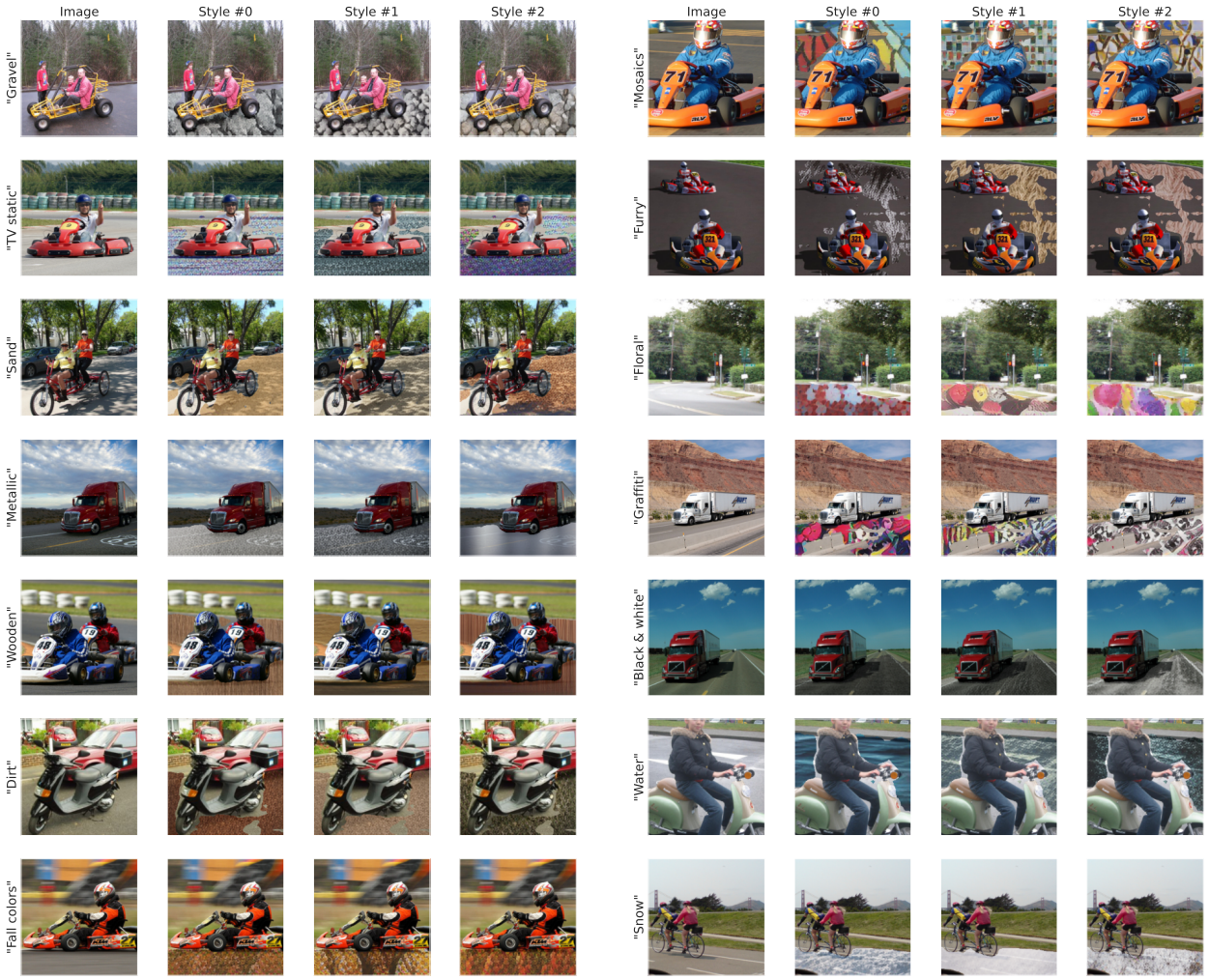
Table C.10: Accuracy of each model architecture on the datasets used in our analysis.

C.3.2 Concept transformation pipeline

Recall that our pipeline consists of two steps: concept detection and concept transformation (Section 3.2). We describe each step below and provide examples in Figure C.11.

We detect concepts using pre-trained object detectors trained on MS-COCO [Lin+14] and LVIS [GDG19]. For MS-COCO, we use a model with a ResNet-101 backbone[3] which is trained on COCO-Stuff[4] annotations and can detect 182 concepts. For LVIS, we use a pre-trained model from the Detectron [Gir+18] model zoo[5], which can detect 1230 classes. We only consider a prediction as

[2] https://pytorch.org/vision/stable/models.html
[3] https://github.com/kazuto1011/deeplab-pytorch
[4] https://github.com/nightrome/cocostuff
[5] https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md



(a) (b)

Figure C.11: Illustration of concept-level transformations in ImageNet: 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., "gravel"). We collect three examples per style, which are then split across training and testing.

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 of Ghiasi et al. [Ghi+17] using their pre-trained model[6]. 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 C.11) 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

[6] https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2

used.

All the pre-trained models used are open-sourced and freely available for non-commercial research.

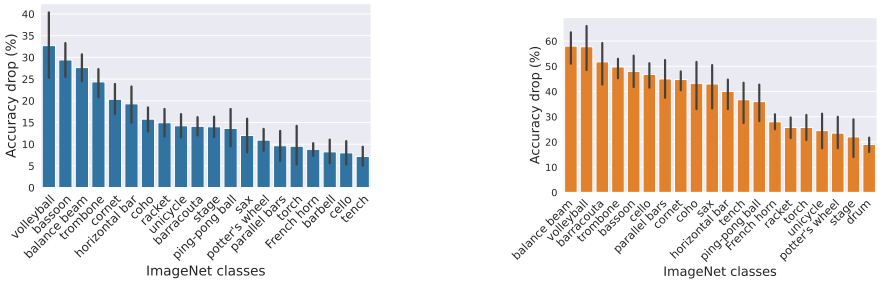
C.4 Additional experiments

Here, we expand on our analysis in Section 3.2 so as to characterize the effect of concept-level transformations on classifiers.

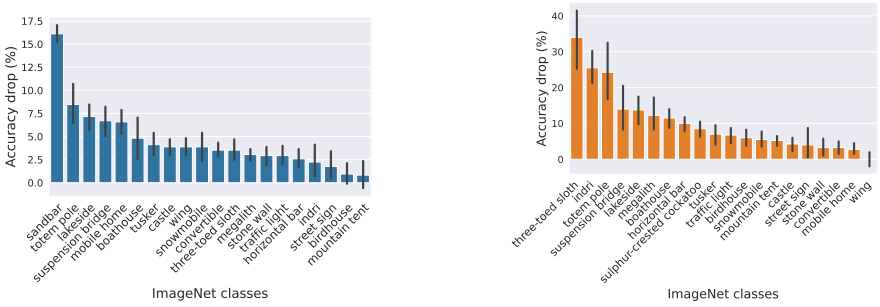
Per concept. In Figure C.12, we visualize the accuracy drop induced by transformations of a specific concept for classifiers trained on ImageNet and Places-365 (similar to Figure 3.7a). Here, the accuracy drop post-transformation is measured only on images that contain the concept of interest. We then 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 C.13, 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".

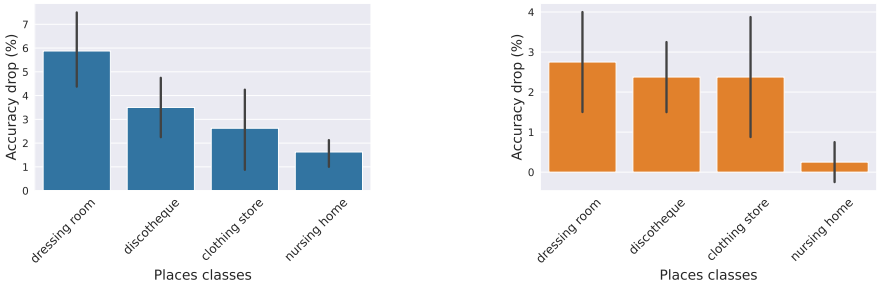
Per class (prediction-rules). In Figure C.14, we provide additional examples of class-level prediction rules identified using our methodology. Specifically, for each class, the highlighted concepts are those that hurt model accuracy when transformed.



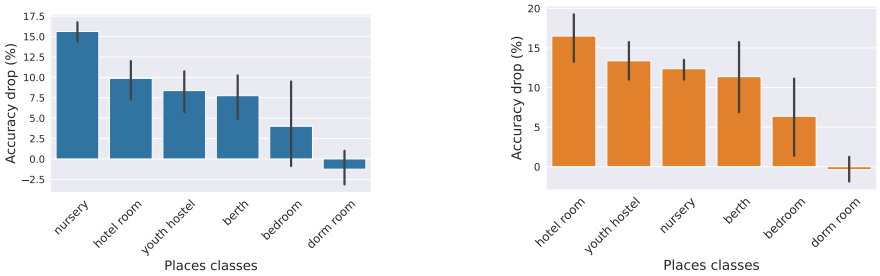
(a) Concept: "person"; Models: VGG16 (*left*) and ResNet-50 (*right*) trained on ImageNet.



(b) Concept: "tree"; Models: VGG16 (*left*) and ResNet-50 (*right*) trained on ImageNet.

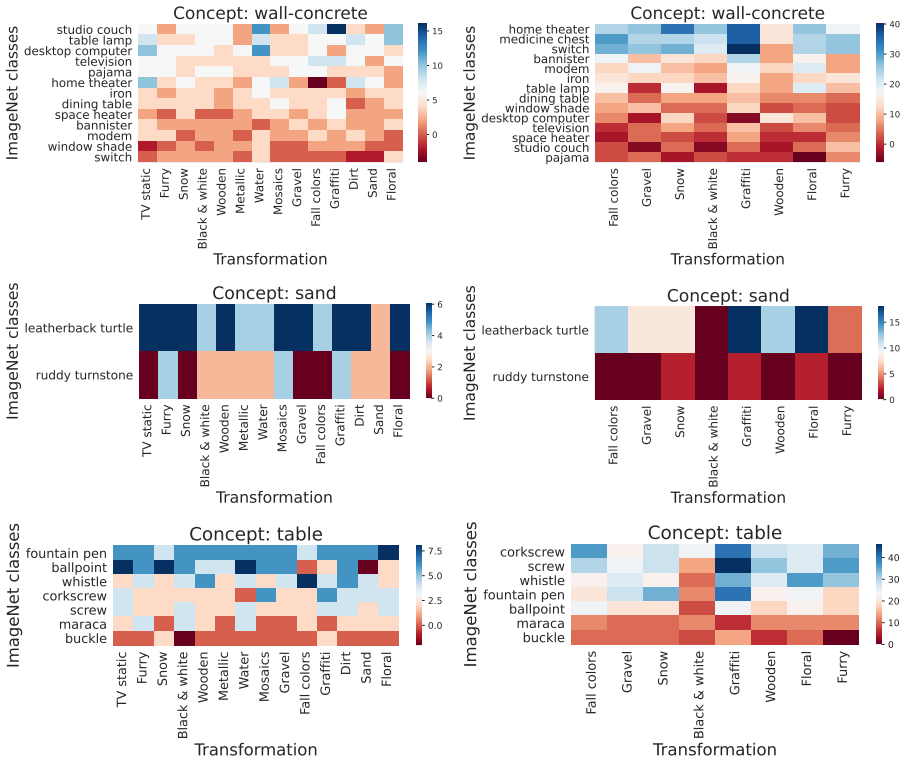


(c) Concept: "dress"; Models: VGG16 (*left*) and ResNet-18 (*right*) trained on Places-365.

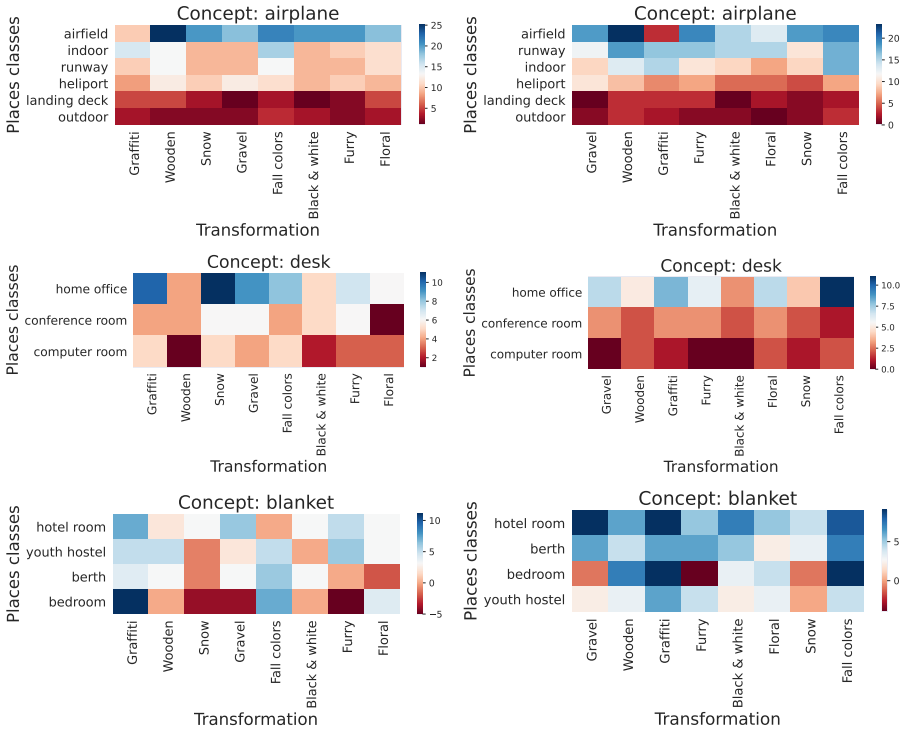


(d) Concept: "bed"; Models: VGG16 (*left*) and ResNet-18 (*right*) trained on Places-365.

Figure C.12: 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.

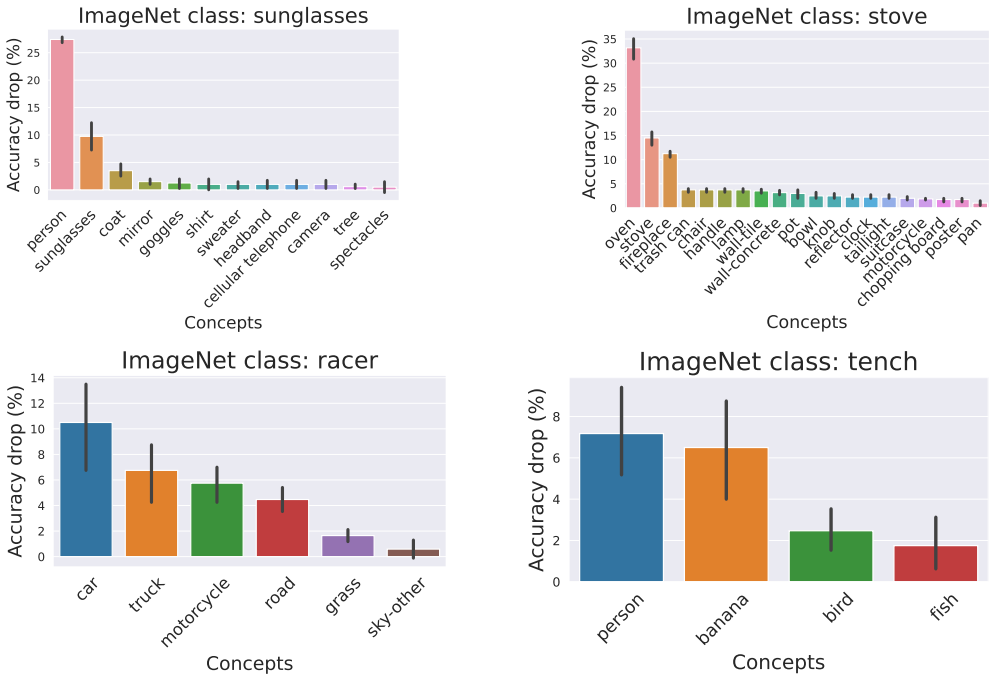


(a) VGG16 (*left*) and ResNet-50 (*right*) models trained on ImageNet.

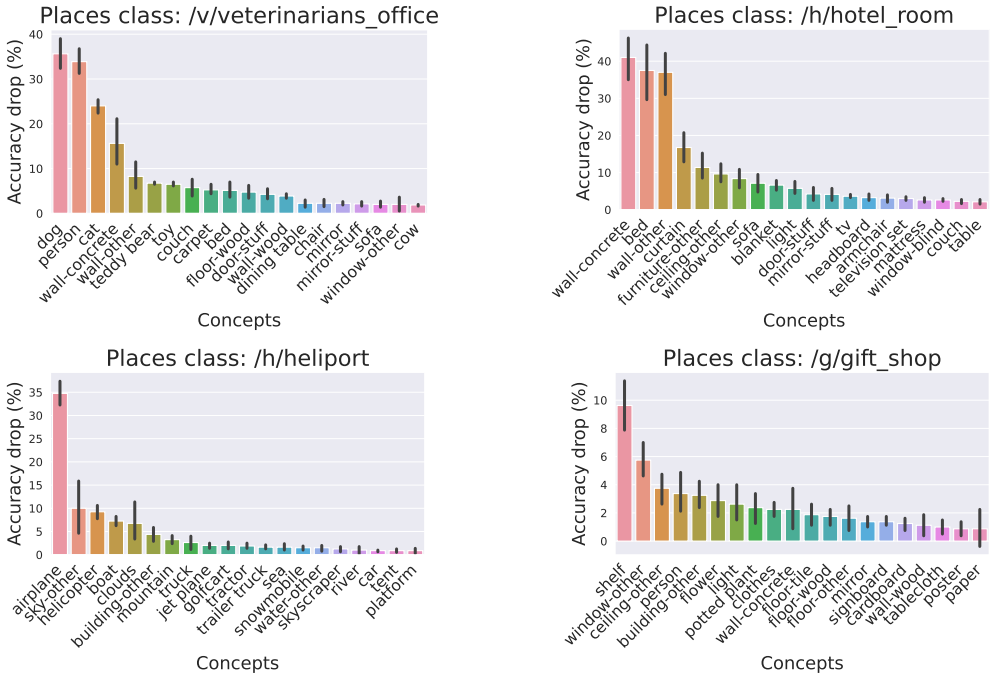


(b) VGG16 (*left*) and ResNet-18 (*right*) models trained on Places-365.

Figure C.13: Heatmaps illustrating classifier sensitivity to various concept-level transformations. Here, we measure model sensitivity in terms of the per class accuracy drop induced by the transformation on images of that class which contain the concept of interest.



(a) VGG16 classifier trained on the ImageNet dataset.



(b) ResNet-18 classifier trained on the Places-365 dataset.

Figure C.14: 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.

Appendix D

Additional details for Chapter 4

D.1 Details for Section 4.1

D.1.1 Experimental setup

For model training, we use the hyperparameters provided in Appendix C.1.7. Additional intervention-specific hyperparameters are listed in Appendix Table D.1. Due to computational constraints, we trained a restricted set of model architectures with robustness interventions—ResNet-18 and ResNet-50 for adversarial training, and ResNet-18 and ResNet-34 for all others. Adversarial training was implemented using the `robustness` library,[1] while random erasing using the PyTorch `transforms`.[2]

Eps	Step size	#Steps
0.5	0.4	3
1	0.8	3

(a) PGD-training [Mad+18]

Mean	StdDev
0	0.2

(b) Gaussian noise

Probability	Scale	Ratio
0.5	0.02 - 0.33	0.3 - 3.3

(c) Random erasing

Table D.1: Additional hyperparameters for robustness interventions.

D.1.2 Full experimental results

In Tables D.2 and D.3, we present the raw accuracies of models trained using various train-time robustness interventions.

[1] https://github.com/MadryLab/robustness
[2] https://pytorch.org/docs/stable/torchvision/transforms.html

ResNet-18				
Task	ε	Accuracy (%) Source	Target	Target-RT
ENTITY-13	0	**90.91 ± 0.73**	**61.52 ± 1.23**	**76.71 ± 1.09**
	0.5	89.23 ± 0.80	**61.10 ± 1.23**	74.92 ± 1.04
	1.0	88.45 ± 0.81	58.53 ± 1.26	73.35 ± 1.11
ENTITY-30	0	**87.88 ± 0.89**	**49.96 ± 1.31**	**73.05 ± 1.17**
	0.5	85.68 ± 0.91	**48.93 ± 1.34**	71.34 ± 1.14
	1.0	84.23 ± 0.91	47.66 ± 1.23	70.27 ± 1.17
LIVING-17	0	**92.01 ± 1.30**	**58.21 ± 2.32**	**83.38 ± 1.79**
	0.5	90.35 ± 1.35	55.79 ± 2.44	**83.00 ± 1.89**
	1.0	88.56 ± 1.50	53.89 ± 2.36	80.90 ± 1.92
NON-LIVING-26	0	**88.09 ± 1.28**	**41.87 ± 2.01**	**73.52 ± 1.71**
	0.5	86.28 ± 1.32	**41.02 ± 1.91**	**72.41 ± 1.71**
	1.0	85.19 ± 1.38	**40.23 ± 1.92**	70.61 ± 1.73

ResNet-50				
Task	ε	Accuracy (%) Source	Target	Target-RT
ENTITY-13	0	**91.54 ± 0.64**	**62.48 ± 1.16**	**79.32 ± 1.01**
	0.5	89.87 ± 0.80	**63.01 ± 1.15**	**80.14 ± 1.00**
	1.0	89.71 ± 0.74	61.21 ± 1.22	78.58 ± 0.98
ENTITY-30	0	**89.26 ± 0.78**	**51.18 ± 1.24**	77.60 ± 1.17
	0.5	87.51 ± 0.88	**50.72 ± 1.28**	**78.92 ± 1.06**
	1.0	86.63 ± 0.88	**50.99 ± 1.27**	**78.63 ± 1.03**
LIVING-17	0	**92.40 ± 1.28**	**58.22 ± 2.42**	**85.96 ± 1.72**
	0.5	90.79 ± 1.55	**55.97 ± 2.38**	**87.22 ± 1.66**
	1.0	89.64 ± 1.47	54.64 ± 2.48	**85.63 ± 1.73**
NON-LIVING-26	0	**88.13 ± 1.30**	**41.82 ± 1.86**	76.58 ± 1.69
	0.5	**88.20 ± 1.20**	**42.57 ± 2.03**	**78.84 ± 1.62**
	1.0	86.17 ± 1.36	**41.69 ± 1.96**	76.16 ± 1.61

Table D.2: Effect of adversarial training on model robustness to subpopulation shift. All models are trained on samples from the source domain—either using standard training ($\varepsilon = 0.0$) or using adversarial training. Models are then evaluated in terms of: (a) source accuracy, (b) target accuracy and (c) target accuracy after retraining the linear layer of the model with data from the target domain. Confidence intervals (95%) obtained via bootstrapping. Maximum task accuracy over ε (taking into account confidence interval) shown in bold.

ResNet-18				
Task	Intervention	Accuracy (%) Source	Target	Target-RT
ENTITY-13	Standard	**90.91 ± 0.73**	**61.52 ± 1.23**	76.71 ± 1.09
	Erase Noise	**91.01 ± 0.68**	**62.79 ± 1.27**	**78.10 ± 1.09**
	Gaussian Noise	77.00 ± 1.04	47.90 ± 1.21	70.37 ± 1.17
	Stylized ImageNet	76.85 ± 1.00	50.18 ± 1.21	65.91 ± 1.17
ENTITY-30	Standard	**87.88 ± 0.89**	**49.96 ± 1.31**	73.05 ± 1.17
	Erase Noise	**88.09 ± 0.80**	**49.98 ± 1.31**	**74.27 ± 1.15**
	Gaussian Noise	74.12 ± 1.16	35.79 ± 1.21	65.62 ± 1.28
	Stylized ImageNet	70.96 ± 1.16	37.67 ± 1.21	60.45 ± 1.22
LIVING-17	Standard	**92.01 ± 1.30**	**58.21 ± 2.32**	**83.38 ± 1.79**
	Erase Noise	**93.09 ± 1.27**	**59.60 ± 2.40**	**85.12 ± 1.71**
	Gaussian Noise	80.13 ± 1.99	46.16 ± 2.57	77.31 ± 2.08
	Stylized ImageNet	79.21 ± 1.85	43.96 ± 2.38	72.74 ± 2.09
NON-LIVING-26	Standard	**88.09 ± 1.28**	**41.87 ± 2.01**	**73.52 ± 1.71**
	Erase Noise	**88.68 ± 1.18**	**43.17 ± 2.10**	**73.91 ± 1.78**
	Gaussian Noise	78.14 ± 1.60	35.13 ± 1.94	67.79 ± 1.79
	Stylized ImageNet	71.43 ± 1.73	30.56 ± 1.75	61.83 ± 1.98

ResNet-34				
Task	Intervention	Accuracy (%) Source	Target	Target-RT
ENTITY-13	Standard	**91.75 ± 0.70**	**63.45 ± 1.13**	**78.07 ± 1.02**
	Erase Noise	**91.76 ± 0.70**	**62.71 ± 1.25**	**77.43 ± 1.06**
	Gaussian Noise	81.60 ± 0.97	50.69 ± 1.28	71.50 ± 1.13
	Stylized ImageNet	78.66 ± 0.94	51.05 ± 1.30	67.38 ± 1.16
ENTITY-30	Standard	**88.81 ± 0.81**	**51.68 ± 1.28**	**75.12 ± 1.11**
	Erase Noise	**89.07 ± 0.82**	**51.04 ± 1.27**	**74.88 ± 1.08**
	Gaussian Noise	75.05 ± 1.11	38.31 ± 1.26	67.47 ± 1.22
	Stylized ImageNet	72.51 ± 1.10	38.98 ± 1.22	61.65 ± 1.25
LIVING-17	Standard	**92.83 ± 1.19**	59.74 ± 2.27	**85.46 ± 1.83**
	Erase Noise	**92.96 ± 1.32**	**61.13 ± 2.30**	**85.66 ± 1.78**
	Gaussian Noise	84.06 ± 1.71	48.38 ± 2.44	78.79 ± 1.91
	Stylized ImageNet	80.94 ± 2.00	44.16 ± 2.43	72.77 ± 2.18
NON-LIVING-26	Standard	**89.64 ± 1.17**	**43.03 ± 1.99**	**74.99 ± 1.66**
	Erase Noise	**89.62 ± 1.31**	**43.53 ± 1.89**	**75.04 ± 1.70**
	Gaussian Noise	79.26 ± 1.61	34.89 ± 1.91	68.07 ± 1.78
	Stylized ImageNet	71.49 ± 1.65	31.10 ± 1.80	62.94 ± 1.90

Table D.3: Effect of various train-time interventions on model robustness to subpopulation shift. All models are trained on samples from the source domain.

D.2 Details for Section 4.2

D.2.1 Datasets

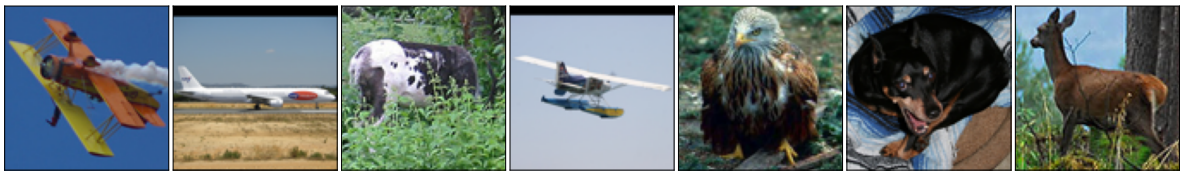
For our first set of experiments (Section 4.2.2), we focus on a canonical setting where a small portion of the training set if labeled and we have access to a pool of unlabeled data.

STL-10. The STL-10 [CNL11] dataset contains 5,000 training and 8,000 test images of size 96×96 from 10 classes. We designate 1,000 of the 5,000 (20%) training examples to be the labeled training set, 500 (10%) to be the validation set, and the rest are used as unlabeled data.

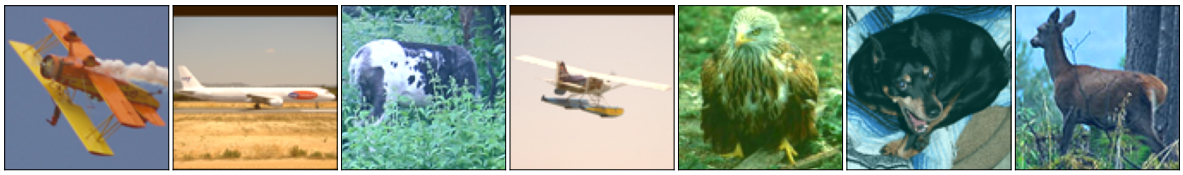
CIFAR-10. The CIFAR-10 [Kri09] dataset contains 50,000 training and 8,000 test images of size 32×32 from 10 classes. We designate 1,000 of the 50,000 (2%) training examples to be the labeled training set, 5000 (10%) to be the validation set, and the rest as unlabeled data.

In both cases, we report the final performance on the standard test set of that dataset. We also create two datasets that each contain a different spurious correlation.

Tinted STL-10. We reuse the STL-10 setup described above, but we add a class-specific tint to each image in the (labeled) training set. Specifically, we hand-pick a different color for each of the 10 classes and then add this color to each of the pixels (ensuring that each RGB channel remains within the valid range)—see Figure D.4 for examples. This tint is only present in the labeled part of the training set, the unlabeled and test parts of the dataset are left unaltered.



(a) Original



(b) Tinted

Figure D.4: Tinted STL-10 images. The tint is class-specific and thus models can learn to predict based mostly on that tint.

Biased CelebA. We consider the task of predicting gender in the CelebA [Liu+15] dataset. In order to create a biased training set, we choose a random sample of 500 non-blond males and 500 blond females. We then use a balanced unlabeled dataset consisting of 1,000 random samples for each of: blond males, blond females, non-blond males, and non-blond females. We use the standard CelebA test set which consists of 12.41% blond females, 48.92% non-blond females, 0.90% blond

males, and 37.77% non-blond males. (Note that a classifier predicting purely based on hair color with have an accuracy of 50.18% on that test set.)

All of the datasets that we use are freely available for non-commercial research purposes. Moreover, to the best of our knowledge, they do not contain offensive content or identifiable information (other than publicly available celebrity photos).

D.2.2 Model architectures and input preprocessing

For both the standard model and the models trained on images processed by edge detection algorithm, we use a standard model architecture—namely, VGG16 [SZ15] with the addition of batch normalization [IS15] (often referred to as VGG16-BN). We describe the exact edge detection process as well as the architecture of the BagNet model (texture prior) below. We visualize these priors in Figure D.6.

Canny edge detection. Given an image, we first smooth it with a 5 pixel bilateral filter [TM98], with filter σ in the coordinate and color space set to 75. After smoothing, the image is converted to gray-scale. Finally, a Canny filter [Can86] is applied to the image, with hysteresis thresholds 100 and 200, to extract the edges.

Sobel edge detection. Given an image, we first upsample it to 128×128 pixels. Then we convert it to gray-scale and apply a Gaussian blur (kernel size=5, $\sigma = 5$). The image is then passed through a Sobel filter [SF68] with a kernel size of 3 in both the horizontal and the vertical direction to extract the image gradients.

BagNet. For our texture-biased model, we use a slimmed down version of the BagNet architecture from Brendel and Bethge [BB19]. The goal of this architecture is to limit the receptive field of the model, hence forcing it to make predictions based on local features. The exact architecture we used is shown in Figure D.5. Intuitively, the top half of the network—i.e., the green and blue blocks—construct features on patches of size 20×20 for 96×96 images and 10×10 for 32×32 images. The rest of the network consists only of 1×1 convolutions and max-pooling, hence not utilizing the image's spatial structure.

D.2.3 Training setup

Basic training

We train all our models using stochastic gradient descent (SGD) with momentum (a coefficient of 0.9) and a decaying learning rate. We add weight decay regularization with a coefficient of 10^{-4}. In terms of data augmentation, we apply random cropping with a padding of 4 pixels, random horizontal flips, and a random rotation of ±2 degrees. These transformations are applied after the edge detection processing. We train all models with a batch size of 64 for 96×96-sized images and 128 for 32×32-sized images for a total of 300 epochs. All our experiments are performed using our internal cluster which mainly consists of NVIDIA 1080 Ti GTX GPUs.

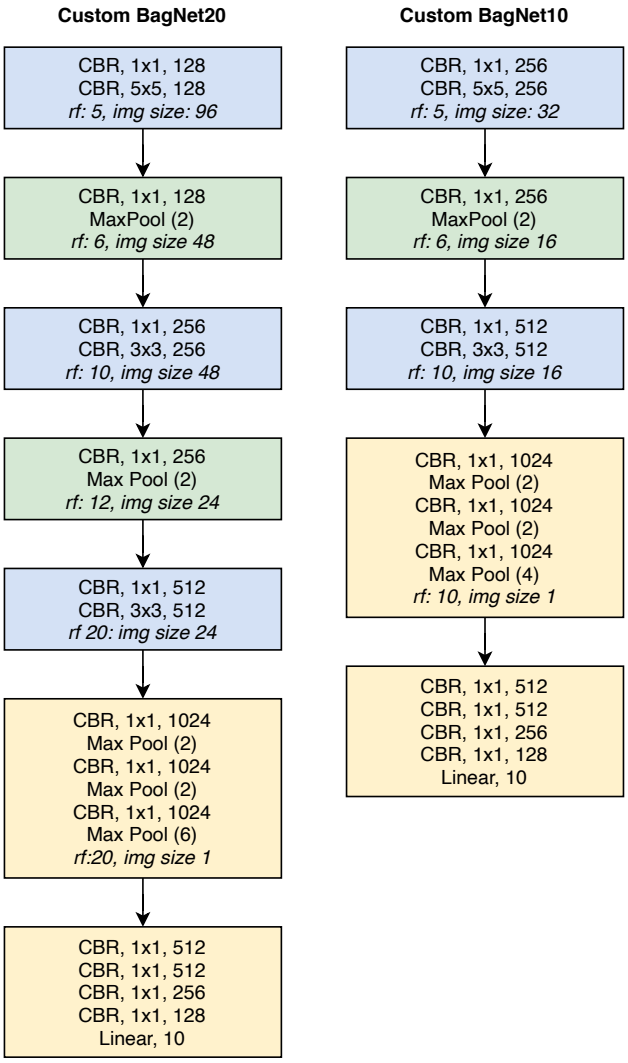
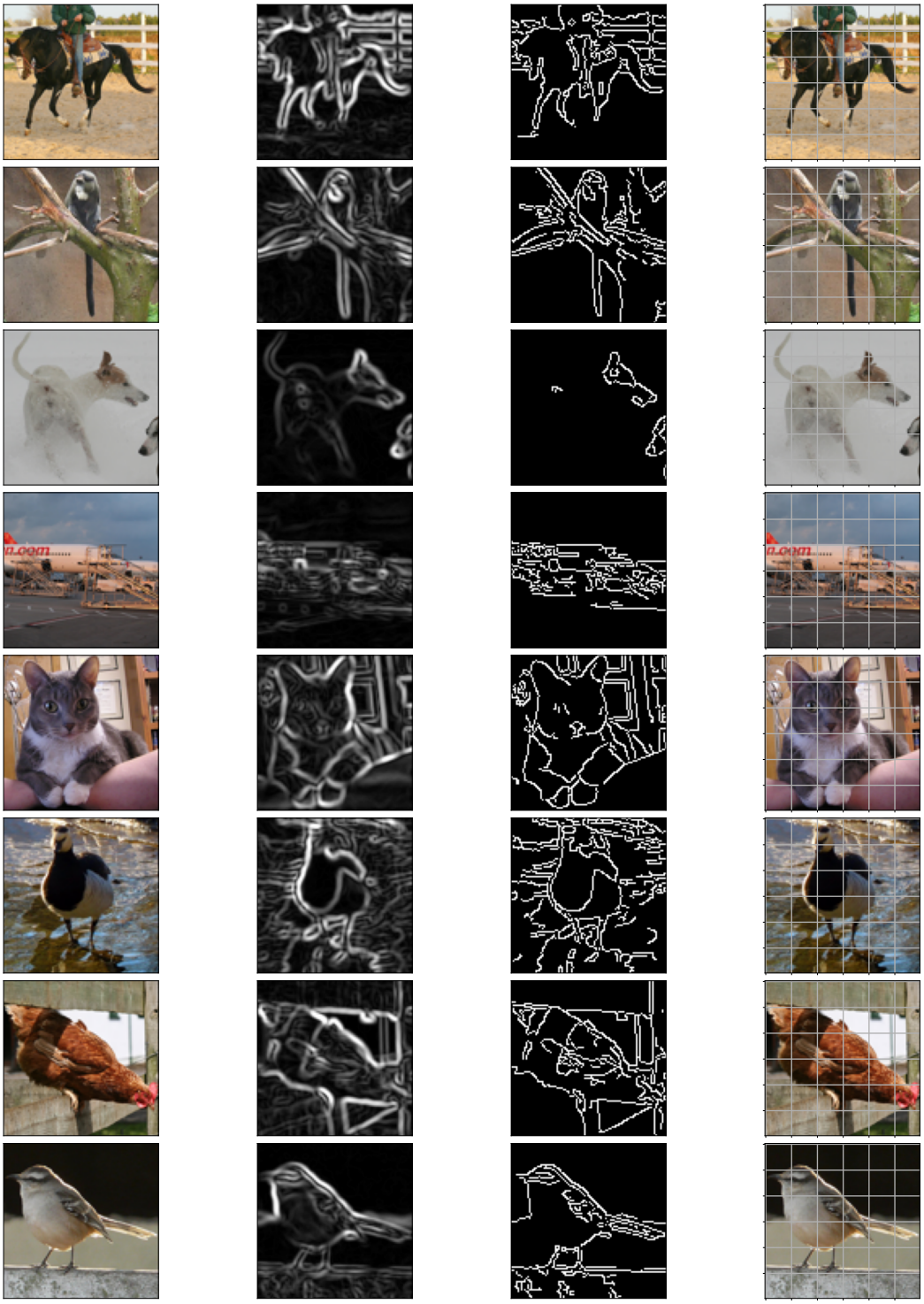


Figure D.5: The customized BagNet architecture used for training texture-biased models. The basic building block consists of a convolutional layer, followed by batch normalization and finally a ReLU non-linearity (denoted collectively as *CBR*).



(a) Original (b) Sobel (c) Canny (d) BagNet

Figure D.6: Further visualizations of the different feature priors we introduce. For each original image (a), we visualize the output of both edge detection algorithms—Sobel (b) and Canny (c)—as well as the receptive field of the BagNet model.

Hyperparameter tuning. To ensure a fair comparison across feature priors, we selected the hyperparameters for each dataset-prior pair separately, using the held-out validation set (separate from the final test used for reporting performance). Specifically, we performed a grid search choosing the learning rate (LR) from $[0.1, 0.05, 0.02, 0.01, 0.005]$, the number of epochs between each learning rate drop (K) from $[50, 100, 300]$ and the factor with which the learning rate is multiplied (γ) from $[0.5, 1]$. The parameters chosen are shown in Table D.7. We found that all models achieved near-optimal performance strictly within the range of each hyperparameters. Thus, we did not consider a wider grid.

Dataset	Prior	LR	γ	K
STL-10	Standard	0.01	0.5	100
	Canny	0.01	0.5	100
	Sobel	0.005	0.5	100
	BagNet	0.05	0.5	100
CIFAR-10	Standard	0.01	0.5	100
	Canny	0.01	0.5	100
	Sobel	0.01	0.5	100
	BagNet	0.01	0.1	100
CelebA	Standard	0.005	0.5	50
	Canny	0.005	0.1	100
	Sobel	0.01	0.5	50
	BagNet	0.02	0.5	100

Table D.7: Hyperparameters chosen through grid search for each dataset-prior pair (we used the STL-10 hyperparameters for the tinted STL-10 dataset). LR corresponds to the learning rate, γ to the factor used to decay the learning rate at each drop, and K to the train epochs between each learning rate drop.

D.2.4 Ensembles

In order to leverage prior diversity, we ensemble models trained with (potentially) different priors. We use the following ensembles:

1. **Take Max:** Predict based on the model assigning the highest probability on this example.
2. **Average:** Average the (softmax) output probabilities of the models, predict the class assigned the highest probability.
3. **Rank:** Each model ranks all test examples based on the probability assigned to their predicted labels. Then, for each example, we predict using the model which has a lower rank on this example.

We then report the maximum of these ensemble methods in Table 4.4.

D.2.5 Self-training and co-training schemes

In the setting that we are focusing on, we are provided with a labeled dataset $\mathbf{X}$ and an unlabeled dataset $\mathbf{U}$, where typically there is much more unlabeled data ($|\mathbf{U}| \gg |\mathbf{X}|$). We are then choosing a set of (one or more) feature priors each of which corresponds to a different way of training a model (e.g., using edge detection preprocessing).

General methodology. We start by training each of these models on the labeled dataset. Then, we combine the predictions of these models to produce pseudo-labels for the unlabeled dataset. Finally, we choose a fraction of the unlabeled data and train the models on that set using the produced pseudo-labels (in additional to the original labeled set X). This process is repeated using increasing fractions of the unlabeled dataset until, eventually, models are trained on its entirety. We refer to each such phase as an *era*. We include an additional 5% of the unlabeled data per era, resulting in a total of 20 eras. During each era, we use the training process described in Appendix D.2.3 without re-initializing the models (warm start). After completing this process, we train a standard model from scratch using both the labeled set and resulting pseudo-labels. The methodology used for choosing and combining pseudo-labels is described below for each scheme.

Self-training. Since we are only training one model, we only need to decide how to choose the pseudo-labels to use for each era. We do this in the simplest way: at ear t, we pick the subset $\mathbf{U_t} \subseteq \mathbf{U}$ of examples that are assigned the highest probability on their predicted label. We attempt to produce a class-balanced training set by applying this process separately on each class (as predicted by the model). The pseudocode for the method is provided in Algorithm 2.

Algorithm 2 Self-training

Parameters: Number of eras T. Fraction added per era k.
Input : Labeled data $\mathbf{X}$ with n classes, unlabeled data $\mathbf{U}$, model trained on $\mathbf{X}$.
for *era* $t \in 1...T$ **do**
  forward-pass $\mathbf{U}$ through the model to create pseudo-labels $\mathbf{U_t} = []$ **for** *each class* c **do**
    Select the $\frac{kt|\mathbf{U}|}{n}$ most confident examples from $\mathbf{U}$ predicted by the model as class c
    Add those examples to $\mathbf{U_t}$ with class c
  Re-train (warm start) the model on $\mathbf{X} \cup \mathbf{U_t}$ until convergence
Train a standard model from scratch on $\mathbf{X} \cup \mathbf{U_T}$.

Standard co-training. Here, we train multiple models (in our experiments two) based on a common pool of pseudo-labeled examples in each era. In each era t, each model labels the unlabeled dataset $\mathbf{U}$. Then, for each class, we alternate between models, adding the next most confident example predicted as that class for that model to $\mathbf{U_t}$, until we reach a fixed number of unique examples have been added for that class (5% of the size of the unlabeled dataset per era). Note that this process allows both conflicts and duplicates: if multiple models are confident about a specific example, that example may be added more than once (potentially with a different label each time). Finally, we train each model (without re-initializing) on $\mathbf{X} \cup \mathbf{U_t}$. The pseudocode for this method can be found in Algorithm 3.

Algorithm 3 Standard Co-Training

Parameters: Number of eras T. Fraction added per era k.
Input : Labeled data $\mathbf{X}$ with n classes, unlabeled data $\mathbf{U}$, models trained on $\mathbf{X}$.
for *era* $t \in 1...T$ **do**
  forward-pass $\mathbf{U}$ through each model to create pseudo-labels $\mathbf{U_t} = []$ **for** *each class* c **do**
    $\mathbf{U_t^{(c)}} = []$ **while** *the number of unique examples in* $\mathbf{U_t^{(c)}} < \frac{kt|\mathbf{U}|}{n}$ **do**
      for *each model* m **do**
        Add the next most confident example predicted by m as class c to $\mathbf{U_t^{(c)}}$
    Add $\mathbf{U_t^{(c)}}$ to $\mathbf{U_t}$
  Re-train (warm start) each model on $\mathbf{X} \cup \mathbf{U_t}$ until convergence
Train a standard model from scratch on $\mathbf{X} \cup \mathbf{U_T}$.

D.2.6 Experiment organization

We now provide the full experimental results used to create the plots in the main body as well as additional analysis. Specifically, in Appendix D.2.7 and D.2.8 we present the performance of individual ensemble schemes for pre-trained and self-trained models respectively. Then, in Appendix D.2.9 we present the performance of co-training for each combination of feature priors. In Appendix D.2.10 we analyse the effect that co-training has on model similarity after training. Finally, in Appendix D.2.11 we evaluate model ensembles on datasets with spurious correlations and in Appendix D.2.12 we breakdown the performance of co-training on the skewed CelebA dataset according to different input attributes.

D.2.7 Full pre-trained ensemble results

In Table 4.4, we reported the best ensemble method for each pair of models trained with different priors on the labeled data. In Table D.8, we report the full results over the individual ensembles.

Feature Priors	Model 1	Model 2	Max Conf.	Avg Conf.	Rank	Best
Standard + Standard	52.54 ± 0.85	51.82 ± 0.85	53.98 ± 0.83	54.02 ± 0.85	53.98 ± 0.83	54.02 ± 0.82
Sobel + Sobel	51.94 ± 0.88	53.69 ± 0.86	54.62 ± 0.83	54.68 ± 0.86	54.61 ± 0.85	54.68 ± 0.83
Canny + Canny	45.48 ± 0.84	44.19 ± 0.88	46.46 ± 0.82	46.48 ± 0.86	46.70 ± 0.83	46.70 ± 0.79
BagNet + BagNet	42.22 ± 0.80	42.56 ± 0.83	43.32 ± 0.82	43.49 ± 0.82	43.33 ± 0.85	43.49 ± 0.84
Standard + Sobel	52.54 ± 0.79	51.94 ± 0.82	58.14 ± 0.82	58.21 ± 0.88	58.12 ± 0.82	**58.21 ± 0.90**
Standard + Canny	52.54 ± 0.87	45.48 ± 0.81	55.18 ± 0.82	55.49 ± 0.83	54.41 ± 0.81	55.49 ± 0.83
Standard + BagNet	52.54 ± 0.85	42.22 ± 0.80	52.89 ± 0.84	53.03 ± 0.89	50.69 ± 0.81	53.03 ± 0.85
Sobel + Canny	51.94 ± 0.82	45.48 ± 0.85	53.81 ± 0.84	53.95 ± 0.80	53.18 ± 0.91	53.95 ± 0.85
Sobel + BagNet	51.94 ± 0.86	42.22 ± 0.82	54.42 ± 0.84	55.14 ± 0.83	53.50 ± 0.82	55.14 ± 0.84
Canny + BagNet	45.48 ± 0.78	42.22 ± 0.79	49.95 ± 0.84	50.57 ± 0.82	49.64 ± 0.81	50.57 ± 0.84

(a) Ensemble Baselines for CIFAR-10

Feature Priors	Model 1	Model 2	Max Conf.	Avg Conf.	Rank	Best
Standard + Standard	53.73 ± 0.86	55.38 ± 1.00	56.95 ± 0.94	57.06 ± 0.91	56.94 ± 0.97	57.06 ± 0.91
Sobel + Sobel	55.49 ± 0.94	55.64 ± 0.98	56.71 ± 0.92	56.83 ± 0.90	56.66 ± 0.89	56.83 ± 0.94
Canny + Canny	56.29 ± 0.92	54.99 ± 0.96	58.04 ± 0.94	58.23 ± 0.94	57.95 ± 0.89	58.23 ± 0.93
BagNet + BagNet	52.04 ± 0.92	50.34 ± 0.90	53.40 ± 0.98	53.42 ± 0.91	53.29 ± 0.96	53.42 ± 0.98
Standard + Sobel	53.73 ± 0.94	55.49 ± 0.95	59.01 ± 0.90	59.08 ± 0.91	58.94 ± 0.96	59.08 ± 0.95
Standard + Canny	53.73 ± 1.00	56.29 ± 0.94	60.90 ± 0.94	60.96 ± 0.94	60.85 ± 0.87	**60.96 ± 0.94**
Standard + BagNet	53.73 ± 0.95	52.04 ± 0.90	56.99 ± 0.94	57.17 ± 0.92	57.04 ± 0.91	57.17 ± 0.94
Sobel + Canny	55.49 ± 0.91	56.29 ± 0.94	59.92 ± 0.95	60.02 ± 0.97	59.77 ± 0.91	60.02 ± 0.91
Sobel + BagNet	55.49 ± 0.94	52.04 ± 0.95	59.17 ± 0.94	59.76 ± 0.96	59.08 ± 0.89	59.76 ± 0.87
Canny + BagNet	56.29 ± 0.96	52.04 ± 0.95	61.09 ± 0.92	61.42 ± 0.94	60.68 ± 0.92	**61.42 ± 0.93**

(b) Ensemble Baselines for STL-10

Table D.8: Full results for ensembles of pre-trained models.

D.2.8 Ensembling self-trained models

In Table D.9, we report the best ensemble method for pairs of self-trained models with different priors. In Table D.10, we report the full results over the individual ensembles. We find that, similar to the ensembles of models trained on the labeled data, models with diverse priors gain more from ensembling. However, co-training models with diverse priors together still outperforms ensembling self-trained models.

	Feature Priors	Model 1	Model 2	Ensemble
Same	Standard + Standard	59.92 ± 0.95	59.34 ± 0.88	62.25 ± 0.93
	Canny + Canny	58.40 ± 0.94	57.69 ± 0.94	60.38 ± 0.92
	BagNet + BagNet	57.80 ± 0.96	58.11 ± 0.85	60.52 ± 0.90
Different	Standard + Canny	59.92 ± 0.90	58.40 ± 0.95	**64.44 ± 0.90**
	Standard + BagNet	59.92 ± 0.94	57.80 ± 0.96	63.19 ± 0.87
	Canny + BagNet	58.40 ± 0.94	57.80 ± 0.96	**64.80 ± 0.91**

(a) STL-10

	Feature Priors	Model 1	Model 2	Ensemble
Same	Standard + Standard	63.65 ± 0.81	61.95 ± 0.82	64.85 ± 0.79
	Sobel + Sobel	63.05 ± 0.81	66.01 ± 0.80	66.25 ± 0.82
	BagNet + BagNet	53.92 ± 0.82	52.90 ± 0.91	55.00 ± 0.83
Different	Standard + Sobel	63.65 ± 0.81	63.05 ± 0.83	**67.52 ± 0.77**
	Standard + BagNet	63.65 ± 0.81	53.92 ± 0.88	64.10 ± 0.79
	Sobel + BagNet	63.05 ± 0.83	53.92 ± 0.89	65.68 ± 0.79

(b) CIFAR-10

Table D.9: Ensemble performance when combining *self-trained* models with Standard, Canny, Sobel, and BagNet priors. When two models of the same prior are ensembled, the models are trained with different random initializations.

Feature Priors	Model 1	Model 2	Max Conf.	Avg Conf.	Rank	Best
Standard + Standard	63.65 ± 0.81	61.95 ± 0.87	64.84 ± 0.77	64.85 ± 0.76	64.83 ± 0.83	64.85 ± 0.79
Sobel + Sobel	63.05 ± 0.87	66.01 ± 0.82	66.19 ± 0.81	66.25 ± 0.79	66.17 ± 0.81	66.25 ± 0.83
BagNet + BagNet	53.92 ± 0.87	52.90 ± 0.83	54.86 ± 0.87	55.00 ± 0.83	54.87 ± 0.82	55.00 ± 0.87
Standard + Sobel	63.65 ± 0.79	63.05 ± 0.80	67.42 ± 0.79	67.52 ± 0.79	67.38 ± 0.79	**67.52 ± 0.77**
Standard + Canny	63.65 ± 0.90	51.82 ± 0.88	63.70 ± 0.81	63.91 ± 0.81	63.02 ± 0.83	63.91 ± 0.82
Standard + BagNet	63.65 ± 0.81	53.92 ± 0.82	64.05 ± 0.85	64.10 ± 0.79	62.69 ± 0.80	64.10 ± 0.86
Sobel + Canny	63.05 ± 0.81	51.82 ± 0.80	61.43 ± 0.80	61.42 ± 0.80	60.66 ± 0.81	61.43 ± 0.83
Sobel + BagNet	63.05 ± 0.78	53.92 ± 0.83	65.45 ± 0.85	65.68 ± 0.82	64.65 ± 0.80	65.68 ± 0.82
Canny + BagNet	51.82 ± 0.81	53.92 ± 0.79	59.60 ± 0.81	59.79 ± 0.83	60.24 ± 0.82	60.24 ± 0.81

(a) Ensemble Baselines for CIFAR-10

Feature Priors	Model 1	Model 2	Max Conf.	Avg Conf.	Rank	Best
Standard + Standard	59.92 ± 0.92	59.34 ± 0.99	62.18 ± 0.92	62.25 ± 0.96	62.16 ± 0.88	62.25 ± 0.94
Canny + Canny	58.40 ± 0.95	57.69 ± 0.89	60.30 ± 0.95	60.36 ± 0.92	60.38 ± 0.91	60.38 ± 0.95
BagNet + BagNet	57.80 ± 0.89	58.11 ± 0.94	60.42 ± 0.90	60.46 ± 0.98	60.52 ± 0.93	60.52 ± 0.90
Standard + Sobel	59.92 ± 0.92	57.86 ± 0.91	62.49 ± 0.89	62.69 ± 0.91	62.66 ± 0.89	62.69 ± 0.94
Standard + Canny	59.92 ± 0.94	58.40 ± 0.95	64.29 ± 0.95	64.44 ± 0.89	64.34 ± 0.95	**64.44 ± 0.95**
Standard + BagNet	59.92 ± 0.89	57.80 ± 0.97	63.01 ± 0.93	63.10 ± 0.89	63.19 ± 0.88	63.19 ± 0.88
Sobel + Canny	57.86 ± 0.91	58.40 ± 0.93	62.20 ± 0.92	62.14 ± 0.92	62.22 ± 0.90	62.22 ± 0.91
Sobel + BagNet	57.86 ± 0.95	57.80 ± 0.95	62.24 ± 0.94	62.58 ± 0.90	63.52 ± 0.91	63.52 ± 0.88
Canny + BagNet	58.40 ± 0.93	57.80 ± 0.95	64.38 ± 0.89	64.64 ± 0.92	64.80 ± 0.90	**64.80 ± 0.92**

(b) Ensemble Baselines for STL-10

Table D.10: Full results for ensembles of self-trained models.

D.2.9 Self-training and co-training on STL-10 and CIFAR-10

Methods	Prior(s)	Labeled Only	+Unlabeled Self/Co-Training	+ Standard model with Pseudo-labels
Self-training	Standard	52.54 ± 0.86	63.65 ± 0.76	64.02 ± 0.82
	Canny	45.48 ± 0.90	51.82 ± 0.82	55.59 ± 0.80
	Sobel	51.94 ± 0.88	63.05 ± 0.84	64.77 ± 0.80
	BagNet	42.22 ± 0.82	53.92 ± 0.89	54.21 ± 0.85
Co-training	Standard	52.54 ± 0.91	65.06 ± 0.76	65.10 ± 0.84
	+Standard	51.82 ± 0.86	64.93 ± 0.80	
	Canny	45.48 ± 0.85	51.15 ± 0.79	55.74 ± 0.80
	+Canny	44.19 ± 0.82	51.65 ± 0.81	
	Sobel	51.94 ± 0.86	67.18 ± 0.80	68.47 ± 0.74
	+Sobel	53.69 ± 0.89	67.35 ± 0.77	
	Canny	45.48 ± 0.79	58.66 ± 0.81	65.34 ± 0.81
	+Sobel	51.94 ± 0.80	64.87 ± 0.79	
	Canny	45.48 ± 0.85	59.19 ± 0.85	67.59 ± 0.74
	+BagNet	42.22 ± 0.85	67.92 ± 0.79	
	Sobel	51.94 ± 0.81	71.88 ± 0.73	$\mathbf{74.25 \pm 0.74}$
	+BagNet	42.22 ± 0.82	73.91 ± 0.71	
	BagNet	42.22 ± 0.79	55.94 ± 0.83	56.05 ± 0.77
	+BagNet	42.56 ± 0.86	55.26 ± 0.88	
	Canny	45.48 ± 0.85	59.23 ± 0.81	67.21 ± 0.77
	+Standard	52.54 ± 0.87	66.92 ± 0.82	
	Sobel	51.94 ± 0.83	71.44 ± 0.76	$\mathbf{73.83 \pm 0.76}$
	+Standard	52.54 ± 0.85	73.59 ± 0.72	
	Standard	52.54 ± 0.88	66.67 ± 0.83	66.77 ± 0.75
	+BagNet	42.22 ± 0.80	67.12 ± 0.75	

Table D.11: Performance of self-training and co-training on CIFAR-10 for each prior combination.

Methods	Prior(s)	Labeled Only	+Unlabeled Self/Co-Training	+ Standard model with Pseudo-labels
Self-training	Standard	53.73 ± 0.95	59.92 ± 0.91	60.52 ± 0.94
	Canny	56.29 ± 0.96	58.40 ± 0.91	62.19 ± 0.92
	Sobel	55.49 ± 0.96	57.86 ± 0.98	60.92 ± 0.89
	BagNet	52.04 ± 0.96	57.80 ± 0.99	61.69 ± 0.95
Co-training	Standard	53.73 ± 0.95	58.05 ± 0.92	61.16 ± 0.95
	+Standard	55.38 ± 0.96	60.44 ± 0.95	
	Canny	56.29 ± 0.92	60.22 ± 0.91	63.24 ± 0.92
	+Canny	54.99 ± 0.94	59.56 ± 0.94	
	Sobel	55.49 ± 0.96	58.93 ± 0.91	60.68 ± 0.94
	+Sobel	55.64 ± 0.95	59.23 ± 0.90	
	Canny	56.29 ± 0.95	62.40 ± 0.99	65.53 ± 0.84
	+Sobel	55.49 ± 0.92	64.11 ± 0.91	
	Canny	56.29 ± 0.92	62.21 ± 0.89	$\mathbf{67.33 \pm 0.88}$
	+BagNet	52.04 ± 0.94	66.74 ± 0.87	
	Sobel	55.49 ± 0.92	62.72 ± 0.94	65.79 ± 0.94
	+BagNet	52.04 ± 1.00	65.44 ± 0.91	
	BagNet	52.04 ± 0.89	59.85 ± 0.89	60.84 ± 0.95
	+BagNet	50.34 ± 0.91	60.16 ± 0.89	
	Canny	56.29 ± 0.94	62.16 ± 0.92	65.67 ± 0.93
	+Standard	53.73 ± 0.92	64.22 ± 0.91	
	Sobel	55.49 ± 0.95	61.15 ± 0.89	63.08 ± 0.91
	+Standard	53.73 ± 0.92	61.74 ± 0.93	
	Standard	53.73 ± 0.94	61.99 ± 0.88	62.34 ± 0.89
	+BagNet	52.04 ± 0.91	62.31 ± 1.00	

Table D.12: Performance of self-training and co-training on STL-10 for each prior combination.

D.2.10 Correlation between the individual feature-biased models and the final standard model

Method	Prior	CIFAR-10 Before	CIFAR-10 After	STL-10 Before	STL-10 After
Self-training	Standard	0.598	0.813	0.554	0.728
	Canny	0.237	0.622	0.305	0.519
	Sobel	0.259	0.76	0.385	0.621
	BagNet	0.38	0.752	0.357	0.516
Co-training	Canny	0.237	0.595	0.305	0.496
	+BagNet	0.38	0.664	0.357	0.538
	Sobel	0.259	0.719	0.385	0.581
	+BagNet	0.38	0.716	0.357	0.554

Table D.13: Similarity between models before and after training on pseudo-labeled data. Our measure of similarity is the (Pearson) correlation between which test examples are correctly predicted by each model. In Columns 3 and 5 we report that notion of similarity between the pre-trained feature-biased models and the pre-trained standard model (the numbers are reproduced from Table 4.1). Then, in columns 4 and 6 we report the similarity between the feature-biased models at the end of self- or co-training and the standard model trained on their (potentially combined) pseudo-labels. We observe that through this process of training a standard model on the pseudo-labels of different feature-biased models, the former behaves more similar to the latter.

D.2.11 Ensembles for spurious datasets

In Table D.14 (full table in Table D.15), we ensemble the self-trained priors for the Tinted STL-10 dataset and the CelebA dataset as in Section 4.2.3. Both of these datasets have a spurious correlation base on color, which results in a weak Standard and BagNet model. As a result, the ensembles with the Standard or BagNet models do not perform well on the test set. However, in Section 4.3, we find that co-training in this setting allows the BagNet model to improve when jointly trained with a shape model, thus boosting the final performance.

Feature Priors	Model 1	Model 2	Ensemble
Standard + Canny	17.56 ± 0.73	$\mathbf{57.31 \pm 0.96}$	44.31 ± 0.90
Standard + Sobel	17.56 ± 0.71	56.12 ± 0.90	46.06 ± 0.95
Standard + BagNet	17.56 ± 0.73	13.53 ± 0.66	16.64 ± 0.66
Canny + BagNet	$\mathbf{57.31 \pm 0.96}$	13.53 ± 0.64	48.30 ± 0.89
Sobel + BagNet	56.12 ± 0.91	13.53 ± 0.69	49.05 ± 0.98

(a) Tinted STL-10

Feature Priors	Model 1	Model 2	Ensemble
Standard + Canny	71.57 ± 0.53	$\mathbf{85.73 \pm 0.40}$	84.05 ± 0.42
Standard + Sobel	71.57 ± 0.55	$\mathbf{85.42 \pm 0.43}$	82.10 ± 0.45
Standard + BagNet	71.57 ± 0.53	64.89 ± 0.56	69.66 ± 0.55
Canny + BagNet	$\mathbf{85.73 \pm 0.42}$	64.89 ± 0.56	84.06 ± 0.45
Sobel + BagNet	$\mathbf{85.42 \pm 0.43}$	64.89 ± 0.57	82.89 ± 0.44

(b) CelebA

Table D.14: Performance of ensembles consisting of models trained with different priors.

Feature Priors	Model 1	Model 2	Max Conf.	Avg Conf.	Rank	Best
Standard + Canny	17.56 ± 0.70	$\mathbf{57.31 \pm 0.95}$	44.31 ± 0.98	43.48 ± 0.94	42.12 ± 0.95	44.31 ± 0.94
Standard + Sobel	17.56 ± 0.66	56.12 ± 0.98	46.06 ± 0.94	44.71 ± 0.91	39.39 ± 0.95	46.06 ± 0.99
Standard + BagNet	17.56 ± 0.71	13.53 ± 0.64	16.59 ± 0.69	16.64 ± 0.71	16.14 ± 0.74	16.64 ± 0.66
Canny + BagNet	$\mathbf{57.31 \pm 0.91}$	13.53 ± 0.62	48.09 ± 0.96	48.30 ± 1.01	39.92 ± 0.92	48.30 ± 0.95
Sobel + BagNet	56.12 ± 0.94	13.53 ± 0.64	49.00 ± 0.95	49.05 ± 0.95	37.67 ± 0.91	49.05 ± 0.93

(a) Tinted STL-10

Feature Priors	Model 1	Model 2	Max Conf.	Avg Conf.	Rank	Best
Standard + Canny	71.57 ± 0.53	$\mathbf{85.73 \pm 0.43}$	83.96 ± 0.44	84.05 ± 0.43	84.00 ± 0.46	84.05 ± 0.43
Standard + Sobel	71.57 ± 0.57	$\mathbf{85.42 \pm 0.41}$	82.06 ± 0.45	82.10 ± 0.45	78.01 ± 0.51	82.10 ± 0.49
Standard + BagNet	71.57 ± 0.56	64.89 ± 0.56	69.66 ± 0.54	69.66 ± 0.54	68.01 ± 0.58	69.66 ± 0.54
Canny + BagNet	$\mathbf{85.73 \pm 0.42}$	64.89 ± 0.57	84.06 ± 0.44	84.06 ± 0.45	72.79 ± 0.51	84.06 ± 0.44
Sobel + BagNet	$\mathbf{85.42 \pm 0.39}$	64.89 ± 0.55	82.89 ± 0.46	82.89 ± 0.46	71.65 ± 0.57	82.89 ± 0.43

(b) CelebA

Table D.15: Performance of individual ensembles on datasets with spurious correlations.

D.2.12 Breakdown of test accuracy for co-training on CelebA

Method	Prior(s)	Female Blond (N=2480)	Female Not Blond (N=9767)	Male Blond (N=180)	Male Not Blond (N=7535)
Self-training	Standard	**97.78 ± 0.52**	47.06 ± 0.83	55.56 ± 6.11	95.94 ± 0.37
	Canny	94.44 ± 0.81	77.27 ± 0.69	**78.33 ± 5.00**	96.19 ± 0.36
	Sobel	95.97 ± 0.60	73.43 ± 0.78	70.56 ± 5.56	96.63 ± 0.37
	BagNet	**97.26 ± 0.60**	35.44 ± 0.80	41.67 ± 6.67	96.30 ± 0.40
Co-training	Canny +BagNet	96.94 ± 0.56	**86.69 ± 0.56**	**79.44 ± 5.00**	97.53 ± 0.31
	Sobel +BagNet	96.81 ± 0.56	84.41 ± 0.63	**79.44 ± 5.00**	**97.89 ± 0.29**

Table D.16: Accuracy of predicting gender on different subpopulations of the CelebA dataset. We show the accuracy of standard models trained on the pseudo-labels produced by different self- or co-training schemes. Recall that in the training set all females are blond and all males are non-blond (while the unlabeled dataset is balanced). It is thus interesting to consider where this correlation is reversed. We observe that, in these cases, both the standard and BagNet models perform quite poorly, even after being self-trained on the unlabeled dataset where this correlation is absent. At the same time, co-training steers the models away from this correlation, resulting in improved performance. 95% confidence intervals computed via bootstrap are shown.

Appendix E

Additional details for Chapter 5

E.1 Experimental details

E.1.1 Datasets

For the bulk of our experimental analysis 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 and editing are performed on (or using) samples from the standard test sets to avoid overlap with the training data used to develop the models.

E.1.2 Models

Here, we describe the exact architecture and training process for each model we use. For 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 [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 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. For the typographic attacks of Section 5.4, we use the ResNet-50 models trained via CLIP [Rad+21], as provided in the original model repository.[2]

[1] https://pytorch.org/vision/stable/models.html
[2] https://github.com/openai/CLIP

Architecture \ Dataset	Test Accuracy (%) ImageNet	Places
VGG	73.70	54.02
ResNet	75.77	54.24
CLIP-ResNet	59.84	-

Table E.1: Accuracy of each model architecture on the datasets used in our analysis.

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 (5.2). 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 (cf. Section 5.2) 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

We now describe the details of our evaluation methodology, 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 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 the performance of the model 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. Appendix Figure C.11): "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 hyper-parameters 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 (5.3) 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 Real-world data collection

In Section 5.4 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[3] 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 E.2.

Figure E.2: 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.

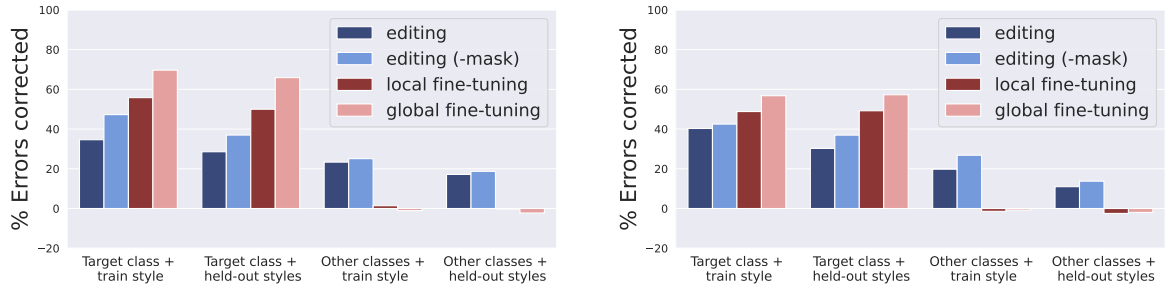
[3] https://www.flickr.com/

E.2 Additional experiments

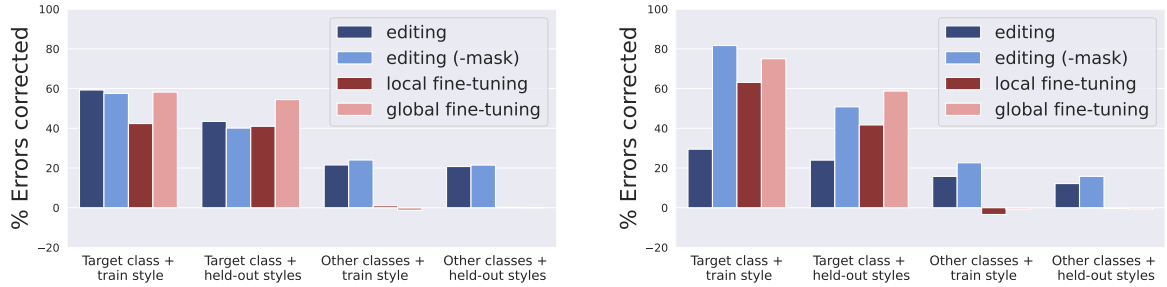
E.2.1 The effectiveness of editing

In Figures E.3- E.6, 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.8-E.11.

We observe that both methods successfully generalize to held-out samples from the target class (used to perform the modification)—even when the transformation is performed using held-out styles. However, while the performance improvements of editing also extend to other classes containing the same concept, this does not seem to be the case for fine-tuning. These trends hold even when we use more exemplars to perform the modification. In Appendix Figure E.7, we illustrate sample error corrections (and failures to do so) due to editing and fine-tuning.

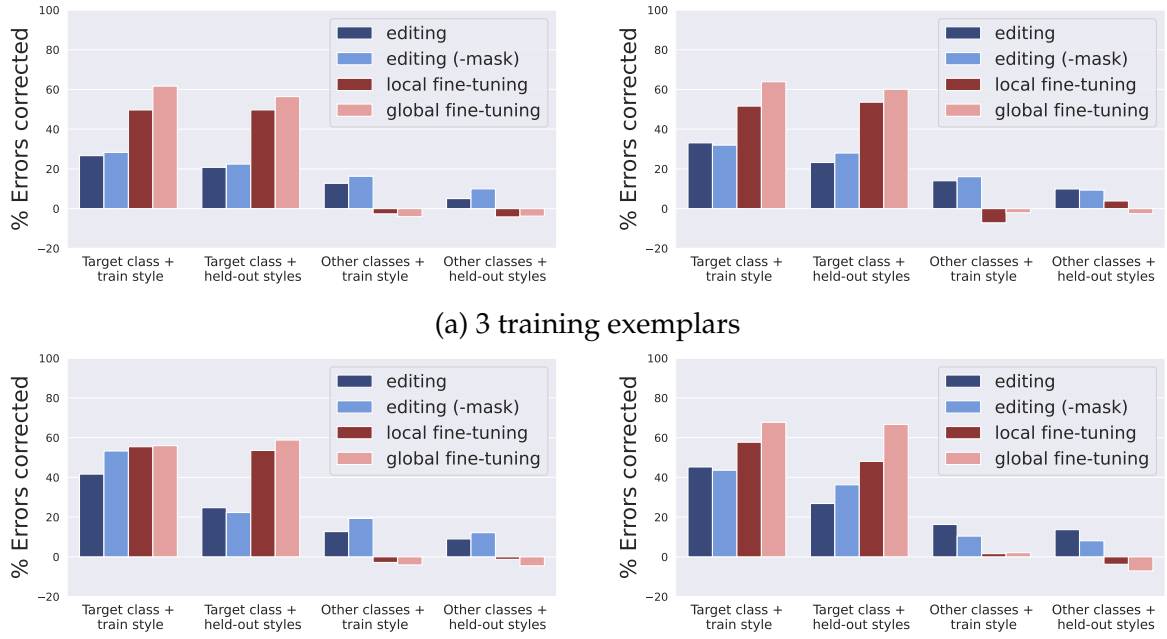


(a) 3 training exemplars



(b) 10 training exemplars

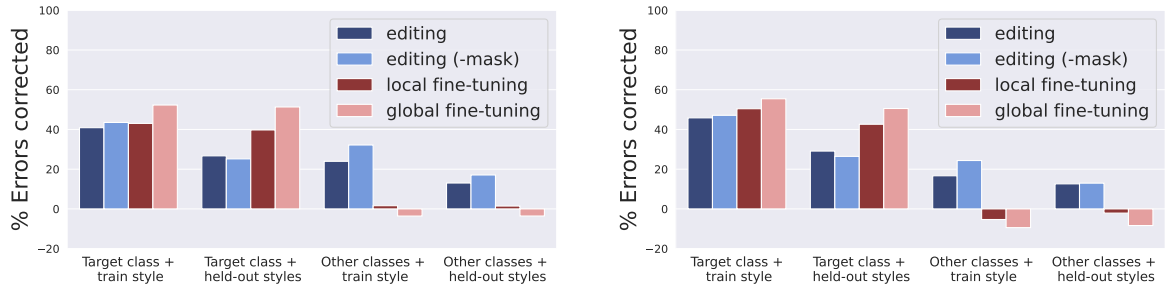
Figure E.3: Editing vs. fine-tuning: average number of misclassifications corrected by the method when applied to an ImageNet-trained VGG-16 classifier. Here, the average is computed over different concept-transformation pairs—with concepts derived from instance segmentation modules trained on MS-COCO (*left*) and LVIS (*right*); and transformations described in Appendix E.1. 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

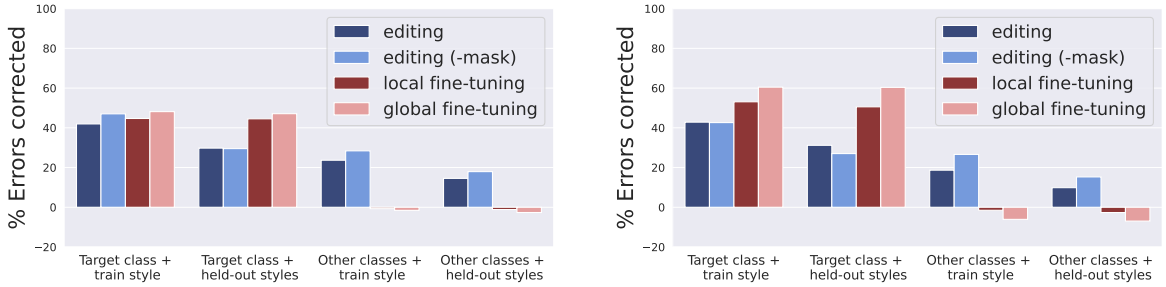
Figure E.4: Repeating the analysis in Appendix Fig. E.3 on an ImageNet-trained ResNet-50.



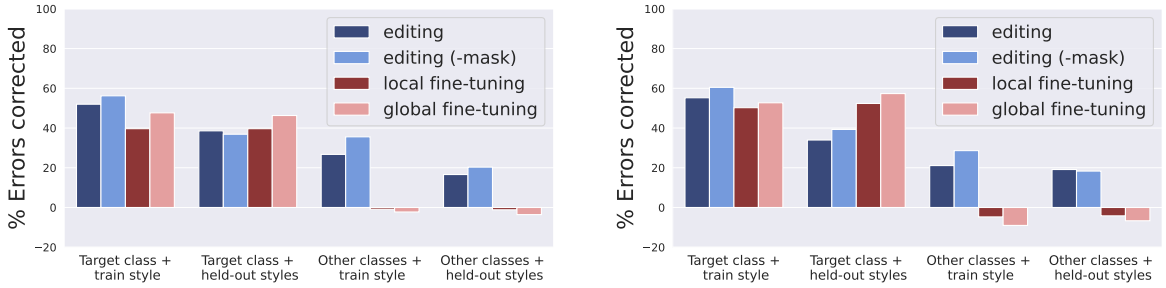
(a) 3 training exemplars

(b) 10 training exemplars

Figure E.5: Repeating the analysis in Appendix Fig. E.3 on an Places365-trained VGG-16.



(a) 3 training exemplars



(b) 10 training exemplars

Figure E.6: Repeating the analysis in Appendix Fig. E.3 on an Places365-trained ResNet-18.

(a) Train exemplars: (*top*) Original and (*bottom*) transformed images.

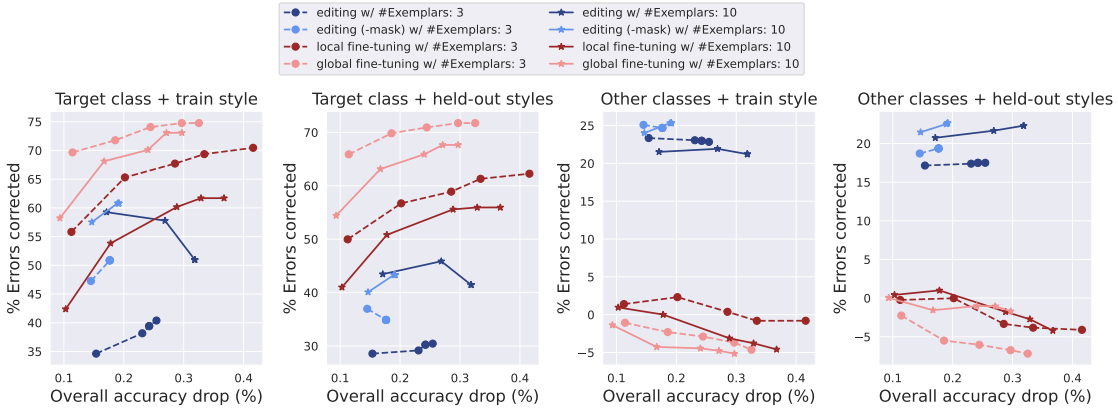
Method: Fine-tuning

(b) Corrections (3.12%) (c) No change (85.63%) (d) New errors (11.25%)

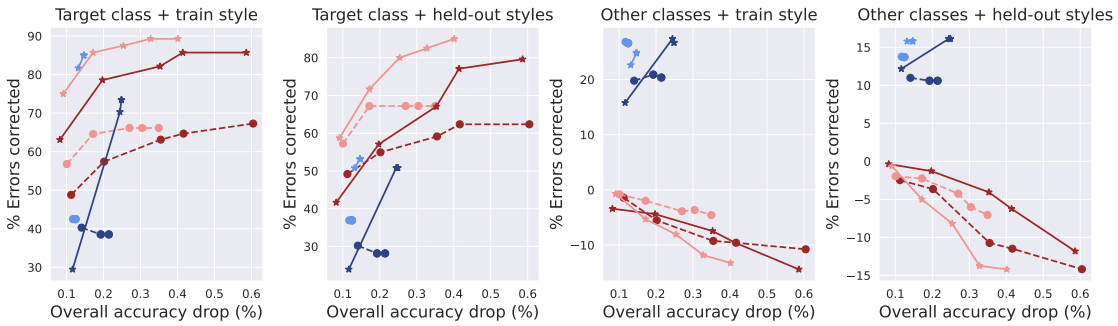
Method: Editing

(e) Corrections (32.89%) (f) No change (59.87%) (g) New errors (7.24%)

Figure E.7: Examples of errors (not) corrected when rewriting "grass" into "snow". The true label, as well as model predictions pre/post-edit for each image are in the title. (a) Train exemplars for editing and fine-tuning. Test set examples where fine-tuning and editing correct the model error on the transformed example (b/e), do not cause any change (c/f) and induce an new error (d/g).

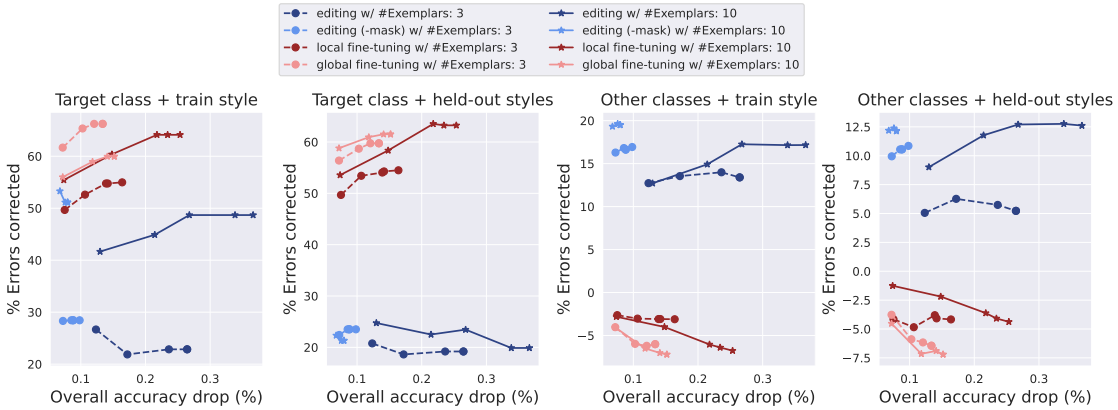


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

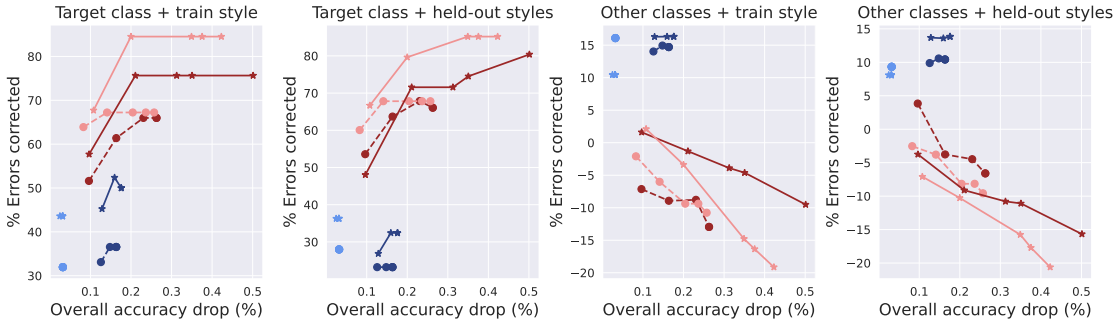


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E.8: Performance vs. drop in overall test set accuracy: Here, we visualize average number of misclassifications corrected by editing and fine-tuning when applied to an ImageNet-trained VGG16 classifier—where the average is computed over different concept-transformation pairs.

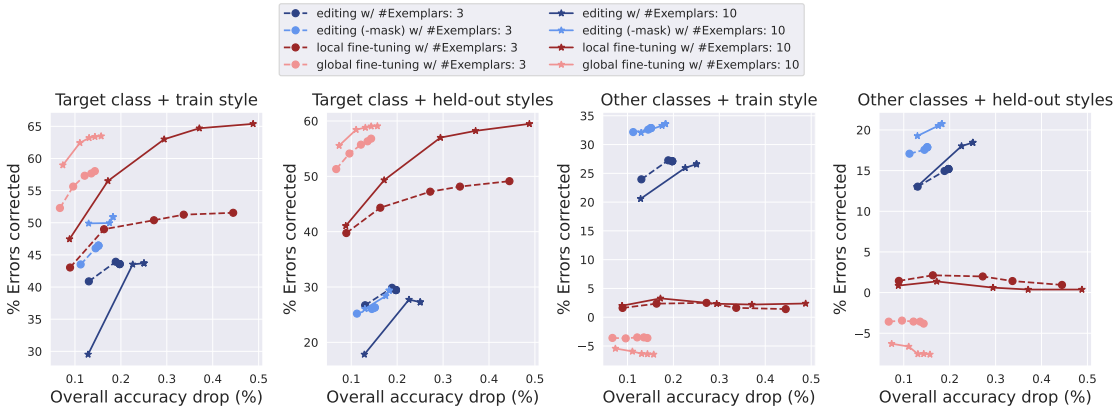


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

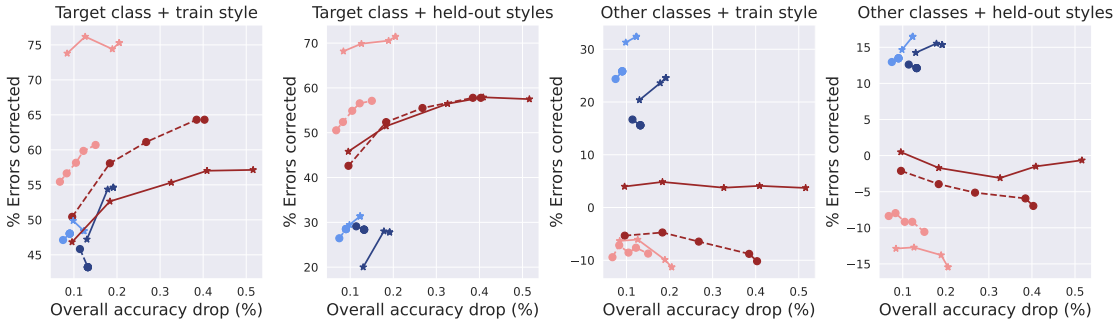


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E.9: Repeating the analysis in Appendix Fig. E.8 on an ImageNet-trained ResNet-50.

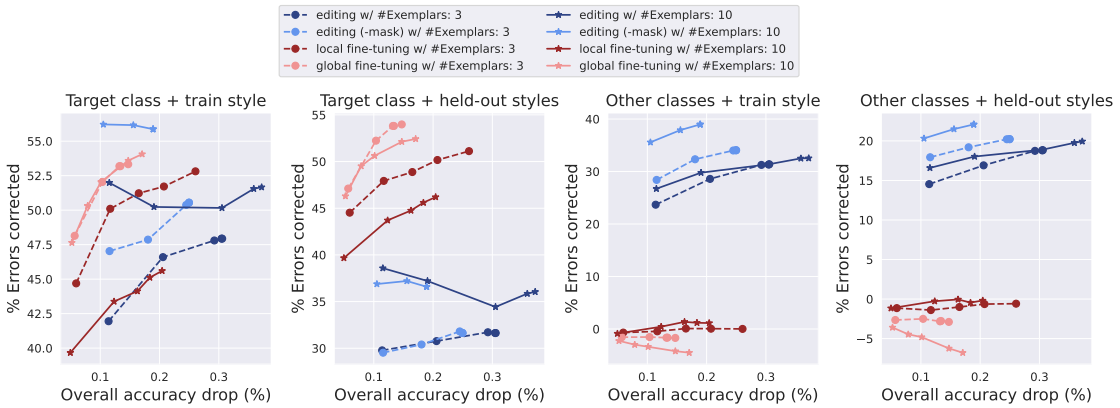


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

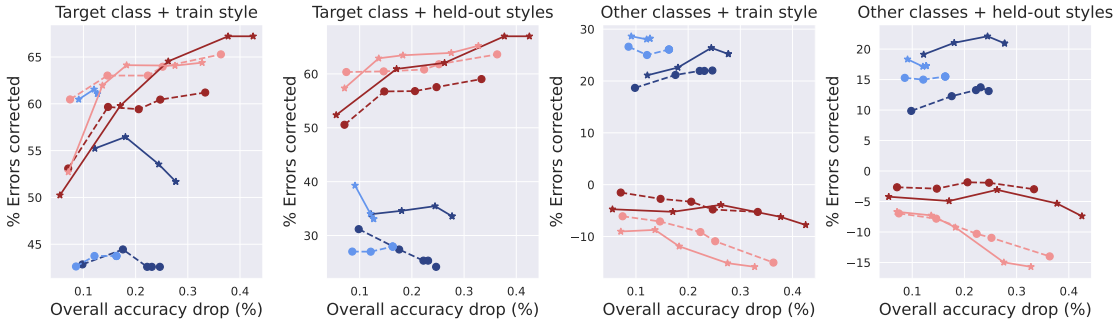


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E.10: Repeating the analysis in Appendix Fig. E.8 on an Places365-trained VGG16.



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

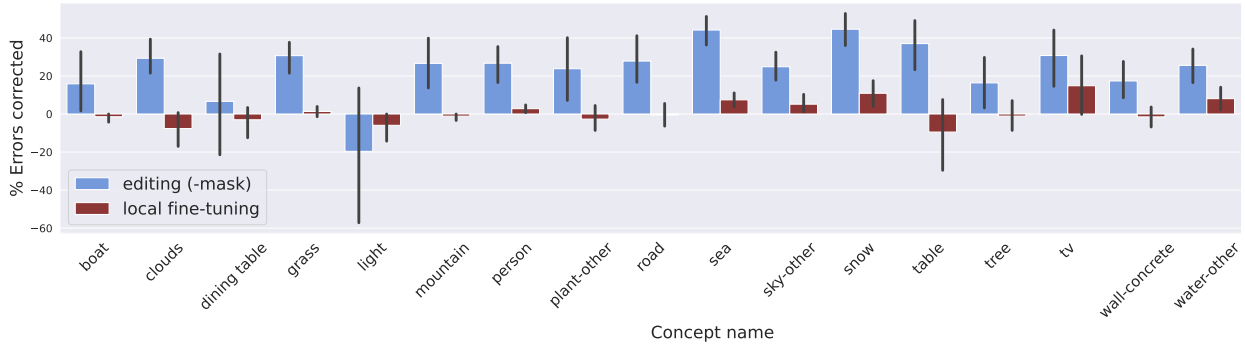


(b) Concepts derived from an instance segmentation model trained on LVIS.

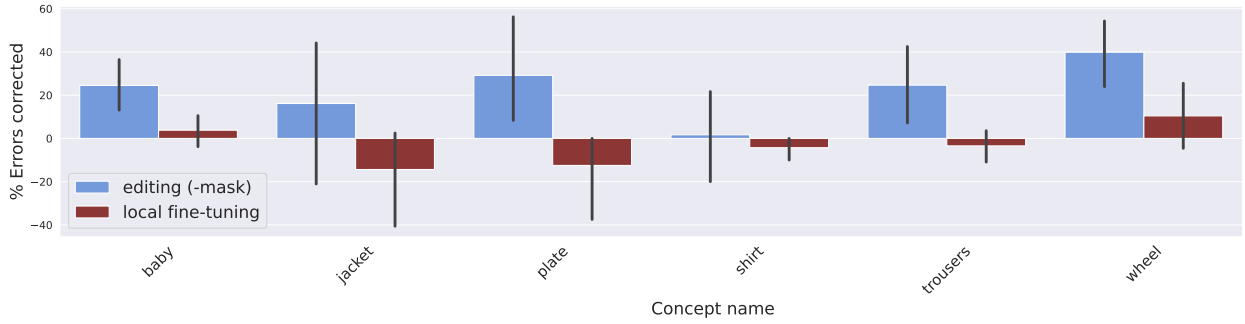
Figure E.11: Repeating the analysis in Appendix Fig. E.8 on an Places365-trained ResNet-18.

E.2.2 A fine-grained look at performance improvements

In Appendix Figures E.12 and E.13 we take a closer look at the performance improvements caused by editing (-mask) and (local) fine-tuning (with 10 exemplars) on an ImageNet-trained VGG16 classifier. In particular, we break down the improvements on test examples from *non-target* classes with the same transformation as training, per-concept and per-style respectively. As before, we only consider hyperparameters that lead to an overall accuracy drop of less than 0.25%.

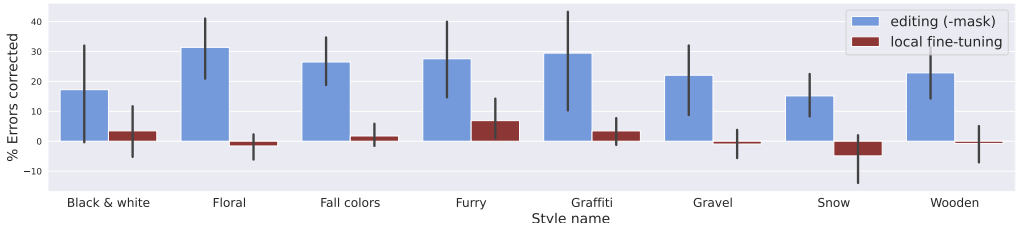


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

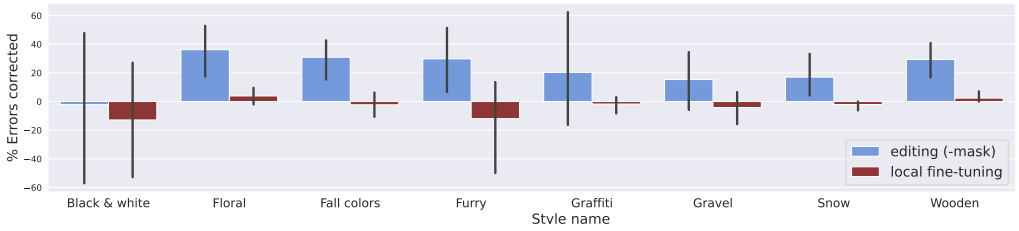


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E.12: Performance of editing and fine-tuning on test examples from non-target classes containing a given concept, averaged across transformations (cf. Appendix E.1).



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

E.2.3 Ablations

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 E.14. 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.

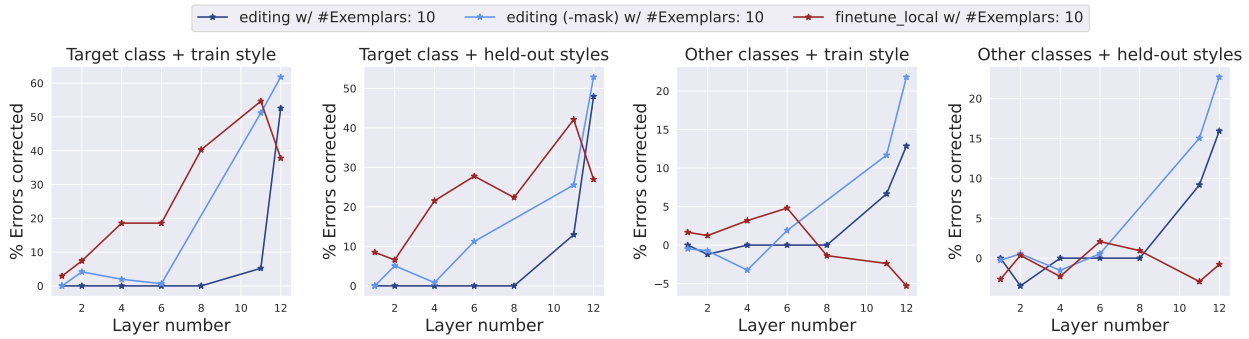


Figure E.14: Editing vs. fine-tuning performance (with 10 exemplars) on an ImageNet-trained VGG-16 classifier, as a function of the layer that is modified. Here, we visualize the average number of misclassifications corrected over different concept-transformation pairs, with concepts derived from instance segmentation modules trained on MS-COCO; and transformations "snow" and "graffiti". For both editing and fine-tuning, the overall drop in model accuracy is less than 0.25%.

Number of exemplars. Increasing the number of exemplars used for each method typically leads to qualitatively the same impact, just more significant, cf. Appendix Figures E.8-E.11. We also perform a more fine-grained ablation for a single model (ImageNet-trained VGG16 on COCO-concepts) in Figure E.15. 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.

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 Appendix Figures E.16-E.19. This suggests that the rank restriction is necessary to prevent the model from overfitting to the few exemplars used.

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

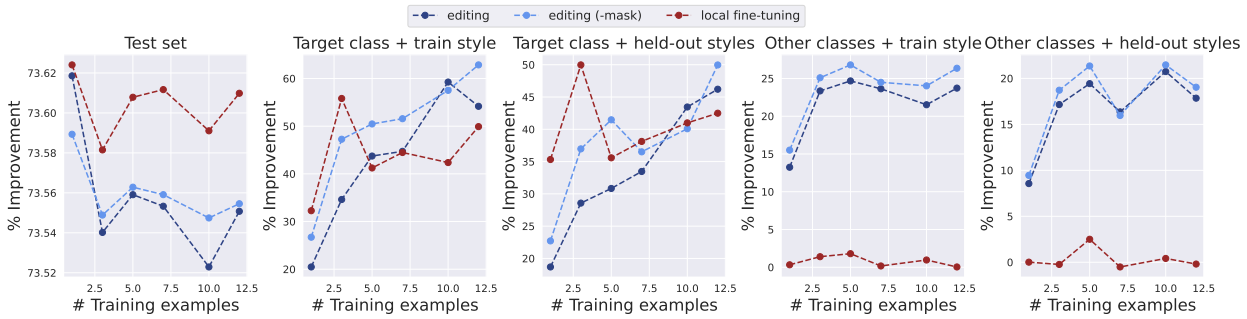
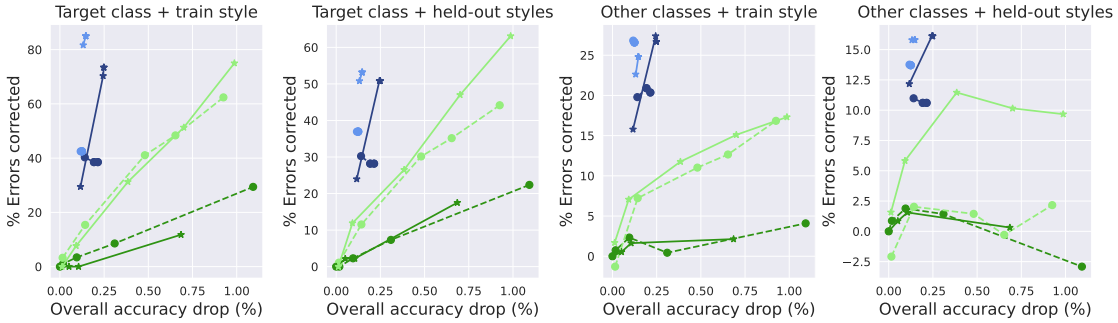


Figure E.15: Editing vs. fine-tuning performance on an ImageNet-trained VGG-16 classifier, as a function of the number of train exemplars. Here, we visualize the average number of misclassifications corrected over different concept-transformation pairs, with concepts derived from instance segmentation modules trained on MS-COCO; and transformations described in Appendix E.1. For both editing and fine-tuning, the overall drop in model accuracy is less than 0.25%.

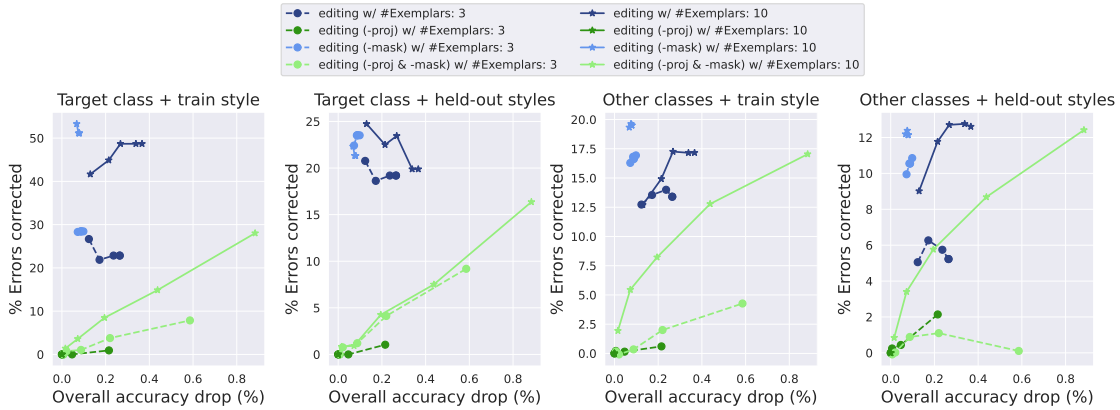
the entirety of the image leads to even better performance—cf. curves corresponding to '-mask' in Appendix Figures E.16-E.19. We hypothesize that this has a regularizing effect as it constrains the weights to preserve the original mapping between keys and values in regions without the concept.

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

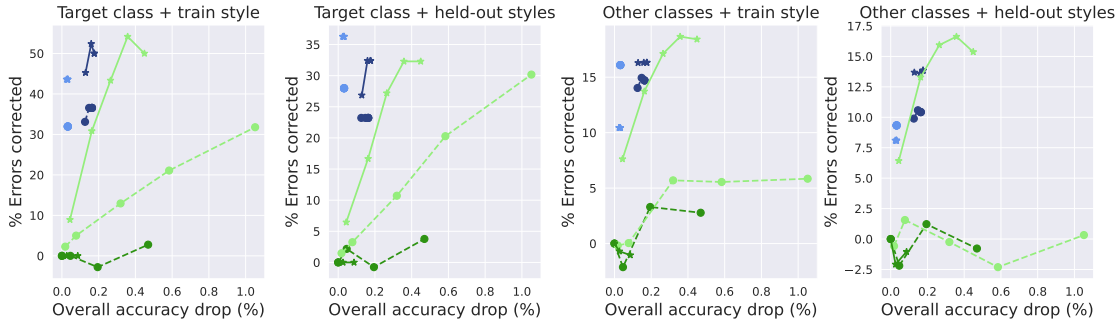


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E.16: Performance vs. drop in overall test set accuracy: Here, we visualize average number of mistakes 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.

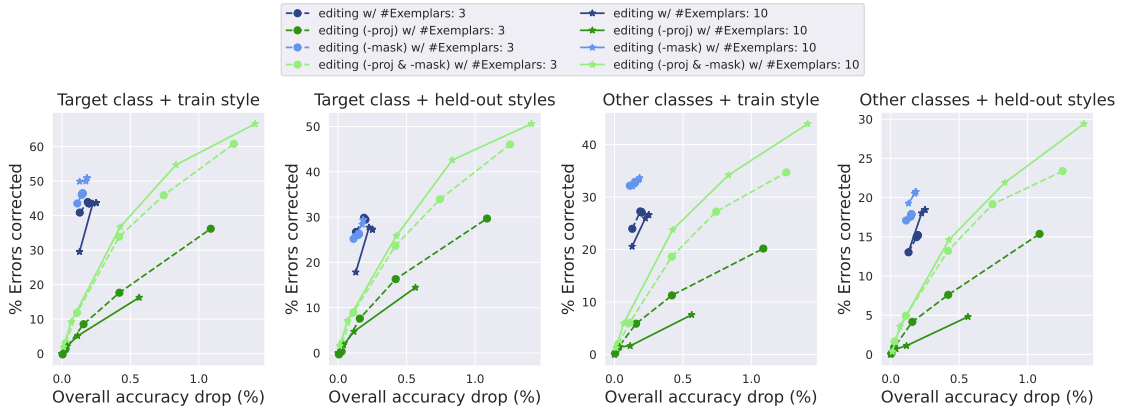


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

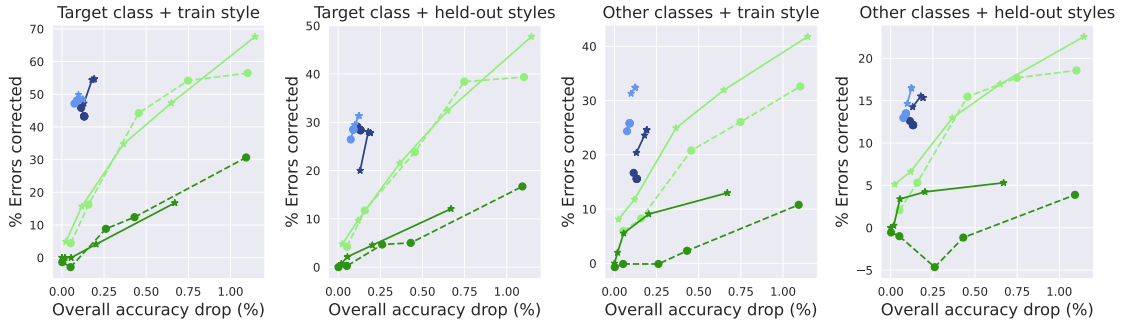


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E.17: Repeating the analysis in Figure E.16 on an ImageNet-trained ResNet-50.

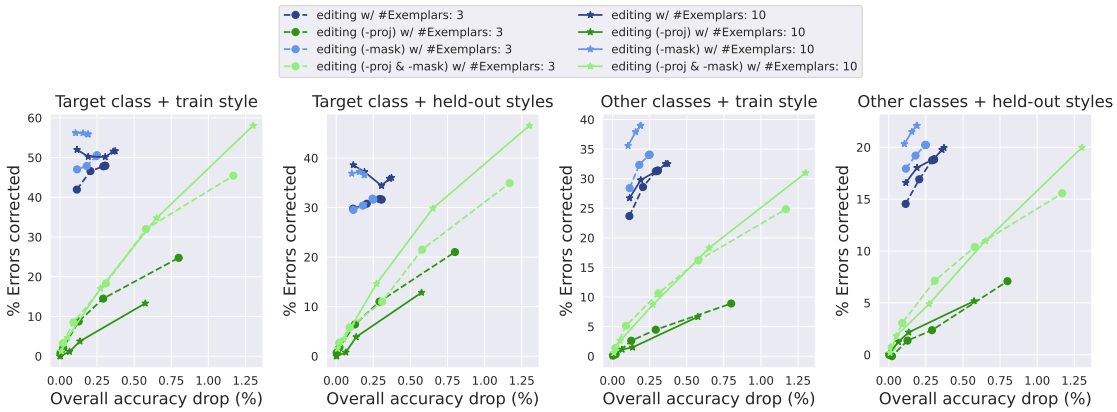


(a) Concepts derived from an instance segmentation model trained on MS-COCO.

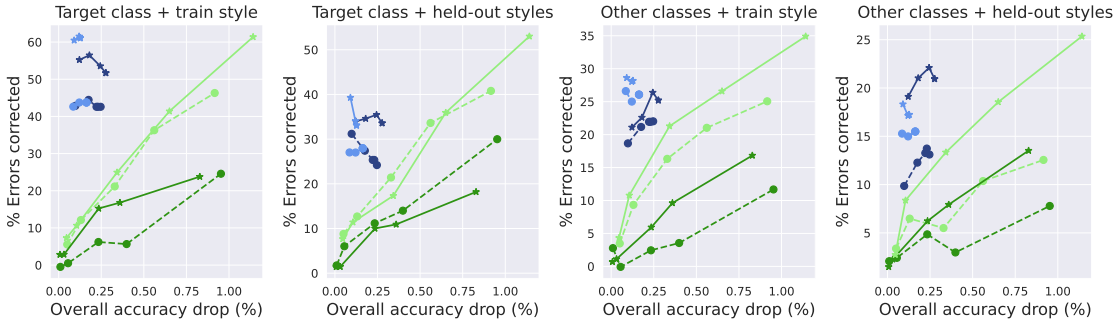


(b) Concepts derived from an instance segmentation model trained on LVIS.

Figure E.18: Repeating the analysis in Appendix Fig. E.16 on an Places365-trained VGG16.



(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.19: Repeating the analysis in Appendix Fig. E.16 on an Places365-trained ResNet-18 classifier.

E.2.4 Fine-grained model behavior on typographic attacks

In Figure E.20, we take a closer look at how effective different rewriting methods (with one train exemplar) are in mitigating typographic attacks. We find that:

- *Local-finetuning:* Corrects only a subset of the errors.
- *Global-finetuning:* Corrects most errors on the attacked images. However, on the flip side it: (i) causes the model to spuriously associate other images with the target class ("teapot") used for fine-tuning and (ii) significantly reduces model accuracy on clean images of "iPod".
- *Editing:* Corrects all errors without substantially hurting model accuracy on clean images.

(a)

(b)

Figure E.20: Effectiveness of different methods in preventing typographic attacks. (a) Model predictions after the rewrite—local fine-tuning often fails to prevent such attacks, while global fine-tuning results in the model associating "iPod" with the class used for fine-tuning ("teapot"). (b) Accuracy on the original test set and clean samples of class "ipod" before and after the rewrite. While global fine-tuning is fairly effective at mitigating typographic attacks, it disproportionately reduces model accuracy on images of iPods.