Probing, Improving, and Verifying Machine Learning Model Robustness

by

Kai Yuanqing Xiao

B.S., EECS, Massachusetts Institute of Technology (2017)
B.S., Math, Massachusetts Institute of Technology (2017)
M.Eng, EECS, Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

September 2022

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

Certified by .....
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

Probing, Improving, and Verifying Machine Learning Model Robustness

by

Kai Yuanqing Xiao

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

Abstract

Machine learning models turn out to be brittle when faced with distribution shifts, making them hard to rely on in real-world deployment. This motivates developing methods that enable us to detect and alleviate such model brittleness, as well as to verify that our models indeed meet desired robustness guarantees.

This thesis presents a set of tools that help us detect model vulnerabilities and biases. This set comprises, on the one hand, a suite of new datasets that allow us to obtain a finer-grained understanding of model reliance on backgrounds. On the other hand, it involves 3DB, a framework that leverages photorealistic simulation, to probe model vulnerabilities to more varied distribution shifts.

In addition to identifying these vulnerabilities, we discuss interventions that can make models more robust to distribution shifts, including using more training data. As we demonstrate, indiscriminately using more auxiliary data is not always beneficial, and we thus develop dataset projection, a method to choose the "right" auxiliary data to use.

Finally, we show how to efficiently and formally verify that our models are robust to one of the most well-studied types of distribution shift: pixel-wise adversarial perturbations.

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

Acknowledgements

No Ph.D. is complete without an acknowledgement of those who were a part of my Ph.D. journey. Whether it was through mentorship, collaboration, or simply being a part of my day-to-day life, everyone mentioned here (and truly, many others not listed here) made the years of my Ph.D. a thoroughly enjoyable experience.

I want to begin by thanking my Ph.D. advisor, Aleksander Mądry. Aleksander has been a great mentor, both in research and in life. What I appreciate most about Aleksander is that while he does care that I succeed, he cares more that I am happy. I am extremely grateful that I was able to work with such a kind and humorous advisor.

I also want to thank my Ph.D. thesis committee, Costis Daskalakis and Martin Rinard, for their feedback and insightful questions. Next, I want to thank my Master's thesis advisor Erik Demaine and my undergraduate research advisors Shafi Goldwasser and Dana Moshkovitz for introducing me to the world of computer science research[1]. Finally, I want to thank my research internship mentors Sven Gowal and Jerry Li for teaching me how research is conducted and applied in industry.

My research has always been significantly improved as a result of the efforts of my collaborators. I am so thankful to have been able to work together with many brilliant and hard-working researchers, including Vincent Tjeng, Russ Tedrake, Mahi Shafiullah, Aleksander Mądry, Sven Gowal, Todd Hester, Rae Jeong, Daniel Mankowitz, Yuanyuan Shi, Tsui-Wei Weng, Nir Levine, Timothy Mann, Dj Dvijotham, Jonathan Uesato, Robert Stanforth, Pushmeet Kohli, Logan Engstrom, Andrew Ilyas, Guillaume Leclerc, Hadi Salman, Sai Vemprala, Vaibhav Vineet, Pengchuan Zhang, Shibani Santurkar, Greg Yang, Ashish Kapoor, Sam Park, Andy Wei, Jerry Li, and Eric Wong.

A wise person once told me, you can't put your life on hold while you're doing your Ph.D. Thus, I want to thank many amazing friends who were a part of my life these past few years. First, thank you to my labmates for making the MadryLab such a fun environment to work in. I will just list labmates who were not already mentioned as collaborators; they definitely made me excited to show up to lab every day — Dimitris Tsipras, Brandon Tran, Aleksander Makelov, Saachi Jain, Alaa Khaddaj, Kristian Georgiev, Harshay Jain, Hedi Driss, Sarah Cen, and Kamila, Natalia, and Samanta as well! And of course, so many things I appreciate about our lab (e.g., free food) would not be possible without Debbie. Second, thank you to Michael "k3soju" Zhang, BLACKPINK (especially Rosé), Jenny Xu and Abs with Jenny, and Kendall Square's local lunch locations (Cava, Sweetgreen, and Chipotle) for keeping me fed. Next, I'd

[1]It was in theoretical computer science, and not in machine learning!

like to thank my Boston family, who made Boston feel like home. First, thank you to all the wonderful roommates I've had the pleasure of living with during my Ph.D. — Ben Chan, Mike Sun, Weilian Chu, Raymond Ng, Brandon Zeng, Kevin Lu, Mark Chounlakone, Nalini Singh, Regina Apodaca, and last but certainly not least, Rumya Raghavan. Also a huge shoutout to my "neighbhors" who I see just as often as my roommates, for all the meals, board game nights, and other shenanigans — Tanya Talkar, Shruthi Narayanan, Angela Song, Taylor Tang, Sadie Swift, and Claire Luo.

Additionally, I want to give a shoutout to my friends whom I've really enjoyed spending time with during my Ph.D. years, including (but definitely not limited to) Sunny Tian, Kevin Tian, Kevin Li, Daniel Sun, Alyssa Chen, CK Ieng, John Cherian, Hao Shen, Surya Bhupatiraju, Sam Judd, Michael Shum, Benji Lin, Stuart Finney, Shraraman Ray Chandroori, Kevin "umi" Peng, Liang Zhou, Mycal Tucker, Daniel Zuo, Vickie Ye, Jeffrey Ling, Sindy Tan, Wenkai Lei, Victor Xu, Johnny Ho, Aidi Zhang, Helen Jiang, Audrey Li, Jimmy Wu, Deniz Oktay, Giri Anand, Andrew Xia, Kimberley Yu, Mo Zhao, Claire Nobuhara, Matt Tung, Bryan Chen, MIT Movementality, the fantasy football leagues I'm a part of, Ginny Sun, Michelle Tang, Tony Zeng, Albert Gu, Eshaan Nischani, Edward Park, James Roggeveen, Lisa Ruan, Jeana Choi, and Colin Wei. If you are reading this and feel like your name should have been on this list — please reach out to me and let's hang out in the future! :P

And finally, I want to thank my family, especially my mom and dad, for always being unconditionally supportive. They were with me for every step of my journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning (ML) is, in many ways, a success story with the potential to shape society. Driven by advances in algorithms, faster hardware and more compute, and ever-growing datasets, ML, and deep learning in particular, has shown immense promise. In 2012, deep learning grew in popularity after surpassing the previous state-of-the-art on the popular computer vision benchmark ImageNet [Rus+15]. Since then, deep learning has achieved state-of-the-art results on a variety of other domains, including natural language processing [Dev+19; Bro+20; Cho+22], recommender systems [He+17b; Zha+19], object detection [He+17a; RF18], biology [Jum+21], and game-playing [Sil+16; Sil+17; Ber+19; Sch+20b]. It has enabled the automation of driverless vehicles, drug discovery, and financial strategies. Given the promise and potential of ML, we may be tempted to use ML for more and more economically and societally important decisions.

1.1 ML models make brittle predictions

However, ML is not a silver bullet that works in every scenario. While ML performs impressively well on average, predictions of ML models are extremely brittle, and ML models may learn spurious correlations that perpetuate existing biases.

One of the most striking demonstrations that predictions of ML models are brittle is the phenomenon of *adversarial examples* [Big+13; Sze+14]. In Figure 1-1, a high-accuracy ML model correctly classifies the image on the left as a pig with high confidence. However, after adding an imperceptible amount of carefully-crafted noise to the image, we can create the adversarial image on the right. This image still looks like a pig to humans, but the same high-accuracy ML model incorrectly predicts that

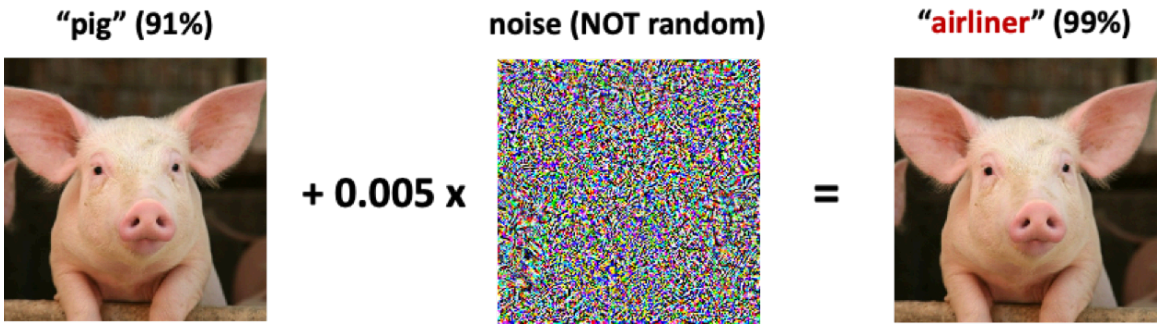


Figure 1-1: A high-accuracy ML model correctly classifies the pig (left). Adding a small amount of imperceptible noise results in an adversarial example (right), which the ML model now confidently yet incorrectly predicts as an airliner.

it is an airliner with even higher confidence.

The phenomenon of adversarial examples is just one of many cases where ML models lack robustness. More broadly, ML predictions are brittle when the data ML models see at test time comes from a different distribution than the data these models see at training time. This change in the data distribution is often called *distribution shift*. We show two examples of such distribution shifts in Figure 1-2. This state of affairs raises questions about whether ML is truly ready for real-world deployment.

Nevertheless, practitioners have already begun to use ML for critical systems in the the real-world. At times, it can fail catastrophically, and in other cases, it can make the right decision based on the wrong reasons. Although rare, self-driving car have gotten into accidents [PMP20] and even caused casualties. ML models have been used for critical decisions such as predictive policing [Pea10], estimating criminal recidivism [RWC18], and loan approval [SGK20], but such models have also been shown to make unfair and biased decisions based on factors such as race and gender [BG18]. Finally, ML models have been used for predictions in health care, but such models can depend on spurious correlations such as hospital-specific markers [Zec+18] or the presence of a ruler (rather than a tumor) in an image [Est+17].

Thus, it is crucial to remedy the disconnect between the brittleness of ML and its extensive usage in real-world settings. In this thesis, we aim to answer the central question:

How can we ensure that our ML models are reliable enough for real-world deployment?

We approach this question from three angles. First, ML models have vulnerabilities and biases that make them difficult to rely on. Diagnosing exactly what these vulnerabilities and biases are can help us determine whether our ML models have

Figure 1-2: ML models are also sensitive to changes such as common corruptions (left, [HD19]) and simple geometric transformations such as rotations (right, [Eng+19b]). ML model predictions change even when the ground-truth labels have not changed.

learned the right correlations. If we expect these correlations to still hold in real-world situations, this can increase our confidence in deploying such models. Thus, our first objective is to create tools to probe model robustness. Using these tools, we seek a finer-grained understanding of exactly which signals models are most sensitive to.

Second, when model vulnerabilities and biases can be identified, it is critical to perform interventions to make models more robust. Understanding what elements of the training process improve robustness to model vulnerabilities is a key step to improving the reliability of ML models.

Finally, even after performing robustness interventions, ML models may still have vulnerabilities. To ensure the reliability of ML models, we further seek to formally verify that these models satisfy certain robustness guarantees. Having mathematically sound guarantees of robustness makes ML models much more deployment-ready in safety-critical settings where these guarantees must be satisfied. By probing, improving, and verifying ML model robustness, we make strides toward deployable ML.

1.2 Why does ML brittleness exist?

Before delving into ways to understand and improve model robustness, we begin by discussing why ML model brittleness exists in the first place. Part of the strength, but also the danger, of ML models is that they are excellent correlation extractors. In fact, they often succeed by exploiting any correlations between inputs and their labels that can improve test accuracy. As a result, models tend to overuse some correlations, such as prioritizing object textures over object shape [Gei+19a], and also exploit other correlations that humans cannot even perceive, such as non-robust features [Ily+19]. Indeed, while the added adversarial noise in Figure 1-1 may seem incomprehensible to

us, it could be a pattern that occurs more frequently with the "airliner" class, and therefore a valid correlation for the ML model to use.

Models use these correlations because they improves test accuracy; however, this gives rise to model brittleness to changes in these features as well. This can be especially concerning if the correlations models use are different from the ones that humans consider to be important. If a correlation is not human-aligned, then the correlations that exist on the training set will not necessarily hold under distribution shift in the test set. Thus, the fact that ML models exploit not human-aligned correlations can have negative implications for model reliability and robustness.

1.3 Probing model robustness

Although we know that models have vulnerabilities and biases due to the set of correlations they learn from training data, we do not always have tools to characterize exactly what these correlations are. Indeed, a multitude of studies have tried to shed light on some of these correlations. In computer vision, ML models are sensitive to a variety of distribution shifts, including slight rotations and translations [Eng+19b; KMF18], image background changes [RSG16; ZXY17], new object orientations [Alc+19; Bar+19], sub-population frequency shifts [STM21], image corruptions [HD19], and changes to the data collection pipeline [Rec+19; Eng+20]. In this thesis, we develop multiple toolkits to help us understand model vulnerabilities and biases, both more deeply and more broadly.

Fine-grained understanding of image backgrounds. We begin with a deep dive into a well-known correlation that models rely on—image backgrounds. Image backgrounds are a natural source of correlation between images and their labels in object recognition. Indeed, prior work has shown that models may use backgrounds in classification [ZXY17; RSG16; BHP18], and suggests that even human vision makes use of image context for scene and object recognition [Tor03]. However, most prior work focused on anecdotes or newly-curated small datasets, and it was unclear whether findings generalized to modern classifiers and datasets like ImageNet [Rus+15]. Thus, we aim to study how current image classifiers utilize image backgrounds by focusing on ImageNet and analyzing state-of-the-art training methods, architectures, and pre-trained models tuned to work well for it. To this end, in Chapter 2 we create a toolkit for disentangling foreground and background signal on ImageNet images, and find that (a) models can achieve non-trivial accuracy by relying on the background alone, (b) models often misclassify images even in the presence of correctly

classified foregrounds—up to 88% of the time with adversarially chosen backgrounds, and (c) more accurate models tend to depend on backgrounds less. Our analysis of backgrounds brings us closer to understanding which correlations machine learning models use, and how they determine models' out of distribution performance.

Leveraging synthetic data. Next, we answer the question more broadly. We develop 3DB, a framework that leverages photorealistic simulation to automatically probe model vulnerabilities to a large variety of distribution shifts, and present findings from 3DB in Chapter 3. By using a system that integrates with photorealistic renderers, we can progammatically control what model biases we want to probe. This helps us reproduce previous robustness analyses with ease, and also allows us to study model sensitivity to the composition of different biases. Studying synthetic images helps us discover situations where ML models are expected to perform well, but instead fail to make the right decision. Further, we show that many insights generated in the synthetic data setting transfer to real-world data as well.

Overall, these toolkits enable us to better pinpoint the exact set of correlations ML models pick up from training data. This tells us if ML models are succeeding based on human-aligned signals, and informs us on if we should rely on such models for unseen data from the real-world.

1.4 Improving model robustness and performance

In the previous section, we discussed creating new tools to understand the vulnerabilities and biases that ML models have learned. However, merely understanding what signals models use is not enough; to prepare ML for the real-world, we must find interventions to train models that are more robust and perform better. Prior work shows that techniques such as data augmentation [Cub+19] and adversarial training [GSS15; Mad+18] can improve model robustness to some distribution shifts. Building off the tools introduced in the previous section, we seek to understand what general changes to the training procedure improve our robustness and performance.

We begin with our case study on image backgrounds, and investigate what factors during the model training process affect model robustness to image background changes. In the latter half of Chapter 2, we find that models that perform better on average are also more robust to background changes. Further, we observe that various factors relating to the training dataset, as well as changes to the training algorithm itself, can improve robustness.

The best way to use auxiliary data to improve performance. One general

trend in machine learning is that increased training dataset size frequently leads to improved performance and robustness [Kap+20; Ros+20]. We show in Chapter 2 that this trend also holds for background robustness, while other work shows that the same trend holds in other robustness settings too [Sch+20a; Mår+20]. Unfortunately, sometimes it is difficult to acquire more data, as doing so can involve expensive data collection and labeling procedures.

Thus, to obtain more training data for a target task, one can draw upon related but distinct datasets, or auxiliary datasets. However, in Chapter 4, we show that indiscriminately using auxiliary datasets can actually harm performance instead. In particular, if an auxiliary dataset has relevant but biased data, it could potentially harm model performance. Thus, we put forth the problem of *dataset projection*—finding subsets of auxiliary datasets that are most aligned with a target dataset. These so-called projected datasets can be used as training data to improve performance on target tasks while being substantially smaller than the auxiliary dataset. We then develop a framework for solving such dataset projection problems and demonstrate in a variety of vision and language settings that the resulting projected datasets, when compared to the original auxiliary datasets, (1) are closer approximations of target datasets and (2) can be used to improve test performance or provide analysis for the target datasets.

1.5 Verifying model robustness

After performing interventions to improve our models, we hope that our models are indeed more robust and real-world ready. However, some robustness interventions that appear to work well at first do not actually improve robustness. In particular, models that are trained to be robust to adversarial examples (such as the one in Figure 1-1) can be ineffective when faced with more carefully crafted adversarial examples [CW17a; Ues+18; ACW18; Tra+20]. This state of affairs gives rise to the need for verification of networks, i.e., the task of formally proving that no small perturbations of a given input can cause it to be misclassified by the model.

Although some verifiers have been designed to solve this problem, the verification process is often intractably slow. For example, researchers developed the Reluplex verifier [Kat+17], but verifying robustness for even a small neural network model with less than 100 neurons turns out to be computationally infeasible. Thus, we want to develop methods to efficiently verify ML model robustness to adversarial examples.

Improving the verification procedure via MILP. First, we propose im-

proving the verifier itself, by formulating the problem of verifying model robustness to adversarial examples as a Mixed-Integer Linear Program (MILP) in Chapter 5. This formulation allows us to take advantage of off-the-shelf MILP solvers that have been optimized for decades [Gur17b]. To improve the speed further, we introduce tight formulations for non-linearities in neural networks, and we use a novel presolve algorithm that makes full use of all information available in the adversarial robustness setting. Our MILP verifier is two to three orders of magnitude quicker than prior state-of-the-art. This computational speedup allows us to verify the robustness of larger convolutional networks, and determine, for the first time, the exact adversarial accuracy of an MNIST classifier to norm-bounded perturbations.

Training models that are easier to verify. The verifier can be optimized to improve the speed of ML model robustness verification, but another big factor that determines verification speed is the ML model being verified. Thus, we propose changing the model training process to make the ML model itself more amenable to verification in Chapter 6. To achieve this, we begin by investigating what properties of a model make the MILP verification procedure faster to solve. We find that weight sparsity and so-called stable ReLUs in neural networks lead to verification problems that can be solved more efficiently. Subsequently, we explore co-designing neural networks to be simultaneously robust and easily verifiable by introducing regularization terms in the model training process. These regularization terms achieve the primary goals of weight sparsity and ReLU stability during training without significantly hurting the neural network's accuracy. We show that our techniques can be used in conjunction with any standard training procedure, and that they allows us to train MNIST and CIFAR-10 ML models that are provably robust on most test inputs and can be verified an order of magnitude faster.

Overall, we take two steps forward to ML models that come with provable guarantees of robustness. Ensuring that our models are provably robust improves our confidence that these models are ready to be deployed in the real world, especially for safety critical scenarios where such robustness guarantees are necessary.

1.6 Outlook: Towards deployable machine learning

By probing, improving, and verifying ML model robustness, this thesis builds toward ML that is safe and reliable. Despite the progress, there remains much work to be done. Going forward, I believe that research in two key directions will play an important role in ensuring that ML models can be deployed more extensively and safely in the

real world.

The many faces of robustness. In this work, we primarily focused on robustness to adversarial examples and specific types of distribution shift such as changes to image backgrounds. While these are important stepping stones, the real world comes with far more types of distribution shift that we would want our models to be robust to. For example, what if a self-driving car company trained their autopilot system on millions of images and videos in sunny weather in Phoenix, but now the car needs to operate in rainy weather in Singapore? What if a loan approval system is trained on all data available until today, but a massive overhaul to the financial system means that useful predictive features no longer apply in the future?

One of the key challenges of tackling distribution shift robustness is that distribution shift encompasses such a broad range of changes. In order to move forward, we need varied benchmarks for different types of distribution shifts, including the backgrounds datasets benchmark presented in Chapter 2 and related works such as WILDS [Koh+20] and those of Hendrycks et al. [Hen+20]. Along with curating more realistic benchmarks, we must continue to develop synthetic benchmarks like 3DB. While real-world data is the gold standard, generating synthetic data is far more scalable. We already found that insights from photorealistically rendered 3DB images transfer well to real data in Section 3.4—and with recent advances in deep generative models such as DALL-E 2 [Ram+22], generative models could also be used to simulate real data effectively. These synthetic data sources can be scaled both for more extensive testing of model sensitivity to different factors of variation, and also for usage as training data to encourage model invariance and robustness to those factors.

How does robustness transfer? Due to the trend that ML model perform significantly better when dataset size, model size, and compute are scaled to extreme amounts, most state-of-the-art models in today's world are large models trained by large tech companies. Notably, individuals or academic labs are unlikely to be able to train ML models of such scale by themselves. As a result, the future of ML is likely to involve individual ML practitioners starting with big foundation models [Bom+21] trained by a big company, and fine-tuning [Rad+19; Dev+19] or otherwise tuning [LL21] it for their own use case. Thus, understanding how robustness of the original large models transfer upon fine-tuning is an important direction for ensuring the safety of real-world ML use cases.

1.7 Thesis organization

We now describe how the rest of the thesis is organized.

Chapter 2 explores the role of image backgrounds in object recognition, and introduces a toolkit of new evaluation datasets to help probe model reliance on background signals over time. The material presented in this chapter is based on joint work with Logan Engstrom, Andrew Ilyas, and Aleksander Mądry [Xia+21b].

Chapter 3 presents 3DB, a framework for debugging computer vision models. We use this framework to automate discovery of ML model biases and vulnerabilities. The material presented in this chapter is based on joint work with Logan Engstrom, Andrew Ilyas, Guillaume Leclerc, Hadi Salman, Sai Vemprala, Vibhav Vineet, Pengchuan Zhang, Shibani Santurkar, Greg Yang, Ashish Kapoor, and Aleksander Mądry [Lec+21a].

Chapter 4 discusses a refinement of the idea that more data is always useful for improving model performance and robustness. It puts forth dataset projection, a method to find the most useful subset of additional data to improve model performance. The material presented in this chapter is based on joint work with Aleksander Mądry and Eric Wong [WXM22].

Chapter 5 tackles the problem of verification of ML model robustness to adversarial examples by formulating the problem as a mixed-integer linear program. The material presented in this chapter is based on joint work with Russ Tedrake and Vincent Tjeng [TXT19].

Chapter 6 details how to improve the robustness verification process by co-designing ML models that are simultaneously robust and easy-to-verify. The material presented in this chapter is based on joint work with Aleksander Mądry, Mahi Shafiullah, and Vincent Tjeng [Xia+19b].

Part I

Probing model robustness

Chapter 2

Investigating the role of image backgrounds

2.1 Introduction

Object recognition models are typically trained to minimize loss on a given dataset, and evaluated by the accuracy they attain on the corresponding test set. In this paradigm, model performance can be improved by incorporating any generalizing correlation between images and their labels into decision-making. However, the actual model reliability and robustness depend on the specific set of correlations that is used, and on how those correlations are combined. Indeed, outside of the training distribution, model predictions can deviate wildly from human expectations either due to relying on correlations that humans do not perceive [JLT18; Ily+19; Jac+19], or due to overusing correlations, such as texture [Gei+19a; Bak+18] and color [YS02], that humans do use (but to a lesser degree). Characterizing the correlations that models depend on thus has important implications for understanding model behavior, in general.

Image backgrounds are a natural source of correlation between images and their labels in object recognition. Indeed, prior work has shown that models may use backgrounds in classification [Zha+07; RSG16; ZXY17; RZT18; Zec+18; Bar+19; SSF19; Sag+20; Gei+20], and suggests that even human vision makes use of image context for scene and object recognition [Tor03]. In this work, we aim to obtain a deeper and more holistic understanding of how current state-of-the-art image classifiers utilize image backgrounds. To this end, in contrast to most of the prior work (which

tends to study relatively small and often newly-curated image datasets[1]), our focus is on ImageNet [Rus+15]—one of the largest and most widely used datasets, with state-of-the-art training methods, architectures, and pre-trained models tuned to work well for it.

Zhu, Xie, and Yuille [ZXY17] analyze ImageNet classification (focusing on the older, AlexNet model) to find that AlexNet achieves small but non-trivial test accuracy on a dataset consisting of only backgrounds (where foreground objects are replaced by black rectangles). While sufficient for establishing that backgrounds can be used for classification, we aim to go beyond those initial explorations to get a more fine-grained understanding of the relative importance of backgrounds and foregrounds, for newer, state-of-the-art models, and to provide a versatile toolkit for others to use. Specifically, we investigate the extent to which models rely on backgrounds, the implications of this reliance, and how models' use of backgrounds has evolved over time. Concretely:

- We create a suite of datasets that help disentangle (and control for different aspects of) the impact of foreground and background signals on classification. The code and datasets are publicly available for others to use in this repository: `https://github.com/MadryLab/backgrounds_challenge`.

- Using the aforementioned toolkit, we characterize models' reliance on image backgrounds. We find that image backgrounds alone suffice for fairly successful classification and that changing background signals decreases average-case performance. In fact, we further show that by choosing backgrounds in an adversarial manner, we can make standard models misclassify 88% of images as the background class.

- We demonstrate that standard models not only use but *require* backgrounds for correctly classifying large portions of test sets (35% on our benchmark).

- We study the impact of backgrounds on classification for a variety of classifiers, and find that models with higher ImageNet test accuracy tend to simultaneously have higher accuracy on image backgrounds alone and have greater robustness to changes in image background.

[1]We discuss these works in greater detail in Section 2.5, Related Works.

2.2 Methodology

To properly gauge image backgrounds' role in image classification, we construct a synthetic dataset for disentangling background from foreground signal: ImageNet-9.

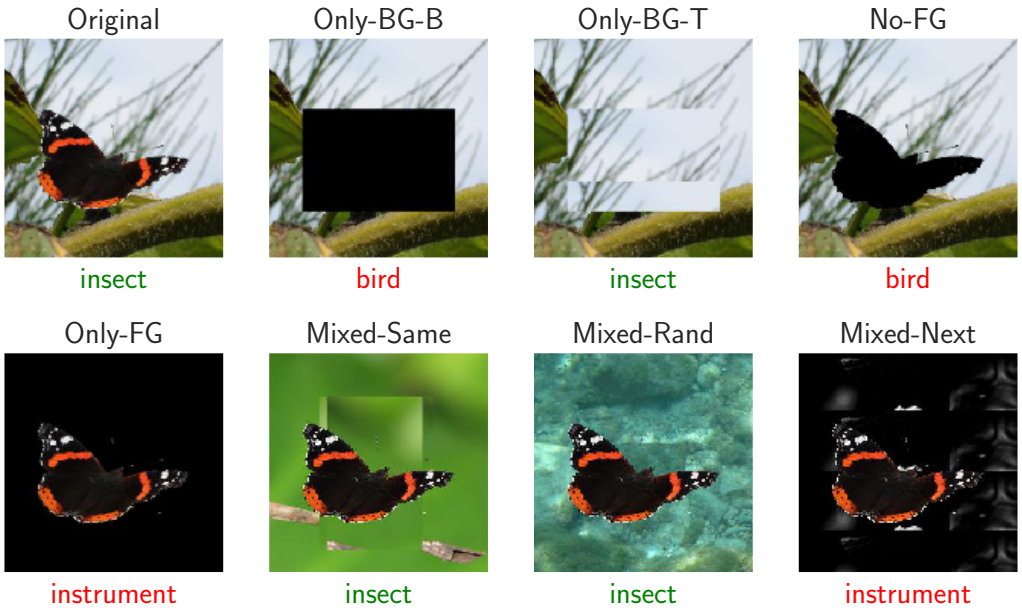


Figure 2-1: Variations of the synthetic dataset ImageNet-9, as described in Table 2.1. We label each image with its pre-trained ResNet-50 classification—green, if corresponding with the original label; red, if not. The model correctly classifies the image as "insect" when given: the original image, only the background, and two cases where the original foreground is present but the background changes. Note that, in particular, the model fails in two cases when the original foreground is present but the background changes (as in MIXED-NEXT or ONLY-FG).

Base dataset: ImageNet-9. We organize a subset of ImageNet into a new dataset with nine coarse-grained classes and call it ImageNet-9 (IN-9) [2]. To create it, we group together ImageNet classes sharing an ancestor in the WordNet [Mil95] hierarchy. We use coarse-grained classes because there are not enough images with annotated bounding boxes (which we need to disentangle backgrounds and foregrounds) to use the standard labels. The resulting IN-9 dataset is class-balanced and has 45405 training images and 4050 testing images. While we can (and do) apply our methods on the full ImageNet dataset as well, we choose to focus on this coarse-grained version

[2] These classes are `dog`, `bird`, `vehicle`, `reptile`, `carnivore`, `insect`, `instrument`, `primate`, and `fish`.

of ImageNet because of its higher-fidelity images. We describe the dataset creation process in detail and discuss the advantages of focusing on IN-9 in Appendix A.1.

Variations of ImageNet-9 From this base set of images, which we call the ORIGINAL version of IN-9, we create seven other synthetic variations designed to understand the impact of backgrounds. We use both rectangular bounding boxes and the foreground segmentation algorithm GrabCut [RKB04], as implemented in OpenCV, to disentangle backgrounds and foregrounds. We visualize these variations in Figure 2-1, and provide a detailed reference in Table 2.1. These subdatasets of IN-9 differ only in how they process the foregrounds and backgrounds of each constituent image.

Larger dataset: IN-9L We finally create a dataset called IN-9L that consists of all the images in ImageNet corresponding to the classes in ORIGINAL (rather than just the images that have associated bounding boxes). This dataset has about 180k training images in total. We leverage this larger dataset to train better generalizing models, and prefer to analyze models trained on IN-9L whenever possible.

Table 2.1: The 8 modified subdatasets created from ImageNet-9, which are visualized in Figure 2-1. The foreground detection method refers to how the pixels corresponding to the foreground are found. GrabCut refers to the foreground segmentation algorithm implemented in OpenCV. Random backgrounds in the last three datasets are taken from ONLY-BG-T. For more details see Appendix A.1.

Name	Foreground	Background	Foreground Detection Method
ORIGINAL	Unmodified	Unmodified	—
ONLY-BG-B	Black	Unmodified	Bounding Box
ONLY-BG-T	Tiled background	Unmodified	Bounding Box
NO-FG	Black	Unmodified	GrabCut
ONLY-FG	Unmodified	Black	GrabCut
MIXED-SAME	Unmodified	Random BG of the same class	GrabCut
MIXED-RAND	Unmodified	Random BG of a random class	GrabCut
MIXED-NEXT	Unmodified	Random BG of the next class	GrabCut

2.3 Quantifying reliance on background signals

With ImageNet-9 in hand, we now assess the role of image backgrounds in classification.

Backgrounds suffice for classification. Prior work has found that models are able to make accurate predictions based on backgrounds alone; we begin by directly quantifying this ability. Looking at the ONLY-BG-T, ONLY-BG-B, and NO-FG

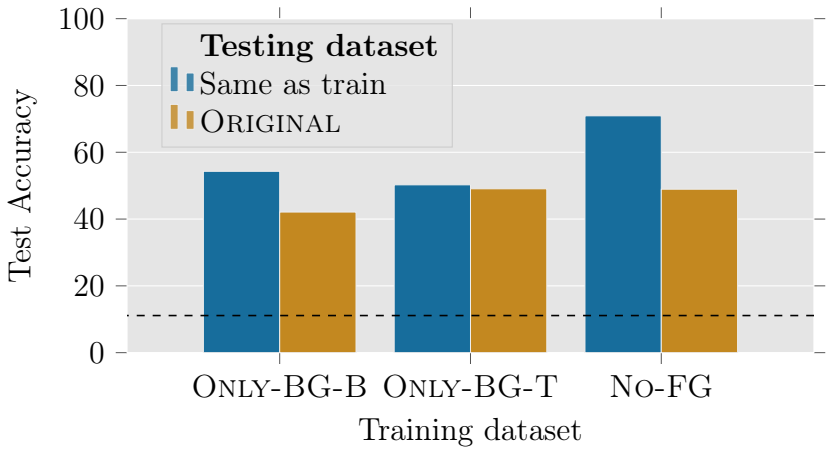


Figure 2-2: We train models on each of the "background-only" datasets, then evaluate each on its corresponding test set as well as the ORIGINAL test set. Even though the model only learns from background signal, it achieves (much) better than random performance on *both* the corresponding test set and ORIGINAL. Here, random guessing would give 11.11% (the dotted line).

datasets, we find (cf. Figure 2-2) that models trained on these background-only training sets generalize reasonably well to both their corresponding test sets and to unmodified images from the ORIGINAL test set (around 40-50% for every model, far above the baseline of 11% representing random classification). Our results confirm that image backgrounds contain signal that models can accurately classify standard images with.

Models exploit background signal for classification. We discover that models can misclassify due to background signal, especially when the background class does not match that of the foreground. As a demonstration, we study model accuracies on the MIXED-RAND dataset, where image backgrounds are randomized and thus provide no information about the correct label. By comparing test accuracies on MIXED-RAND and MIXED-SAME [3], where images have class-consistent backgrounds, we can measure classifiers' dependence on the correct background. We denote the resulting accuracy gap between MIXED-SAME and MIXED-RAND as the BG-GAP; this difference represents the drop in model accuracy due to changing the class signal from the background. In Table 2.2, we observe a BG-GAP of 13-22% and 4-11% for models trained on IN-9L and ImageNet, respectively, suggesting that backgrounds often mislead state-of-the-art models *even when the correct foreground is present.*

[3] MIXED-SAME controls for artifacts from image processing presented in MIXED-RAND. For further discussion, see Appendix A.4.

Table 2.2: Performance of state-of-the-art computer vision models on selected test sets of ImageNet-9. We include both pre-trained ImageNet models and models of different architectures that we train on IN-9L. The BG-Gap is defined as the difference in test accuracy between Mixed-Same and Mixed-Rand and helps assess the tendency of such models to rely on background signal. Architectures are sorted by their test accuracies on ImageNet and Original for pre-trained and IN-9L-trained models, respectively. Shaded in grey are the two architectures that can be directly compared across datasets (ResNet-50 and Wide-ResNet-50x2).

	Pre-trained on ImageNet					**Trained on IN-9L**				
Test dataset	*MobileNet-v3*	*EfficientNet-b0*	*ResNet-50*	*WRN-50x2*	*DPN-92*	*AlexNet*	*ShuffleNet*	*ResNet-50*	*WRN-50x2*	*VGG16-BN*
ImageNet	67.9%	77.2%	77.6%	78.5%	80.0%			—		
Original	95.5%	96.1%	96.9%	96.6%	97.2%	86.7%	95.7%	96.3%	97.2%	97.6%
Only-BG-T	16.3%	16.5%	17.4%	18.8%	17.6%	41.5%	43.6%	43.6%	45.1%	45.7%
Mixed-Same	84.0%	86.2%	91.0%	88.3%	90.5%	76.2%	86.7%	89.9%	90.6%	91.0%
Mixed-Rand	73.2%	76.3%	84.3%	81.4%	86.1%	54.2%	69.4%	75.6%	78.0%	78.0%
BG-gap	10.8%	9.9%	6.7%	6.9%	4.4%	22.0%	17.3%	14.3%	12.6%	13.0%

More Training Data can reduce the BG-Gap. Our results indicate that ImageNet-trained models are less dependent on backgrounds than their IN-9L-trained counterparts—they have a smaller (but still significant) BG-Gap, and perform worse when predicting solely based on backgrounds (i.e., on the Only-BG-T dataset). We explore two ways that ImageNet differs from IN-9L to understand this phenomena—ImageNet has (a) more datapoints than IN-9L, and (b) a more fine-grained class structure. Figure 2-3 shows that more training data reduces the BG-Gap, particularly when the training dataset size approaches the size of ImageNet. This indicates that training on much more data (and thus, more backgrounds) can reduce (but not eliminate) the effect of backgrounds on model predictions. An ablation study of ImageNet's more fine-grained class structure does not find strong evidence supporting its helpfulness (cf. Appendix A.2).

Models are vulnerable to adversarial backgrounds. To understand how worst-case backgrounds impact models' performance, we evaluate model robustness to adversarially chosen backgrounds. We find that 88% of foregrounds are susceptible to such backgrounds; that is, for these foregrounds, there is a background that causes the classifier to classify the resulting foreground-background combination as

Figure 2-3: We compare test accuracies on MIXED-SAME and MIXED-RAND and observe that training with more data reduces the BG-GAP (BG-GAP measures the effect of backgrounds on model predictions). While this trend is true for models trained on both IN-9 and ImageNet, the trend is most noticeable for models trained on the largest training set, the full ImageNet dataset—this is shown on the far right side of the graph.

the background class. For a finer grained look, we also evaluate image backgrounds based on their attack success rate (ASR), i.e., how frequently they cause models to predict the (background) class in the presence of a conflicting foreground class. As an example, Figure 2-4 shows the five backgrounds with the highest ASR for the insect class—these backgrounds (extracted from insect images in ORIGINAL) fool a IN-9L-trained ResNet-50 model into predicting insect on up to 52% of non-insect foregrounds. We plot a histogram of ASR over all insect backgrounds in Figure A-16 of the Appendix—it has a long tail. Similar results are observed for other classes as well (cf. Appendix A.7).

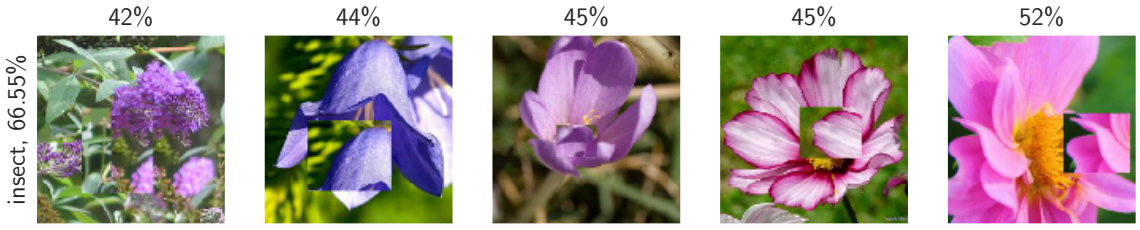


Figure 2-4: The adversarial backgrounds that most frequently fool IN-9L-trained models into classifying a given foreground as insect, ordered by the percentage of foregrounds fooled. The total portion of images that can be fooled (by any background from this class) is 66.55%.

Training on MIXED-RAND reduces background dependence. Next, we explore how to reduce models' dependence on background. To this end, we train models on MIXED-RAND, a synthetic dataset where background signals are decorrelated from class labels. As we would expect, MIXED-RAND-trained models extract less signal from backgrounds: evaluation results show that MIXED-RAND models perform poorly (15% accuracy—barely higher than random) on datasets with only backgrounds and no foregrounds, (ONLY-BG-T or ONLY-BG-B).

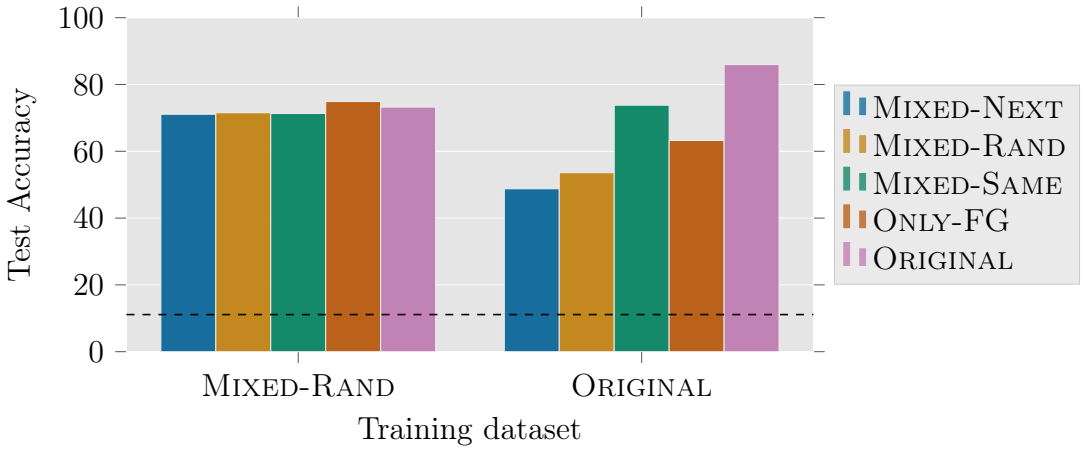


Figure 2-5: We compare the test performance of a model trained on the synthetic MIXED-RAND dataset with a model trained on ORIGINAL. We evaluate these models on variants of IN-9 that contain identical foregrounds. For the ORIGINAL-trained model, test performance decreases significantly when the background signal is modified during testing. However, the MIXED-RAND-trained model is robust to background changes, albeit at the cost of lower accuracy on images from ORIGINAL.

Indeed, such models are also more accurate on datasets where backgrounds do not match foregrounds. In Figure 2-5, we observe that a MIXED-RAND-trained model has 17.3% higher accuracy than its ORIGINAL-trained counterpart on MIXED-RAND, and 22.3% higher accuracy on MIXED-NEXT, a dataset where background signals class-consistently mismatch foregrounds. (Recall that MIXED-NEXT images have foregrounds from class y mixed with backgrounds from class $y+1$, labeled as class y.) The MIXED-RAND-trained model also has little variation (at most 3.8%) in accuracy across all five test sets that contain the correct foreground.

Qualitatively, the MIXED-RAND-trained model also appears to place more relative importance on foreground pixels than the ORIGINAL-trained model; the saliency maps of the two models in Figure 2-6 show that the MIXED-RAND-trained model's saliency maps highlight more foreground pixels than those of ORIGINAL-trained models.

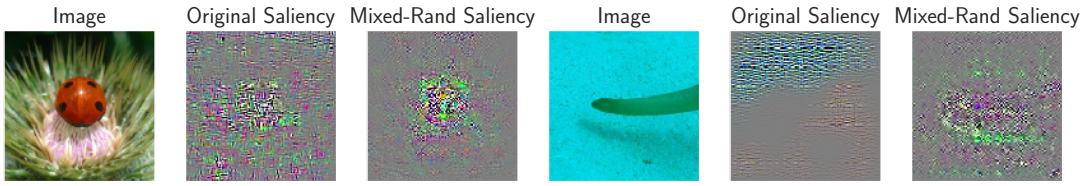


Figure 2-6: Saliency maps for the the ORIGINAL and MIXED-RAND models on two images. As expected, the MIXED-RAND model appears to place more importance on foreground pixels.

Table 2.3: Prediction categories we study for a given image-model pair. For a given image, a model can make differing predictions based on the presence or absence of its foreground/background. We label each possible case based on how the background classification relates to the full image classification and the foreground classification. To proxy classifying full images, foregrounds, and backgrounds separately, we classify ORIGINAL, MIXED-RAND, and ONLY-BG-T (respectively). "BG Irrelevant" demarcates images where the foreground classification result is the same as that of the full image (in terms of correctness). We show illustrative examples of BG Required and BG+FG Required below.

Label	Correct Prediction on Full Image	Correct Prediction on Foreground	Correct Prediction on Background
BG Required	✓	✗	✓
BG Fools	✗	✓	✗
BG+FG Required	✓	✗	✗
BG+FG Fools	✗	✓	✓
BG Irrelevant	✓/✗	✓/✗	—

A fine grained look at dependence on backgrounds. We now analyze models' reliance on backgrounds at an image-by-image level and ask: for which images does introducing backgrounds help or hurt classifiers' performance? To this end, for each image in ORIGINAL, we decompose how models use foreground and background signals by examining classifiers' predictions on the corresponding image in MIXED-RAND and ONLY-BG-T. Here, we use the MIXED-RAND and ONLY-BG-T predictions as a proxy for which class the foreground and background signals (alone) point towards, respectively. We categorize each image based on how its background and foreground signals impact classification; we list the categories in Table 2.3 and show the counts for each category as a histogram per classifier in Figure 2-8. Our results show that while few backgrounds induce misclassification (see Appendix A.8 for examples), a large fraction of images require backgrounds for correct classification—approximately 35% on the ORIGINAL trained classifiers, as calculated by combining the "BG Required" and "BG+FG Required" categories.

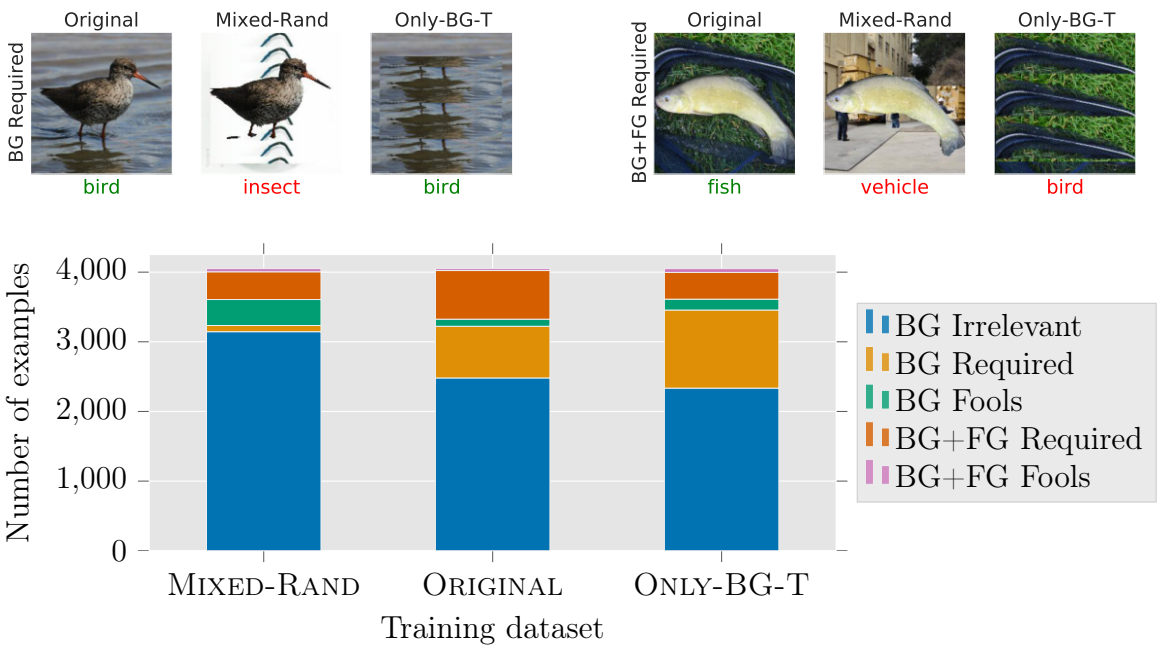


Figure 2-8: We categorize each test set image based on how a model classifies the full image, the background alone, and the foreground alone (cf. Table 2.3). The model trained on ORIGINAL needs the background for correct classification on 35% of images (measured by adding "BG Required" and "BG+FG Required), while a model trained on MIXED-RAND is much less reliant on background. The model trained on ONLY-BG-T requires the background most, as expected; however, the model often misclassifies both the full image and the background, so the "BG Irrelevant" subset is still sizable.

Further insights derived from IN-9 are discussed in the Appendix A.4. We focus on key findings in this section, but also include more comprehensive results and examples of other questions that can be explored by using the toolkit of IN-9 in the Appendix.

2.4 Benchmark progress and background dependence

In the previous sections, we demonstrated that standard image classification models exploit signals from backgrounds. Considering that these models result from progress on standard computer vision benchmarks, a natural question is: *to what extent have improvements on image classification benchmarks resulted from exploiting background correlations?* And relatedly, *how has model robustness to misleading background signals evolved over time?*

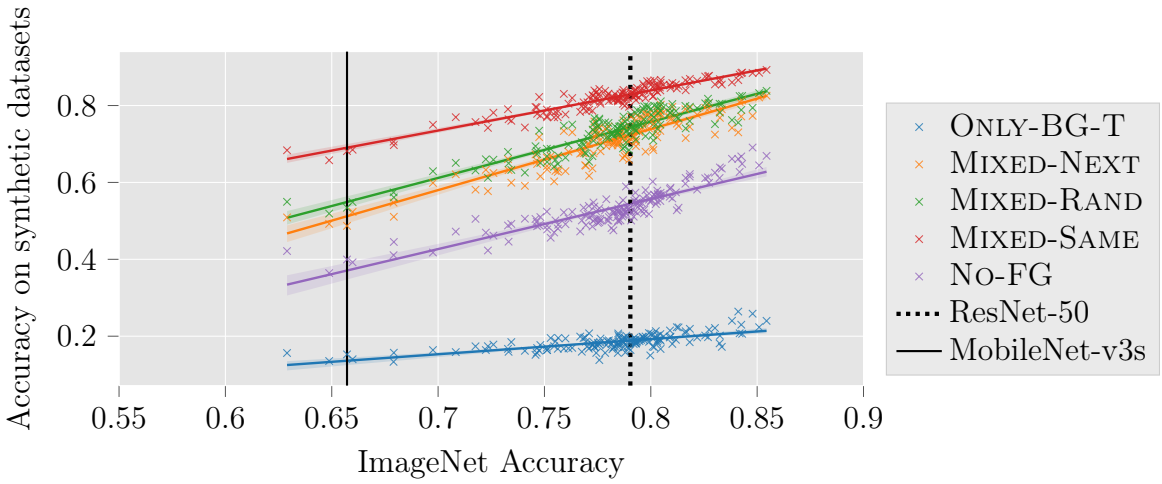


Figure 2-9: Measuring progress on each of the synthetic ImageNet-9 datasets with respect to progress on the standard ImageNet test set. Higher accuracy on ImageNet generally corresponds to higher accuracy on each of the constructed datasets, but the rate at which accuracy grows varies based on the types of features present in each dataset. Each pre-trained model corresponds to a vertical line on the plot—we mark ResNet-50 and MobileNet-v3s models for reference.

As a first step towards answering these questions, we study the progress made by ImageNet models on our synthetic IN-9 dataset variations. In Figure 2-9 we plot accuracy on our synthetic datasets against ImageNet accuracy for each of the architectures considered. As evidenced by the lines of best fit in Figure 2-9, accuracy increases on the original ImageNet benchmark generally correspond to accuracy increases on all of the synthetic datasets. This includes the Only-BG datasets—indicating that models *do* improve at extracting correlations from image backgrounds.

Indeed, the Only-BG trend observed in Figure 2-9 suggests that either (a) image classification models can only attain their reported accuracies in the presence of background signals; or (b) these models carry an implicit bias towards features in the background, as a result of optimization technique, model class, etc.—in this case, we may need explicit regularization (e.g., through distributionally robust optimization [Sag+20] or related techniques) to obtain models invariant to these background features. The Only-BG trend does not indicate that models are failing per se; it could also indicate that models learn to depend on backgrounds because they are necessary for correctly classifying certain images due to quirks in the ImageNet dataset.

Still, models' *relative* improvement in accuracy across dataset variants is promising—models improve on classifying Only-BG-T at a slower (absolute) rate than Mixed-

Rand, Mixed-Same and Mixed-Next. Furthermore, the performance gap between the Mixed datasets and the others (most notably, between Mixed-Rand and Mixed-Same; between Mixed-Next and Mixed-Rand; and consequently between Mixed-Next and Mixed-Same) trends towards closing, indicating that models not only are becoming better at using foreground features, but also are becoming more robust to misleading background features (Mixed-Rand and Mixed-Next). Finally, models also improve in accuracy faster on No-FG (which has foreground shape but no texture) than on Only-BG-T, which implies that better models are using foreground shape features more effectively.

Overall, the accuracy trends observed from testing ImageNet models on our synthetic datasets reveal that better models (a) are capable of exploiting background correlations, but (b) are increasingly robust to changes in background, suggesting that invariance to background features may not necessarily come at the cost of benchmark accuracy.

2.5 Related work

Prior works on contextual bias from image backgrounds[4] show that background correlations can be predictive [Tor03] and can influence model decisions. Zhang et al. [Zha+07] find that (a) a bag-of-features object detection algorithm depends on image backgrounds in the PASCAL dataset and (b) using this algorithm on a training set with varying backgrounds leads to better generalization. Beery, Horn, and Perona [BHP18] and Barbu et al. [Bar+19] collect new test datasets of animals and objects, respectively. Barbu et al. [Bar+19] focus on object classes that also exist in ImageNet, and their new test set contains objects photographed in front of unconventional backgrounds and in unfamiliar orientations. Both works show that computer vision models experience significant accuracy drops when trained on data with one set of backgrounds and tested on data with another. Sagawa et al. [Sag+20] create a synthetic dataset of Waterbirds, where waterbirds and landbirds from one dataset are combined with water and land backgrounds from another. They show that a model's reliance on spurious correlations with the background can be harmful for small subgroups of data where those spurious correlations no longer hold (e.g. landbirds on water backgrounds). Rosenfeld, Zemel, and Tsotsos [RZT18] analyze background dependence for object detection (as opposed to classification) models

[4]Here, we highlight works analyzing contextual bias from image backgrounds specifically, and discuss contextual bias generally and foreground segmentation in Appendix A.5.

on the MS-COCO dataset. They transplant an object from one image to another image, and find that object detection models may detect the transplanted object differently depending on its location, and that the transplanted object may also cause mispredictions on other objects in the image. Zech et al. [Zec+18] study medical imaging and show that a model learned to detect a hospital-specific metal token on medical scans. The model then used each hospital's pneumonia prevalence rate to predict pneumonia fairly well (without learning much about actually detecting pneumonia).

The only work that, similarly to us, studies a large-scale dataset in this context is Zhu, Xie, and Yuille [ZXY17], who analyze ImageNet and show that AlexNet can achieve nontrivial accuracy on a dataset similar to our Only-BG-B dataset. While sufficient for establishing that backgrounds can be used for classification, this dataset also introduces biases by adding large black rectangular patches to all of the images (which our Only-BG-T dataset fixes). In comparison to [ZXY17] and the other prior works, we: (a) properly segment foregrounds and backgrounds using the GrabCut algorithm instead of relying on rectangular bounding boxes; (b) create dataset variations that allow us to measure not just model performance without foregrounds, but also the relative influence of foregrounds and backgrounds on model predictions; (c) control for the effect of image artifacts by focusing on comparisons between the Mixed-Same and Mixed-Rand datasets; (d) study model robustness to adversarial backgrounds; (e) study a larger and more recent set of classifiers [He+16; ZK16; TL19]; (f) show how improvements they give on ImageNet relate to background dependence; and (g) make our benchmarking toolkit publicly accessible for others to use and build on.

2.6 Discussion and conclusion

In this work, we study the extent to which classifiers rely on image backgrounds. To this end, we create a toolkit for measuring the precise role of background and foreground signal that involves constructing new test datasets that contain different amounts of each. Through these datasets we establish both the usefulness of background signal and the tendency of our models to depend on backgrounds, even when relevant foreground features are present. Our results show that our models are not robust to changes in the background, either in the adversarial case, or in the average case.

As most ImageNet images have human-recognizable foreground objects, our models appear to rely on background more than humans on that dataset. The fact that models

can be fooled by adversarial background changes on 88% of all images highlights how poorly computer vision models may perform in an out-of-distribution setting. However, contextual information like the background can still be useful in certain settings. After all, humans do use backgrounds as context in visual processing, and the background may be necessary if the foreground is blurry or distorted [Tor03]. Therefore, reliance on background is a nuanced question that merits further study.

On one hand, our findings provide evidence that models succeed by using background correlations, which may be undesirable in some applications. On the other hand, we find that advances in classifiers have given rise to models that use foregrounds more effectively and are more robust to changes in the background. To obtain even more robust models, we may want to draw inspiration from successes in training on the Mixed-Rand dataset (a dataset designed to neutralize background signal—cf. Table 2.1), related data-augmentation techniques [SSF19], and training algorithms like distributionally robust optimization [Sag+20] and model-based robust learning [RHP20]. Overall, the toolkit and findings in this work help us to better understand models and to monitor our progress toward the goal of reliable machine learning.

Chapter 3

Debugging computer vision models with 3DB

3.1 Introduction

Modern machine learning models turn out to be remarkably brittle under distribution shift. Indeed, in the context of computer vision, models exhibit an abnormal sensitivity to slight input rotations and translations [Eng+19b; KMF18], synthetic image corruptions [HD19; Kan+19], and changes to the data collection pipeline [Rec+19; Eng+20]. Still, while such brittleness is widespread, it is often hard to understand its root causes, or even to characterize the precise situations in which this unintended behavior arises.

How do we then comprehensively diagnose model failure modes? Stakes are often too high to simply deploy models and collect eventual "real-world" failure cases. There has thus been a line of work in computer vision focused on identifying systematic sources of model failure such as unfamiliar object orientations [Alc+19], misleading backgrounds [ZXY17; Xia+21b], or shape-texture conflicts [Gei+19b; Ath+18]. These analyses—a selection of which is visualized in Figure 3-1—reveal patterns or situations that degrade performance of vision models, providing invaluable insights into model robustness. Still, carrying out each such analysis requires its own set of (often complex) tools and techniques, usually accompanied by a significant amount of manual labor (e.g., image editing, style transfer, etc.), expertise, and data cleaning. This prompts the question:

Can we support reliable discovery of model failures in a systematic, automated, and unified way?

Contributions. In this work, we propose *3DB*, a framework for automatically identifying and analyzing the failure modes of computer vision models. This framework makes use of a 3D simulator to render realistic scenes that can be fed into any computer vision system. Users can specify a set of transformations to apply to the scene—such as pose changes, background changes, or camera effects—and can also customize and compose them. The system then performs a guided search, evaluation, and aggregation over these user-specified configurations and presents the user with an interactive, user-friendly summary of the model's performance and vulnerabilities. *3DB* is general enough to enable users to, with little-to-no effort, re-discover insights from prior work on robustness to pose, background, and texture bias (cf. Figure 3-2), among others. Further, while prior studies have largely been focused on examining model sensitivities along a single axis, *3DB* allows users to compose various transformations to understand the interplay between them, while still being able to disentangle their individual effects.

The remainder of this paper is structured into the following parts: in Section 3.3 we illustrate the utility of *3DB* through a series of case studies uncovering biases in an ImageNet-pretrained classifier. Next, we show (in Section 3.4) that the vulnerabilities

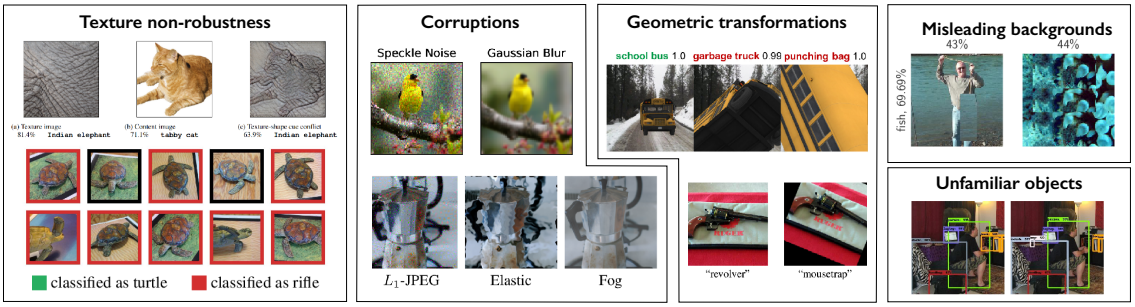


Figure 3-1: Examples of vulnerabilities of computer vision systems identified through prior in-depth robustness studies. Figures reproduced from [Gei+19b; Ath+18; HD19; Kan+19; Alc+19; Eng+19b; Xia+21b; RZT18].

Figure 3-2: The *3DB* framework is modular enough to facilitate—among other tasks—efficient rediscovery of all the types of brittleness shown in Figure 3-1 in an integrated manner. It also allows users to realistically compose transformations (right) while still being able to disentangle the results.

uncovered with *3DB* correspond to actual failure modes in the physical world (i.e., they are not specific to simulation). Finally, we discuss Related Work in Section 3.5.

3.2 Designing *3DB*

The goal of *3DB* is to leverage photorealistic simulation in order to effectively diagnose failure modes of computer vision models. To this end, the following set of principles guide the design of *3DB*:

(a) **Generality**: *3DB* should support any type of computer vision model (i.e., not necessarily a neural network) trained on any dataset and task (i.e., not necessarily classification). Furthermore, the framework should support diagnosing non-robustness with respect to any parameterizable three-dimensional scene transformation.

(b) **Compositionality**: Data corruptions and transformations rarely occur in isolation. Thus, *3DB* should allow users to investigate robustness along many different axes simultaneously.

(c) **Physical realism**: The vulnerabilities extracted from *3DB* should correspond to models' behavior in the real (physical) world, and, in particular, not depend on artifacts of the simulation process itself. Specifically, the insights that *3DB* produces should not be affected by a simulation-to-reality gap, and still hold when models are deployed in the wild.

(d) **User-friendliness**: *3DB* should be simple to use and should relay insights to the user in an easy-to-understand manner. Even non-experts should be able to look at the result of a *3DB* experiment and easily understand what the weak points of their model are, as well as gain insight into how the model behaves more generally.

(e) **Scalability**: *3DB* should be performant and parallelizable.

Capabilities and workflow. To achieve the goals articulated above, we design *3DB* in a modular manner, i.e., as a combination of swappable components. This combination allows the user to specify transformations they want to test, search over the space of these transformations, and aggregate the results of this search in a concise way. More specifically, the *3DB* workflow revolves around five steps (visualized in Figure 3-3):

1. **Setup**: The user collects one or more 3D meshes that correspond to objects the model is trained to recognize, as well as a set of environments to test against.

2. **Search space design**: The user defines a *search space* by specifying a set of transformations (which *3DB* calls *controls*) that they expect the computer vision model to be robust to (e.g., rotations, translations, zoom, etc.). Controls are grouped into "rendered controls" (applied during the rendering process) and "post-processor controls" (applied after the rendering as a 2D image transformation).

3. **Policy-guided search**: After the user has specified a set of controls, *3DB* instantiates and renders a myriad of object configurations derived from compositions of the given transformations. It records the behavior of the ML model on each constructed scene for later analysis. A user-specified *search policy* over the space of all possible combinations of transformations determines the exact scenes for *3DB* to render.

4. **Model loading**: The only remaining step before running a *3DB* analysis is loading the vision model that the user wants to analyze (e.g., a pre-trained classifier or object detection model).

5. **Analysis and insight extraction**: Finally, *3DB* is equipped with a model *dashboard* (cf. Appendix B.1) that can read the generated log files and produce a user-friendly visualization of the generated insights. By default, the dashboard has three panels. The first of these is failure mode display, which highlights configurations, scenes, and transformations that caused the model to misbehave. The per-object analysis pane allows the user to inspect the model's performance on a specific 3D mesh (e.g., accuracy, robustness, and vulnerability to groups of transformations). Finally, the aggregate analysis pane extracts insights about the model's performance averaged over all the objects and environments collected and thus allows the user to notice consistent trends and vulnerabilities in their model.

Each of the aforementioned components (the controls, policy, renderer, inference module, and logger) are fully customizable and can be extended or replaced by the user without altering the core code of *3DB*. For example, while *3DB* supports more than 10 types of controls out-of-the-box, users can add custom ones (e.g., geometric transformations) by implementing an abstract function that maps a 3D state and a set of parameters to a new state. Similarly, *3DB* supports debugging classification and

object detection models by default, and by implementing a custom evaluator module, users can extend support to a wide variety of other vision tasks and models.

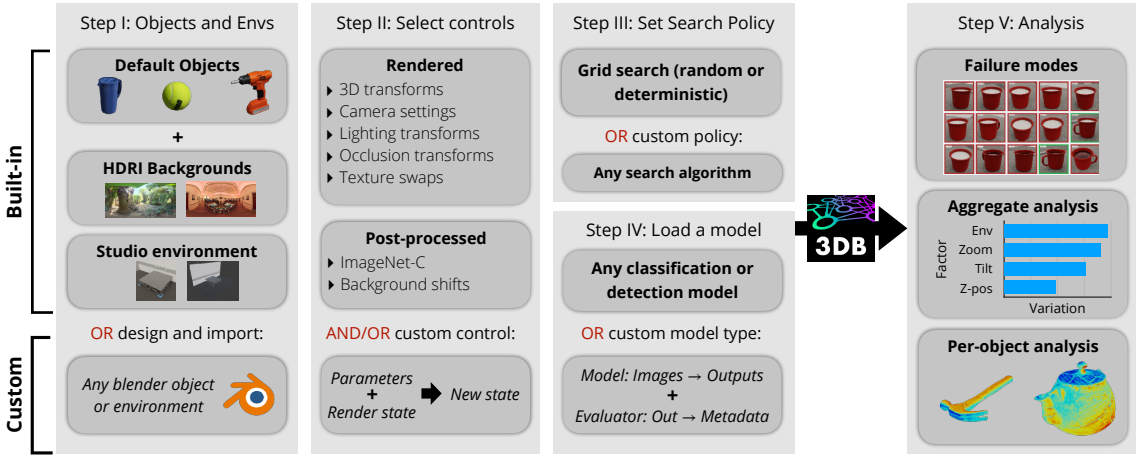


Figure 3-3: An overview of the *3DB* workflow: First, the user specifies a set of 3D object models and environments to use for debugging. The user also enumerates a set of (in-built or custom) transformations, known as controls, to be applied by *3DB* while rendering the scene. Based on a user-specified search policy over all these controls (and their compositions), *3DB* then selects the exact scenes to render. The computer vision model is finally evaluated on these scenes and the results are logged in a user-friendly manner in a custom dashboard.

3.3 Debugging and analyzing models with *3DB*

In this section, we illustrate through case studies how to analyze and debug vision models with *3DB*. In each case, we follow the workflow outlined in Section 3.2—importing the relevant objects, selecting the desired transformations (or constructing custom ones), selecting a search policy, and finally analyzing the results.

In all our experiments, we analyze a ResNet-18 [He+16] trained on the ImageNet [Rus+15] classification task (its validation set accuracy is 69.8%). Note that *3DB* is classifier-agnostic (i.e., ResNet-18 can be replaced with any PyTorch classification module), and even supports object detection tasks. For our analysis, we collect 3D models for 16 ImageNet classes (see Appendix B.4 for more details on each experiment). We ensure that in "clean" settings, i.e., when rendered in simple poses on a plain white background, the 3D models are correctly classified at a reasonable rate (cf. Table 3.1) by our pre-trained ResNet.

	banana	baseball	bowl	drill	golf ball	hammer	lemon	mug
Simulated accuracy (%)	96.8	100.0	17.5	63.3	95.0	65.6	100.0	13.4
ImageNet accuracy (%)	82.0	66.0	84.0	40.0	82.0	54.0	76.0	42.0
	orange	pitcher base	power drill	sandle	shoe	spatula	teapot	tennis ball
Simulated accuracy (%)	98.5	7.9	87.5	88.0	59.2	76.1	47.8	100.0
ImageNet accuracy (%)	72.0	52.0	40.0	66.0	82.0	18.0	80.0	68.0

Table 3.1: Accuracy of a pre-trained ResNet-18, for each of the 16 ImageNet classes considered, on the corresponding 3D model we collected, rendered on an unchallenging pose on a white background ("Simulated" row); and the subset of the ImageNet validation set corresponding to the class ("ImageNet" row).

3.3.1 Sensitivity to image backgrounds

We begin our exploration by using *3DB* to confirm ImageNet classifiers' reliance on background signal, as pinpointed by several recent in-depth studies [Zha+07; ZXY17; Xia+21b]. Out-of-the-box, *3DB* can render 3D models onto HDRI files using image-based lighting; we downloaded 408 such background environments from `hdrihaven.com`. We then used the pre-packaged "camera" and "orientation" controls to render (and evaluate our classifier on) scenes of the pre-collected 3D models at random poses, orientations, and scales on each background. Figure 3-5 shows some (randomly sampled) example scenes generated by *3DB* for the "coffee mug" model.

Analyzing a subset of backgrounds. In Figure 3-4, we visualize the performance of a ResNet-18 classifier on the 3D models from 16 different ImageNet classes—in random positions, orientations, and scales—rendered onto 20[1] of the collected HDRI backgrounds. One can observe that background dependence indeed varies widely across different objects—for example, the "orange" and "lemon" 3D models depend much more on background than the "tennis ball." We also find that certain backgrounds yield systemically higher or lower accuracy; for example, average accuracy on "gray pier" is five times lower than that of "factory yard."

Analyzing all backgrounds with the "coffee mug" model. The previous study broadly characterizes classifier sensitivity classifiers to different models and environments. Now, to gain a deeper understanding of this sensitivity, we focus our analysis only a single 3D model (a "coffee mug") rendered in all 408 environments. We find that the highest-accuracy backgrounds had tags such as *skies*, *field*, and *mountain*,

[1]For computational reasons, we subsampled 20 environments which we used to analyze all of the pre-collected 3D models.

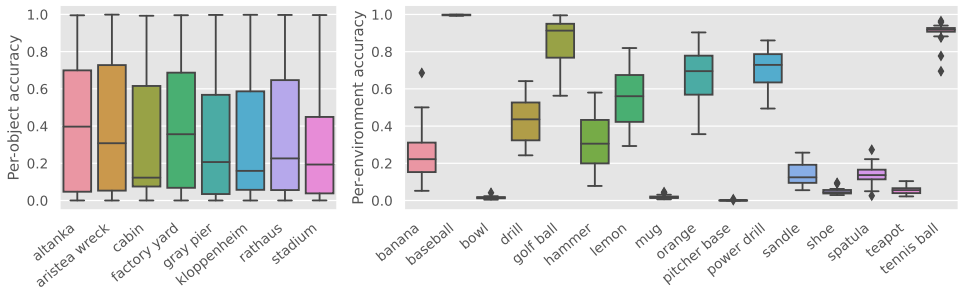


Figure 3-4: Visualization of accuracy on controls from Section 3.3.1. **(Left)** We compute the accuracy of the model conditioned on each object-environment pair. For each environment on the x-axis, we plot the variation in accuracy (over the set of possible objects) using a boxplot. We visualize the per-object accuracy spread by including the median line, the first and third quartiles box edges (the interval between which is called the inter-quartile range, IQR), the range, and the outliers (points that are outside the IQR by $^3/_2|\text{IQR}|$). **(Right)** Using the same format, we track how the classified object (on the x-axis) impacts variation in accuracy (over different environments) on the y-axis.

while the lowest-accuracy backgrounds had tags *indoor*, *city*, and *building*.

At first, this observation seems to be at odds with the idea that the classifier relies heavily on context clues to make decisions. After all, the backgrounds where the classifier seems to perform well (poorly) are places that we would expect a coffee mug to be rarely (frequently) present in the real world. Visualizing the best and worst backgrounds in terms of accuracy (Figure 3-6) suggests a possible explanation for this: the best backgrounds tend to be clean and distraction-free. Conversely, complicated backgrounds (e.g., some indoor scenes) often contain context clues that make the mug difficult for models to detect. Comparing a "background complexity" metric (based on the number of edges in the image) to accuracy (Figure 3-7) supports this explanation: mugs overlaid on more complex backgrounds are more frequently misclassified by the model. In fact, some specific backgrounds even result in the model "hallucinating" objects; for example, the second-most frequent predictions for the *pond* and *sidewalk* backgrounds were *birdhouse* and *traffic light* respectively, despite the fact that neither object is present in the environment.

Zoom/background interactions case study: the advantage of composable controls. Finally, we leverage *3DB*'s composability to study interactions between controls. In Figure 3-8 we plot the mean classification accuracy of our "orange" model while varying background and scale factor. We, for example, find that while the model generally is highly accurate at classifying "orange" with a 2x zoom factor, such a zoom

Figure 3-5: Examples of rendered scenes of the coffee mug 3D model in different environments, labeled with a pre-trained model's top prediction.

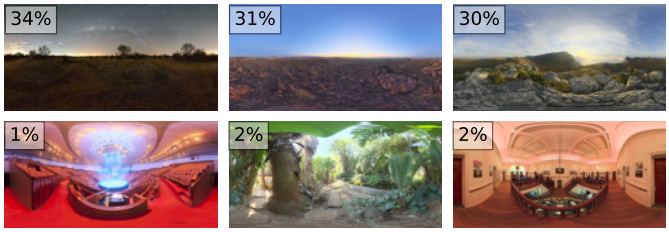


Figure 3-6: **(Top)** Best and **(Bottom)** worst background environments for classification of the coffee mug, and their respective accuracies (averaged over camera positions and zoom factors).

factor induces failure in a well lit mountainous environment ("kiara late-afternoon")—a fine-grained failure mode that we would not catch without explicitly capturing the interaction between background choice and zoom.

3.3.2 Texture-shape bias

We now demonstrate how *3DB* can be straightforwardly extended to discover more complex failure modes in computer vision models. Specifically, we will show how to rediscover the "texture bias" exhibited by ImageNet-trained neural networks [Gei+19b] in a systematic and (near-)photorealistic way. Geirhos et al. [Gei+19b] fuse pairs of images—combining texture information from one with shape and edge information from the other—to create so-called "cue-conflict" images. They then demonstrate that on these images (cf. Figure 3-9), ImageNet-trained convolutional neural networks (CNNs) typically predict the class corresponding to the texture component, while humans typically predict based on shape features.

Cue-conflict images identify a concrete difference between human and CNN decision mechanisms. However, the fused images are unrealistic and can be cumbersome to generate (e.g., even the simplest approach uses style transfer [GEB16]). *3DB* gives us an opportunity to rediscover the influence of texture in a more streamlined fashion.

Specifically, we implement a control (now pre-packaged with *3DB*) that replaces an object's texture with a random (or user-specified) one. We use this control to create cue-conflict objects out of eight 3D models[2] and seven animal-skin texture

[2]Object models: mug, helmet, hammer, strawberry, teapot, pitcher, bowl, lemon, banana and spatula

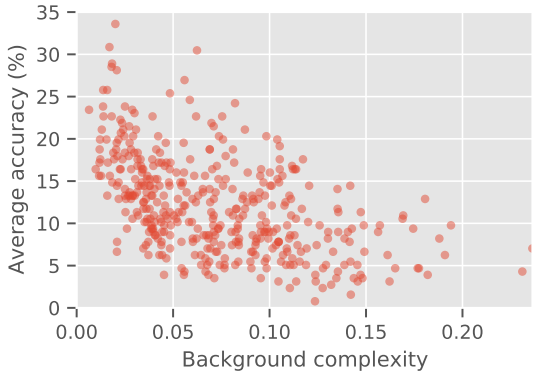


Figure 3-7: Relation between the complexity of a background and its average accuracy. Here complexity is defined as the average pixel value of the image after applying an edge detection filter.

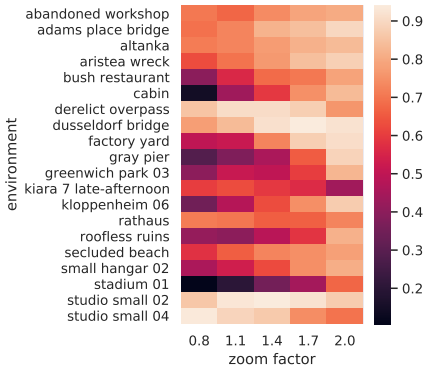


Figure 3-8: *3DB*'s focus on composability enables us to study robustness along multiple axes simultaneously. Here we study average model accuracy (computed over pose randomization) as a function of *both* zoom level and background.

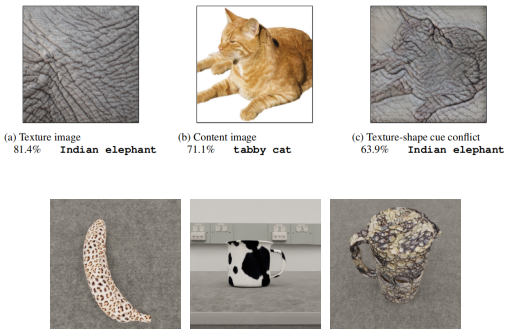


Figure 3-9: Texture vs. shape cue-conflict images generated by Geirhos et al. [Gei+19b] (*top*) and *3DB* (*bottom*).

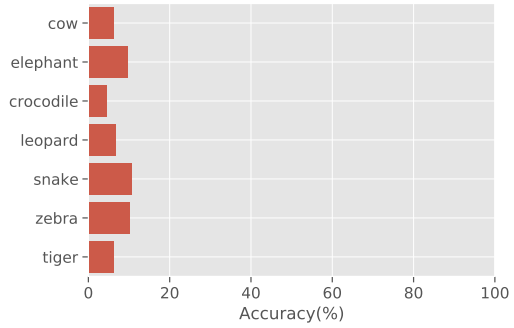


Figure 3-10: Model accuracy on previously correctly-classified images after their texture is altered via *3DB*, as a function of texture-type.

images[3] (i.e., 56 objects in total). We test our pre-trained ResNet-18 on images of these objects rendered in a variety of poses and camera locations. Figure 3-9 displays sample cue-conflict images generated using *3DB*.

Our study confirms the findings of Geirhos et al. [Gei+19b] and indicates that texture bias indeed extends to (near-)realistic settings. For images that were originally correctly classified (i.e., when rendered with the original texture), changing the texture reduced accuracy by 90-95% uniformly across textures (Figure 3-10). Furthermore, we observe that the model predictions usually align better with the texture of the objects rather than their geometry (Figure 3-11). One notable exception is the pitcher object, for which the most common prediction (aggregated over all textures) was *vase*. A possible explanation for this (based on inspection of the training data) is that due to high variability of vase textures in the train set, the classifier was forced to rely more on shape.

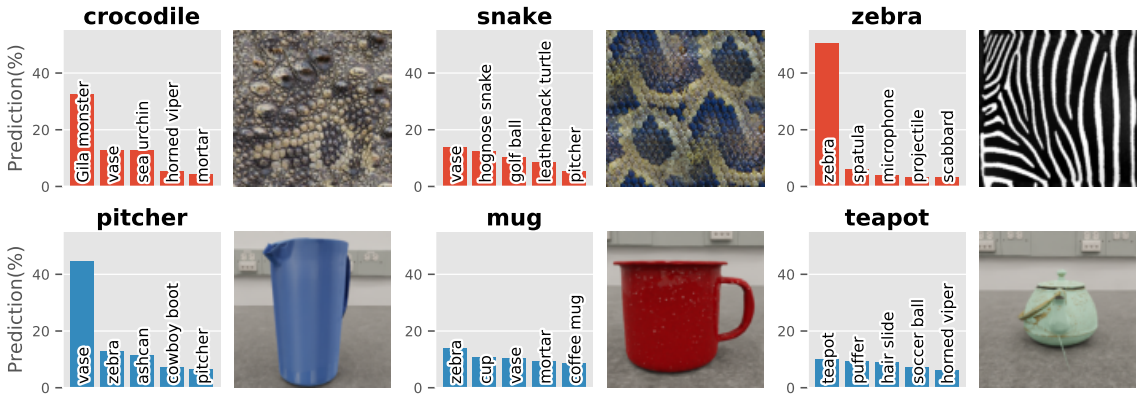


Figure 3-11: Distribution of classifier predictions after the texture of the 3D object model is altered. In the top row, we visualize the most frequently predicted classes for each texture (averaged over all objects). In the bottom row, we visualize the most frequently predicted classes for each object (averaged over all textures). We find that the model tends to predict based on the texture more often than based on the object.

3.3.3 Orientation and scale dependence

Image classification models are brittle to object orientation in both real and simulated settings [KMF18; Eng+19b; Bar+19; Alc+19]. As was the case for both background and texture sensitivity, reproducing and extending such observations is straightforward with *3DB*. Once again, we use the built-in controls to render objects at varying poses, orientations, scales, and environments before stratifying on properties of interest.

[3]Texture types: cow, crocodile, elephant, leopard, snake, tiger and zebra

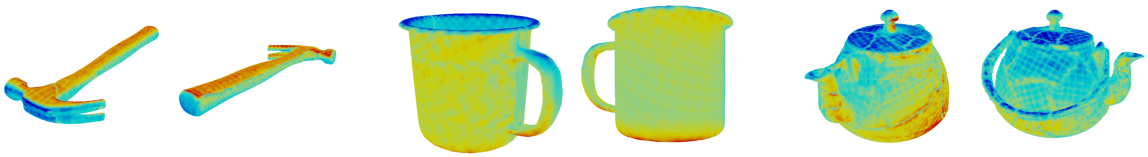


Figure 3-12: Model sensitivity to pose. The heatmaps denote the accuracy of the model in predicting the correct label, conditioned on a specific part of the object being visible in the image. Here, red and blue denotes high and low accuracy respectively.

Indeed, we find that classification accuracy is highly dependent on object orientation (Figure 3-13 left) and scale (Figure 3-13 right). However, this dependence is not uniform across objects. As one would expect, the classifier's accuracy is less sensitive to orientation on more symmetric objects (like "tennis ball" or "baseball"), but can vary widely on more uneven objects (like "drill").

For a more fine-grained look at the importance of object orientation, we can measure the classifier accuracy conditioned on a given part of each 3D model being visible. This analysis is once again straightforward in *3DB*, since each rendering is (optionally) accompanied by a UV map which maps pixels in the scene back to locations on on the object surface. Combining these UV maps with accuracy data allows one to construct the "accuracy heatmaps" shown in Figure 3-12, wherein each part of an object's surface corresponds to classifier accuracy on renderings in which the part is visible. The results confirm that atypical viewpoints adversely impact model performance, and also allow users to draw up a variety of testable hypotheses regarding performance on specific 3D models (e.g., for the coffee mug, the bottom rim is highlighted in red—is it the case that mugs are more accurately classified when viewed from the bottom)? These hypotheses can then be investigated further through natural data collection, or—as we discuss in the upcoming section—through additional experimentation with *3DB*.

3.3.4 Case study: using *3DB* to dive deeper

Our heatmap analysis in the previous section (cf. Figure 3-12) showed that classification accuracy for the mug decreases when its interior is visible. What could be causing this effect? One hypothesis is that in the ImageNet training set, objects are captured in context, and thus ImageNet-trained classifiers rely on this context to make decisions. Inspecting the ImageNet dataset, we notice that coffee mugs in context usually contain coffee in them. Thus, the aforementioned hypothesis would suggest that the pre-trained model relies, at least partially, on the contents of the mug to correctly classify

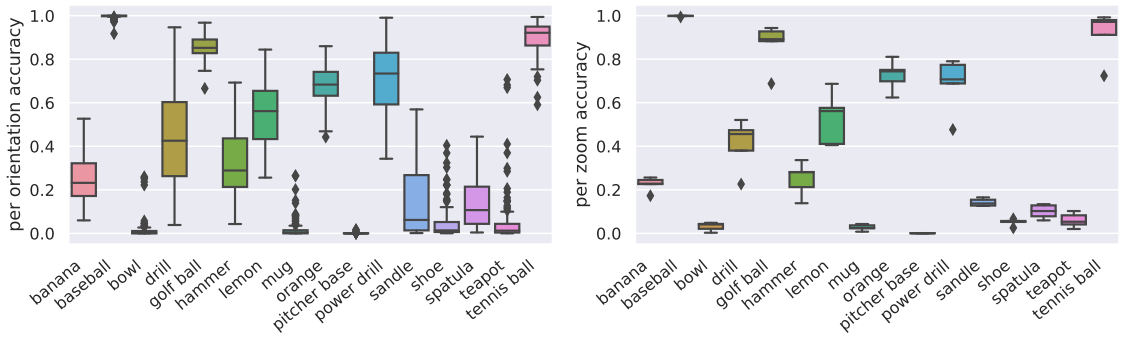
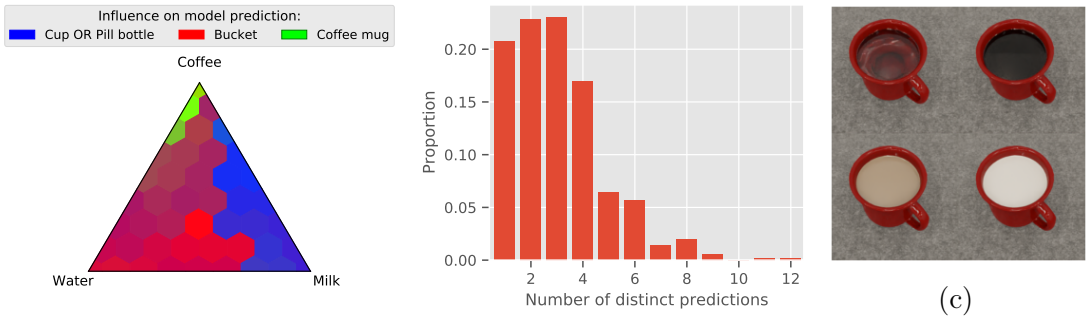


Figure 3-13: **(Left)** We compute the accuracy of the model for each object-orientation pair. For each object on the x-axis, we plot the variation in accuracy (over the set of possible orientations) using a boxplot. We visualize the per-orientation accuracy spread by including the median line, the first and third quartiles box edges, the range, and the outliers. **(Right)** Using the same format as the left hand plot, we plot how the classified object (on the x-axis) impacts variation in accuracy (over different zoom values) on the y-axis.

it. *Can we leverage 3DB to confirm or refute this hypothesis?*

To test this, we implement a custom control that can render a liquid inside the "coffee mug" model. Specifically, this control takes water:milk:coffee ratios as parameters, then uses a parametric Blender shader (cf. Appendix B.5) to render a corresponding mixture of the liquids into the mug. We used the pre-packaged grid search policy, (programmatically) restricting the search space to viewpoints from which the interior of the mug was visible.

The results of the experiment are shown in Figure 3-14. It turns out that the model is indeed sensitive to changes in liquid, supporting our hypothesis: model predictions stayed constant (over all liquids) for only 20.7% of the rendered viewpoints (cf. Figure 3-14b). The *3DB* experiment provides further support for the hypothesis when we look at the correlation between the liquid mixture and the predicted class: Figure 3-14a visualizes this correlation in a normalized heatmap (for the unnormalized version, see Figure B-5b in the Appendix B.5). We find that the model is most likely to predict "coffee mug" when coffee is added to the interior (unsurprisingly); as the coffee is mixed with water or milk, the predicted label distribution shifts towards "bucket" and "cup" or "pill bottle," respectively. Overall, our experiment suggests that current ResNet-18 classifiers are indeed sensitive to object context—in this case, the fluid composition of the mug interior. More broadly, this illustration highlights how a system designer can quickly go from hypothesis to empirical verification with minimal effort using *3DB*. (In fact, going from the initial hypothesis to Figure 3-14 took less



(a) (b) (c)

Figure 3-14: Testing classifier sensitivity to context: Figure (a) shows the correlation of the liquid mixture in the mug on the prediction of the model, averaged over random viewpoints (see Figure B-5b for the raw frequencies). Figure (b) shows that for a fixed viewpoint, model predictions are unstable with respect to the liquid mixture. Figure (c) shows examples of rendered liquids (water, black coffee, milk, and milk/coffee mix).

than a single day of work for one author.)

3.4 Physical realism

The previous sections have demonstrated various ways in which we can use *3DB* to obtain insights into model behavior in simulation. Our overarching goal, however, is to understand when models will fail in the physical world. Thus, we would like for the insights extracted by *3DB* to correspond to naturally-arising model behavior, and not just artifacts of the simulation itself [4]. To this end, we now test the *physical realism* of *3DB*: can we understand model performance (and uncover vulnerabilities) on real photos using only a high-fidelity simulation?

To answer this question, we collected a set of physical objects with corresponding 3D models, and set up a physical room with its corresponding 3D environment. We used *3DB* to identify strong points and vulnerabilities of a pre-trained ImageNet classifier in this environment, mirroring our methodology from Section 3.3. We then recreated each scenario found by *3DB* in the physical room, and took photographs that matched the simulation as closely as possible. Finally, we evaluated the physical realism of the system by comparing models' performance on the photos (i.e., whether they classified each photo correctly) to what *3DB* predicted.

[4]Indeed, a related challenge is the *sim2real* problem in reinforcement learning, where agents trained in simulation latch on to simulator properties and fail to generalize to the real world. In both cases, we are concerned about artifacts or spurious correlations that invalidate conclusions made in simulation.

Setup. We performed the experiment in the studio room shown in Appendix Figure B-3b for which we obtained a fairly accurate 3D model (cf. Appendix Figure B-3a). We leverage the `YCB` [Cal+15] dataset to guide our selection of real-world objects, for which 3D models are available. We supplement these by sourcing additional objects (from `amazon.com`) and using a 3D scanner to obtain corresponding meshes. [5]

We next used *3DB* to analyze the performance of a pre-trained ImageNet ResNet-18 on the collected objects in simulation, varying over a set of realistic object poses, locations, and orientations. For each object, we selected 10 rendered situations: five where the model made the correct prediction, and five where the model predicted incorrectly. We then tried to recreate each rendering in the physical world. First we roughly placed the main object in the location and orientation specified in the rendering, then we used a custom-built iOS application (see Appendix B.2) to more precisely match the rendering with the physical setup.

Results. Figure 3-15 visualizes a few samples of renderings with their recreated physical counterparts, annotated with model correctness. Overall, we found a 85% agreement rate between the model's correctness on the real photos and the synthetic renderings—agreement rates per class are shown in Figure 3-15. Thus, despite imperfections in our physical reconstructions, the vulnerabilities identified by *3DB* turned out to be physically realizable vulnerabilities (and conversely, the positive examples found by *3DB* are usually also classified correctly in the real world). We found that objects with simpler/non-metallic materials (e.g., the bowl, mug, and sandal) tended to be more reliable than metallic objects such as the hammer and drill. It is thus possible that more precise texture tuning of 3D models object could increase agreement further (although a more comprehensive study would be needed to verify this).

3.5 Related work

3DB builds on a growing body of work that looks beyond accuracy-based benchmarks in order to understand the *robustness* of modern computer vision models and their failure modes. In particular, our goal is to provide a unified framework for reproducing these studies and for conducting new analyses. In this section, we discuss the existing

[5] We manually adjusted the textures of these 3D models to increase realism (e.g., by tuning reflectance or roughness). In particular, classic photogrammetry is unable to model the metallicness and reflectivity of objects. It also tends to embed reflections as part of the color of the object

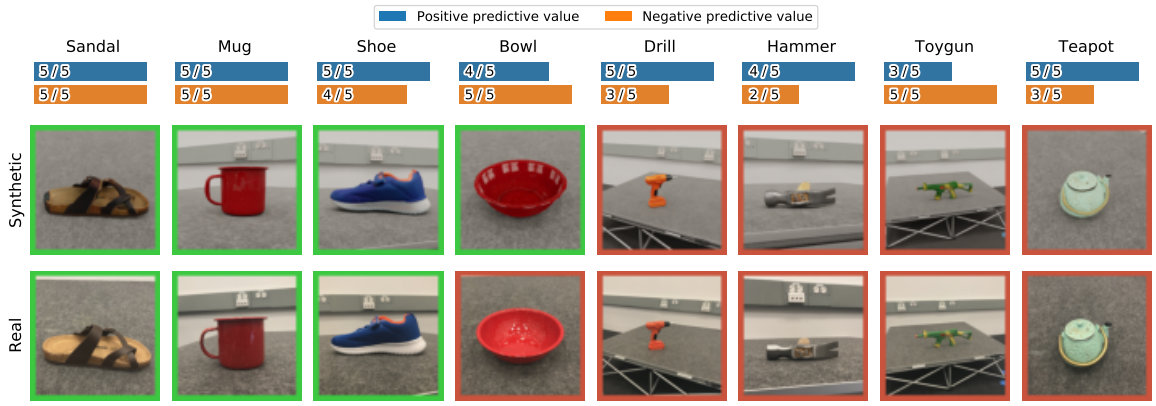


Figure 3-15: **(Top)** Agreement, in terms of model correctness, between model predictions within *3DB* and model predictions in the real world. For each object, we selected five rendered scenes found by *3DB* that were misclassified in simulation, and five that were correctly classified; we recreated and deployed the model on each scene in the physical world. The *positive (resp., negative) predictive value* is rate at which correctly (resp. incorrectly) classified examples in simulation were also correctly (resp., incorrectly) classified in the physical world. **(Bottom)** Comparison between example simulated scenes generated by *3DB* (first row) and their recreated physical counterparts (second row). Border color indicates whether the model was correct on this specific image.

research in robustness, interpretability, and simulation that provide the context for our work.

Adversarial robustness. Several recent works propose analyzing model robustness by crafting adversarial, i.e., *worst-case*, inputs. For example, [Sze+14] discovered that a carefully chosen but imperceptible perturbation suffices to change classifier predictions on virtually any natural input. Subsequently, the study of such "adversarial examples" has extended far beyond the domain of image classification: e.g., recent works have studied worst-case inputs for object detection and image segmentation [Eyk+18; Xie+17; Fis+17]; generative models [KFS18]; and reinforcement learning [Hua+17a]. More closely related to our work are studies focused on three-dimensional or physical-world adversarial examples [Eyk+18; Bro+18; Ath+18; Xia+19a; Liu+19]. These studies typically use differentiable rendering and perturb object texture, geometry, or lighting to induce misclassification. Alternatively, Li, Schmidt, and Kolter [LSK19] modify the camera itself via an adversarial camera lens that consistently cause models to misclassify inputs.

In our work, we have primarily focused on using non-differentiable but high-fidelity rendering to analyze a more *average-case* notion of model robustness to

semantic properties such as object orientation or image backgrounds. Nevertheless, the extensibility of *3DB* means that users can reproduce such studies (by swapping out the Blender rendering module for a differentiable renderer, writing a custom control, and designing a custom search policy) and use our framework to attain a more realistic understanding of the worst-case robustness of vision models.

Robustness to synthetic perturbations. Another popular approach to analyzing model robustness involves applying transformations to natural images and measuring the resultant changes in model predictions. For example, Engstrom et al. [Eng+19b] measure robustness to image rotations and translations; Geirhos et al. [Gei+19b] study robustness to style transfer (i.e., texture perturbations); and a number of works have studied robustness to common corruptions [HD19; Kan+19], changes in image backgrounds [ZXY17; Xia+21b], Gaussian noise [For+19], and object occlusions [RZT18], among other transformations.

A more closely related approach to ours analyzes the impact of factors such as object pose and geometry by applying synthetic perturbations in three-dimensional space [HG19; Shu+20; HMG18; Alc+19]. For example, Hamdi and Ghanem [HG19] and Jain et al. [Jai+20] use a neural mesh renderer [KUH18] and Redner [Li+18], respectively, to render images to analyze the failure modes of vision models. Alcorn et al. [Alc+19] present a system for discovering neural networks' failure modes as a function of object orientation, zoom, and (two-dimensional) background and perform a thorough study on the impact of these factors on model decisions.

3DB draws inspiration from the studies listed above and tries to provide a unified framework for detecting *arbitrary* model failure modes. For example, our framework provides explicit mechanisms for users to make custom controls and custom search strategies, and includes built-in controls designed to range across many possible failure modes encompassing nearly all of the aforementioned studies (cf. Section 3.3). Users can also *compose* different transformations in *3DB* to get an even more fine-grained understanding of model robustness.

Other types of robustness. An oft-studied but less related branch of robustness research tests model performance on unaltered images from distributions that are close to but not identical to that of the training set. Examples of such investigations include studies of newly collected datasets such as ImageNet-v2 [Rec+19; Eng+20; Tao+20], ObjectNet [Bar+19], and others (e.g., [Hen+19; Sha+19]). In a similar vein, Torralba and Efros [TE11] study model performance when trained on one standard

dataset and tested on another. We omit a detailed discussion of these works since *3DB* is synthetic by nature (and thus less photorealistic than the aforementioned studies). As shown in Section 3.4, however, *3DB* is indeed realistic enough to be indicative of real-world performance.

Interpretability, counterfactuals, and model debugging. *3DB* can be cast as a method for *debugging* vision models that provides users fine-grained control over the rendered scenes and thus enables them to find specific modes of failure (cf. Sections 3.3 and 3.4). Model debugging is also a common goal in intepretability research, where methods generally seek to provide justification for model decisions based on either local features (i.e., specific to the image at hand) or global ones (i.e., general biases of the model). Local explanation methods, including saliency maps [SVZ13; DG17; STY17], surrogate models such as LIME [RSG16], and counterfactual image pairs [FV17; ZXY17; Goy+19], can provide insight into specific model decisions but can also be fragile [GAZ19; AJ18] or misleading with respect to global model behaviour [Ade+18; STY17; Ade+20; Lip18]. Global interpretability methods include concept-based explanations [Bau+17; Kim+18; Yeh+20; WSM21] (though such explanations can often lack causal links to the features models actually use [Goy+19]), but also encompass many of the robustness studies highlighted earlier in this section, which can be cast as uncovering global biases of vision models.

Simulated environments and training data. Finally, there has been a long line of work on developing simulation platforms that can serve as both a source of additional (synthetic) training data, and as a proxy for real-world experimentation. Such simulation environments are thus increasingly playing a role in fields such as computer vision, robotics, and reinforcement learning (RL). For instance, OpenAI Gym [Bro+16] and DeepMind Lab [Bea+16] provide simulated RL training environments with a fleet of control tasks. Other frameworks such as UnityML [Jul+20] and RoboSuite [Zhu+20] were subsequently developed to cater to more complex agent behavior.

In computer vision, the Blender rendering engine [Ble20] has been used to generate synthetic training data through projects such as BlenderProc [Den+19] and BlendTorch [Hei+20]. Similarly, HyperSim [RP] is a photorealistic synthetic dataset focused on multimodal scene understanding. Another line of work learns optimal simulation parameters for synthetic data generation according to user-defined objectives, such as minimizing the distribution gap between train and test environments [Kar+19b;

DKF20b; Beh+20]. Simulators such as AirSim [Sha+18], FlightMare [Son+20], and CARLA [Dos+17] (built on top of video game engines Unreal Engine and Unity) allow for the collection of synthetic training data for perception and control. In robotics, simulators include environments that model typical household layouts for robot navigation [Kol+17; Wu+18; Pui+18], interactive ones with objects that can be actuated [Xia+18; Xia+20a; Xia+20b], and those that include support for tasks such as question answering and instruction following [Sav+19].

While some of these platforms may share components with *3DB* (e.g., the physics engine, photorealistic rendering), they do not share the same goals as *3DB*, i.e., diagnosing specific failures in existing models.

3.6 Conclusion

In this work, we introduced *3DB*, a unified framework for diagnosing failure modes in vision models based on high-fidelity rendering. We demonstrate the utility of *3DB* by applying it to a number of model debugging use cases—such as understanding classifier sensitivities to realistic scene and object perturbations, and discovering model biases. Further, we show that the debugging analysis done using *3DB* in simulation is actually predictive of model behavior in the physical world. Finally, we note that *3DB* was designed with extensibility as a priority; we encourage the community to build upon the framework so as to uncover new insights into the vulnerabilities of vision models.

Part II

Improving model robustness and performance

Chapter 4

Improving performance by finding target-aligned subsets of auxiliary data

4.1 Introduction

The use of larger datasets has been a key driver of recent progress in machine learning. Indeed, language and computer vision models trained on datasets containing billions of documents [Bro+20] and images [Sun+17; Mah+18] make the classic ImageNet [Rus+15] benchmark look small in comparison. So, when training machine learning models, there is a strong inclination to use as much data as possible. This can amount to the costly process of collecting and labeling more data, or incorporating *auxiliary data* from related but separate datasets.

Indeed, auxiliary datasets can be used to improve generalization [Pen+19; Bee+20] and out of distribution robustness [Sch+20a; Mår+20]. They can also give rise to learning richer representations [MPR16; Qiu+21] and better initializations via pretraining [Rad+18; Car+19]. All of these reinforce the conventional wisdom that *more data leads to better performance* [Ros+20; Kap+20]. As a result, when given the option of using an auxiliary dataset, a typical strategy is to use all of it.

However, blindly using auxiliary data turns out to hurt model performance in certain cases. For example, pooling medical imaging data from multiple hospitals causes models to detect hospital sources instead of just medical symptoms [Zec+18]. Large language datasets can over-represent certain populations while excluding marginalized populations [Ben+21]. The widely used ImageNet benchmark [Rus+15] is often used

as an auxiliary dataset for other tasks, yet was shown to contain spurious correlations such as men holding fish [Xia+21a]. In light of the fact that auxiliary data may include irrelevant or harmful biases, how can we best use auxiliary data for a particular task while avoiding potential downsides?

To answer this question, we consider the task of identifying a subset of the auxiliary data that is most aligned with the target data, which we call the *dataset projection* problem. As we will demonstrate, using the "right" subset of auxiliary data can be important for certain target tasks. In particular, intelligently and automatically selected subsets of auxiliary data have the potential to decrease sources of unwanted biases, amplify useful features, and ultimately improve task performance.

Our contributions. In this paper, we analyze the limitations of indiscriminately using auxiliary data, and develop methods to better leverage such data. Specifically:

1. Through three experiments, we exhibit scenarios where using the entire auxiliary dataset can hurt model performance, and demonstrate how manually selecting what auxiliary data to use improves performance.

2. We formulate the problem of projecting datasets, and develop methods to solve this problem that can *automatically* find target-aligned subsets of auxiliary datasets.

3. We demonstrate on a variety of vision and language tasks that our methods find projected datasets that are better approximations of target datasets than the original auxiliary dataset. In particular, we find that these datasets can improve downstream performance when augmenting the target dataset, even when compared to the original auxiliary dataset.

4. Our framework enables a new form of analysis that uses projections of auxiliary data to reveal insights about the composition of the target dataset.

4.2 Can indiscriminate use of auxiliary datasets hurt performance?

The current trend in machine learning is to use more data when it is available. However, recent studies have raised concerns about the unintended consequences of blindly

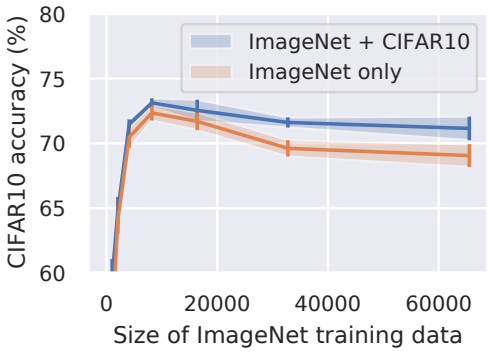


Figure 4-1: Adding auxiliary ImageNet data to a small CIFAR10 dataset (blue curve) can improve performance, but adding too much hurts performance. Training exclusively on auxiliary ImageNet data isolates this behavior (orange curve). This indicates that ImageNet contains patterns that hurt CIFAR10 generalization.

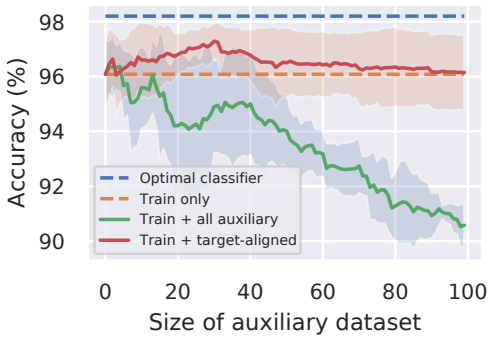
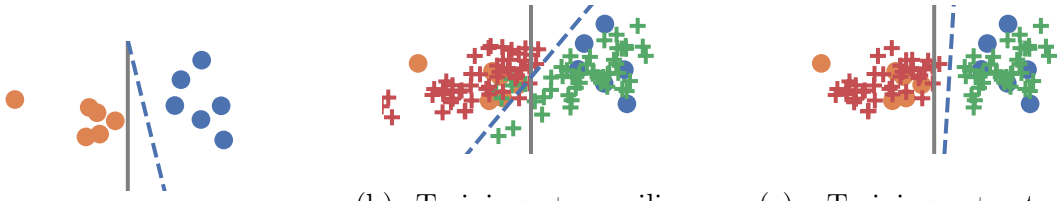


Figure 4-2: Accuracy of a linear classifier when augmenting a small dataset with biased auxiliary data for the Gaussian example in Figure 4-3. Increasing the amount of auxiliary data causes the classifier to lose accuracy. Adding target-aligned data can mitigate the bias and improve performance over the original training data.

incorporating new dataset sources. For instance, pneumonia detectors learned to identify hospital-specific markers rather than pneumonia itself when data was pooled [Zec+18]. Furthermore, combining data from multiple surveys can introduce new sources of sampling error and nonresponse bias [LR17].

Motivating example with CIFAR10 and ImageNet. The consequences of using auxiliary data manifest themselves in commonly-studied machine learning settings as well. Suppose we want to train a CIFAR10 classifier, but only have a limited amount of CIFAR10 data. To improve performance, we can augment our limited CIFAR10 dataset using relevant classes of ImageNet as auxiliary data. As we increase the amount of ImageNet data, performance initially improves (see blue curve in Figure 4-1). However, we find that adding *too much* ImageNet data degrades performance instead.

Why does this happen? It turns out that the model is overfitting to the patterns in the ImageNet data. This trend becomes even clearer when we train a model only on the data sourced from ImageNet (see orange curve in Figure 4-1). While some amount of ImageNet data alone can provide useful features for CIFAR10, too much ImageNet data introduces irrelevant patterns that hurt performance.



(a) Training data

(b) Training + auxiliary data

(c) Training + target-aligned data

Figure 4-3: A synthetic example showing how auxiliary data can harm performance. The optimal and estimated linear classifiers are plotted in the solid and dashed lines respectively. (a) Training data is sampled from Gaussian distributions, and (b) auxiliary data comes from a skewed Gaussian that biases the estimated classifier. (c) Subsampling the auxiliary data to be aligned with training data can reduce the auxiliary bias and improve performance.

Linear example. To better understand this phenomenon, we construct a simplified linear example which replicates the empirical trends observed with CIFAR10 and ImageNet. Consider a binary classification problem with labels $y \in \{-1, +1\}$ and data x drawn from a class conditional Gaussian distribution with mean $y \cdot \mu$:

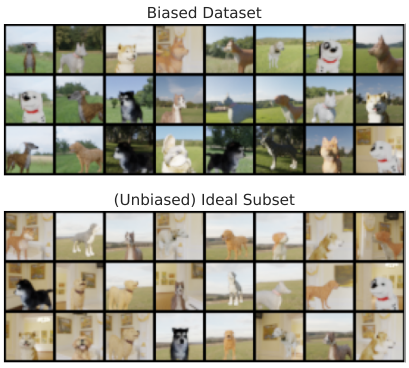
$$x \sim \mathcal{N}(y \cdot \mu, I) \tag{4.1}$$

With only a few training datapoints, the estimated linear classifier can have high error (see Figure 4-3a). To improve performance, we can augment the training data with an auxiliary dataset. In this case, our auxiliary data z comes from a skewed and rotated Gaussian distribution instead:

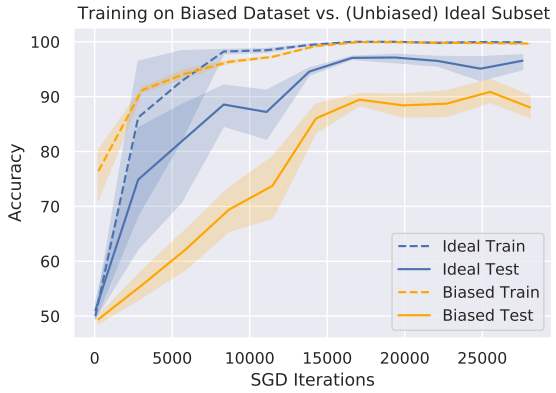
$$z \sim \mathcal{N}(y \cdot \mu, \Sigma) \tag{4.2}$$

where Σ is different from the identity. Adding too much of this auxiliary data biases the estimated linear classifier and actually hurts performance (see Figure 4-3b).

However, not all of the auxiliary data is harmful. Ideally, we would like to keep auxiliary data that is most aligned with our original training dataset, and discard irrelevant data. A natural way to achieve this is to fit multivariate Gaussian distributions to the training data, and restrict the auxiliary data to those that fall within a 95% confidence region (see Figure 4-3c). Augmenting the training data with this target-aligned subset reduces the bias and improves performance, as measured in Figure 4-2.



(a) Biased dataset and unbiased ideal subset of synthetic images generated via 3DB.



(b) Train and test accuracy curves.

Figure 4-4: (a) Training on the 15k-image biased dataset (top) results in 7% lower accuracy than training on the 3k-image unbiased, ideal subset of the dataset (bottom) when the model is tested on an unbiased test set. (b) Accuracy curves for each training set.

Controlled synthetic setting. We further recreate this phenomenon in a controlled, synthetic setting to show that indiscriminately using auxiliary data can be harmful due to spurious correlations. Our synthetic setting involves rendering photorealistic images with Blender and the 3DB framework [Lec+21b]. First, we generate a biased auxiliary dataset of cats and dogs, where 90% of cats are indoors and 90% of dogs are outdoors. For the target test dataset, we generate an unbiased dataset where the background is not correlated with the class at all. Training models on the full auxiliary dataset (containing the spurious correlation) leads to an average test accuracy of 90.4% while training models on a manually chosen subset of the auxiliary dataset (without the spurious correlation) leads to a significantly higher average test accuracy of 97.6%. Here, using strictly more data leads to worse performance due to the spurious background correlation that models learn from the full auxiliary dataset. Figure 4-4 shows examples of the biased auxiliary data and the unbiased subset.

Full details for reproducing all three of these examples are in Appendix C.1.

4.3 Dataset projection

Motivated by the limits of indiscriminately using auxiliary datasets, we aim to answer: *How can we extract the relevant portions of auxiliary datasets for a target dataset?* Unlike two of the three examples in Section 4.2 (linear example and the controlled synthetic example), real-world datasets do not always have clear patterns for manually

filtering out irrelevant auxiliary data. Instead, we would like an automatic way to extract a target-aligned subset of auxiliary data.

More formally, let p be the target distribution, and let q be the auxiliary distribution. We assume that q is composed of k source distributions, $q_1, \ldots, q_k$. Our goal is to find a linear combination of source distributions $\{q_i\}$ that best matches the target distribution p. We can formulate this task as the following optimization problem:

$$\begin{aligned} &\min_{\alpha} \mathbb{E}_{X\sim p, Y\sim \hat{q}}\left[d(X,Y)\right] \\ &\text{subject to } \hat{q} = \sum_{i=1}^{k} \alpha_i q_i, \quad \sum_{i=1}^{k} \alpha_i = 1, \quad \alpha_i \geq 0 \end{aligned} \tag{4.3}$$

where d is a distance metric over datasets, (X, Y) are datasets sampled from $(p, \hat{q})$, and α_i denotes the proportion of each source distribution q_i used for approximating the target distribution p. Intuitively, this can be thought of as "projecting" the target distribution p onto the space spanned by source distributions q_i. Hence, we refer to this optimization problem as *dataset projection*.

Source distributions. To apply our dataset projection framework, an auxiliary distribution q needs to be split into multiple source distributions q_i. How can we get these source distributions? In supervised settings, a natural way to split an auxiliary dataset is to use existing class or attribute labels. In completely unsupervised settings, we can automatically split an auxiliary distribution with unsupervised clustering methods. Finally, labels can be combined with clustering methods to generate even finer-grained source distributions. Crucially, our framework can project *any* auxiliary dataset onto a target dataset, including those with different labels or no labels at all. We defer a detailed discussion of these splitting strategies for dataset projection to Appendix C.2.

Distance metric. Our dataset projection framework requires a metric to measure the distance between two datasets. We employ the Maximum Mean Discrepancy (MMD) score [Gre+12], a statistic that measures the similarity of two distributions. This metric has been successfully used in prior works to learn generative models [Kar+19a; DRG15] and detect distribution shift [RGL19]. In Appendix C.2, we describe in detail how we compute this distance metric.

Algorithm 1 Active set algorithm for projecting datasets with soft active set estimate A_i

$\alpha_i^0 = 1/k$, $A_i = 0.5$ for $i = 1 \dots k$ *// Initialize feasible point and soft active set estimate*
for $t = 0, 1, \dots$ **do**
  if α^t is stationary point **then**
    Return α^t
  end if
  $g = \nabla f(\alpha^t)$ *// Estimate numerical gradient of the dataset projection objective*
  $(\tilde{\alpha}^t, \tilde{A}^t) = \texttt{DampedUpdate}(\alpha^t, A^t, g)$ *// Active set update*
  $(\alpha^{t+1}, A^{t+1}) = \texttt{SearchUpdate}(\tilde{\alpha}^t, \tilde{A}^t, g)$ *// Line search update*
end for

Solving the dataset projection problem. Solving the dataset projection problem has several challenges. First, there is a simplex constraint on the optimization variables α. Second, the objective is both stochastic and non-differentiable with respect to the optimization variables α. To tackle this problem, we develop two numerical solvers: an active set solver motivated by simplex optimization methods with theoretical convergence guarantees, and a projected gradient descent (PGD) solver based on the widely used proximal method for constrained minimization. In practice, the active set solver converges faster to sparser solutions and tends to perform better, while the PGD solver is more stable but slower to converge. We provide a brief summary of the active set solver in Algorithm 1, and give a more detailed discussion of the details of each solver, how we handle these challenges, and their corresponding strengths and differences in Appendix C.2.

4.4 Projected datasets in practice

How well does our framework for projecting datasets work empirically? First, we perform an extensive evaluation of our framework, in order to validate that our projected datasets are indeed more target-aligned than the original auxiliary datasets. We then highlight potential use-cases for projected datasets. In particular, we find that projected datasets can improve downstream performance and provide insights on the composition of the target dataset.

Experimental setup. Our benchmark for projecting datasets spans five vision datasets for image classification and five language datasets for sentiment analysis,

Domain	Dataset
Vision	CIFAR10 [Kri09]
	STL10 [Ada11]
	Oxford-IIIT Pet [Par+12]
	ImageNet [Rus+15]
	3DB [Lec+21b]
Language	SST [Soc+13]
	Yelp [ZZL15]
	Emoji [Bar+18]
	Emotion [Moh+18]
	DailyDialog [Cha+20]

Table 4.1: All datasets used in our dataset projection benchmark.

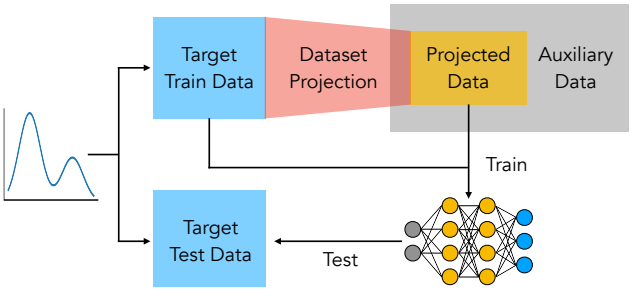


Figure 4-5: Pipeline for validating alignment and augmenting target data via training. After projecting onto the auxiliary dataset, we can (1) train a model on the projected data to validate target-alignment and (2) augment target training data to improve test performance.

summarized in Table 4.1. Each scenario in the benchmark consists of one auxiliary dataset and one target dataset, resulting in a total of 36 scenarios. In all scenarios, we use our framework to project each class from the target dataset onto the auxiliary dataset, using either our active set or PGD solver, and compare to the random baseline of uniformly using all sources (i.e. $\alpha_i = 1/k$ for $i = 1 \dots k$). The complete description of the experimental setup and corresponding datasets is deferred to Appendix C.3.

4.4.1 Validating alignment with the target dataset

To solve the dataset projection problem from Equation (4.3), our framework aims to maximize alignment with the target dataset. But how can we validate whether our solvers actually achieve this goal?

To this end, we leverage a suite of metrics that quantify the alignment of a projected dataset with the target dataset. Note that it may not always be possible to reach perfect alignment with a projected dataset. This can occur when the auxiliary and target datasets are too distinct, or if the sources are too coarse. Nonetheless, we can still project datasets with our framework to find the *most* target-aligned subset. In this section, we run experiments to validate whether projected datasets are more target-alignment than the original auxiliary dataset.

Measuring target-alignment via training. One approach to validate our projected datasets is to train a model on the projected data, and evaluate that model on the target dataset, as depicted in Figure 4-5. Intuitively, a better alignment with the

Auxiliary	Target	Random	PGD-PD	Not PGD-PD	AS-PD	Not AS-PD
ImageNet	3DB	27.0 ± 5.0	22.2 ± 1.7	16.3 ± 2.4	$\mathbf{29.5} \pm 1.7$	22.1 ± 1.6
CIFAR10		30.8 ± 3.7	32.7 ± 4.5	19.6 ± 3.1	$\mathbf{35.4} \pm 1.0$	23.8 ± 0.9
Oxford-IIIT		66.8 ± 8.8	68.4 ± 14.4	55.9 ± 12.8	$\mathbf{71.2} \pm 8.0$	56.3 ± 12.1
STL10		12.6 ± 4.0	34.8 ± 3.5	5.7 ± 4.0	$\mathbf{37.7} \pm 4.0$	12.2 ± 3.4

Table 4.2: Approximating target datasets with auxiliary data. In this experiment, we train on auxiliary data and test on target data. Higher test accuracy corresponds to better approximation quality. AS-PD typically performs the best, and PGD-PD also typically outperforms Random. The complementary subsets of Not AS-PD and Not PGD-PD often perform worse than Random, as expected.

target dataset should lead to better accuracy. Indeed, we find that our framework can find projections that are more accurate at predicting the target dataset. We highlight a subset of our vision results in Table 4.2, which shows that models trained on projected datasets (aligned with the target dataset 3DB) selected with active set (AS-PD) and PGD (PGD-PD) consistently outperform models trained on the random baseline (Random). We defer the full table of results evaluating our projections for the rest of our benchmark to Appendix C.4.

Ablation: alignment of the complementary subset. Since our framework searches for the most target-aligned subset, we would expect the complementary subset to have poor alignment with the target dataset. Indeed, this sanity check turns out to be the case: in Table 4.2 we evaluate the complementary subsets of those chosen by our framework (Not PGD-PD and Not AS-PD), and find that these subsets have even worse performance than the random baseline.

Measuring target-alignment via distance metrics. Another natural approach to evaluate the projection is to calculate the MMD score from Equation (4.3) between the target dataset and the projected dataset. However, since our framework directly optimizes for this specific distance, a more holistic approach is to evaluate with additional distance metrics not used in our framework. To validate our results, we use distance metrics from [ZLB17] for the vision settings and similarity metrics for the natural language settings [MRS10]. We confirm in Figure 4-6 that the projected datasets are indeed closer to the target than the original auxiliary dataset. This improvement in target approximation holds for nearly all choices of the auxiliary and target datasets, which we evaluate and plot for the rest of our benchmark in Appendix C.4.

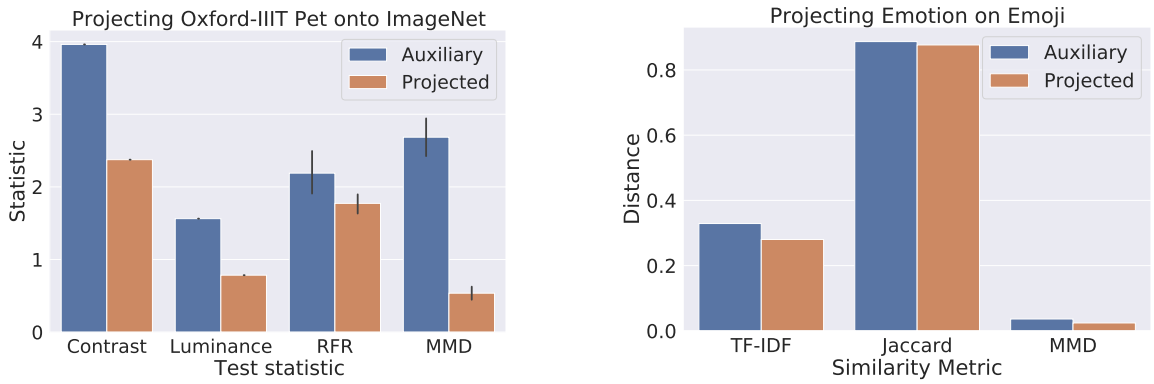


Figure 4-6: We validate that projected datasets have lower distance to the target dataset, using additional distance metrics. These metrics are (left) 1D-statistics based on contrast, luminance, and random filter response (RFR) for vision datasets and (right) similarity metrics based on TF-IDF and Jaccard coefficients for language datasets.

Figure 4-7: Visually comparing AS-PD (Middle) with random data from the original auxiliary dataset (Left) and the target dataset (Right). While the full auxiliary dataset includes wild outdoor cats, the projected dataset consists primarily of indoor household cats which aligns with the target dataset.

Qualitative comparison via visualization. We provide a qualitative evaluation of our framework in Figure 4-7, where we plot examples of the subset selected by AS-PD and compare it to the auxiliary dataset and the target dataset. We find that the images in AS-PD appear to have much more target-aligned backgrounds and types of cats when compared to the auxiliary dataset, although differences still do exist because the target is likely to be different from *any* subset of the auxiliary dataset.

4.4.2 Effectiveness at augmenting the target dataset

A natural use case for auxiliary data is to augment the target dataset in the hopes of achieving higher test accuracy. In this section, we measure how useful the projected

Auxiliary	Target	Target Only	Target + Random	Target + PGD-PD	Target + AS-PD
ImageNet	CIFAR10	43.6 ± 0.4	54.8 ± 2.1	55.3 ± 1.8	**57.0** ± 3.0
CIFAR10	Oxford-IIIT	52.1 ± 1.9	54.2 ± 2.5	55.4 ± 2.0	**59.5** ± 2.9
Oxford-IIIT	STL10	54.2 ± 2.8	54.3 ± 3.1	**65.1** ± 1.4	61.6 ± 1.1
Oxford-IIIT	3DB	69.8 ± 1.6	73.4 ± 7.2	82.2 ± 3.3	**82.3** ± 10.2

Table 4.3: Augmenting a target dataset with auxiliary data for image classification. In this experiment, we add auxiliary data to a target dataset for training, and test on target data. In most cases, augmenting with AS-PD performs the best.

Auxiliary	Target	Target Only	Target + Random	Target + PGD-PD	Target + AS-PD
Yelp	SST	74.2 ± 6.0	60.8 ± 8.9	65.6 ± 12.0	**78.6** ± 2.9
DailyDialog	Emoji	12.4 ± 3.6	14.0 ± 3.3	12.8 ± 2.9	**24.4** ± 1.0
SST	Emotion	37.6 ± 4.3	39.4 ± 4.1	42.8 ± 7.8	**45.2** ± 4.7
Emoji	Yelp	28.6 ± 1.9	29.2 ± 3.0	27.0 ± 5.4	**34.4** ± 5.7
Emotion	DailyDialog	37.4 ± 6.3	37.6 ± 5.0	35.0 ± 8.0	**67.4** ± 6.0

Table 4.4: Augmenting a target dataset with auxiliary data for sentiment analysis. In this experiment, we add auxiliary data to a target dataset for training, and test on target data. In most cases, augmenting with AS-PD performs the best.

dataset is for our full experimental benchmark. Specifically, we use the projected dataset as extra training data (in addition to the target data itself), and evaluate whether it can increase performance on the target test data, as illustrated in Figure 4-5. We compare to the baselines of using no auxiliary data (Target Only) and augmenting uniformly with all auxiliary sources (Target + Random). Tables 4.3, 4.4, and 4.5 highlight a subset of our vision and language results, and the full suite of experiments are deferred to Appendix C.4.

Augmenting with auxiliary data. We first highlight that augmenting the target dataset with projected datasets can often improve test performance in both image classification (see Table 4.3) and sentiment analysis (see Table 4.4). For example, augmenting STL10 with projected data from Oxford-IIIT (PGD-PD) increases test accuracy by 11%, compared with a 0% increase when augmenting with random auxiliary data. Similarly, augmenting DailyDialog with projected data from Emotion (AS-PD) can increase test accuracy by 30% over both baselines. This demonstrates that auxiliary data can be used much more effectively to augment existing data for model training if we first apply dataset projection.

In our full benchmark, AS-PD outperforms Random at augmenting the target in 10 out of 16 vision settings (Table C.5 in the Appendix) and all 20 language

Auxiliary	Target	DA	Random + DA	PGD-PD + DA	AS-PD + DA
DailyDialog	SST	78.4 ± 3.5	76.2 ± 4.7	76.0 ± 2.6	**80.8** ± 2.7
DailyDialog	Emoji	22.8 ± 3.2	23.8 ± 2.0	23.6 ± 1.9	**25.0** ± 2.4
Emoji	Emotion	50.6 ± 9.0	52.6 ± 6.7	52.6 ± 5.1	**59.6** ± 4.4
Emoji	Yelp	32.4 ± 4.3	33.8 ± 6.6	33.6 ± 6.6	**36.8** ± 4.2
Yelp	DailyDialog	76.2 ± 2.0	74.8 ± 2.9	**76.8** ± 1.5	75.0 ± 1.8

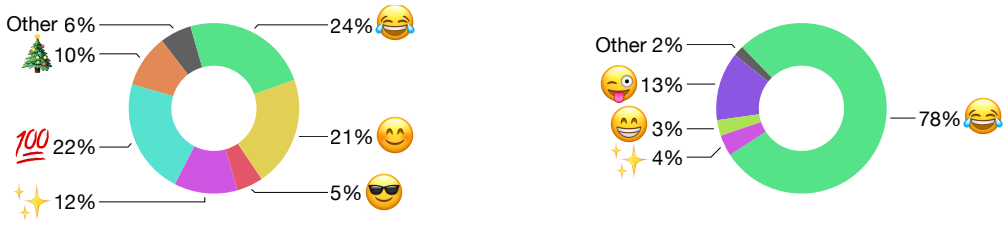
Table 4.5: Augmenting a target dataset with both auxiliary data and back-translation for sentiment analysis. We find that combining auxiliary data from AS-PD with data augmentation (DA) gives the best results, showing that the performance gains of AS-PD are complementary to data augmentation.

settings (Table C.8 in the Appendix). We find that in most cases, AS-PD tends to achieve better results over PGD-PD due to its sparser solutions and faster convergence properties.

Using auxiliary data with traditional data augmentation. Can auxiliary data be used effectively with more traditional data augmentation techniques? To measure this, we evaluate our benchmark with TrivialAugment [MH21] for vision and back-translation augmentation [SHB15] for language. Indeed, we find that the improvements from augmenting with auxiliary data are complementary to traditional data augmentation techniques. For example, Table 4.5 highlights a subset of our language results, which shows that AS-PD combined with traditional data augmentation (DA) achieves better results than traditional data augmentation alone. We see similar results in our full benchmark for both vision and language (see Table C.6 and Table C.9 in the Appendix).

4.4.3 Analyzing the target dataset

In this section, we highlight how our framework can further be used as an analysis tool for the target dataset. Specifically, the subset of sources identified by our framework provides a natural description of the composition of the target dataset. For example, in Figure 4-8, we plot the projection of the Emotion dataset onto the Emoji dataset. The resulting proportions of the projection informs us about what kinds of Emoji data is closest to each Emotion class. "Optimistic" emotion data tends to be closest to data with Christmas tree and sparkling emojis, while "sad" emotion data tends to be closest to data with crying emojis. Figure 4-9 shows a similar analysis for projections of cats from various datasets onto ImageNet cats, revealing that CIFAR10 contains



(a) Optimistic (b) Sad

Figure 4-8: A visualization of the proportions of each Emoji source that are most aligned with classes of the Emotion dataset (Optimistic and Sad), as identified via dataset projection. Additional examples showing the projections of the Angry and Joy classes onto the Emoji dataset are in Appendix C.5.

(a) CIFAR10 (b) STL10

Figure 4-9: A visualization of the proportions of each ImageNet cat source that is most aligned with cat classes from CIFAR10 and STL10, as identified via dataset projection. Additional examples showing the projections of Oxford-IIIT Pet and 3DB cats are in Appendix C.5.

primarily household cats, while STL10 also contains cats found in the wild. Additional examples can be found in Appendix C.5.

4.5 Related work

Dataset projection is related to various methods including dataset pruning [AAP05], core-set construction [TKC05], and dataset distillation [Wan+18; Ngu+21]. These methods attempt to reduce the size of a single dataset while still retaining the most "valuable" training data signal. In contrast, our goal in dataset projection is to match a target dataset to the span of multiple auxiliary sources.

The use of an auxiliary dataset is related to the field of auxiliary learning [Vin+08; Zha+14; Mor+18; LDJ19], in which additional tasks are used during training to help improve performance on a target task. Crucially, these approaches train on auxiliary tasks alongside the primary task, whereas dataset projection extracts target-aligned

subsets of auxiliary data without requiring any auxiliary task.

Works have also explored modifying synthetic data generation processes to more closely resemble a target dataset. Kar et al. [Kar+19a] and Devaranjan, Kar, and Fidler [DKF20a] used a parameterized graphics engine and probabilistic scene grammars to match a target dataset of scenes. On the other hand, dataset projection is agnostic to the way in which auxiliary data is generated and can be used to select from any type of data source. Our work is also broadly related to the field of domain adaptation [WD18; WC20; Far+21], which aims to adapt *models* trained on an auxiliary dataset to improve performance on a target dataset. In contrast, dataset projection is completely model-agnostic and answers the fundamental question of how to align the *data*. Using our framework with domain adaptation to align both the model and data jointly is an interesting future direction.

Our method builds on an extensive line of work on minimization over simplex constraints [Ber82; BM02], as well as distance metrics that measure the similarity of two image distributions [Gre+12]. Our simplex formulation of aligning multiple dataset sources uses constraints with similarities to the problems posed by Hashimoto [Has21], but for a different purpose and objective. While Hashimoto [Has21] uses their framework to study theoretical scaling laws for models generalization, we solve the optimization problem to search for target-aligned subsets of auxiliary data.

4.6 Conclusion

In this work, we pose the problem of dataset projection and develop methods to find the most target-aligned subset of auxiliary data. When using a biased auxiliary dataset, we demonstrate that it can be beneficial to use just a portion of the data rather than all of it. In these situations, dataset projection can select subsets of the data that better approximate the target dataset and can lead to better test performance when augmenting the target dataset. We highlight these trends empirically on a variety of language and vision datasets and further highlight the use of our framework as an analysis tool. Our work takes a step towards understanding when more data is useful, and finding more useful data.

Part III

Verifying model robustness

Chapter 5

Evaluating model robustness with mixed integer programming

5.1 Introduction

Neural networks trained only to optimize for training accuracy have been shown to be vulnerable to *adversarial examples*: perturbed inputs that are very similar to some regular input but for which the output is radically different [Sze+14]. There is now a large body of work proposing defense methods to produce classifiers that are more robust to adversarial examples. However, as long as a defense is evaluated only via heuristic attacks (such as the Fast Gradient Sign Method (FGSM) [GSS15] or [CW17c]'s attack (CW)), we have no guarantee that the defense actually increases the robustness of the classifier produced. Defense methods thought to be successful when published have often been later found to be vulnerable to a new class of attacks. For instance, multiple defense methods are defeated in [CW17b] by constructing defense-specific loss functions and in [ACW18] by overcoming obfuscated gradients.

Fortunately, we *can* evaluate robustness to adversarial examples in a principled fashion. One option is to determine (for each test input) the minimum distance to the closest adversarial example, which we call the *minimum adversarial distortion* [Car+17]. Alternatively, we can determine the *adversarial test accuracy* [Bas+16], which is the proportion of the test set for which no perturbation in some bounded class causes a misclassification. An increase in the mean minimum adversarial distortion or in the adversarial test accuracy indicates an improvement in robustness.[1]

[1]The two measures are related: a solver that can find certificates for bounded perturbations can be used iteratively (in a binary search process) to find minimum distortions.

We present an efficient implementation of a mixed-integer linear programming (MILP) verifier for properties of piecewise-linear feed-forward neural networks. Our tight formulation for non-linearities and our novel presolve algorithm combine to minimize the number of binary variables in the MILP problem and dramatically improve its numerical conditioning. Optimizations in our MILP implementation improve performance by several orders of magnitude when compared to a naïve MILP implementation, and we are two to three orders of magnitude faster than the state-of-the-art Satisfiability Modulo Theories (SMT) based verifier, Reluplex [Kat+17]

We make the following key contributions:

- We demonstrate that, despite considering the full combinatorial nature of the network, our verifier *can* succeed on larger neural networks, including those with convolutional and residual layers, when evaluating the robustness of these networks to bounded perturbations.

- We identify *why* we can succeed on larger neural networks with hundreds of thousands of units. First, a large fraction of the ReLUs can be shown to be either always active or always inactive over the bounded input domain. Second, since the predicted label is determined by the unit in the final layer with the maximum activation, proving that a unit *never* has the maximum activation over all bounded perturbations eliminates it from consideration. We fully exploit both phenomena, reducing the overall number of non-linearities considered.

- We determine for the first time the exact adversarial accuracy for MNIST classifiers to perturbations with bounded l_∞ norm ϵ. We are also able to certify more samples than the state-of-the-art *and* find more adversarial examples across MNIST and CIFAR-10 classifiers with different architectures trained with a variety of robust training procedures.

Our code will be made available after review.

5.2 Related work

Our work relates most closely to other work on verification of piecewise-linear neural networks; [Bun+17] provides a good overview of the field. We categorize verification procedures as *complete* or *incomplete*. To understand the difference between these two types of procedures, we consider the example of evaluating adversarial accuracy.

As in [WK17], we call the exact set of all final-layer activations that can be achieved by applying a bounded perturbation to the input the *adversarial polytope*. Incomplete verifiers reason over an *outer approximation* of the adversarial polytope. As a result, when using incomplete verifiers, the answer to some queries about the adversarial polytope may not be decidable. In particular, incomplete verifiers can only certify robustness for a fraction of robust input; the status for the remaining input is undetermined. In contrast, complete verifiers reason over the *exact* adversarial polytope. Given sufficient time, a complete verifier can provide a definite answer to any query about the adversarial polytope. In the context of adversarial accuracy, complete verifiers will obtain a valid adversarial example or a certificate of robustness for *every* input. When a time limit is set, complete verifiers behave like incomplete verifiers, and resolve only a fraction of queries. However, complete verifiers do allow users to answer a larger fraction of queries by extending the set time limit.

Incomplete verifiers for evaluating network robustness employ a range of techniques, including duality [Dvi+18a; WK17; RSL18], layer-by-layer approximations of the adversarial polytope [XTJ18], discretizing the search space [Hua+17b], abstract interpretation [Geh+18], bounding the local Lipschitz constant [Wen+18], or bounding the activation of the ReLU with linear functions [Wen+18].

Complete verifiers typically employ either MILP solvers as we do [CNR17; Dut+18; FJ18; LM17a] or SMT solvers [Car+17; Ehl17; Kat+17; Sch+15]. Our approach improves upon existing MILP-based approaches with a tighter formulation for non-linearities and a novel presolve algorithm that makes full use of all information available, leading to solve times several orders of magnitude faster than a naïvely implemented MILP-based approach. When comparing our approach to the state-of-the-art SMT-based approach (Reluplex) on the task of finding minimum adversarial distortions, we find that our verifier is two to three orders of magnitude faster. Crucially, these improvements in performance allow our verifier to verify a network with over 100,000 units — several orders of magnitude larger than the largest MNIST classifier previously verified with a complete verifier.

A complementary line of research to verification is in robust training procedures that train networks *designed* to be robust to bounded perturbations. Robust training attempts to minimize the "worst-case loss" for each example — that is, the maximum loss over all bounded perturbations of that example [WK17]. Since calculating the exact worst-case loss can be computationally costly, robust training procedures typically instead minimize an estimate of the worst-case loss: either a *lower bound* as in the case of adversarial training [GSS15], or an *upper bound* as for certified training approaches

[HA17; WK17; RSL18]. Complete verifiers such as ours can augment robust training procedures by resolving the status of input for which heuristic attacks cannot find an adversarial example but incomplete verifiers cannot guarantee robustness, enabling more accurate comparisons between different training procedures.

5.3 Background and notation

We denote a neural network by a function $f(\cdot;\theta) : \mathbb{R}^m \to \mathbb{R}^n$ parameterized by a (fixed) vector of weights θ. For a classifier, the output layer has a neuron for each target class.

Verification as solving an MILP. The general problem of verification is to determine whether some property P on the output of a neural network holds for all input in a bounded input domain $\mathcal{C} \subseteq \mathbb{R}^m$. For the verification problem to be expressible as solving an MILP, P must be expressible as the conjunction or disjunction of linear properties $P_{i,j}$ over some set of polyhedra $\mathcal{C}_i$, where $\mathcal{C} = \cup\mathcal{C}_i$.

In addition, $f(\cdot)$ must be composed of piecewise-linear layers. This is not a particularly restrictive requirement: piecewise-linear layers include layers that are linear transformations (such as fully-connected, convolution, and average-pooling layers) and layers that use piecewise-linear functions (such as ReLU or maximum-pooling layers). We provide details on how to express these piecewise-linear functions in Section 5.4.1. The "shortcut connections" used in architectures such as ResNet [He+16] are also linear, and batch normalization [IS15] or dropout [Sri+14] are linear transformations at *evaluation time* [Bun+17].

5.4 Formulating robustness evaluation of classifiers as an MILP

Evaluating Adversarial Accuracy. Let $\mathcal{G}(x)$ denote the region in the input domain corresponding to all allowable perturbations of a particular input x. In general, perturbed inputs must also remain in the domain of valid inputs $\mathcal{X}_{valid}$. For example, for normalized images with pixel values ranging from 0 to 1, $\mathcal{X}_{valid} = [0,1]^m$. As in [Mad+18], we say that a neural network is robust to perturbations on x if the predicted probability of the true label $\lambda(x)$ exceeds that of every other label for all

perturbations:

$$\forall x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid}) : \ \arg\max_i(f_i(x')) = \lambda(x) \tag{5.1}$$

Equivalently, the network is robust to perturbations on x if and only if Equation 5.2 is infeasible for x'.

$$(x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid})) \wedge \left(f_{\lambda(x)}(x') < \max_{\mu \in [1,n] \setminus \{\lambda(x)\}} f_\mu(x') \right) \tag{5.2}$$

where $f_i(\cdot)$ is the i^{th} output of the network. For conciseness, we call x *robust* with respect to the network if $f(\cdot)$ is robust to perturbations on x. If x is not robust, we call any x' satisfying the constraints a *valid* adversarial example to x. The *adversarial accuracy* of a network is the fraction of the test set that is robust; the *adversarial error* is simply the complement of the adversarial accuracy.

As long as $\mathcal{G}(x) \cap \mathcal{X}_{valid}$ can be expressed as the union of a set of polyhedra, the feasibility problem can be expressed as an MILP. The four robust training procedures we consider [WK17; Won+18; Mad+18; RSL18] are designed to be robust to perturbations with bounded l_∞ norm, and the l_∞-ball of radius ϵ around each input x can be succinctly represented by the set of linear constraints $\mathcal{G}(x) = \{x' \mid \forall i : -\epsilon \leq (x - x')_i \leq \epsilon\}$.

Evaluating Mean Minimum Adversarial Distortion. Let $d(\cdot, \cdot)$ denote a distance metric that measures the perceptual similarity between two input images. The minimum adversarial distortion under d for input x with true label $\lambda(x)$ corresponds to the solution to the optimization:

$$\min_{x'} d(x', x) \tag{5.3}$$

$$\text{subject to} \quad \arg\max_i(f_i(x')) \neq \lambda(x) \tag{5.4}$$

$$x' \in \mathcal{X}_{valid} \tag{5.5}$$

We can *target* the attack to generate an adversarial example that is classified in one of a set of target labels T by replacing Equation 5.4 with $\arg\max_i(f_i(x')) \in T$.

The most prevalent distance metrics in the literature for generating adversarial examples are the l_1 [CW17c; Che+18], l_2 [Sze+14], and l_∞ [GSS15; Pap+16] norms. All three can be expressed in the objective without adding any additional integer variables to the model [BV04]; details are in Appendix D.1.3.

5.4.1 Formulating piecewise-linear functions in the classifier

Tight formulations of the rectifier and maximum functions are critical to good performance of the MILP solver; we thus present these formulations in detail with accompanying proofs.[2]

Formulating ReLU Let $y = \max(x, 0)$, and $l \leq x \leq u$. There are three possibilities for the *phase* of the ReLU. If $u \leq 0$, we have $y \equiv 0$. We say that such a unit is *stably inactive*. Similarly, if $l \geq 0$, we have $y \equiv x$. We say that such a unit is *stably active*. Otherwise, the unit is *unstable*. For unstable units, we introduce an indicator decision variable $a = \mathbb{1}_{x \geq 0}$. As we prove in Appendix D.1.1, $y = \max(x, 0)$ is equivalent to the set of linear and integer constraints in Equation 5.6.

$$(y \leq x - l(1 - a)) \wedge (y \geq x) \wedge (y \leq u \cdot a) \wedge (y \geq 0) \wedge (a \in \{0, 1\}) \tag{5.6}$$

Formulating the Maximum Function Let $y = \max(x_1, x_2, \ldots, x_m)$, and $l_i \leq x_i \leq u_i$.

Proposition 1 *Let $l_{max} \triangleq \max(l_1, l_2, \ldots, l_m)$. We can eliminate from consideration all x_i where $u_i \leq l_{max}$, since we know that $y \geq l_{max} \geq u_i \geq x_i$.*

We introduce an indicator decision variable a_i for each of our input variables, where $a_i = 1 \implies y = x_i$. Furthermore, we define $u_{max,-i} \triangleq \max_{j \neq i}(u_j)$. As we prove in Appendix D.1.2, the constraint $y = \max(x_1, x_2, \ldots, x_m)$ is equivalent to the set of linear and integer constraints in Equation 5.7.

$$\bigwedge_{i=1}^{m} ((y \leq x_i + (1 - a_i)(u_{max,-i} - l_i)\ ) \wedge (y \geq x_i\ )) \wedge \left(\sum\nolimits_{i=1}^{m} a_i = 1\right) \wedge (a_i \in \{0, 1\}) \tag{5.7}$$

5.4.2 Progressive bounds tightening

We previously assumed that we had some element-wise bounds on the inputs to non-linearities. In practice, we have to carry out a presolve step to determine these bounds. Determining tight bounds is critical for problem tractability: tight bounds strengthen the problem formulation and thus improve solve times [Vie15]. For instance, if we can prove that the phase of a ReLU is stable, we can avoid introducing a binary variable.

[2] [HV17] presents formulations for general piecewise linear functions.

More generally, loose bounds on input to some unit will propagate downstream, leading to units in later layers having looser bounds.

We used two procedures to determine bounds: Interval Arithmetic (ia), also used in [CNR17; Dut+18], and the slower but tighter Linear Programming (lp) approach. Implementation details are in Appendix D.2.

Since faster procedures achieve efficiency by compromising on tightness of bounds, we face a trade-off between higher *build times* (to determine tighter bounds to inputs to non-linearities), and higher *solve times* (to solve the main MILP problem in Equation 5.2 or Equation 5.3-5.5). While a degree of compromise is inevitable, our knowledge of the non-linearities used in our network allows us to reduce average build times without affecting the strength of the problem formulation.

The key observation is that, for piecewise-linear non-linearities, there are thresholds beyond which further refining a bound will not improve the problem formulation. With this in mind, we adopt a progressive bounds tightening approach: we begin by determining coarse bounds using fast procedures and only spend time refining bounds using procedures with higher computational complexity if doing so could provide additional information to improve the problem formulation.[3] Pseudocode demonstrating how to efficiently determine bounds for the tightest possible formulations for the ReLU and maximum function is provided below and in Appendix D.3 respectively.

GetBoundsForReLU(x, fs)

$\triangleright$ fs are the procedures to determine bounds
$\triangleright$ fs sorted in increasing computational complexity.
$l_{best} = -\infty$; $u_{best} = \infty$ $\triangleright$ initialize best known upper and lower bounds on x
for f in fs: $\triangleright$ carrying out progressive bounds tightening
    do $u = f(x, boundType = upper)$; $u_{best} = \min(u_{best}, u)$
        if $u_{best} \leq 0$ **return** (l_{best}, u_{best})
        $\triangleright$ Early return: $x \leq u_{best} \leq 0$; thus $\max(x, 0) \equiv 0$.
        $l = f(x, boundType = lower)$; $l_{best} = \max(l_{best}, l)$
        if $l_{best} \geq 0$ **return** (l_{best}, u_{best})
        $\triangleright$ Early return: $x \geq l_{best} \geq 0$; thus $\max(x, 0) \equiv x$
return (l_{best}, u_{best}) $\triangleright$ x could be either positive or negative.

The process of progressive bounds tightening is naturally extensible to more procedures. [WK17; Won+18; Dvi+18a; Wen+18] each discuss procedures to determine

[3] As a corollary, we always use only ia for the output of the first layer, since the independence of network input implies that ia is provably optimal for that layer.

bounds with computational complexity and tightness intermediate between IA and LP. Using one of these procedures in addition to IA and LP has the potential to further reduce build times.

5.5 Experiments

Dataset. All experiments are carried out on classifiers for the MNIST dataset of handwritten digits or the CIFAR-10 dataset of color images.

Architectures. We conduct experiments on a range of feed-forward networks. In all cases, ReLUs follow each layer except the output layer. **MLP-**$m\times[n]$ refers to a multilayer perceptron with m hidden layers and n units per hidden layer. We further abbreviate MLP-1×[500] and MLP-2×[200] as $\mathbf{MLP_A}$ and $\mathbf{MLP_B}$ respectively. **CNN** refers to the ConvNet architecture used for the robust MNIST classifier in [WK17]. The network has two convolutional layers (stride length 2) with 16 and 32 filters respectively (size 4×4 in both layers), followed by a fully-connected layer with 100 units. **RES** refers to the ResNet architecture used in [Won+18], with 9 convolutional layers in four blocks, followed by two fully-connected layers with 4096 and 1000 units respectively.

Training Methods. We conduct experiments on networks trained with a regular loss function and networks trained to be robust. Networks trained to be robust are identified by a prefix corresponding to the method used to approximate the worst-case loss: $\mathbf{LP_d}$[4] when the dual of a linear program is used, as in [WK17]; $\mathbf{SDP_d}$ when the dual of a semidefinite relaxation is used, as in [RSL18]; and **Adv** when adversarial examples generated via Projected Gradient Descent (PGD) are used, as in [Mad+18]. Full details on each network are in Appendix D.4.1.

Experimental Setup. We run experiments on a modest 8 CPUs@2.20 GHz with 8GB of RAM. Appendix D.4.2 provides additional details about the computational environment. Maximum build effort is LP. Unless otherwise noted, we report a timeout if solve time for some input exceeds 1200s.

[4]This is unrelated to the procedure to determine bounds named LP.

5.5.1 Performance comparisons

Comparisons to other MILP-based Complete Verifiers

Our MILP approach implements three key optimizations: we use *progressive tightening*, make use of the information provided by the *restricted input domain* $\mathcal{G}(x)$, and use *asymmetric bounds* in the ReLU formulation in Equation 5.6. None of the four other MILP-based complete verifiers implement *progressive tightening* or use the *restricted input domain*, and only [FJ18] uses *asymmetric bounds*. Since none of the four verifiers have publicly available code, we use ablation tests to provide an idea of the difference in performance between our verifier and these existing ones.

When removing *progressive tightening*, we directly use LP rather than doing IA first. When removing *using restricted input domain*, we determine bounds under the assumption that our perturbed input could be anywhere in the full input domain $\mathcal{X}_{valid}$, imposing the constraint $x' \in \mathcal{G}(x)$ only after all bounds are determined. Finally, when removing *using asymmetric bounds*, we replace l and u in Equation 5.6 with $-M$ and M respectively, where $M \triangleq \max(-l, u)$, as is done in [CNR17; Dut+18; LM17a]. We carry out experiments on an MNIST classifier, and results from performing these experiments are reported in Table 5.1.

Table 5.1: Results of ablation testing on our verifier, where each test removes a single optimization. The task was to determine the adversarial accuracy of the MNIST classifier LP_d-CNN to perturbations with l_∞ norm-bound $\epsilon = 0.1$. *Build* time refers to time used to determine bounds, while *solve* time refers to time used to solve the main MILP problem in Equation 5.2 once all bounds have been determined. During solve time, we solve a linear program for each of the *nodes explored* in the MILP search tree.
[†]We exclude the initial build time required (3593s) to determine reusable bounds.

Optimization Removed	Mean Time / s			Nodes Explored		Fraction Timed Out
	Build	Solve	Total	Mean	Median	
(Control)	3.44	0.08	3.52	2.05	0	0
Progressive tightening	7.66	0.11	7.77	2.05	0	0
Using restricted input domain[†]	1.49	56.47	57.96	1343.21	67	0.0047
Using asymmetric bounds	4465.11	133.03	4598.15	1498.35	113	0.0300

The ablation tests demonstrate that each optimization is critical to the performance of our verifier. In terms of performance comparisons, we expect our verifier to have a runtime several orders of magnitude faster than any of the three verifiers not using *asymmetric bounds*. While [FJ18] do use *asymmetric bounds*, they do not use information from the *restricted input domain*; we thus expect our verifier to have a

runtime at least an order of magnitude faster than theirs.

Comparisons to Other Complete and Incomplete Verifiers

We also compared our verifier to other verifiers on the task of finding minimum targeted adversarial distortions for MNIST test samples. Verifiers included for comparison are 1) `Reluplex` [Kat+17], a complete verifier also able to find the true minimum distortion; and 2) `LP`[5], `Fast-Lip`, `Fast-Lin` [Wen+18], and `LP-full` [WK17], incomplete verifiers that provide a certified *lower* bound on the minimum distortion.

Verification Times, vis-à-vis the state-of-the-art SMT-based complete verifier `Reluplex`. Figure 5-1 presents average verification times per sample. All solves for our method were run to completion. On the l_∞ norm, we improve on the speed of `Reluplex` by two to three orders of magnitude.

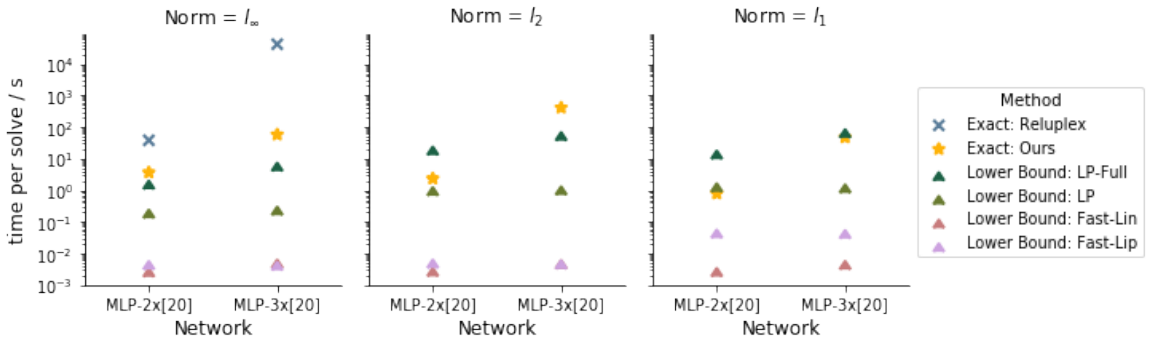


Figure 5-1: Average times for determining bounds on / exact values of the minimum targeted adversarial distortion for MNIST test samples. We improve on the speed of the state-of-the-art complete verifier `Reluplex` by two to three orders of magnitude. Results for methods other than ours are from [Wen+18]; results for `Reluplex` were only available in [Wen+18] for the l_∞ norm.

Minimum Targeted Adversarial Distortions, vis-à-vis incomplete verifiers. Figure 5-2 compares lower bounds from the incomplete verifiers to the exact value we obtain. The gap between the best lower bound and the true minimum adversarial distortion is significant even on these small networks. This corroborates the observation in [RSL18] that incomplete verifiers provide weak bounds if the network they are applied to is not optimized for that verifier. For example, under the l_∞ norm, the best certified lower bound is less than half of the true minimum distortion. In context: a network robust to perturbations with l_∞ norm-bound $\epsilon = 0.1$ would only be verifiable to $\epsilon = 0.05$.

[5]This is unrelated to the procedure to determine bounds named LP, or the training procedure LP_d.

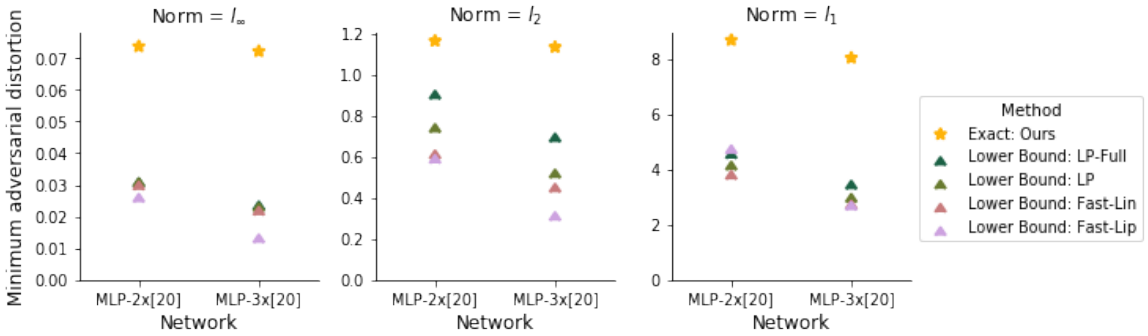


Figure 5-2: Average of bounds on / exact values of the minimum targeted adversarial distortion for MNIST test samples. The gap between the true minimum adversarial distortion and the best lower bound is significant and increases for deeper networks.

5.5.2 Determining adversarial accuracy of MNIST and CIFAR-10 classifiers

We use our verifier to determine the adversarial accuracy of classifiers trained by a range of robust training procedures on the MNIST and CIFAR-10 datasets. Table 5.2 presents the test error and estimates of the adversarial error for these classifiers.[6] For MNIST, we verified a range of networks trained to be robust to attacks with bounded l_∞ norm $\epsilon = 0.1$, as well as networks trained to be robust to larger attacks of $\epsilon = 0.2, 0.3$ and 0.4. Lower bounds on the adversarial error are proven by providing adversarial examples for input that is not robust. We compare the number of samples for which we successfully find adversarial examples to the number for PGD, a strong first-order attack. Upper bounds on the adversarial error are proven by providing certificates of robustness for input that is robust. We compare our upper bounds to the previous state-of-the-art for each network.

While performance depends on the training method and architecture, we improve on both the lower and upper bounds for *every* network tested.[7] For lower bounds, we successfully find an adversarial example for every test sample that PGD finds an adversarial example for. In addition, we observe that PGD 'misses' some valid adversarial examples: it fails to find these adversarial examples even though they are within the norm bounds. As the last three rows of Table 5.2 show, PGD misses for a

[6] As mentioned in Section 5.2, complete verifiers will obtain either a valid adversarial example or a certificate of robustness for every input given enough time. However, we do not *always* have a guarantee of robustness or a valid adversarial example for every test sample since we terminate the optimization at 1200s to provide a better picture of how our verifier performs within reasonable time limits.

[7] On SDP_d-MLP_A, the verifier in [RSL18] finds certificates for 372 samples for which our verifier reaches its time limit.

Table 5.2: Adversarial accuracy of MNIST and CIFAR-10 classifiers to perturbations with l_∞ norm-bound ϵ. In every case, we improve on both 1) the lower bound on the adversarial error, found by PGD, and 2) the previous state-of-the-art (SOA) for the upper bound, generated by the following methods: [1] [WK17], [2] [Dvi+18a], [3] [RSL18]. For classifiers marked with a ✓, we have a guarantee of robustness or a valid adversarial example for *every* test sample. Gaps between our bounds correspond to cases where the solver reached the time limit for some samples. Additional solve statistics on nodes explored are in Appendix D.5.

Dataset	Network	ϵ	Test Error	Certified Bounds on Adversarial Error					Mean Time / s
				Lower Bound		Upper Bound		No Gap?	
				PGD	Ours	SOA	Ours		
MNIST	LP_d-CNN	0.1	1.89%	4.11%	**4.38%**	5.82%[1]	**4.38%**	✓	3.52
	Adv-CNN	0.1	0.96%	4.10%	**4.21%**	—	**7.21%**		135.74
	Adv-MLP_B	0.1	4.02%	9.03%	**9.68%**	15.41%[2]	**9.68%**	✓	3.69
	SDP_d-MLP_A	0.1	4.18%	11.51%	**14.36%**	34.77%[3]	**30.81%**		312.43
	LP_d-CNN	0.2	4.23%	9.54%	**10.68%**	17.50%[1]	**10.68%**	✓	7.32
	LP_d-CNN	0.3	11.40%	22.70%	**25.79%**	35.03%[1]	**25.79%**	✓	5.13
	LP_d-CNN	0.4	26.13%	39.22%	**48.98%**	62.49%[1]	**48.98%**	✓	5.07
CIFAR-10	LP_d-RES	$\frac{8}{255}$	72.93%	76.52%	**77.29%**	78.52%[1]	**77.60%**		15.23

larger fraction of test samples when ϵ is larger. We also found that PGD is far more likely to miss for some test sample if the minimum adversarial distortion for that sample is close to ϵ; this observation is discussed in more depth in Appendix D.6. For upper bounds, we improve on the bound on adversarial error even when the upper bound on the worst-case loss — which is used to generate the certificate of robustness — is *explicitly* optimized for during training (as is the case for LP_d and SDP_d training). Our method also scales well to the more complex CIFAR-10 dataset and the larger LP_d-RES network (which has 107,496 units), with the solver reaching the time limit for only 0.31% of samples.

Most importantly, we are able to determine the **exact adversarial accuracy** for Adv-MLP_B and LP_d-CNN for all ϵ tested, finding either a certificate of robustness or an adversarial example for *every* test sample. For Adv-MLP_B and LP_d-CNN, running our verifier over the full test set takes approximately 10 hours — the same order of magnitude as the time to train each network on a single GPU. Better still, verification of individual samples is fully parallelizable.

Observations on Determinants of Verification Time

Ceteris paribus, we might expect verification time to be correlated to the total number of ReLUs, since the solver may need to explore both possibilities for the phase of each ReLU. However, there is clearly more at play: even though LP_d-CNN and Adv-CNN have identical architectures, verification time for Adv-CNN is two orders of magnitude higher.

Table 5.3: Determinants of verification time: mean verification time is 1) inversely correlated to the number of labels that can be eliminated from consideration and 2) correlated to the number of ReLUs that are not provably stable. Results are for $\epsilon = 0.1$ on MNIST.

Network	Mean Time / s	Number of Labels Eliminated	Number of ReLUs			
			Possibly Unstable	Provably Stable		Total
				Active	Inactive	
LP_d-CNN	3.52	6.57	121.18	1552.52	3130.30	4804
Adv-CNN	135.74	3.14	545.90	3383.30	874.80	4804
Adv-MLP_B	3.69	4.77	55.21	87.31	257.48	400
SDP_d-MLP_A	312.43	0.00	297.66	73.85	128.50	500

The key lies in the restricted input domain $\mathcal{G}(x)$ for each test sample x. When input is restricted to $\mathcal{G}(x)$, we can prove that many ReLUs are stable (with respect to $\mathcal{G}$). Furthermore, we can eliminate some labels from consideration by proving that the upper bound on the output neuron corresponding to that label is lower than the lower bound for some other output neuron. As the results in Table 5.3 show, a significant number of ReLUs can be proven to be stable, and a significant number of labels can be eliminated from consideration. Rather than being correlated to the *total* number of ReLUs, solve times are instead more strongly correlated to the number of ReLUs that are not provably stable, as well as the number of labels that cannot be eliminated from consideration.

5.6 Discussion

This paper presents an efficient complete verifier for piecewise-linear neural networks. While we have focused on evaluating networks on the class of perturbations they are *designed* to be robust to, *defining* a class of perturbations that generates images perceptually similar to the original remains an important direction of research. Our

verifier *is* able to handle new classes of perturbations (such as convolutions) as long as the set of perturbed images is a union of polytopes in the input space.

We close with ideas on improving verifiability of neural networks. As previously discussed, increasing the number of locally stable ReLUs speeds up verification. We also observed (see Appendix D.7) that sparsifying weights promotes verifiability. Adopting a principled sparsification approach (for example, l_1 regularization during training, or pruning and retraining [HMD16]) could potentially further increase verifiability without compromising on the true adversarial accuracy.

Chapter 6

Speeding up robustness verification via inducing ReLU stability

6.1 Introduction

Deep neural networks (DNNs) have recently achieved widespread success in image classification [KSH12], face and speech recognition [Tai+14; Hin+12], and game playing [Sil+16; Sil+17]. This success motivates their application in a broader set of domains, including more safety-critical environments. This thrust makes understanding the reliability and robustness of the underlying models, let alone their resilience to manipulation by malicious actors, a central question. However, predictions made by machine learning models are often brittle. A prominent example is the existence of adversarial examples [Sze+14]: imperceptibly modified inputs that cause state-of-the-art models to misclassify with high confidence.

There has been a long line of work on both generating adversarial examples, called *attacks* [CW17a; CW17b; ACW18; AS17; Ues+18; Evt+17], and training models robust to adversarial examples, called *defenses* [GSS15; Pap+16; Mad+18; KKG18]. However, recent research has shown that most defenses are ineffective [CW17b; ACW18; Ues+18]. Furthermore, even for defenses such as that of [Mad+18] that have seen empirical success against many attacks, we are unable to conclude yet with certainty that they are robust to all attacks that we want these models to be resilient to.

This state of affairs gives rise to the need for *verification of networks*, i.e., the task of *formally* proving that no small perturbations of a given input can cause it to be misclassified by the network model. Although many exact verifiers[1] have been designed

[1] Also sometimes referred to as combinatorial verifiers.

to solve this problem, the verification process is often intractably slow. For example, when using the Reluplex verifier of [Kat+17], even verifying a small MNIST network turns out to be computationally infeasible. Thus, addressing this intractability of exact verification is the primary goal of this work.

Our contributions. Our starting point is the observation that, typically, model training and verification are decoupled and seen as two distinct steps. Even though this separation is natural, it misses a key opportunity: the ability to align these two stages. Specifically, applying the principle of *co-design* during model training is possible: training models in a way to encourage them to be simultaneously robust and easy-to-exactly-verify. This insight is the cornerstone of the techniques we develop in this paper.

In this work, we use the principle of co-design to develop training techniques that give models that are both robust and easy-to-verify. Our techniques rely on improving two key properties of networks: weight sparsity and ReLU stability. Specifically, we first show that natural methods for improving weight sparsity during training, such as ℓ_1-regularization, give models that can already be verified much faster than current methods. This speedup happens because in general, exact verifiers benefit from having fewer variables in their formulations of the verification task. For instance, for exact verifiers that rely on linear programming (LP) solvers, sparser weight matrices means fewer variables in those constraints.

We then focus on the major speed bottleneck of current approaches to exact verification of ReLU networks: the need of exact verification methods to "branch," i.e., consider two possible cases for each ReLU (ReLU being active or inactive). Branching drastically increases the complexity of verification. Thus, well-optimized verifiers will not need to branch on a ReLU if it can determine that the ReLU is *stable*, i.e. that the ReLU will always be active or always be inactive for any perturbation of an input. This motivates the key goal of the techniques presented in this paper: we aim to minimize branching by maximizing the number of stable ReLUs. We call this goal *ReLU stability* and introduce a regularization technique to induce it.

Our techniques enable us to train weight-sparse and ReLU stable networks for MNIST and CIFAR-10 that can be verified significantly faster. Specifically, by combining natural methods for inducing weight sparsity with a robust adversarial training procedure (cf. [GSS15]), we are able to train networks for which almost 90% of inputs can be verified in an amount of time that is small[2] compared to previous verification techniques. Then, by also adding our regularization technique for inducing

[2]We chose our time budget for verification to be 120 seconds per input image.

ReLU stability, we are able to train models that can be verified an additional 4–13x times as fast while maintaining state-of-the-art accuracy on MNIST. Our techniques show similar improvements for exact verification of CIFAR models. In particular, we achieve the following verification speed and provable robustness results for ℓ_∞ norm-bound adversaries:

Dataset	Epsilon	Provable Adversarial Accuracy	Average Solve Time (s)
MNIST	$\epsilon = 0.1$	94.33%	0.49
	$\epsilon = 0.2$	89.79%	1.13
	$\epsilon = 0.3$	80.68%	2.78
CIFAR	$\epsilon = 2/255$	45.93%	13.50
	$\epsilon = 8/255$	20.27%	22.33

Our network for $\epsilon = 0.1$ on MNIST achieves provable adversarial accuracies comparable with the current best results of [Won+18] and [Dvi+18b], and our results for $\epsilon = 0.2$ and $\epsilon = 0.3$ achieve the best provable adversarial accuracies yet. To the best of our knowledge, we also achieve the first nontrivial provable adversarial accuracy results using exact verifiers for CIFAR-10.

Finally, we design our training techniques with universality as a goal. We focus on improving the *input* to the verification process, regardless of the verifier we end up using. This is particularly important because research into network verification methods is still in its early stages, and our co-design methods are compatible with the best current verifiers (LP/MILP-based methods) and should be compatible with any future improvements in verification.

Our code is available at `https://github.com/MadryLab/relu_stable`.

6.2 Background and related work

Exact verification of networks has been the subject of many recent works [Kat+17; Ehl17; Car+17; TXT19; LM17b; CNR17]. To understand the context of these works, observe that for linear networks, the task of exact verification is relatively simple and can be done by solving a LP. For more complex models, the presence of nonlinear ReLUs makes verification over all perturbations of an input much more challenging. This is so as ReLUs can be active or inactive depending on the input, which can cause exact verifiers to "branch" and consider these two cases separately. The number of such cases that verifiers have to consider might grow exponentially with the number of ReLUs, so verification speed will also grow exponentially in the worst case. [Kat+17]

further illustrated the difficulty of exact verification by proving that it is NP-complete. In recent years, formal verification methods were developed to verify networks. Most of these methods use satisfiability modulo theory (SMT) solvers [Kat+17; Ehl17; Car+17] or LP and Mixed-Integer Linear Programming (MILP) solvers [TXT19; LM17b; CNR17]. However, all of them are limited by the same issue of scaling poorly with the number of ReLUs in a network, making them prohibitively slow in practice for even medium-sized models.

One recent approach for dealing with the inefficiency of exact verifiers is to focus on certification methods[3] [WK17; Won+18; Dvi+18b; RSL18; MGV18; SND18]. In contrast to exact verification, these methods do not solve the verification task directly; instead, they rely on solving a *relaxation* of the verification problem. This relaxation is usually derived by overapproximating the adversarial polytope, or the space of outputs of a network for a region of possible inputs. These approaches rely on training models in a specific manner that makes certification of those models easier. As a result, they can often obtain provable adversarial accuracy results faster. However, certification is fundamentally different from verification in two primary ways. First, it solves a relaxation of the original verification problem. As a result, certification methods can fail to certify many inputs that are actually robust to perturbations – only exact verifiers, given enough time, can give conclusive answers on robustness for every single input. Second, certification approaches fall under the paradigm of co-training, where a certification method only works well on models specifically trained for that certification step. When used as a black box on arbitrary models, the certification step can yield a high rate of false negatives. For example, [RSL18] found that their certification step was significantly less effective when used on a model trained using [WK17]'s training method, and vice versa. In contrast, we design our methods to be universal. They can be combined with any standard training procedure for networks and will improve exact verification speed for any LP/MILP-based exact verifier. Our methods can also decrease the amount of overapproximation incurred by certification methods like [WK17; Dvi+18b]. Similar to most of the certification methods, our technique can be made to have very little training time overhead.

Finally, subsequent work of [Gow+18] shows how applying interval bound propagation during training, combined with MILP-based exact verification, can lead to provably robust networks.

[3]These works use both "verification" and "certification" to describe their methods. For clarity, we use "certification" to describe their approaches, while we use "verification" to describe *exact* verification approaches. For a more detailed discussion of the differences, see Appendix E.6.

6.3 Training verifiable ML models

We begin by discussing the task of verifying a network and identify two key properties of networks that lead to improved verification speed: weight sparsity and so-called ReLU stability. We then use natural regularization methods for inducing weight sparsity as well as a new regularization method for inducing ReLU stability. Finally, we demonstrate that these methods significantly speed up verification while maintaining state-of-the-art accuracy.

6.3.1 Verifying adversarial robustness of ML models

Deep neural network models. Our focus will be on one of the most common architectures for state-of-the-art models: k-layer fully-connected feed-forward DNN classifiers with ReLU non-linearities[4]. Such models can be viewed as a function $f(\cdot, W, b)$, where W and b represent the weight matrices and biases of each layer. For an input x, the output $f(x, W, b)$ of the DNN is defined as:

$$z_0 = x \tag{6.1}$$

$$\hat{z}_i = z_{i-1}W_i + b_i \quad \text{for } i = 1, 2, \ldots, k-1 \tag{6.2}$$

$$z_i = \max(\hat{z}_i, 0) \quad \text{for } i = 1, 2, \ldots, k-2 \tag{6.3}$$

$$f(x, W, b) = \hat{z}_{k-1} \tag{6.4}$$

Here, for each layer i, we let $\hat{z}_{ij}$ denote the j^{th} ReLU pre-activation and let $\hat{z}_{ij}(x)$ denote the value of $\hat{z}_{ij}$ on an input x. $\hat{z}_{k-1}$ is the final, output layer with an output unit for each possible label (the logits). The network will make predictions by selecting the label with the largest logit.

Adversarial robustness. For a network to be reliable, it should make predictions that are robust – that is, it should predict the same output for inputs that are very similar. Specifically, we want the DNN classifier's predictions to be robust to a set $\text{Adv}(x)$ of possible adversarial perturbations of an input x. We focus on ℓ_∞ norm-bound adversarial perturbations, where $\text{Adv}(x) = \{x' : ||x' - x||_\infty \leq \epsilon\}$ for some constant ϵ, since it is the most common one considered in adversarial robustness and verification literature (thus, it currently constitutes a canonical benchmark). Even so, our methods can be applied to other ℓ_p norms and broader sets of perturbations.

[4]Note that this encompasses common convolutional network architectures because every convolutional layer can be replaced by an equivalent fully-connected layer.

Verification of network models. For an input x with correct label y, a perturbed input x' can cause a misclassification if it makes the logit of some incorrect label $\hat{y}$ larger than the logit of y on x'. We can thus express the task of finding an adversarial perturbation as the optimization problem:

$$\begin{aligned} &\min_{x',\hat{y}} f(x', W)_y - f(x', W)_{\hat{y}} \\ &\text{subject to} \quad x' \in \text{Adv}(x) \end{aligned}$$

An adversarial perturbation exists if and only if the objective of the optimization problem is negative.

Adversarial accuracies. We define the *true adversarial accuracy* of a model to be the fraction of test set inputs for which the model is robust to all allowed perturbations. By definition, evaluations against specific adversarial attacks like PGD or FGSM provide an upper bound to this accuracy, while certification methods provide lower bounds. Given sufficient time for each input, an exact verifier can prove robustness to perturbations, or find a perturbation where the network makes a misclassification on the input, and thus exactly determine the true adversarial accuracy. This is in contrast to certification methods, which solve a relaxation of the verification problem and can not exactly determine the true adversarial accuracy no matter how much time they have.

However, such exact verification may take impractically long for certain inputs, so we instead compute the *provable adversarial accuracy*, which we define as the fraction of test set inputs for which the verifier can prove robustness to perturbations within an allocated time budget (timeout). Similarly to certifiable accuracy, this accuracy constitutes a lower bound on the true adversarial accuracy. A model can thus, e.g., have high true adversarial accuracy and low provable adversarial accuracy if verification of the model is too slow and often fails to complete before timeout.

Also, in our evaluations, we chose to use the MILP exact verifier of [TXT19] when performing experiments, as it is both open source and the fastest verifier we are aware of.

6.3.2 Weight sparsity and its impact on verification speed

The first property of network models that we want to improve in order to speed up exact verification of those models is weight sparsity. Weight sparsity is important for verification speed because many exact verifiers rely on solving LP or MILP systems,

which benefit from having fewer variables. We use two natural regularization methods for improving weight sparsity: ℓ_1-regularization and small weight pruning. These techniques significantly improve verification speed – see Table 6.1. Verifying even small MNIST networks is almost completely intractable without them. Specifically, the verifier can only prove robustness of an adversarially-trained model on 19% of inputs with a one hour budget per input, while the verifier can prove robustness of an adversarially-trained model with ℓ_1-regularization and small weight pruning on 89.13% of inputs with a 120 second budget per input.

Interestingly, even though adversarial training improves weight sparsity (see Appendix E.2) it was still necessary to use ℓ_1-regularization and small weight pruning. These techniques further promoted weight sparsity and gave rise to networks that were much easier to verify.

Dataset	Epsilon		Training Method	Test Set Accuracy	Provable Adversarial Accuracy	Average Solve Time (s)
MNIST	$\epsilon = 0.1$	1	Adversarial Training	99.17%	19.00%	2970.43
		2	+ℓ_1-Regularization	99.00%	82.17%	21.99
		3	+Small Weight Pruning	98.99%	89.13%	11.71
		4	+ReLU Pruning (control)	98.94%	91.58%	6.43

Table 6.1: Improvement in provable adversarial accuracy and verification solve times when incrementally adding natural regularization methods for improving weight sparsity and ReLU stability into the model training procedure, before verification occurs. Each row represents the addition of another method – for example, Row 3 uses adversarial training, ℓ_1-regularization, and small weight pruning. Row 4 adds ReLU pruning (see Appendix E.1). Row 4 is the control model for MNIST and $\epsilon = 0.1$ that we present again in comparisons in Tables 6.2 and 6.3. We use a 3600 instead of 120 second timeout for Row 1 and only verified the first 100 images (out of 10000) because verifying it took too long.

6.3.3 ReLU stability

Next, we target the primary speed bottleneck of exact verification: the number of ReLUs the verifier has to branch on. In our paper, this corresponds to the notion of inducing ReLU stability. Before we describe our methodology, we formally define ReLU stability.

Given an input x, a set of allowed perturbations $\text{Adv}(x)$, and a ReLU, exact verifiers may need to branch based on the possible pre-activations of the ReLU, namely $\hat{z}_{ij}(\text{Adv}(x)) = \{\hat{z}_{ij}(x') : x' \in \text{Adv}(x)\}$ (cf. (6.2)). If there exist two perturbations

x', x'' in the set $\text{Adv}(x)$ such that $\text{sign}(\hat{z}_{ij}(x')) \neq \text{sign}(\hat{z}_{ij}(x''))$, then the verifier has to consider that for some perturbed inputs the ReLU is active ($z_{ij} = \hat{z}_{ij}$) and for other perturbed inputs inactive ($z_{ij} = 0$). The more such cases the verifier faces, the more branches it has to consider, causing the complexity of verification to increase exponentially. Intuitively, a model with 1000 ReLUs among which only 100 ReLUs require branching will likely be much easier to verify than a model with 200 ReLUs that all require branching. Thus, it is advantageous for the verifier if, on an input x with allowed perturbation set $\text{Adv}(x)$, the number of ReLUs such that

$$\text{sign}(\hat{z}_{ij}(x')) = \text{sign}(\hat{z}_{ij}(x)) \quad \forall x' \in \text{Adv}(x) \tag{6.5}$$

is maximized. We call a ReLU for which (6.5) holds on an input x a *stable ReLU* on that input. If (6.5) does not hold, then the ReLU is an *unstable ReLU*.

Directly computing whether a ReLU is stable on a given input x is difficult because doing so would involve considering all possible values of $\hat{z}_{ij}(\text{Adv}(x))$. Instead, exact verifiers compute an upper bound $\hat{u}_{ij}$ and a lower bound $\hat{l}_{ij}$ of $\hat{z}_{ij}(\text{Adv}(x))$. If $0 \leq \hat{l}_{ij}$ or $\hat{u}_{ij} \leq 0$, they can replace the ReLU with the identity function or the zero function, respectively. Otherwise, if $\hat{l}_{ij} < 0 < \hat{u}_{ij}$, these verifiers then determine that they need to "branch" on that ReLU. Thus, we can rephrase (6.5) as

$$\text{sign}(\hat{u}_{ij}) = \text{sign}(\hat{l}_{ij}) \tag{6.6}$$

We will discuss methods for determining these upper and lower bounds $\hat{u}_{ij}$, $\hat{l}_{ij}$ in Section 6.3.3.

A regularization technique for inducing ReLU stability: RS Loss

As we see from equation (6.6), a function that would indicate exactly when a ReLU is stable is $F^*(\hat{u}_{ij}, \hat{l}_{ij}) = \text{sign}(\hat{u}_{ij}) \cdot \text{sign}(\hat{l}_{ij})$. Thus, it would be natural to use this function as a regularizer. However, this function is non-differentiable and if used in training a model, would provide no useful gradients during back-propagation. Thus, we use the following smooth approximation of F^* (see Fig. 6-1) which provides the desired gradients:

$$F(\hat{u}_{ij}, \hat{l}_{ij}) = -\tanh(1 + \hat{u}_{ij} \cdot \hat{l}_{ij})$$

We call the corresponding objective RS Loss, and show in Fig. 6-2a that using this loss function as a regularizer effectively decreases the number of unstable ReLUs. RS Loss thus encourages *ReLU stability*, which, in turn, speeds up exact verification -

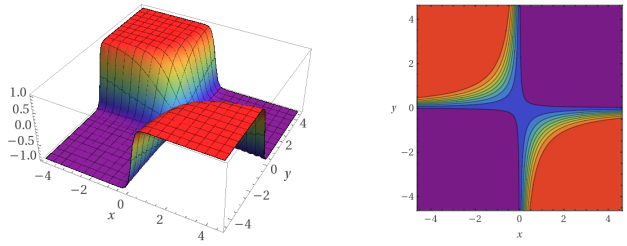


Figure 6-1: Plot and contour plot of the function $F(x, y) = -\tanh(1 + x \cdot y)$

see Fig. 6-2b.

Estimating ReLU upper and lower bounds on activations

A key aspect of using RS Loss is determining the upper and lower bounds $\hat{u}_{ij}, \hat{l}_{ij}$ for each ReLU (cf. (6.6)). The bounds for the inputs z_0 (cf. (6.1)) are simple – for the input x, we know $x - \epsilon \leq z_0 \leq x + \epsilon$, so $\hat{u}_0 = x - \epsilon$, $\hat{l}_0 = x + \epsilon$. For all subsequent z_{ij}, we estimate bounds using either the naive interval arithmetic (IA) approach described in [TXT19] or an improved version of it. The improved version is a tighter estimate but uses more memory and training time, and thus is most effective on smaller networks. We present the details of naive IA and improved IA in Appendix E.3.

Interval arithmetic approaches can be implemented relatively efficiently and work well with back-propagation because they only involve matrix multiplications. This contrasts with how exact verifiers compute these bounds, which usually involves solving LPs or MILPs. Interval arithmetic also overestimates the number of unstable ReLUs. This means that minimizing unstable ReLUs based on IA bounds will provide an upper bound on the number of unstable ReLUs determined by exact verifiers. In particular, IA will properly penalize every unstable ReLU.

Improved IA performs well in practice, overestimating the number of unstable ReLUs by less than 0.4% in the first 2 layers of MNIST models and by less than 36.8% (compared to 128.5% for naive IA) in the 3rd layer. Full experimental results are available in Table E.1 of Appendix E.3.3.

Impact of ReLU stability improvements on verification speed

We provide experimental evidence that RS Loss regularization improves ReLU stability and speeds up average verification times by more than an order of magnitude in Fig. 6-2b. To isolate the effect of RS Loss, we compare MNIST models trained in exactly the same way other than the weight on RS Loss. When comparing a network trained

with a RS Loss weight of 5e−4 to a network with a RS Loss weight of 0, the former has just 16% as many unstable ReLUs and can be verified 65x faster. The caveat here is that the former has 1.00% lower test set accuracy.

We also compare verification speed with and without RS Loss on MNIST networks for different values of ϵ (0.1, 0.2, and 0.3) in Fig. 6-2c. We choose RS Loss weights that cause almost no test set accuracy loss (less than 0.50% - See Table 6.3) in these cases, and we still observe a 4–13x speedup from RS Loss. For CIFAR, RS Loss gives a smaller speedup of 1.6–3.7x (See Appendix E.5).

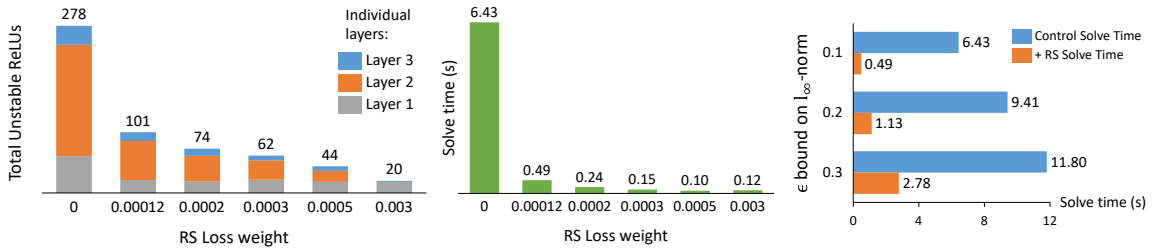


Figure 6-2: (a) Average number of unstable ReLUs by layer and (b) average verification solve times of 6 networks trained with different weights on RS Loss for MNIST and $\epsilon = 0.1$. Averages are taken over all 10000 MNIST test set inputs. Both metrics improve significantly with increasing RS Loss weight. An RS Loss weight of 0 corresponds to the control network, while an RS Loss weight of 0.00012 corresponds to the "+RS" network for MNIST and $\epsilon = 0.1$ in Tables 6.2 and 6.3. (c) Improvement in the average time taken by a verifier to solve the verification problem after adding RS Loss to the training procedure, for different ϵ on MNIST. The weight on RS Loss was chosen so that the "+RS" models have test set accuracies within 0.50% of the control models.

Impact of ReLU Stability improvements on provable adversarial accuracy

As the weight on the RS Loss used in training a model increases, the ReLU stability of the model will improve, speeding up verification and likely improving provable adversarial accuracy. However, like most regularization methods, placing too much weight on RS Loss can decrease the model capacity, potentially lowering both the true adversarial accuracy and the provable adversarial accuracy. Therefore, it is important to choose the weight on RS Loss carefully to obtain both high provable adversarial accuracy and faster verification speeds.

To show the effectiveness of RS Loss in improving provable adversarial accuracy, we train two networks for each dataset and each value of ϵ. One is a "*control*"

network that uses all of the natural improvements for inducing both weight sparsity (ℓ_1-regularization and small weight pruning) and ReLU stability (ReLU pruning - see Appendix E.1). The second is a "*+RS*" network that uses RS Loss in addition to all of the same natural improvements. This lets us isolate the incremental effect of adding RS Loss to the training procedure.

In addition to attaining a 4–13x speedup in MNIST verification times (see Fig. 6-2c), we achieve higher provable adversarial accuracy in every single setting when using RS Loss. This is especially notable for the hardest verification problem we tackle – proving robustness to perturbations with ℓ_∞ norm-bound 8/255 on CIFAR-10 – where adding RS Loss nearly triples the provable adversarial accuracy from 7.09% to 20.27%. This improvement is primarily due to verification speedup induced by RS Loss, which allows the verifier to finish proving robustness for far more inputs within the 120 second time limit. These results are shown in Table 6.2.

Table 6.2: Provable Adversarial Accuracies for the control and "+RS" networks in each setting.

	MNIST, $\epsilon = 0.1$	MNIST, $\epsilon = 0.2$	MNIST, $\epsilon = 0.3$	CIFAR, $\epsilon = 2/255$	CIFAR, $\epsilon = 8/255$
Control	91.58	86.45	77.99	45.53	7.09
+RS	94.33	89.79	80.68	45.93	20.27

6.4 Experiments

6.4.1 Experiments on MNIST and CIFAR

In addition to the experimental results already presented, we compare our control and "+RS" networks with the best available results presented in the state-of-the-art certifiable defenses of [Won+18], [Dvi+18b], and [MGV18] in Table 6.3. We compare their test set accuracy, PGD adversarial accuracy (an evaluation of robustness against a strong 40-step PGD adversarial attack), and provable adversarial accuracy. Additionally, to show that our method can scale to larger architectures, we train and verify a "*+RS (Large)*" network for each dataset and ϵ.

In terms of provable adversarial accuracies, on MNIST, our results are significantly better than those of [Won+18] for larger perturbations of $\epsilon = 0.3$, and comparable for $\epsilon = 0.1$. On CIFAR-10, our method is slightly less effective, perhaps indicating that more unstable ReLUs are necessary to properly learn a robust CIFAR classifier. We also experienced many more instances of the verifier reaching its allotted 120

second time limit on CIFAR, especially for the less ReLU stable control networks. Full experimental details for each model in Tables 6.1, 6.2, and 6.3, including a breakdown of verification solve results (how many images did the verifier **A.** prove robust **B.** find an adversarial example for **C.** time out on), are available in Appendix E.5.

Table 6.3: Comparison of test set, PGD adversarial, and provable adversarial accuracy of networks trained with and without RS Loss. We also provide the best available certifiable adversarial and PGD adversarial accuracy of any single models from [Won+18], [Dvi+18b], and [MGV18] for comparison, and highlight the best provable accuracy for each ϵ.
* The provable adversarial accuracy for "+RS (Large)" is only computed for the first 1000 images because the verifier performs more slowly on larger models.
** [Dvi+18b; MGV18] use a slightly smaller $\epsilon = 0.03 = 7.65/255$.
† [MGV18] computes results over 500 images instead of all 10000.
†† [MGV18] uses a slightly smaller $\epsilon = 0.007 = 1.785/255$.

Dataset	Epsilon	Training Method	Test Set Accuracy	PGD Adversarial Accuracy	Provable/Certifiable Adversarial Accuracy
MNIST	$\epsilon = 0.1$	Control	98.94%	95.12%	91.58%
		+RS	98.68%	95.13%	94.33%
		+RS (Large)*	98.95%	96.58%	95.60%
		Wong et al.	98.92%	-	96.33%
		Dvijotham et al.	98.80%	97.13%	95.56%
		Mirman et al.†	99.00%	97.60%	**96.60**%
MNIST	$\epsilon = 0.2$	Control	98.40%	93.14%	86.45%
		+RS	98.10%	93.14%	**89.79%**
		+RS (Large)*	98.21%	94.19%	89.10%
MNIST	$\epsilon = 0.3$	Control	97.75%	91.64%	77.99%
		+RS	97.33%	92.05%	80.68%
		+RS (Large)*	97.54%	93.25%	59.60%
		Wong et al.	85.13%	-	56.90%
		Mirman et al.†	96.60%	93.80%	**82.00**%
CIFAR	$\epsilon = 2/255$	Control	64.64%	51.58%	45.53%
		+RS	61.12%	49.92%	45.93%
		+RS (Large)*	61.41%	50.61%	41.40%
		Wong et al.	68.28%	-	**53.89%**
		Mirman et al.†,††	62.00%	54.60%	52.20%
CIFAR	$\epsilon = 8/255$	Control	50.69%	31.28%	7.09%
		+RS	40.45%	26.78%	20.27%
		+RS (Large)*	42.81%	28.69%	19.80%
		Wong et al.	28.67%	-	21.78%
		Dvijotham et al.**	48.64%	32.72%	26.67%
		Mirman et al.†, **	54.20%	40.00%	**35.20**%

6.4.2 Experimental methods and details

In our experiments, we use robust adversarial training [GSS15] against a strong adversary as done in [Mad+18] to train various DNN classifiers. For each setting of dataset (MNIST or CIFAR) and ϵ, we find a suitable weight on RS Loss via line search (See Table E.3 in Appendix E.4). The same weight is used for each ReLU. During training, we used improved IA for ReLU bound estimation for "+RS" models and use naive IA for "+RS (Large)" models because of memory constraints. For ease of comparison, we trained our networks using the same convolutional DNN architecture as in [Won+18]. This architecture uses two 2x2 strided convolutions with 16 and 32 filters, followed by a 100 hidden unit fully connected layer. For the larger architecture, we also use the same "large" architecture as in [Won+18]. It has 4 convolutional layers with 32, 32, 64, and 64 filters, followed by 2 fully connected layers with 512 hidden units each.

For verification, we used the most up-to-date version of the exact verifier from [TXT19]. Model solves were parallelized over 8 CPU cores. When verifying an image, the verifier of [TXT19] first builds a model, and second, solves the verification problem (See Appendix E.4.2 for details). We focus on reporting solve times because that is directly related to the task of verification itself. All build times for the control and "+RS" models on MNIST that we presented were between 4 and 10 seconds, and full results on build times are also presented in Appendix E.5.

Additional details on our experimental setup (e.g. hyperparameters) can be found in Appendix E.4.

6.5 Conclusion

In this paper, we use the principle of co-design to develop training methods that emphasize verification as a goal, and we show that they make verifying the trained model much faster. We first demonstrate that natural regularization methods already make the exact verification problem significantly more tractable. Subsequently, we introduce the notion of ReLU stability for networks, present a method that improves a network's ReLU stability, and show that this improvement makes verification an additional 4–13x faster. Our method is universal, as it can be added to any training procedure and should speed up any exact verification procedure, especially MILP-based methods.

Prior to our work, exact verification seemed intractable for all but the smallest

models. Thus, our work shows progress toward reliable models that can be proven to be robust, and our techniques can help scale verification to even larger networks.

Many of our methods appear to compress our networks into more compact, simpler forms. We hypothesize that the reason that regularization methods like RS Loss can still achieve very high accuracy is that most models are overparametrized in the first place. There exist clear parallels between our methods and techniques in model compression [HMD16; Che+17] – therefore, we believe that drawing upon additional techniques from model compression can further improve the ease-of-verification of networks. We also expect that there exist objectives other than weight sparsity and ReLU stability that are important for verification speed. If so, further exploring the principle of co-design for those objectives is an interesting future direction.

Bibliography

[AAP05] A. Angelova, Y. Abu-Mostafam, and P. Perona. "Pruning training sets for learning of object categories". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005.

[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.

[Ada11] Andrew Y. Ng Adam Coates Honglak Lee. "An Analysis of Single Layer Networks in Unsupervised Feature Learning". In: *AISTATS*. 2011.

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

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

[AJ18] David Alvarez-Melis and Tommi S Jaakkola. "On the robustness of interpretability methods". In: *arXiv preprint arXiv:1806.08049* (2018).

[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.

[AS17] Anish Athalye and Ilya Sutskever. "Synthesizing robust adversarial examples". In: *arXiv preprint arXiv:1707.07397* (2017).

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

[Bak+18] Nicholas Baker et al. "Deep convolutional networks do not classify based on global object shape." In: *PLOS Computational Biology*. 2018.

[Bar+18] Francesco Barbieri et al. "Semeval 2018 task 2: Multilingual emoji prediction". In: *Proceedings of The 12th International Workshop on Semantic Evaluation*. 2018, pp. 24–33.

[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*. 2019.

[Bar+20] Francesco Barbieri et al. “TweetEval:Unified Benchmark and Comparative Evaluation for Tweet Classification”. In: *Proceedings of Findings of EMNLP*. 2020.

[Bas+16] Osbert Bastani et al. “Measuring neural net robustness with constraints”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2613–2621.

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

[Bea+16] Charles Beattie et al. “Deepmind lab”. In: *arXiv preprint arXiv:1612.03801* (2016).

[Bee+20] Sara Beery et al. “Synthetic examples improve generalization for rare classes”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 863–873.

[Beh+20] Harkirat Singh Behl et al. “Autosimulate: (quickly) learning synthetic data generation”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 255–271.

[Ben+21] Emily M Bender et al. “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 2021, pp. 610–623.

[Ber+19] Christopher Berner et al. “Dota 2 with Large Scale Deep Reinforcement Learning”. In: (2019). URL: `https://arxiv.org/abs/1912.06680`.

[Ber82] Dimitri P Bertsekas. “Projected Newton methods for optimization problems with simple constraints”. In: *SIAM Journal on control and Optimization* 20.2 (1982), pp. 221–246.

[Bez+17] Jeff Bezanson et al. “Julia: A fresh approach to numerical computing”. In: *SIAM Review* 59.1 (2017), pp. 65–98.

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

[BHP18] Sara Beery, Grant van Horn, and Pietro Perona. “Recognition in Terra Incognita”. In: *European Conference on Computer Vision (ECCV)*. 2018.

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

[Ble20] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2020. URL: `http://www.blender.org`.

[BM02] Ernesto G Birgin and José Mario Martínez. “Large-scale active-set box-constrained optimization method with spectral projected gradients”. In: *Computational Optimization and Applications* 23.1 (2002), pp. 101–125.

[Bom+21] Rishi Bommasani et al. "On the Opportunities and Risks of Foundation Models". In: *arXiv preprint arXiv:2108.07258* (2021).

[Brá+17] Carmo P Brás et al. "A block active set algorithm with spectral choice line search for the symmetric eigenvalue complementarity problem". In: *Applied Mathematics and Computation* 294 (2017), pp. 36–48.

[Bro+16] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[Bro+18] Tom B. Brown et al. *Adversarial Patch*. 2018. arXiv: 1712.09665 [cs.CV].

[Bro+20] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[Bun+17] Rudy Bunel et al. "Piecewise Linear Neural Network verification: A comparative study". In: *arXiv preprint arXiv:1711.00455* (2017).

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Cal+15] Berk Calli et al. "Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols". In: *arXiv preprint arXiv:1502.03143* (2015).

[Car+17] Nicholas Carlini et al. "Ground-Truth Adversarial Examples". In: *arXiv preprint arXiv:1709.10207* (2017).

[Car+19] Mathilde Caron et al. "Unsupervised pre-training of image features on non-curated data". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2959–2968.

[Cha+20] Emile Chapuis et al. "Hierarchical Pre-training for Sequence Labelling in Spoken Dialog". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 2636–2648. DOI: 10.18653/v1/2020.findings-emnlp.239. URL: https://www.aclweb.org/anthology/2020.findings-emnlp.239.

[Che+17] Yu Cheng et al. "A Survey of Model Compression and Acceleration for Deep Neural Networks." In: *CoRR* abs/1710.09282 (2017).

[Che+18] Pin-Yu Chen et al. "EAD: elastic-net attacks to deep neural networks via adversarial examples". In: *AAAI Conference on Artificial Intelligence*. 2018.

[Cho+22] Aakanksha Chowdhery et al. *PaLM: Scaling Language Modeling with Pathways*. 2022. DOI: 10.48550/ARXIV.2204.02311. URL: https://arxiv.org/abs/2204.02311.

[CNR17] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. "Maximum resilience of artificial neural networks". In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2017, pp. 251–268.

[CPT04] Antonio Criminisi, Patrick Pérez, and Kentaro Toyama. "Region Filling and Object Removal by Exemplar-Based Image Inpainting". In: *IEEE Transactions on Image Processing*. 2004.

[Cri+17] Andrea Cristofari et al. "A two-stage active-set algorithm for bound-constrained optimization". In: *Journal of Optimization Theory and Applications* 172.2 (2017), pp. 369–401.

[Cri+20] Andrea Cristofari et al. "An active-set algorithmic framework for non-convex optimization problems over the simplex". In: *Computational Optimization and Applications* 77 (2020), pp. 57–89.

[Cub+19] Ekin D Cubuk et al. "Randaugment: Practical data augmentation with no separate search". In: *arXiv preprint arXiv:1909.13719* 2.4 (2019), p. 7.

[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.

[CW17b] Nicholas Carlini and David Wagner. "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 2017, pp. 3–14.

[CW17c] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE. 2017, pp. 39–57.

[Den+19] Maximilian Denninger et al. "BlenderProc". In: *arXiv preprint arXiv:1911.01911* (2019).

[Dev+19] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: (2019).

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

[DHL17] Iain Dunning, Joey Huchette, and Miles Lubin. "JuMP: A Modeling Language for Mathematical Optimization". In: *SIAM Review* 59.2 (2017), pp. 295–320. DOI: 10.1137/15M1020575.

[DKF20a] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. "Meta-Sim2: Learning to Generate Synthetic Datasets". In: *ECCV*. 2020.

[DKF20b] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. "Meta-Sim2: Unsupervised Learning of Scene Structure for Synthetic Data Generation". In: *European Conference on Computer Vision*. Springer. 2020, pp. 715–733.

[Dos+17] Alexey Dosovitskiy et al. "CARLA: An open urban driving simulator". In: *arXiv preprint arXiv:1711.03938* (2017).

[DRG15] Gintare Karolina Dziugaite, Daniel M. Roy, and Zoubin Ghahramani. "Training generative neural networks via Maximum Mean Discrepancy optimization". In: *Uncertainty in Artificial Intelligence (UAI)*. 2015.

[Dut+18] Souradeep Dutta et al. "Output Range Analysis for Deep Feedforward Neural Networks". In: *NASA Formal Methods Symposium*. Springer. 2018, pp. 121–138.

[Dvi+18a] Krishnamurthy Dvijotham et al. "A Dual Approach to Scalable Verification of Deep Networks". In: *Conference on Uncertainty in Artificial Intelligence*. 2018.

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

[Ehl17] Ruediger Ehlers. "Formal verification of piece-wise linear feed-forward neural networks". In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2017, pp. 269–286.

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

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

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

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

[Est+17] Andre Esteva et al. "Dermatologist-level classification of skin cancer with deep neural networks". In: *nature* 542.7639 (2017), pp. 115–118.

[Evt+17] Ivan Evtimov et al. "Robust Physical-World Attacks on Machine Learning Models". In: *CoRR* abs/1707.08945 (2017). arXiv: `1707.08945`. URL: `http://arxiv.org/abs/1707.08945`.

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

[Far+21] Abolfazl Farahani et al. "A brief review of domain adaptation". In: *Advances in Data Science and Information Engineering* (2021), pp. 877–894.

[FFK98] Francisco Facchinei, Andreas Fischer, and Christian Kanzow. "On the accurate identification of active constraints". In: *SIAM Journal on Optimization* 9.1 (1998), pp. 14–32.

[Fis+17] Volker Fischer et al. "Adversarial examples for semantic image segmentation". In: *Arxiv preprint arXiv:1703.01101*. 2017.

[FJ18] Matteo Fischetti and Jason Jo. "Deep Neural Networks and Mixed Integer Linear Optimization". In: *Constraints* 23.3 (July 2018), pp. 296–309. ISSN: 1383-7133. DOI: 10.1007/s10601-018-9285-6. URL: https://doi.org/10.1007/s10601-018-9285-6.

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

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

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

[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.

[Geh+18] Timon Gehr et al. "Ai 2: Safety and robustness certification of neural networks with abstract interpretation". In: *Security and Privacy (SP), 2018 IEEE Symposium on*. 2018.

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

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

[Gei+20] Robert Geirhos et al. "Shortcut Learning in Deep Neural Networks". In: *arXiv preprint arXiv:2004.07780*. 2020.

[Gow+18] Sven Gowal et al. "On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models". In: *arXiv preprint arXiv:1810.12715* (2018).

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

[Gre+12] Arthur Gretton et al. "A Kernel Two-Sample Test". In: *Journal of Machine Learning Research (JMLR)* 13.25 (2012), pp. 723–773. URL: http://jmlr.org/papers/v13/gretton12a.html.

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

[Gur17a] Gurobi. *Gurobi Guidelines for Numerical Issues*. 2017. URL: http://files.gurobi.com/Numerics.pdf.

[Gur17b] Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2017. URL: http://www.gurobi.com.

[HA17] Matthias Hein and Maksym Andriushchenko. "Formal guarantees on the robustness of a classifier against adversarial manipulation". In: *Advances in Neural Information Processing Systems*. 2017, pp. 2263–2273.

[Has21] Tatsunori Hashimoto. "Model Performance Scaling with Multiple Data Sources". In: *International Conference on Machine Learning (ICML)*. 2021.

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

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

[He+17a] Kaiming He et al. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[He+17b] Xiangnan He et al. "Neural Collaborative Filtering". In: *Proceedings of the 26th International Conference on World Wide Web (WWW)*. 2017.

[Hei+20] Christoph Heindl et al. "BlendTorch: A Real-Time, Adaptive Domain Randomization Library". In: *arXiv preprint arXiv:2010.11696* (2020).

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

[Hen+20] Dan Hendrycks et al. *The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization*. 2020. arXiv: 2006.16241 [cs.CV].

[HG19] Abdullah Hamdi and Bernard Ghanem. "Towards Analyzing Semantic Robustness of Deep Neural Networks". In: *arXiv preprint arXiv:1904.04621* (2019).

[HGW01] Michael Harville, Gaile Gordon, and John Woodfill. "Foreground segmentation using adaptive mixture models in color and depth". In: *IEEE Workshop on Detection and Recognition of Events in Video*. 2001.

[Hin+12] G. Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. ISSN: 1053-5888. DOI: 10.1109/MSP.2012.2205597.

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

[HMG18] Abdullah Hamdi, Matthias Muller, and Bernard Ghanem. "SADA: Semantic Adversarial Diagnostic Attacks for Autonomous Applications". In: *arXiv preprint arXiv:1812.02132* (2018).

[Hua+17a] Sandy Huang et al. “Adversarial Attacks on Neural Network Policies”. In: *ArXiv preprint arXiv:1702.02284*. 2017.

[Hua+17b] Xiaowei Huang et al. “Safety verification of deep neural networks”. In: *International Conference on Computer Aided Verification*. Springer. 2017, pp. 3–29.

[HV17] Joey Huchette and Juan Pablo Vielma. “Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools”. In: *arXiv preprint arXiv:1708.00050* (2017).

[HZ06] William W Hager and Hongchao Zhang. “A new active set algorithm for box constrained optimization”. In: *SIAM Journal on Optimization* 17.2 (2006), pp. 526–557.

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

[IS15] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015.

[Jac+19] Jorn-Henrik Jacobsen et al. “Excessive Invariance Causes Adversarial Vulnerability”. In: *International Contemporary on Learning Representations (ICLR)*. 2019.

[Jai+20] Lakshya Jain et al. “Analyzing and Improving Neural Networks by Generating Semantic Counterexamples through Differentiable Rendering”. In: *arXiv preprint arXiv:1910.00727* (2020).

[JLT18] Saumya Jetley, Nicholas Lord, and Philip Torr. “With friends like these, who needs adversaries?” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[Jul+20] Arthur Juliani et al. *Unity: A General Platform for Intelligent Agents*. 2020. arXiv: `1809.02627 [cs.LG]`.

[Jum+21] John Jumper et al. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature*. 2021.

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

[Kap+20] Jared Kaplan et al. “Scaling Laws for Neural Language Models”. In: *Arxiv preprint arXiv:2001.08361*. 2020.

[Kar+19a] Amlan Kar et al. “Meta-Sim: Learning to Generate Synthetic Datasets”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[Kar+19b] Amlan Kar et al. “Meta-sim: Learning to generate synthetic datasets”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 4551–4560.

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

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. 2015.

[KFS18] Jernej Kos, Ian Fischer, and Dawn Song. “Adversarial examples for generative models”. In: *IEEE Security and Privacy Workshops (SPW)*. 2018.

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

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

[KKG18] Harini Kannan, Alexey Kurakin, and Ian J. Goodfellow. “Adversarial Logit Pairing”. In: *CoRR* abs/1803.06373 (2018). arXiv: 1803.06373. URL: http://arxiv.org/abs/1803.06373.

[KMF18] Can Kanbak, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. “Geometric robustness of deep networks: analysis and improvement”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

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

[Kol+17] Eric Kolve et al. “Ai2-thor: An interactive 3d environment for visual ai”. In: *arXiv preprint arXiv:1712.05474* (2017).

[Kri09] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. 2009.

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

[KUH18] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. “Neural 3D Mesh Renderer”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[LDJ19] Shikun Liu, Andrew J Davison, and Edward Johns. “Self-supervised generalisation with meta auxiliary learning”. In: *arXiv preprint arXiv:1901.08933* (2019).

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

[Lec+21b] Guillaume Leclerc et al. “3DB: A Framework for Debugging Computer Vision Models”. In: *Arxiv preprint arXiv:2106.03805*. 2021.

[Li+18] Tzu-Mao Li et al. "Differentiable Monte Carlo Ray Tracing through Edge Sampling". In: *SIGGRAPH Asia 2018 Technical Papers*. 2018.

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

[Liu+19] Hsueh-Ti Derek Liu et al. "Beyond Pixel Norm-Balls: Parametric Adversaries Using An Analytically Differentiable Renderer". In: *International Conference on Learning Representations (ICLR)*. 2019.

[LL21] Xiang Lisa Li and Percy Liang. *Prefix-Tuning: Optimizing Continuous Prompts for Generation*. 2021. arXiv: 2101.00190 [cs.CL].

[LM17a] Alessio Lomuscio and Lalit Maganti. "An approach to reachability analysis for feed-forward ReLU neural networks". In: *arXiv preprint arXiv:1706.07351* (2017).

[LM17b] Alessio Lomuscio and Lalit Maganti. "An approach to reachability analysis for feed-forward ReLU neural networks". In: *CoRR* abs/1706.07351 (2017). arXiv: 1706.07351. URL: http://arxiv.org/abs/1706.07351.

[LR17] Sharon L Lohr and Trivellore E Raghunathan. "Combining survey data with other data sources". In: *Statistical Science* 32.2 (2017), pp. 293–312.

[LSK19] Juncheng Li, Frank R. Schmidt, and J. Zico Kolter. "Adversarial camera stickers: A physical camera-based attack on deep learning systems". In: *Arxiv preprint arXiv:1904.00759*. 2019.

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

[Mah+18] Dhruv Mahajan et al. "Exploring the Limits of Weakly Supervised Pretraining". In: *European Conference on Computer Vision (ECCV)*. 2018.

[Mår+20] Gustav Mårtensson et al. "The reliability of a deep learning model in clinical out-of-distribution MRI data: a multicohort study". In: *Medical Image Analysis* 66 (2020), p. 101714.

[MGV18] Matthew Mirman, Timon Gehr, and Martin Vechev. "Differentiable Abstract Interpretation for Provably Robust Neural Networks". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 3575–3583. URL: http://proceedings.mlr.press/v80/mirman18b.html.

[MH21] Samuel G Müller and Frank Hutter. "Trivialaugment: Tuning-free yet state-of-the-art data augmentation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 774–782.

[Mil95] George Miller. "WordNet: a lexical database for English". In: *Communications of the ACM*. 1995.

[MKC09] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*. SIAM, 2009.

[Moh+18] Saif Mohammad et al. "Semeval-2018 task 1: Affect in tweets". In: *Proceedings of the 12th international workshop on semantic evaluation*. 2018, pp. 1–17.

[Mor+18] Taylor Mordan et al. "Revisiting Multi-Task Learning with ROCK: a Deep Residual Auxiliary Block for Visual Detection". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[MPR16] Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. "The benefit of multitask representation learning". In: *Journal of Machine Learning Research* 17.81 (2016), pp. 1–32.

[MRS10] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. "Introduction to information retrieval". In: *Natural Language Engineering* 16.1 (2010), pp. 100–103.

[MTW12] Choi Myung Jin, Antonio Torralba, and Alan S. Willsky. "Context models and out-of-context objects". In: *Pattern Recognition Letters*. 2012.

[Ngu+21] Timothy Nguyen et al. "Dataset Distillation with Infinitely Wide Convolutional Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[Pap+16] Nicolas Papernot et al. "Distillation as a defense to adversarial perturbations against deep neural networks". In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 582–597.

[Par+12] Omkar M Parkhi et al. "Cats and Dogs". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[Pea10] Beth Pearsall. "Predictive Policing: The Future of Law Enforcement?" In: 2010.

[Pen+19] Xingchao Peng et al. "Moment matching for multi-source domain adaptation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1406–1415.

[PMP20] Đorđe Petrović, Radomir Mijailović, and Dalibor Pešić. "Traffic Accidents with Autonomous Vehicles: Type of Collisions, Manoeuvres and Errors of Conventional Vehicles' Drivers". In: *Transportation Research Procedia* 45 (2020). Transport Infrastructure and systems in a changing world. Towards a more sustainable, reliable and smarter mobility. TIS Roma 2019 Conference Proceedings, pp. 161–168. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2020.03.003. URL: https://www.sciencedirect.com/science/article/pii/S2352146520301654.

[Pui+18] Xavier Puig et al. "Virtualhome: Simulating household activities via programs". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[Qiu+21] Zhaofan Qiu et al. “Boosting Video Representation Learning With Multi-Faceted Integration”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 14030–14039.

[Rad+18] Alec Radford et al. “Improving language understanding by generative pre-training”. In: 2018.

[Rad+19] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: (2019).

[Ram+22] Aditya Ramesh et al. “Hierarchical Text-Conditional Image Generation with CLIP Latents”. In: *arXiv preprint arXiv:2204.06125* (2022).

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

[RF18] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: abs/1804.02767 (2018). arXiv: `1804.02767`. URL: `http://arxiv.org/abs/1804.02767`.

[RGL19] Stephan Rabanser, Stephan Günnemann, and Zachary C. Lipton. “Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[RHP20] Alexander Robey, Hamed Hassani, and George J. Pappas. “Model-Based Robust Deep Learning”. In: *arXiv preprint arXiv:2005.10247*. 2020.

[RKB04] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. “GrabCut: Interactive Foreground Extraction Using Iterated Graph Cuts”. In: *ACM Transactions on Graphics*. 2004.

[Ros+20] Jonathan S. Rosenfeld et al. “A Constructive Prediction of the Generalization Error Across Scales”. In: *International Conference on Learning Representations (ICLR)*. 2020.

[RP] Mike Roberts and Nathan Paczan. *Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding*. arXiv 2020.

[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.

[RSL18] A. Raghunathan, J. Steinhardt, and P. Liang. “Certified defenses against adversarial examples”. In: *International Conference on Learning Representations (ICLR)*. 2018.

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

[RWC18] Cynthia Rudin, Caroline Wang, and Beau Coker. “The age of secrecy and unfairness in recidivism prediction”. In: *arXiv preprint arXiv:1811.00731* (2018).

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

[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.

[Sav+19] Manolis Savva et al. “Habitat: A platform for embodied ai research”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019.

[Sch+15] Karsten Scheibler et al. “Towards Verification of Artificial Neural Networks.” In: *MBMV*. 2015, pp. 30–40.

[Sch+20a] Steffen Schneider et al. “Improving robustness against common corruptions by covariate shift adaptation”. In: *arXiv preprint arXiv:2006.16971* (2020).

[Sch+20b] Julian Schrittwieser et al. “Mastering Atari, Go, chess and shogi by planning with a learned model”. In: vol. 588. 7839. Springer Science and Business Media LLC, 2020, pp. 604–609.

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

[SGK20] Mohammad Ahmad Sheikh, Amit Kumar Goel, and Tapas Kumar. “An Approach for Prediction of Loan Approval using Machine Learning Algorithm”. In: *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*. 2020, pp. 490–494. DOI: 10.1109/ICESC48915.2020.9155614.

[Sha+18] Shital Shah et al. “Airsim: High-fidelity visual and physical simulation for autonomous vehicles”. In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[Sha+19] Vaishaal Shankar et al. “Do Image Classifiers Generalize Across Time?” In: *arXiv preprint arXiv:1906.02168* (2019).

[SHB15] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Improving neural machine translation models with monolingual data”. In: *arXiv preprint arXiv:1511.06709* (2015).

[Shu+20] Michelle Shu et al. “Identifying Model Weakness with Adversarial Examiner”. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2020.

[Sil+16] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), p. 484.

[Sil+17] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (2017), p. 354.

[SND18] Aman Sinha, Hongseok Namkoong, and John Duchi. “Certifiable Distributional Robustness with Principled Adversarial Training”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[Soc+13] Richard Socher et al. “Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. URL: https://www.aclweb.org/anthology/D13-1170.

[Son+20] Yunlong Song et al. “Flightmare: A Flexible Quadrotor Simulator”. In: *arXiv preprint arXiv:2009.00563* (2020).

[Sri+14] Nitish Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[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.

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

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

[Sun+17] Chen Sun et al. “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era”. In: *International Conference on Computer Vision (ICCV)*. 2017.

[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).

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

[Tai+14] Yaniv Taigman et al. “DeepFace: Closing the Gap to Human-Level Performance in Face Verification”. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR ’14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1701–1708. ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.220. URL: https://doi.org/10.1109/CVPR.2014.220.

[Tao+20] Rohan Taori et al. “Measuring Robustness to Natural Distribution Shifts in Image Classification”. In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[TE11] Antonio Torralba and Alexei Efros. “Unbiased look at dataset bias”. In: *CVPR 2011*. 2011, pp. 1521–1528.

[Tib94] Robert Tibshirani. “Regression Shrinkage and Selection Via the Lasso”. In: *Journal of the Royal Statistical Society, Series B* 58 (1994), pp. 267–288.

[TKC05] Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. "Core Vector Machines: Fast SVM Training on Very Large Data Sets". In: *Journal of Machine Learning Research (JMLR)* (2005).

[TL19] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *International Conference on Machine Learning (ICML)*. 2019.

[Tor03] Antonio Torralba. "Contextual Priming for Object Detection". In: *International Journal of Computer Vision (IJCV)*. 2003.

[Tra+20] Florian Tramer et al. "On adaptive attacks to adversarial example defenses". In: *arXiv preprint arXiv:2002.08347* (2020).

[TT20] Jörg Tiedemann and Santhosh Thottingal. "OPUS-MT — Building open translation services for the World". In: *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*. Lisbon, Portugal, 2020.

[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.

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

[Vie15] Juan Pablo Vielma. "Mixed integer linear programming formulation techniques". In: *SIAM Review* 57.1 (2015), pp. 3–57.

[Vin+08] Pascal Vincent et al. "Extracting and Composing Robust Features with Denoising Autoencoders". In: *International Conference on Machine Learning (ICML)*. 2008.

[Wan+18] Tongzhou Wang et al. "Dataset Distillation". In: *arXiv preprint arXiv:1811.10959* (2018).

[WC20] Garrett Wilson and Diane J Cook. "A survey of unsupervised deep domain adaptation". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 11.5 (2020), pp. 1–46.

[WD18] Mei Wang and Weihong Deng. "Deep visual domain adaptation: A survey". In: *Neurocomputing* 312 (2018), pp. 135–153.

[Wen+18] Tsui-Wei Weng et al. "Towards Fast Computation of Certified Robustness for ReLU Networks". In: *Proceedings of the 35th International Conference on Machine Learning*. 2018.

[WK17] Eric Wong and J Zico Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: *Proceedings of the 35th International Conference on Machine Learning*. 2017.

[Won+18] Eric Wong et al. "Scaling provable adversarial defenses". In: 2018.

[WSM21] Eric Wong, Shibani Santurkar, and Aleksander Madry. "Leveraging Sparse Linear Layers for Debuggable Deep Networks". In: *International Conference on Machine Learning (ICML)*. 2021.

[Wu+18] Yi Wu et al. "Building generalizable agents with a realistic and rich 3d environment". In: *arXiv preprint arXiv:1801.02209* (2018).

[WXM22] Eric Wong, Kai Xiao, and Aleksander Madry. "Dataset Projection: Finding Target-aligned Subsets of Auxiliary Data". In: *arXiv preprint arXiv:TODO* (2022).

[Xia+18] Fei Xia et al. "Gibson env: Real-world perception for embodied agents". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[Xia+19a] Chaowei Xiao et al. "MeshAdv: Adversarial Meshes for Visual Recognition". In: *Computer Vision and Pattern Recognition (CVPR)*. 2019.

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

[Xia+20a] Fei Xia et al. "Interactive Gibson Benchmark: A Benchmark for Interactive Navigation in Cluttered Environments". In: *IEEE Robotics and Automation Letters* (2020).

[Xia+20b] Fanbo Xiang et al. "SAPIEN: A simulated part-based interactive environment". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

[Xia+21a] Kai Xiao et al. "Noise or Signal: The Role of Image Backgrounds in Object Recognition". In: *International Conference on Learning Representations (ICLR)*. 2021.

[Xia+21b] Kai Xiao et al. "Noise or signal: The role of image backgrounds in object recognition". In: *International Conference on Learning Representations (ICLR)*. 2021.

[Xie+17] Cihang Xie et al. "Adversarial examples for semantic segmentation and object detection". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1369–1378.

[XTJ18] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. "Output Reachable Set Estimation and Verification for Multi-Layer Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* (Mar. 2018).

[Yeh+20] Chih-Kuan Yeh et al. "On Completeness-aware Concept-Based Explanations in Deep Neural Networks". In: 2020.

[YS02] Andrew W. Yip and Pawan Sinha. "Contribution of Color to Face Recognition". In: *Perception*. 2002.

[Yu+18] Jiahui Yu et al. “Generative Image Inpainting with Contextual Attention”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[Zec+18] John R. Zech et al. “Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study”. In: *PLoS Medicine*. 2018.

[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 (IJCV)*. 2007.

[Zha+14] Zhanpeng Zhang et al. “Facial Landmark Detection by Deep Multi-task Learning”. In: *European Conference on Computer Vision (ECCV)*. 2014.

[Zha+19] Shuai Zhang et al. “Deep Learning Based Recommender System: A Survey and New Perspectives”. In: *ACM Comput. Surv.* 52.1 (2019).

[Zhu+20] Yuke Zhu et al. “robosuite: A modular simulation framework and benchmark for robot learning”. In: *arXiv preprint arXiv:2009.12293* (2020).

[ZK16] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *British Machine Vision Conference (BMVC)*. 2016.

[ZLB17] Yu Zeng, Huchuan Lu, and Ali Borji. “Statistics of Deep Generated Images”. In: *arXiv preprint arXiv:1708.02688* (2017).

[ZXY17] Zhuotun Zhu, Lingxi Xie, and Alan Yuille. “Object Recognition without and without Objects”. In: *International Joint Conference on Artificial Intelligence*. 2017.

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level convolutional networks for text classification”. In: *Advances in neural information processing systems* 28 (2015).

Appendix A

Additional details for Chapter 2

A.1 Datasets details

We choose the following 9 high-level classes.

Class	WordNet ID	Number of sub-classes
Dog	n02084071	116
Bird	n01503061	52
Vehicle	n04576211	42
Reptile	n01661091	36
Carnivore	n02075296	35
Insect	n02159955	27
Instrument	n03800933	26
Primate	n02469914	20
Fish	n02512053	16

Table A.1: The 9 classes of ImageNet-9.

All datasets used in the paper are balanced by randomly removing images from classes that are over-represented. We only keep as many images as the smallest post-modification synthetic dataset, so all synthetic datasets (except IN-9L) have the same number of images. We also use a custom GUI to manually process the test set to improve data quality. For IN-9L, the only difference from using the corresponding classes in the original ImageNet dataset is that we balance the dataset.

For all images: we apply the following filters before adding each image to our datasets.

- The image must have bounding box annotations.
- For simplicity, each image must have exactly one bounding box. A large majority of images that have bounding box annotations satisfy this.

For images needing a properly segmented foreground: This includes the 3 MIXED datasets, ONLY-FG, and NO-FG. We filter out images based on the following criteria.

- Because images are cropped before they are fed into models, we require that less than 50% of the bounding box is removed by the crop, to ensure that the foreground still exists. Almost all images pass this filter.
- The OpenCV foreground segmentation function `cv2.grabCut` (used to extract the foreground shape) must work on the image. We remove images where it fails.
- For the test set only, we manually remove images with foreground segmentations that retain a significant portion of the background signal.
- For the test set only, we manually remove foreground segmentations that are very bad (e.g. the segmentation selects part of the image, and that part doesn't contain the foreground object).

For images needing only background signal: This includes ONLY-BG-B and ONLY-BG-T. In this case, we apply the following criteria:

- The bounding box must not be too big (more than 90% of the image). The intent here is to avoid ONLY-BG-B images being just a large black rectangle.
- For the test set only, we manually remove ONLY-BG images that still have an instance of the class even after removing the bounding box. This occurs when the bounding boxes are imperfect or incomplete (e.g. only one of two dogs in an image is labeled with a bounding box).

Creating the ONLY-BG-T dataset: We first make a "tiled" version of the background by finding the largest rectangular strip (horizontal or vertical) outside the bounding box, and tiling the entire image with that strip. We then replace the removed foreground with the tiled background. A visual example is provided in Figure A-1. We purposefully choose not to use deep-learning-based inpainting techniques such as [SFS18] to replace the removed foreground, as such methods could lead to biases that

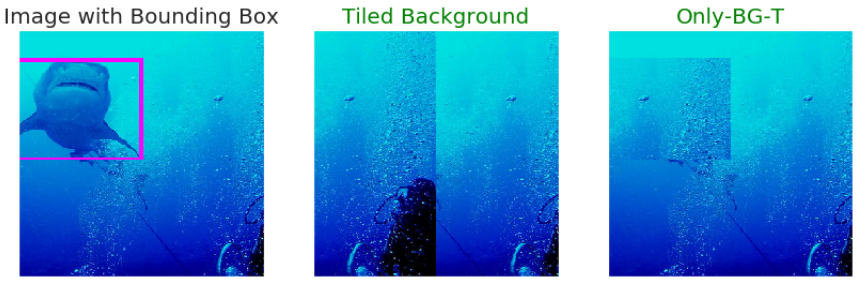


Figure A-1: Visualization of how ONLY-BG-T is created.

the inpainting model has learned from the data. For example, an inpainting model may learn that the best way to inpaint a missing chunk of a flower is to place an insect there, which is something we want to avoid.

Motivation for each IN-9 variation: We create ONLY-BG-B and ONLY-BG-T to remove the foreground completely, including the shape of the foreground object. We intend for ONLY-BG-B to be directly comparable to the prior work of [ZXY17] that uses similar methodology to evaluate older AlexNet models, while ONLY-BG-T is a more natural-looking background that avoids black rectangles introduced in ONLY-BG-B.

The NO-FG dataset is created to retain the foreground shape, but not the texture. We can use it to assess the relative importance of foreground shape compared to foreground texture.

Finally, we create four datasets that have identical foregrounds but each have distinct background signals. ONLY-FG has a pure black background to go with the foreground. MIXED-SAME has background signal from the same class as the foreground. MIXED-RAND has background signal from a random class, so it can be thought of as having neutral background signal. MIXED-NEXT has background signal from the next class, which will always be in conflict with the foreground . Any artifacts in the foreground that result from our image processing pipeline are equally present in all four datasets. Thus, these datasets help to isolate how much backgrounds alone influence model predictions *when the correct foreground exists in the image.*

Full-ImageNet version of each synthetic variation: We also apply the same methodology for disentangling foreground and background signal to the entire ImageNet validation set, creating Full-ImageNet (Full-IN) versions of each of our 7 dataset variations.

We evaluate a pre-trained ResNet-50 on Full-IN for comparison in Table A.2, and observe similar trends to ImageNet-9 that lead to similar conclusions on model

background reliance. We choose to focus on ImageNet-9 results in the main paper because of the following shortcomings of Full-IN.

1. Individual classes are quite small, as some classes have very few (or even zero) images that make it through our filters due to lack of proper annotated bounding boxes.

2. When bounding boxes do exist, their quality is often lower than those in the IN-9 classes. For example, many images of fruit contain multiple fruit, but only one will be properly annotated with a bounding box.

3. When creating the Mixed-Next equivalent for Full-IN, the next class is often similar to the previous one. For example, many dog breeds occur consecutively in ImageNet. Thus, Full-IN's Mixed-Next frequently has backgrounds that are similar to backgrounds from the foreground class.

A.2 Explaining the decreased BG-Gap of pre-trained ImageNet models

We investigate two possible explanations for why pre-trained ImageNet models have a smaller BG-Gap than models trained on ImageNet-9. Understanding this phenomenon can help inform how models should be trained to be more background-robust. We find slight improvements to background-robustness from training on more fine-grained classes. We find that training on larger datasets helps only slightly when the training dataset set size is smaller than IN-9L, but larger improvements occur when the training dataset size is bigger. Thus, we encourage training on larger datasets if reduced background robustness is the goal.

A.2.1 The effect of fine-grainedness on the BG-Gap

One possible explanation is that training models to distinguish between finer-grained classes forces them to focus more on the foreground, which contains relevant features for making those fine-grained distinctions, than the background, which may be fairly similar across sub-classes of a high-level class. This suggests that asking models to solve more fine-grained tasks could improve model robustness to background changes.

To test the effect of fine-grainedness on ImageNet-9, we make a related dataset called IN-9LB that uses the same 9 high-level classes and can be cleanly modified into

more fine-grained versions. Specifically, for IN-9LB we choose exactly 16 sub-classes for each high-level class, for a total of 144 ImageNet classes. To create successively more fine-grained versions of the IN-9LB dataset, we group every n sub-classes together into a higher-level class, for $n \in \{1, 2, 4, 8, 16\}$. Here, $n = 1$ corresponds to keeping all 144 ImageNet classes as they are, while $n = 16$ corresponds to only having 9 high-level classes, like ImageNet-9. Because we keep all images from those original ImageNet classes, this dataset is the same size as IN-9L.

We train models on IN-9LB at different levels of fine-grainedness and evaluate the BG-Gap of those models in Figure A-2. We find that fine-grained models have a smaller BG-Gap as well as better performance on Mixed-Next, but the improvement is very slight and also comes at the cost of decreased accuracy on Original. The BG-Gap of the most fine-grained classifier is 2.3% smaller than the BG-Gap of the most coarse-grained classifier, showing that fine-grainedness does improve background-robustness. However, the improvement is still small compared to the size of the BG-Gap (which is 13.3% for the fine-grained classifier).

A.2.2 The effect of larger dataset size on the BG-Gap

A second possible explanation for why pre-trained ImageNet models have a smaller BG-Gap is that training on larger datasets is important for background-robustness. To evaluate this possibility, we train models on different-sized subsets of IN-9LB. The largest dataset we train on is the full IN-9LB dataset, which is 4 times as large as IN-9, and the smallest is 1/4 as large as IN-9. Figure A-3 shows that increasing the dataset size does increase overall performance but only slightly decreases the BG-Gap.

Next, we train models on different-sized subsets of ImageNet; we use the pre-trained ResNet-50 ImageNet model for full-sized ImageNet, and we train new ResNet-50 models on subsets that are 1/2, 1/4, 1/8, 1/16, and 1/32 as large as ImageNet. In these cases, we observe in Figure A-4 that training on more data does not help significantly when the training dataset sizes are still small, but it does help more noticeably for models trained on 1/2 of ImageNet and all of ImageNet.

It is possible that having both a fine-grained class structure and more training data simultaneously is important for background-robustness. Furthermore, more training data (from other classes that are not in IN-9L) may also be the cause of the increased background-robustness of pre-trained ImageNet models.

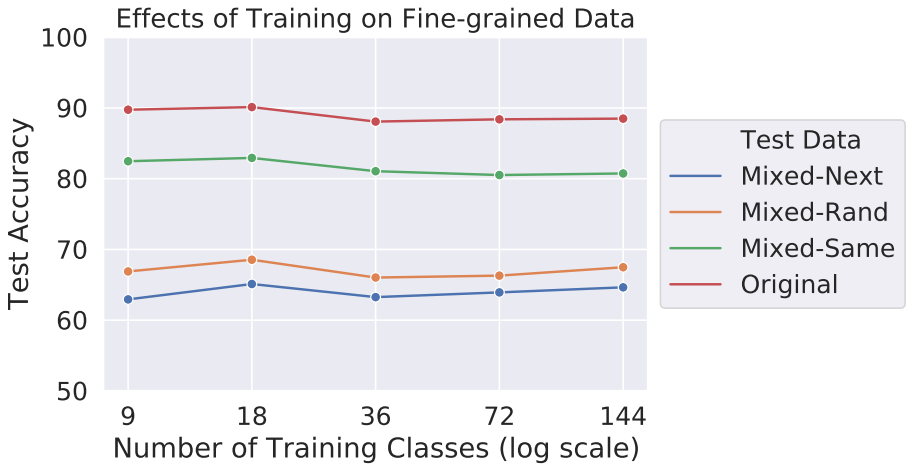


Figure A-2: We train models on IN-9LB at different levels of fine-grainedness (more training classes is more fine-grained). The BG-Gap, or the difference between the test accuracies on Mixed-Same and Mixed-Rand, decreases as we make the classification task more fine-grained, but the decrease is small compared to the size of the BG-Gap.

A.2.3 Summary of methods investigated to reduce the BG-Gap

In Figure A-5, we compare the BG-Gap of ResNet-50 models trained on different datasets and with different methods to a ResNet-50 pre-trained on ImageNet. We explore ℓ_p-robust training, increasing dataset size, and making the classification task more fine-grained, and find that none of these methods reduces the BG-Gap as much as pre-training on ImageNet. The only method that reduces the BG-Gap significantly more is training on Mixed-Rand. Furthermore, the same trends hold true if we measure the difference between Mixed-Same and Mixed-Next as opposed to the BG-Gap (the difference between Mixed-Same and Mixed-Rand).

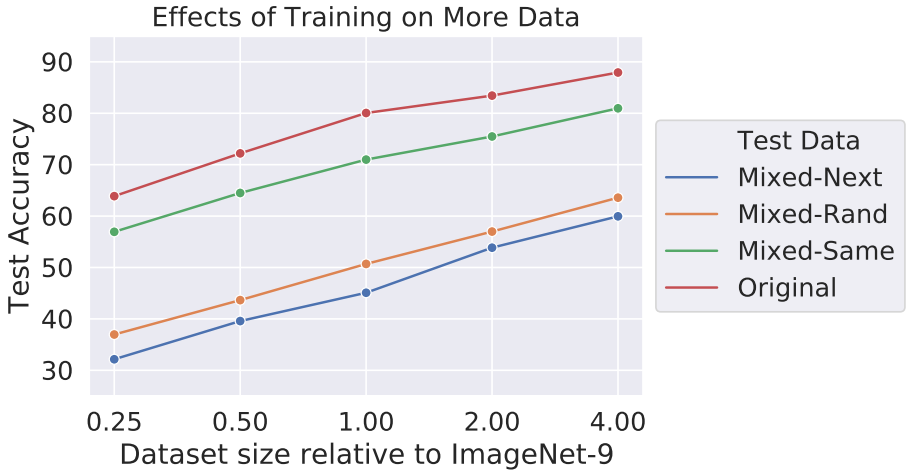


Figure A-3: We train models on different-sized subsets of IN-9LB. The largest training set we use is the full IN-9LB dataset, which is 4 times larger than ImageNet-9. While performance on all test datasets improves as the amount of training data increases, the BG-Gap has almost the same size regardless of the amount of training data used.

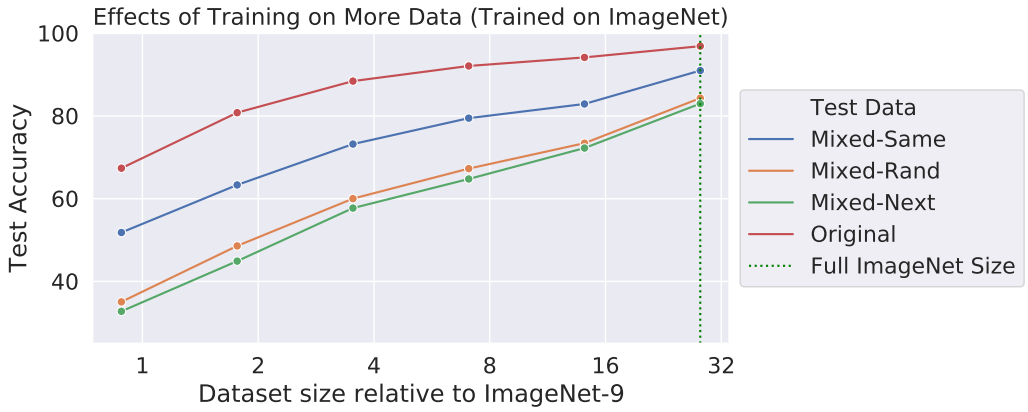


Figure A-4: We train models on different-sized subsets of ImageNet. We use a pre-trained ResNet-50 for the rightmost datapoints corresponding to training on the full ImageNet dataset, which is about 30 times larger than ImageNet-9. The BG-Gap begins to decrease when the training dataset set size is sufficiently large.

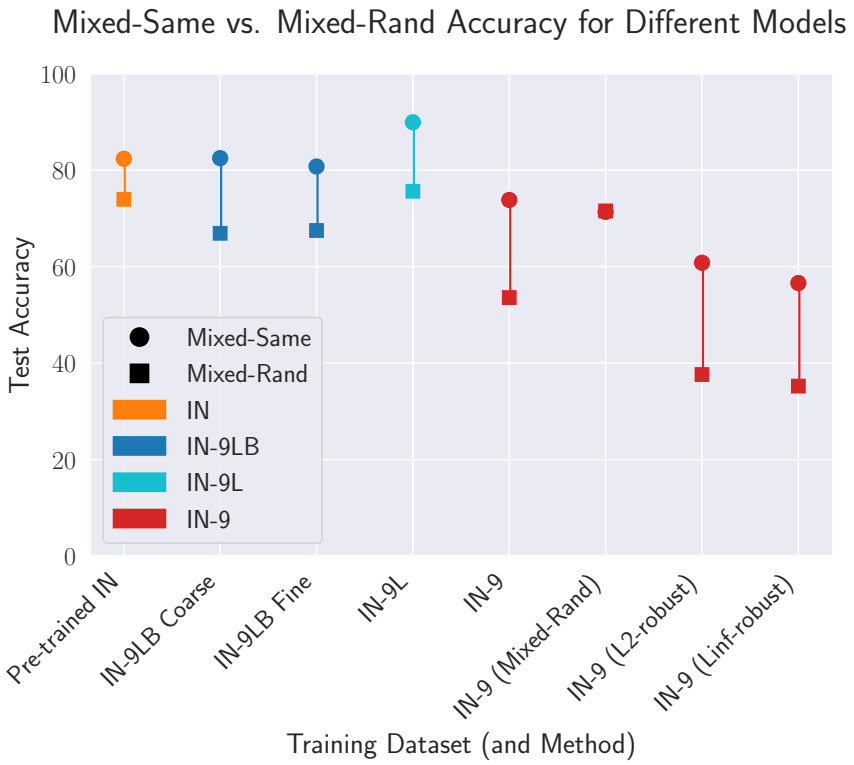


Figure A-5: We compare various different methods of training models and measure their BG-Gap, or the difference between Mixed-Same and Mixed-Rand test accuracy. We find that (1) Pre-trained IN models have surprisingly small BG-Gap. (2) Increasing fine-grainedness (IN-9LB Coarse vs. IN-9LB Fine) and dataset size (IN-9 vs. IN-9L) decreases the BG-Gap only slightly. (3) ℓ_p-robust training does not help. (4) Training on Mixed-Rand (cf. Section 2.3 appears to be the most effective strategy for reducing the BG-Gap. For such a model, the Mixed-Same and Mixed-Rand accuracies are nearly identical.

A.3 Training and evaluation details

For all models, we use fairly standard training settings for ImageNet-style models. We train for 200 epochs using SGD with a batch size of 256, a learning rate of 0.1 (with learning rate drops every 50 epochs), a momentum parameter of 0.9, a weight decay of 1e−4, and data augmentation (random resized crop, random horizontal flip, and color jitter). Unless specified, we always use a standard ResNet-50 architecture [He+16]. For the experiment depicted in Figure A-3, we found that using a smaller learning rate of 0.01 was necessary for training to converge on the smallest training sets. Thus, we used that same learning rate for all models in Figure A-3.

When evaluating ImageNet classifiers on IN-9, we map all ImageNet predictions to their corresponding coarse-grained class in IN-9. For example, we map both `giant schnauzer` and `Irish terrier` to `dog`, and both `goldfish` and `tiger shark` to FISH. If an ImageNet classifier outputs a class that has no corresponding coarse-grained class in IN-9, we consider the prediction incorrect.

A.4 Additional evaluation results

We include full results of training models on every synthetic IN-9 variation and then testing them on every synthetic IN-9 variation in Table A.2. In addition to being more comprehensive, this table and these IN-9 variations can help answer a variety of questions, of which we provide three examples here. Finally, we also evaluate a pre-trained model on Full-ImageNet (Full-IN) versions of each synthetic IN-9 variation for comparison.

How does more training data affect model performance with and without object shape?

We already show closely related results on the effect of more training data on the BG-GAP in Figure A-3. Here, we compare model test performance on the NO-FG and ONLY-BG-B test sets. Both replace the foreground with black, but only NO-FG retains the foreground shape.

By comparing the models trained on ORIGINAL and IN-9L (4x more training data), we find that

1. The ORIGINAL-trained model performs about 9% better on NO-FG than ONLY-BG-B, indicating that it can slightly improve accuracy by using object shape.

2. The IN-9L-trained model performs about 22% better on NO-FG than ONLY-

BG-B, showing that it can improve accuracy far more by using object shape.

Furthermore, both models perform very similarly on ONLY-BG-B. Thus, this suggests that more training data may allow models to learn to use object shape more effectively. Understanding this phenomena further could help inform model training and dataset collection if the goal is to train models that are able to leverage shape effectively.

How much information is leaked from the size of the foreground bounding box?

The scale of an object already gives signal correlated with the object class [Tor03]. Even though they are designed to avoid having foreground signal, the background-only datasets ONLY-BG-B and ONLY-BG-T may inadvertently leak information about object scale due to the bounding box sizes being recognizable.

To gauge the extent of this leakage, we can measure how models trained on datasets where only the foreground signal has useful correlation (MIXED-RAND or ONLY-FG) perform on the background-only test sets. We find that there is small signal leakage from bounding box size alone—a model trained on ONLY-FG achieves about 23% background-only test accuracy, suggesting that it is able to exploit the signal leakage to some degree. A model trained on MIXED-RAND achieves about 15% background-only test accuracy, just slightly better than random, perhaps because it is harder for models to measure (and thus, make use of) object scale when training on MIXED-RAND.

The existence of a small amount of information leakage in this case shows the importance of comparing MIXED-SAME (as opposed to just ORIGINAL) with MIXED-RAND and MIXED-NEXT when assessing model dependence on backgrounds. Indeed, the MIXED datasets may contain (1) image processing artifacts, such as rough edges from the foreground processing, and (2) small traces of the original background. This makes it important to control for both factors when measuring how models react to varying background signal.

How does foreground bounding box size affect accuracy on ONLY-BG-T?

We further find that models are more able to predict accurately using the background signal alone when the foreground object is smaller—this is visualized in Figure A-6. Intuitively this result makes sense, as most state-of-the-art models are trained with cropping-based data augmentation, which can remove small foreground objects from training images. Thus, models are actually trained to succeed when small foreground objects are cropped out, and our toolkit confirms that this is indeed the case.

What about other ways of modifying the background signal?

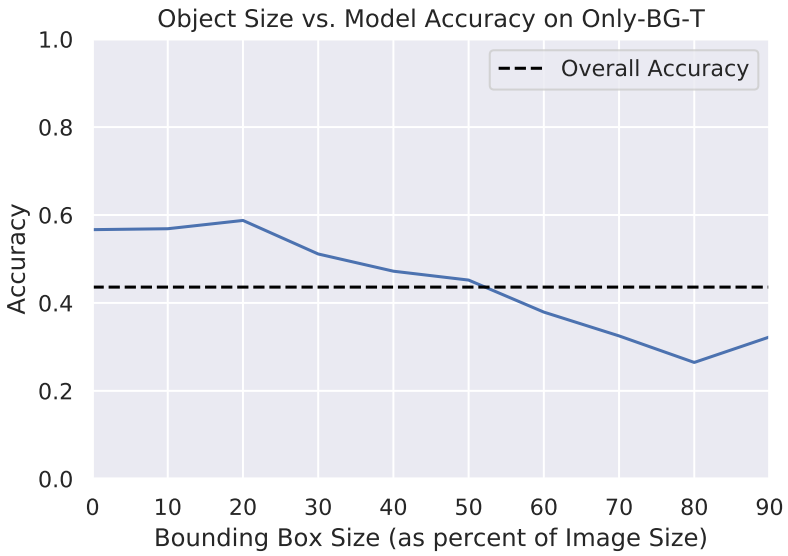


Figure A-6: Comparing model accuracy on ONLY-BG-T across different foreground object bounding box sizes. We observe that the model is more likely to succeed when shown only image backgrounds if the removed foreground objects have smaller bounding boxes. The dotted line represents the overall accuracy of the model on ONLY-BG-T (averaged over all bounding box sizes).

Trained on	**Test Dataset**								
	MIXED-NEXT	MIXED-RAND	MIXED-SAME	NO-FG	ONLY-BG-B	ONLY-BG-T	ONLY-FG	ORIGINAL	IN-9L
MIXED-NEXT	78.07	53.28	48.49	16.20	11.19	8.22	59.60	52.32	46.44
MIXED-RAND	71.09	71.53	71.33	26.72	15.33	14.62	74.89	73.23	67.53
MIXED-SAME	45.41	51.36	74.40	39.85	35.19	41.58	61.65	75.01	69.21
NO-FG	13.70	18.74	42.79	70.91	36.79	42.52	31.48	48.94	47.62
ONLY-BG-B	10.35	15.41	38.37	37.85	54.30	42.54	21.38	42.10	41.01
ONLY-BG-T	11.48	17.09	45.80	40.84	38.49	50.25	19.19	49.06	47.94
ONLY-FG	33.04	35.88	47.63	27.90	23.58	22.59	84.20	54.62	51.50
ORIGINAL	48.77	53.58	73.80	42.22	32.94	40.54	63.23	85.95	80.38
IN-9L	71.21	75.60	89.90	55.78	34.02	43.60	84.12	96.32	94.61
ImageNet	82.99	84.32	90.99	52.69	12.69	17.36	90.17	96.89	95.33
ImageNet (Full-IN)	51.47	48.69	64.34	21.70	7.98	9.51	59.19	76.07	-

Table A.2: The test accuracies, in percentages, of ResNet-50 models trained on all variants of ImageNet-9, and a pre-trained ImageNet ResNet-50. The bottom row and the second-to-last-row test the same pre-trained ImageNet model; however, the bottom row tests the model on the Full-IN version of each dataset variation. Testing on Full-IN shows similar trends as testing on ImageNet-9. Note that the MIXED-NEXT test accuracy is actually higher than the MIXED-RAND test accuracy in the bottom row because the next class is often very similar to the previous class in Full-IN.

One can modify the background in various other ways—for example, instead of replacing the background with black as in ONLY-FG, the background can be blurred as in the BG-BLURRED image of Figure A-7. As expected, blurred backgrounds are still slightly correlated with the correct class. Thus, test accuracies for standard models on this dataset are higher than on ONLY-FG, but lower than on MIXED-SAME (which

Figure A-7: Backgrounds can also be modified in other ways; for example, it can be blurred. Our evaluations on this dataset show similar results.

has signal from random class-aligned backgrounds that are *not* blurred). While we do not investigate all possible methods of modifying background signal, we believe that the variations we do examine in ImageNet-9 already improve our understanding of how background signals matter. Investigating other variations could provide an even more nuanced understanding of what parts of the background are most important.

A.5 Additional related works and explicit comparisons

There has been prior work on mitigating contextual bias in image classification, the influence of background signals on various datasets, and techniques like foreground segmentation that we leverage.

Mitigating Contextual Bias: [Kho+12] focuses on mitigating dataset-specific contextual bias and proposes learning SVMs with both general weights and dataset-specific weights, while [MTW12] creates an out-of-context detection task with 209 out-of-context images and suggests using graphical models to solve it. [SSF19] focuses on the role of co-occurring objects as context in the MS-COCO dataset, and uses object removal to show that (a) models can still predict a removed object when only co-occurring objects are shown, and (b) special data-augmentation can mitigate this.

Explicit Comparison to Prior Works Studying the Influence of Backgrounds: In comparison to prior works on the influence of image backgrounds (described in Section 2.5), our work contributes the following.

- We develop a toolkit for analyzing the background dependence of ImageNet classifiers, the most common benchmark for computer vision progress. Only [ZXY17], which we compare to in Section 2.5, also focuses on ImageNet.

- The test datasets we create separate and mix foreground and background signals in various ways (cf. Table 2.1), allowing us to study the sensitivity of models to these signals in a more fine-grained manner.
- Our toolkit for separating foreground and background can be applied without human-annotated foreground segmentation, which prior works on MS-COCO and Waterbirds rely on. This is important because foreground segmentation annotations are hard to collect and do not exist for ImageNet.
- We study the extent of background dependence in the extreme case of adversarial backgrounds.
- We focus on better vision models, including ResNet [He+16], Wide ResNet[ZK16], and EfficientNet [TL19].
- We evaluate how improvements on the ImageNet benchmark have affected background dependence (cf. Section 2.4).
- We will publicly release our toolkit (code and datasets) for benchmarking background dependence so that others can also use it to better understand their own models. Our toolkit is compatible with any ImageNet-trained model.

Foreground Segmentation and Image Inpainting: In order to create IN-9 and its variants, we rely on OpenCV's implementation of the foreground segmentation algorithm GrabCut [RKB04]. Foreground segmentation is a branch of computer vision that seeks to automatically extract the foreground from an image [HGW01]. After finding the foreground, we remove it and simply replace the foreground with copies of parts of the background. Other works solve this problem, called image inpainting, either using exemplar-based methods [CPT04] or using deep learning [Yu+18; SFS18]. [SFS18] both detects the foreground for removal and inpaints the removed region. However, more advanced inpaintings techniques can be slow and inaccurate when the region that must be inpainted is relatively large [SFS18], which is the case for many ImageNet images. Exploring better ways of segementing the foreground and inpainting the removed foreground could improve our analysis toolkit further.

A.6 Additional examples of synthetic datasets

We randomly sample an image from each class, and display all synthetic variations of that image, as well as the predictions of a pre-trained ResNet-50 (trained on IN-9L) on each variant.

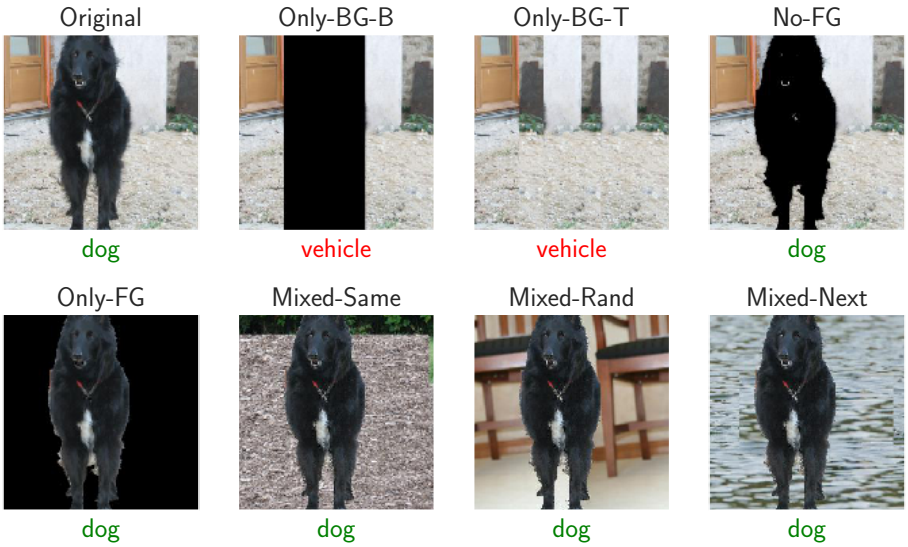


Figure A-8: ImageNet-9 variations—Dog.

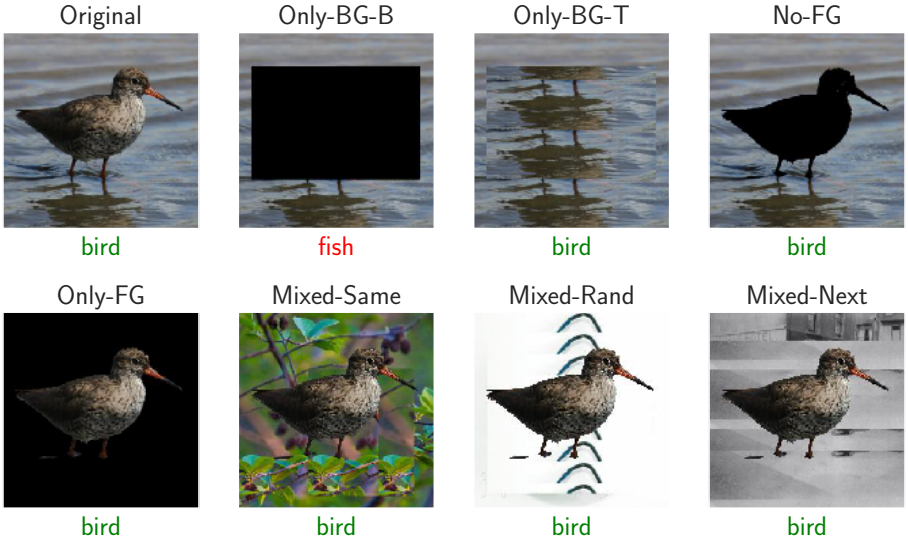


Figure A-9: ImageNet-9 variations—Bird.

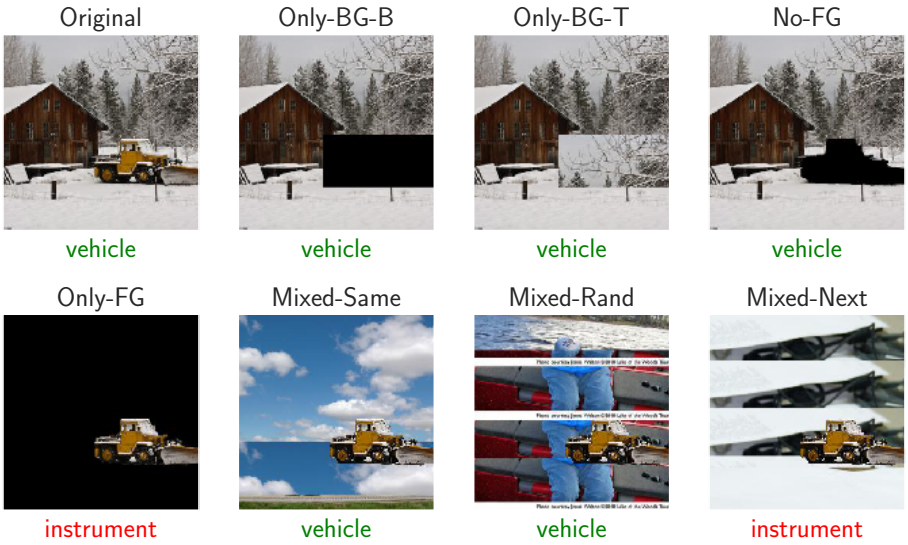


Figure A-10: ImageNet-9 variations—Vehicle.

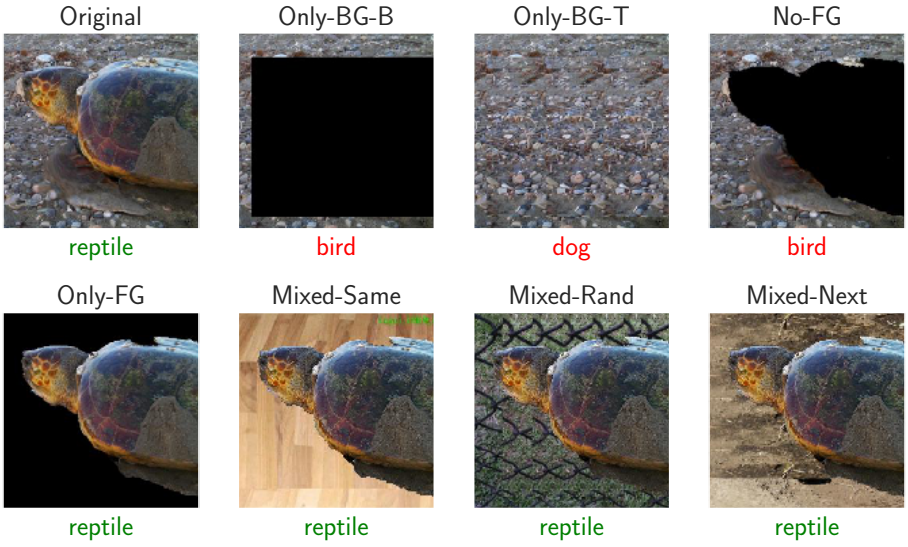


Figure A-11: ImageNet-9 variations—Reptile.

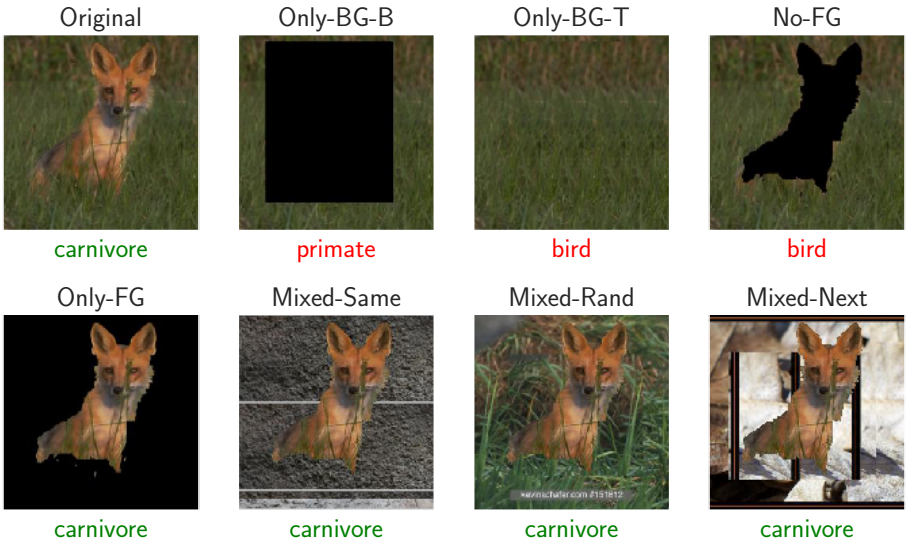


Figure A-12: ImageNet-9 variations—Carnivore.

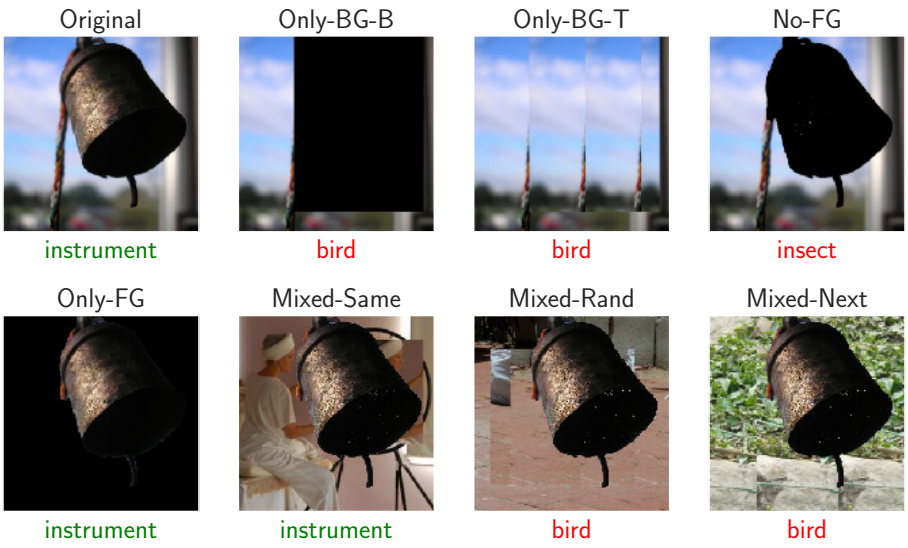


Figure A-13: ImageNet-9 variations—Instrument.

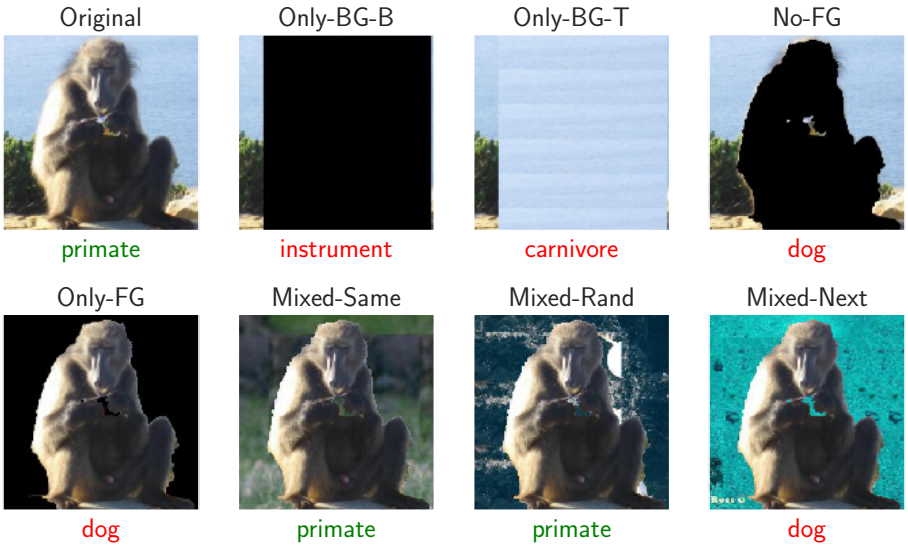


Figure A-14: ImageNet-9 variations—Primate.

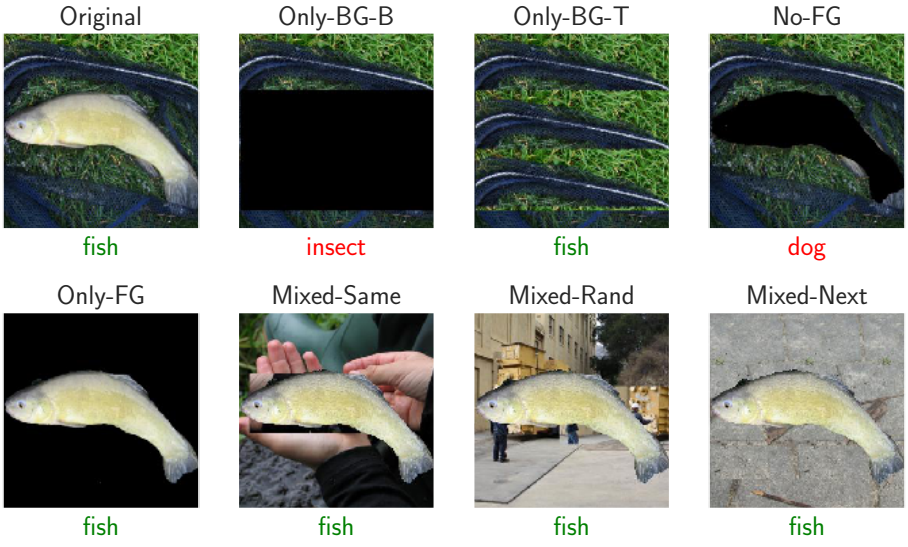


Figure A-15: ImageNet-9 variations—Fish.

A.7 Adversarial backgrounds

We compute the adversarial background attack success rate for 4 models in Table A.3. While the Mixed-Rand model is more adversarially background robust than the Original model, it is less adversarially background robust than the IN-9L model. The model trained on all of ImageNet is the most adversarially background robust of all models. This suggests that increasing training dataset size (IN-9L) has a bigger effect on adversarial background robustness than randomizing backgrounds during training (Mixed-Rand). On the other hand, the Mixed-Rand model has a much lower BG-Gap than the IN-9L model, indicating that models with a smaller BG-Gap are not necessarily robust to adversarial backgrounds, and vice versa.

Training Dataset	Original	Mixed-Rand	IN-9L	ImageNet
Attack Success Rate	99.0%	93.5%	88.0%	77.7%

Table A.3: Adversarial backgrounds attack success rates for 4 models analyzed in this work. The Original and the Mixed-Rand are trained on equally small datasets, IN-9L is trained on 4x more data, and the ImageNet model is trained on the most data.

Next, we visualize the attack success rate distribution of the different backgrounds from the insect class in Figure A-16. The long tail of the distribution indicates that many backgrounds are especially capable of fooling models.

Finally, we include the 5 most fooling backgrounds for all classes, the fool rate for each of those 5 backgrounds, and the total fool rate across all backgrounds from that class (on the left of each row) below.

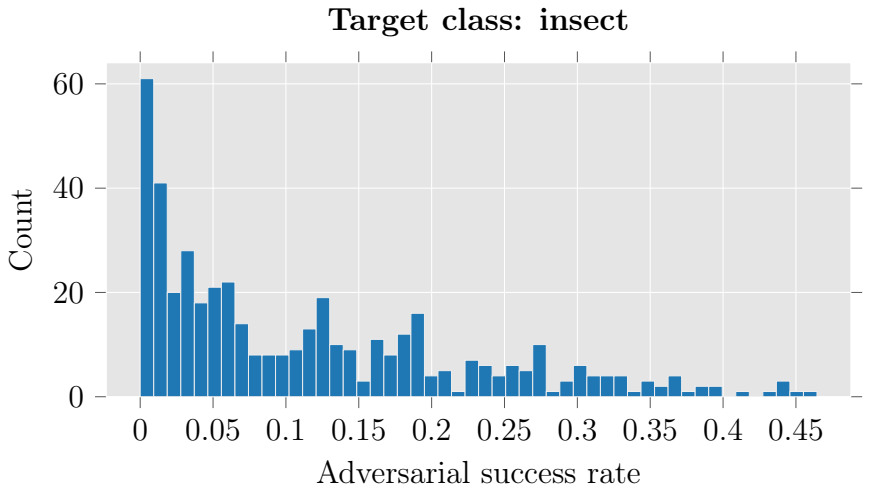


Figure A-16: Histogram of insect backgrounds grouped by how often they cause (non-insect) foregrounds to be classified as insect by a IN-9L-trained model. We visualize the five backgrounds that fool the classifier on the largest percentage of images in Figure 2-4.

Figure A-17: Most adversarial backgrounds—Dog.

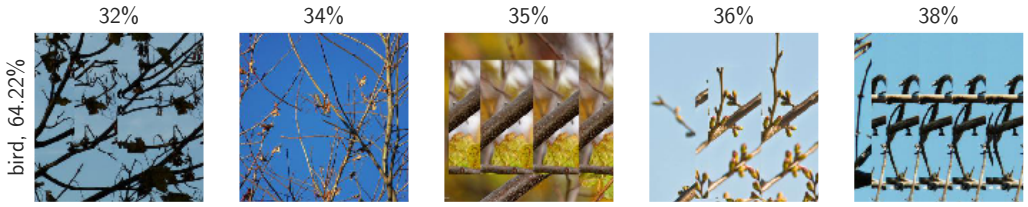


Figure A-18: Most adversarial backgrounds—Bird.

Figure A-19: Most adversarial backgrounds—Vehicle.

Figure A-20: Most adversarial backgrounds—Reptile.

Figure A-21: Most adversarial backgrounds—Carnivore.

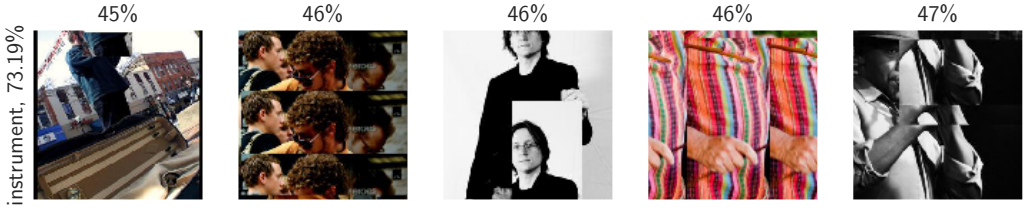


Figure A-22: Most adversarial backgrounds—Instrument.

Figure A-23: Most adversarial backgrounds—Primate.

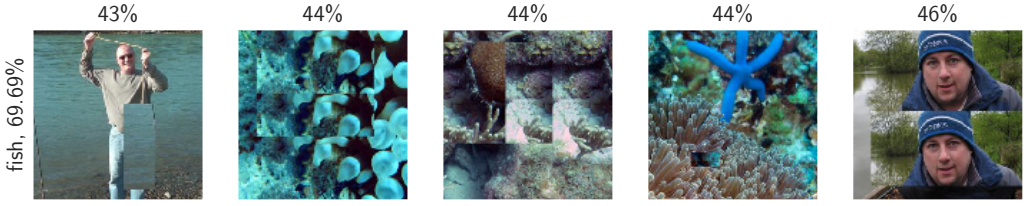


Figure A-24: Most adversarial backgrounds—Fish.

A.8 Examples of fooling backgrounds in unmodified images

We visualize examples of images where the background of the full original image actually fools models in Figure A-25. For these images, models classify the foreground alone correctly, but they predict the same wrong class on the full image and the background. We denote these images as "BG Fools" in Table 2.3 and Figure 2-8. While this category is relatively rare (accounting for just 3% of the ORIGINAL-trained model's predictions), they reveal a subset of original images where background signal hurts classifier performance. Qualitatively, we observe that these images all have confusing or misleading backgrounds.

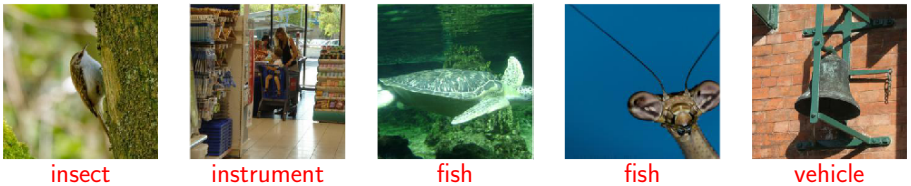


Figure A-25: Images that are incorrectly classified (as the class on the top row, which is the same class that their background alone from ONLY-BG-T is classified as), but are correctly classified (as the class on the bottom row) when the background is randomized. Note that these images have confusing backgrounds that could be associated with another class.

Appendix B

Additional details for Chapter 3

B.1 Experiment dashboard

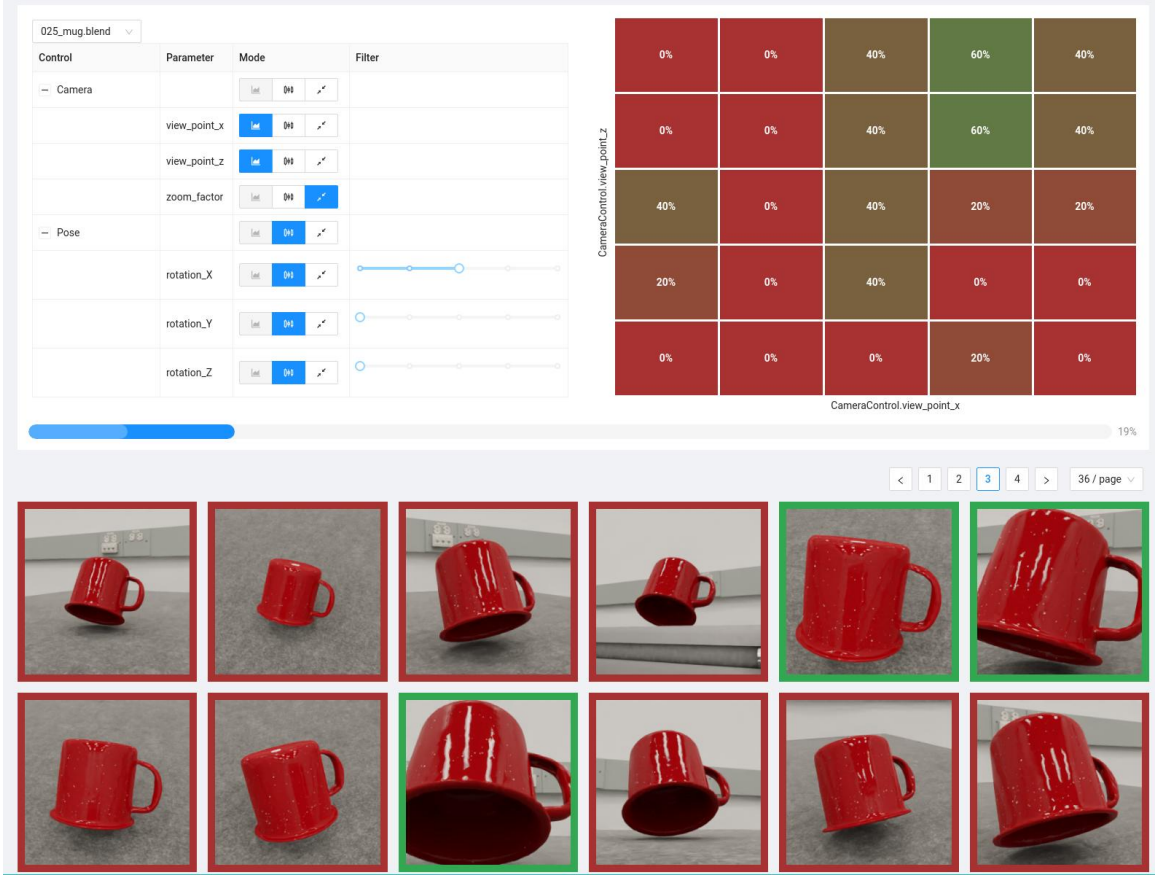


Figure B-1: The *3DB* dashboard used for data exploration.

Since experiments usually produce large amounts of data that can be hard to get a sense of, we created a data visualization dashboard. Given a folder containing the

JSON logs of a job, it offers a user interface to explore the influence of the controls.

For each parameter of each control, we can pick one out three mode:

- **Heat map axis**: This control will be used as the `x` or `y` axis of the heat map. Exactly two controls should be assigned to this mode to enable the visualization. Hovering on cells of the heat map will filter all samples falling in that region.
- **Slider**: This mode enables a slider that is used to only select the samples that match exactly this particular value.
- **Aggregate**: do not filter samples based on this parameter

B.2 iPhone app

We developed a native iOS app to help align objects in the physical experiment (Section 3.4). The app allows the user to enter one or more rendering IDs (corresponding to scenes rendered by *3DB*); the app then brings up a camera with a translucent overlay of either the scene or an edge-filtered version of the scene (cf. Figure B-2). We used the app to align the physical object and environment with their intended place in the rendered scene. The app connects to the same backend serving the experiment dashboard.

B.3 Controls

3DB takes an object-centric perspective, where an object of interest is spawned on a desired background. The scene mainly consists of the object and a camera. The controls in our pipeline affect this interplay between the scene components through various combinations of properties, which subsequently creates a wide variety of rendered images. The controls are implemented using the Blender Python API '`bpy`' that exposes an easy to use framework for controlling Blender. '`bpy`' primarily exposes a scene context variable, which contains references to the properties of the components such as objects and the camera; thus allowing for easy modification.

3DB comes with several predefined controls that are ready to use (see `https://3db.github.io/3db/`). Nevertheless, users are able (and encouraged) to implement custom controls for their use-cases.

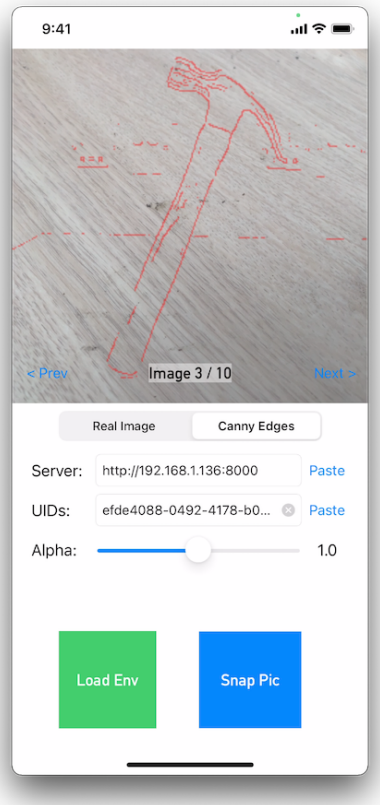
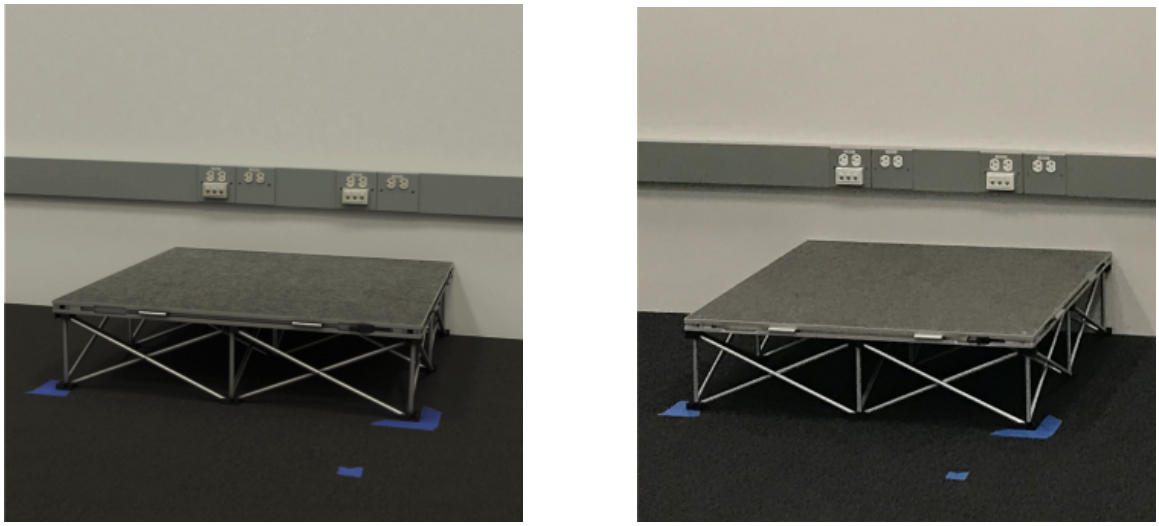


Figure B-2: A screenshot of the iOS app used to align objects for the physical-world experiment. After starting the dashboard server, the user can specify the server location as well as a set of rendering IDs. The corresponding renderings will be displayed over a camera view, allowing the user to correctly position the object in the frame. The user can adjust the object transparency, and can toggle between overlaying the full rendering and overlaying just the edges (shown here).

B.4 Additional experiments details

We refer the reader to our package `https://github.com/3db/3db` for all source code, 3D models, HDRIs, and config files used in the experiments of this paper.

For all experiments we used the pre-trained ImageNet ResNet-18 included in `torchvision`. In this section we will describe, for each experiment the specific 3D-models and environments used by *3DB* to generate the results.



(a) Synthetic (b) Real picture (iPhone 12 Pro)

Figure B-3: Studio used for the real-world experiments (Section 3.4).

B.4.1 Sensitivity to image backgrounds (Section 3.3.1)

Analysing a subset of backgrdounds

Models: We collected 19 3D-models in total. On top of the models shown on figure B-6, we used models for: (1) an orange, (2) two different toy power drills, (3) a baseball ball, (4) a tennis ball, (5) a golf ball, (6) a running shoe, (7) a sandal and (8) a toy gun. Some of these models are from `YCB` [Cal+15] and the rest are purchased from `amazon.com` and then put through a 3D scanner to get corresponding meshes.

Environments: We sourced 20 2k HDRI from the website `https://hdrihaven.com`. In particular we used: `abandoned_workshop, adams_place_bridge, altanka, aristea_wreck, bush_restaurant, cabin, derelict_overpass, dusseldorf_bridge, factory_yard, gray_pier, greenwich_park_03,`

kiara_7_late-afternoon, kloppenheim_06, rathaus, roofless_ruins, secluded_beach, small_hangar_02, stadium_01, studio_small_02, studio_small_04.

Analyzing all backgrounds with the "coffee mug" model.

Models: We used a single model: the coffee mug, in order to keep computational resources under control.

Environments: We used 408 HDRIs from https://hdrihaven.com/ with a 2K resolution.

B.4.2 Texture-shape bias (section 3.3.2)

Textures: To replace the original materials, we collected 7 textures on the internet and we modified them to make them seamlessly tilable. These textures are shown on Figure B-6.

Models: We used all models that are shown on Figure B-6.

Environments: We used the virtual studio environment (Figure B-3).

B.4.3 Orientation and scale dependence (Section 3.3.3)

We use the same models and environments that are used in Appendix B.4.1.

B.4.4 3D models heatmaps (Figure 3-12)

Models: For this experiment we used the set of models shown on Figure B-6.

Environments: We used the virtual studio environment (see Figure B-3).

B.4.5 Case study: using 3DB to dive deeper (Section 3.3.4)

Models: We only used the mug since this experiment is mug specific.

Environments: We used the sudio set shown on Figure B-3.

B.4.6 Physical realism (Section 3.4)

Real-world pictures: All images were taken with an handheld Apple iPhone 12 Pro. To help us align the shots we used the application described in appendix B.2.

Models: We used the models shown in Figure 3-15.

Environments: The environment shown on Figure B-3 was especially designed for this experiment. The goal was to have an environment that matches our studio as closely as possible. The geometry and materials were carefully reproduced using reference pictures. The lighting was reproduce through a high resolution HDRI map.

B.5 Omitted figures

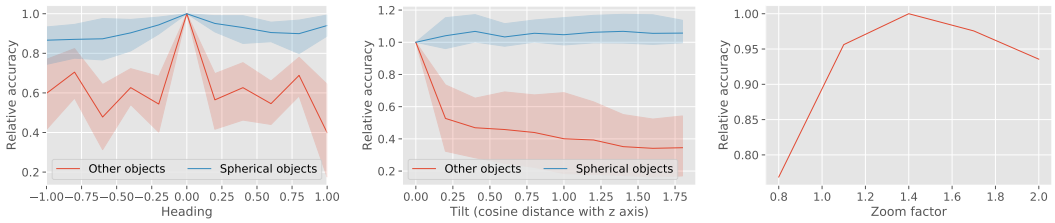
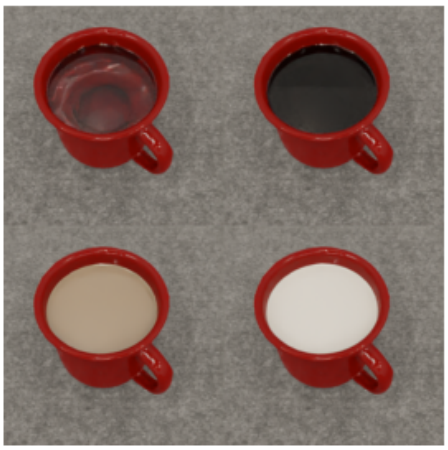
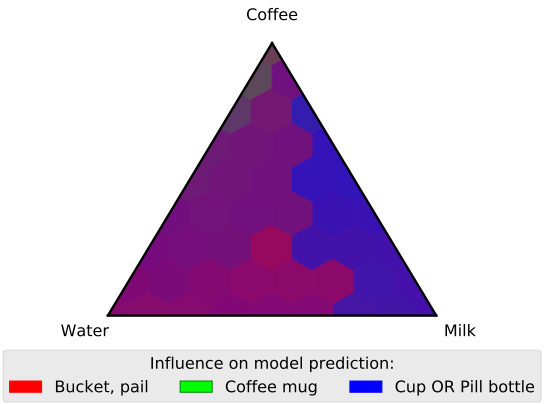


Figure B-4: Additional plots to Figure 3-13. We plot the distribution of model accuracy as a function of object heading (*top*), tilt (*middle*) and zoom (*bottom*), aggregated over variations in controls. For heading and tilt, we separately evaluate accuracy for (non-)spherical objects. Notice how the performance of the model degrades for non-spherical objects as the heading/tilt changes, but not for spherical objects. Also notice how the performance depends on the zoom level of the camera (how large the object is in the frame).



(a) Sample of the images rendered for the experiment presented in section 3.3.4.



(b) Un-normalized version of Figure 3-14-(b).

Figure B-5: Additional illustration for the mug liquid experiment of Figure 3-14. This figure shows the correlation of the liquid mixture in the mug on the prediction of the model, averaged over random viewpoints

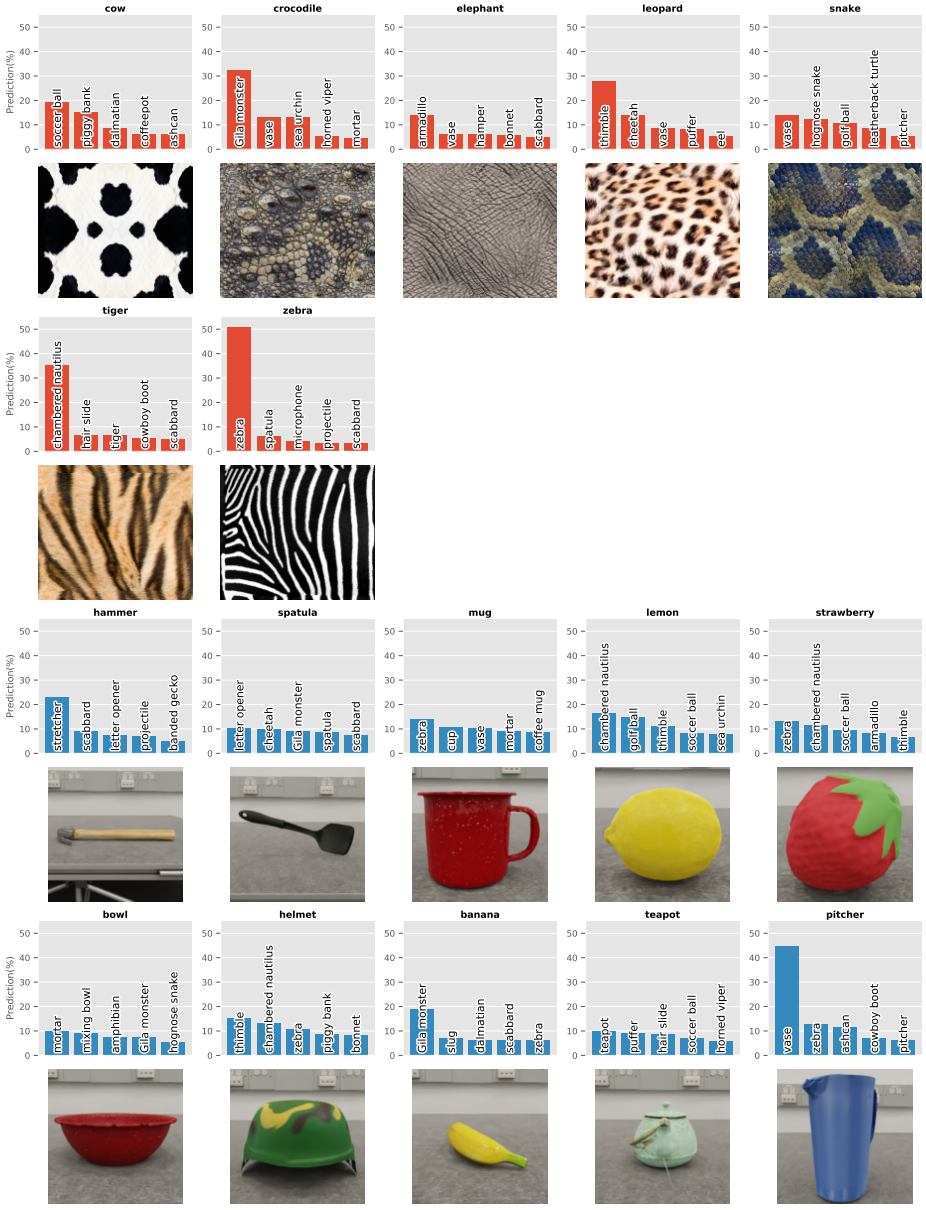


Figure B-6: Additional examples of the experiment in Figure 3-11. Distribution of classifier predictions after the texture of the 3D object model is altered. In the top rows, we visualize the most frequently predicted classes for each texture (averaged over all objects). In the bottom rows, we visualize the most frequently predicted classes for each object (averaged over all textures). We find that the model tends to predict based on the texture more often than based on the object.

Appendix C

Additional details for Chapter 4

C.1 Examples from Section 4.2 on the limits of auxiliary data

In this section, we provide all the specifics for the three experiments discussed in Section 4.2. These experiments demonstrate how indiscriminate usage of *all* auxiliary data can actually harm performance.

C.1.1 Training on ImageNet data for CIFAR10

In the first example (Figure 4-1), we showed that training on more ImageNet data does not always improve CIFAR10 performance. Indeed, although adding a small amount of ImageNet does boost CIFAR10 performance, adding too much ultimately decreases accuracy. The blue line shows the performance of a CIFAR10 classifier when we add auxiliary ImageNet data to one thousand CIFAR10 training examples. The orange line shows the performance of a CIFAR10 classifier when we only train on the auxiliary ImageNet data. In both cases, we find that optimal CIFAR10 performance is reached after adding approximately ten thousand ImageNet datapoints. After this point, adding more ImageNet data degrades the classifier's accuracy.

Our experimental setup for this setting follows our main experimental setup but with one main difference: to examine the effect of auxiliary data, we vary the size of the auxiliary dataset in the amounts of 2^k for $k = 4 \ldots 16$. Otherwise, the remaining specifics (how we select auxiliary data from ImageNet and training parameters) match our benchmark, and are described in Appendix C.3.

C.1.2 Gaussian example

In this section, we provide the specifics of the Gaussian example from Section 4.2, where we showed how restricting data from auxiliary Gaussians can improve the resulting classifier.

Data generation. The target data is generated from 2-dimensional Gaussians with a class-conditional distribution of

$$p(x|y) \sim \mathcal{N}(y \cdot \mu, I) \tag{C.1}$$

for $\mu = (2, 0)$. The auxiliary data is also generated from 2-dimensional Gaussians with the same mean but different covariance. Specifically, the auxiliary data has a conditional distribution of

$$p_{aux}(x|y) \sim \mathcal{N}(y \cdot \mu, \Sigma) \tag{C.2}$$

for $\Sigma = R\,\mathrm{diag}([s, 1/s])R^T$ where $s = 4$ and R is the standard rotation matrix

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{C.3}$$

for $\theta = \pi/8$. In other words, p_{aux} is equivalent to a rotated and scaled version of p with the same mean. This rotation and scaling amounts to the bias introduced from the auxiliary dataset.

We generate $n = 12$ datapoints $\mathcal{D}_{target} = \{x_i, y_i\}$ from the target distribution split evenly between the two classes, and generate $m = 2 \ldots 200$ datapoints $\mathcal{D}_{auxiliary} = \{x'_i, y'_i\}_{i=1\ldots m}$ from the auxiliary distribution, also split between the two classes.

Clipping auxiliary data. To clip the auxiliary data and get a target-aligned subset, we simply throw away all datapoints that lie outside a high confidence region of the Gaussians estimated from the target dataset. Specifically, use the following steps:

1. We estimate the class conditional mean and covariance matrix of the target dataset. Specifically, we calculate the sample mean μ_y and covariance Σ_y of each class in the target data, $\{x_i : y_i = y\} \subseteq \mathcal{D}_{target}$. This gives us an estimated distribution $\hat{p}(x|y) = \mathcal{N}(\mu_y, \Sigma_y)$ for the target distribution.

2. We calculate the Mahalanobis distance of each auxiliary datapoint to the estimated Gaussian of its class $\hat{p}(x|y)$, i.e.

$$MD(x, y) = \sqrt{(x - \mu_y)^T \Sigma_y^{-1} (x - \mu_y)} \tag{C.4}$$

3. We then discard all auxiliary points whose Mahalanobis distances lies outside of a certain threshold. Specifically, the resulting subset is the following:

$$\mathcal{D}_{restricted} = \{(x', y') : MD(x', y') \leq r \text{ for } (x', y') \in \mathcal{D}_{auxiliary}\} \tag{C.5}$$

for $r = 3$.

Fitting the linear classifier. In the experiment shown in Figure 4-2, we measure how adding auxiliary data affects the performance of a linear classifier with a small amount of target data. Similar to the previous experiment on augmenting CIFAR10 data with ImageNet data, we find that adding too much auxiliary data hurts accuracy in this Gaussian example.

Specifically, we add data from the auxiliary dataset to the training data in various amounts from $2 \ldots 100$, and fit a linear classifier. We use the default `LogisticRegression` function from Scikit-learn. Test error is measured over an independent, random sample of 1000 additional samples from the original training distribution p. Error bars are averaged over five random seeds. We also compare to the analytically optimal classifier, which is $h_{opt}(x_1, x_2) = \begin{cases} 1, & \text{if } x_1 \geq 0 \\ -1, & \text{otherwise} \end{cases}$.

C.1.3 Training on biased 3DB data

Inspired by the common story of machine learning models picking up on biases between animals and their backgrounds [RSG16], we construct a binary classification task with explicitly planted biases. In contrast to the Gaussian example, we generate a more realistic setting of cats and dogs using the Blender renderer and the 3DB framework [Lec+21b]. In contrast to the CIFAR10 and ImageNet example, where the exact ImageNet patterns that hurt CIFAR10 generalization are unknown, we can instead construct a setting with an explicit bias. Since this is a controlled experiment, it enables us to do the following:

1. Validate in isolation that biased auxiliary data can hurt performance in situations more complex than the Gaussian example.

Figure C-1: The full 15k-image auxiliary dataset (Left) contains a background bias where dogs appear more frequently outdoors and cats appear more frequently indoors. We only show dog images here to highlight the background bias. The unbiased 3k-image subset (Middle) does not have any background bias. Training on the unbiased subset results in better test accuracy on the unbiased test set (Right), showing that more data is not always more helpful.

2. Demonstrate how an "ideal" target-aligned subset can remove this bias and improve performance over using all of the original auxiliary data

Data generation. We begin by constructing an auxiliary dataset with a background bias. Specifically, we generate 50 images for each (3D-model, background) pairing. We use 15 cat 3D-models and 15 dog 3D-models for the training data. We choose 1 indoor background and 1 outdoor background to be shared by both classes. Then, we choose 8 new indoor backgrounds to appear with cats, and 8 new outdoor backgrounds to appear with dogs. We hold out 1 indoor background, 1 outdoor background, and three 3D-models per class for the test set.

Our resulting training set has a strong background bias; 90% of cats are indoors, and 90% of dogs are outdoors. Meanwhile, the test set consists of two entirely new backgrounds and is unbiased: half of the images use an outdoor background and the other half use an indoor background irrespective of the class. We show examples of generated train and test dog images on the left and right panels of Figure C-1 respectively.

The ideal target-aligned subset. The ideal subset of auxiliary data is one that does not introduce a background bias. Specifically, we can accomplish this goal by choosing images that use the shared indoor and outdoor backgrounds as the “ideal” subset. Then, cats and dogs are equally likely to appear on either indoor or outdoor backgrounds. This ensures that the backgrounds do not confer any useful correlations for predicting the class, removing the background bias. Examples of the ideal dog subset are shown shown in the middle of Figure C-1.

Fitting a classifier. We train ResNet-18 models on both the full auxiliary dataset and the target-aligned subset and evaluate their test performance on the unbiased test set. Most training details are the same as the training details for the vision experiments in the rest of the paper, which are described in Appendix C.3.3. The main differences are the following.

- We train for 120 epochs for the full 15k-image auxiliary dataset, and we train for 600 epochs for the unbiased 3k-image subset (so that the total number of SGD iterations across both settings is the same).
- We use a learning rate of 0.1, and we only do a learning rate drop one time, halfway through training.

Indiscriminately training on the full, biased dataset results in a test accuracy of $90.4\% \pm 2.1$. On the other hand, training on the smaller, target-aligned subset improves the accuracy by over 7% to $97.6\% \pm 0.6$, respectively. This suggests that when a bias exists in the auxiliary data, it can be harmful to use the full dataset as opposed to just a target-aligned subset.

C.2 How to project datasets

In this section, we describe in detail the framework that we developed to solve the dataset projection problem. Specifically:

1. We provide a brief background on the active set optimization method that we build upon (Appendix C.2.1).
2. We describe the exact algorithms of our active set (Appendix C.2.2) and PGD (Appendix C.2.3) solvers.
3. We provide a more detailed description of how we obtain source distributions (Appendix C.2.4).
4. We outline specifically how we compute the distance metric between datasets (Appendix C.2.5).

C.2.1 Overview of active set framework

We build upon an extensive line of work on active set optimization frameworks for minimization over the simplex [Ber82; BM02; Brá+17; Cri+17; FFK98; HZ06]. This

family of algorithms can solve problems of the following form:

$$\min_{\alpha} f(\alpha) \text{ subject to } \sum_{i=1}^{k} \alpha_i = 1, \quad \alpha_i \geq 0 \tag{C.6}$$

where f is the objective function to be minimized over the simplex. Existing algorithms have favorable properties, such as global convergence to stationary points in non-convex settings [Cri+20].

Our algorithm is an adaptation of the framework from Cristofari et al. [Cri+20] to the stochastic and non-differentiable setting, in order to solve the dataset projection problem from Equation (4.3).

As a brief overview, the simplex algorithm from Cristofari et al. [Cri+20] consists of two main steps. First, the method calculates an estimate of the active set, and performs a single coordinate update based on this update. Second, the method then calculates a search direction, and performs a line search in this direction. With these updates, Cristofari et al. [Cri+20] show that this method has linear convergence to a stationary point for non-convex problems with simplex constraints.

However, the method as originally proposed was not intended for stochastic and non-differentiable optimization. Indeed, if we attempt to directly apply this algorithm to solve the dataset projection from Equation (4.3), we run into two main problems. First, the estimate of the active set can vary due to the stochasticity in the objective when sampling a dataset. This causes coordinates to rapidly fluctuate between being in and out of the active set between iterations, which prevents the method from converging. Second, the method requires a gradient calculation for both the active set estimate and the search direction. However, the objective from Equation (4.3) is non-differentiable with respect to the simplex variables.

C.2.2 Extending the active set framework

In order to address these two problems and solve the dataset projection problem, we modify the active set simplex algorithm from Cristofari et al. [Cri+20] in two ways.

Damped updates for estimating the active set. In order to stabilize the algorithm, we need to stop the active set estimates from fluctuating too much. To do this, we instead introduce a soft active set estimate which varies between $[0, 1]$ for each coordinate. Then, to determine if a coordinate is in the active set or not, we simply threshold the soft estimate at 0.5: coordinates with a soft estimate greater

than 0.5 are in the active set, and coordinates below are not. Finally, each iteration we update the soft estimate with a momentum-style update to damp the variability at each iteration. This stabilizes the soft estimate, and creates a more consistent active set across iterations that enables the method to converge. The specific steps for calculating and updating the soft active set are in lines 5-11 of Algorithm 2. The rest of the active set update remains the same as in Cristofari et al. [Cri+20].

Algorithm 2 The damped active set update $\texttt{DampedUpdate}(\alpha, A, g)$ for simplex optimization, which applies a momentum update to the active set A and performs a coordinate step to the iterate α.

1: *// Select coordinate to update and calculate hard active set for current iterate*
2: $j = \arg\min_i \{g_i\}$
3: $\tilde{A} = \{i : \alpha_i \leq \epsilon g^T (e_i - \alpha)\}$
4:
5: *// Apply damped update to the soft active set estimate to stabilize the current active set*
6: $A = \beta A$
7: **for** $i \in \tilde{A}$ **do**
8: $\quad A = A + (1 - \beta)$
9: **end for**
10: $\tilde{A} = \{i : A_i > 0.5\}$
11: $N = \{i : \alpha_i > \epsilon g^T (e_i - \alpha^t), i \neq j\}$
12:
13: *// Apply coordinate update*
14: Set $\tilde{\alpha}_{\tilde{A}} = 0,\ \tilde{\alpha}_N = \alpha_N,\ \tilde{\alpha}_j = \alpha_j + \sum_{h \in \tilde{A}} \alpha_h$
15: **return** $(\tilde{\alpha}, A)$

Numerical estimation for gradients. The active set algorithm has two updates that require gradient directions. However, the dataset projection problem is non-differentiable due to the sampling procedure. Instead, we use numerical estimation to calculate the gradient. Specifically, we use the central difference formula for estimating the gradient. Furthermore, in the second step of the framework, we use a stale gradient from the previous update to keep computational overhead low. This contrasts with the original framework, which uses a fresh gradient for the iterate after the initial active set update step. Otherwise, the remainder of the second update is the same as in Cristofari et al. [Cri+20], and is shown in Algorithm 3.

Algorithm 3 The search direction update $\texttt{SearchUpdate}(\alpha, A, g)$, which uses a stale gradient and line search to update the iterate α with line search parameters (γ, δ). Here, $\mathcal{P}_\Delta$ is the projection operator onto the simplex.

// Compute a search direction that obeys the active set estimate
$d = \mathcal{P}_\Delta(\alpha - sg) - \alpha$
for $i : A_i > 0.5$ **do**
  $d_i = 0$
end for

// Armijo line search
if $g^T d < 0$ **then**
  $\lambda = 1$
  while $f(\alpha + \lambda d) > f(\alpha) + \gamma\lambda g^T d$ **do**
    $\lambda = \delta\lambda$
  end while
else
  $\lambda = 0$
end if
return $\alpha + \lambda d$ *// Second update*

Algorithm 4 Projected gradient descent (PGD) solver for projecting datasets with step size γ

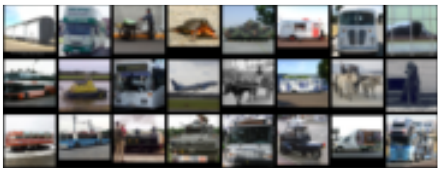
$\alpha_i^0 = 1/k$ for $i = 1 \ldots k$ *// Initialize feasible point*
for $t = 0, 1, \ldots$ **do**
  $g = \nabla f(\alpha^t)$ *// Estimate numerical gradient of the dataset projection objective*
  $\alpha^t = \texttt{Proj}_\Delta(\alpha^t + \gamma \cdot g)$ *// Gradient update with projection onto simplex*
end for

C.2.3 Projected gradient descent

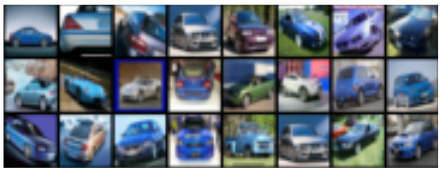
As an another approach, we can apply projected gradient descent (PGD) to solve Equation 4.3. Similarly to the active set approach, we can use numerical gradient estimates to directly optimize α, and project α back to the simplex after every step. Although PGD does not have the global convergence guarantee that the active set method does, PGD is an simple technique for solving constrained optimization problems in deep learning settings. The PGD solver is shown in Algorithm 4.

C.2.4 Determining source distributions

To apply dataset projection to real-world datasets, we need to divide an auxiliary distribution into multiple source distributions to search over. We do so primarily



(a) STL10 cluster.



(b) CIFAR10 "blue" cluster of the car class.

Figure C-2: Examples of clusters created when using a robust representation and unsupervised learning to separate auxiliary datasets into source distributions. (a) A cluster from the STL10 dataset, which has no label information and (b) a cluster from the CIFAR10 car class, which contains mostly blue images.

with two main strategies: either by using existing labels when available, or generating source distributions with unsupervised clustering methods.

Using label information. In some cases, datasets may already have existing class or attribute labels. For example, suppose that the target dataset is CIFAR10 and the auxiliary dataset is ImageNet. When projecting the CIFAR10 *dog* class onto ImageNet, each ImageNet class that is a dog breed can be considered a separate source distribution. Note that the labels for the auxiliary dataset do not need to strictly line up with labels in the target dataset. For example, the Emoji dataset has labels corresponding to emoticons such as a Christmas tree, a camera, and the sun. These labels may not have an obvious correspondence to the labels of other datasets such as the five star ratings from the Yelp dataset. Our framework can simply use the auxiliary labels as a way to divide the data into source distributions, and then project each target class onto the auxiliary data to re-assign labels for the target task.

Unsupervised data. Sometimes no label information is available. For example, suppose we want to project CIFAR10 classes onto the unsupervised STL10 dataset. The STL10 dataset is completely unlabeled and can contain images of objects that are completely irrelevant to the base CIFAR10 class. In such cases, we can use unsupervised clustering techniques to find source distributions. For our vision settings, we use k-means clustering based on robust representations because it has been shown to lead to good visual alignment within each cluster [Eng+19a]. We show several examples from an STL10 cluster in Figure C-2.

Combined label and clustering. The previous two approaches for dividing an auxiliary distribution into source distributions can be combined to obtain even finer-grained source distributions. This can enable source distributions to capture a narrower population, which can then be used to find more accurate projections of the target dataset. For example, when using CIFAR10 as an auxiliary dataset, we can not only break the dataset into subsets corresponding to its classes, but also use clustering techniques to break each class subset into finer-grained categories. We show an example of this in Figure C-2, which shows a blue-car cluster within the CIFAR10 car class.

C.2.5 Distance metric for dataset projection

In this section, we expand on the specific distance metric used in the objective of the dataset projection problem. This objective is the fundamental metric that guides the solvers towards "target-aligned" subsets.

We use what is known as the unbiased estimate for the Maximum Mean Discrepency [Gre+12], as defined in Equation C.7:

$$\mathrm{MMD}(x, y) = \left[\frac{1}{m^2} \sum_{i,j=1}^{m} k(x_i, x_j) + \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(x_i, y_j) + \frac{1}{n^2} \sum_{i,j=1}^{n} k(y_i, y_j) \right]^{\frac{1}{2}} \quad \text{(C.7)}$$

where $k(x, y) = \exp(-\alpha \cdot \frac{1}{2}\|x - y\|_2^2)$ is a kernel with hyperparameter α.

This score was originally proposed as a way to distinguish whether two distributions were different or indistinguishable via a two-sample hypothesis test. In our setting, we consider the datasets as sampled from the source and target distributions, and use the MMD score as the metric for distance without the hypothesis testing component.

However, these MMD scores run into numerical issues when the feature dimension of the dataset is extremely large. These issues make it not possible for the MMD score to distinguish between, for example, two different image distributions. However, Rabanser, Günnemann, and Lipton [RGL19] found that calculating the MMD score in the encoded representation space of a neural network was significantly more effective than the original feature space, and in fact was the most powerful way to perform the traditional two-sample hypothesis test for detecting distribution shift.

Thus, in this work we use the MMD score in the feature space of a neural network, as suggested by Rabanser, Günnemann, and Lipton [RGL19]. For the vision setting we use a randomly initialized encoder network, which found to be as effective as

pretrained variants [RGL19]. For the language setting we use a pretrained BERT model from HuggingFace [**DBLP:journals/corr/abs-1810-04805**]. To select the hyperparameter α, we measured the MMD statistic over a range of possible α and selected one that resulted in a statistically significant hypothesis test in distinguishing the original auxiliary data from the target data. For the vision settings, this turned out to be $\alpha = 500$, and for the language settings this was $\alpha = 0.01$.

C.3 Experimental details

In this section, we provide a complete description of the datasets used and the experimental setup for projecting datasets and training models.

C.3.1 Datasets

In this section, we describe each of the datasets used in the experiments of Section 4.4. A table summary of how we assigned the auxiliary dataset, the target dataset, and the held-out test set of the target dataset is shown in Table C.1.

Our computer vision experiments use four standard classification datasets — ImageNet [Rus+15], Oxford-IIIT Pet [Par+12], CIFAR10 [Kri09], and STL10 [Ada11] — and one synthetic dataset generated using 3DB [Lec+21b]. Our natural language experiments use five sentiment analysis datasets — Stanford Sentiment Treebank [Soc+13], Emotion Recognition [Moh+18; Bar+20], Emoji Prediction [Bar+18; Bar+20], Yelp Reviews [ZZL15], and DailyDialog Act Corpus [Cha+20].

Personally identifiable information or offensive content. The datasets we use are all open source and widely-used in the community. Nonetheless, there is a non-zero chance that the data contains personal information or offensive content. For example, unsupervised datasets such as the images in STL10 may contain such images since the dataset has, by definition, not been supervised. The sentiment analysis datasets may contain negative sentences that are possibly offensive towards people, such as racist messages posted on Twitter or ad-hominen attacks in negative Yelp reviews. To our knowledge, we are not aware of any such data within these datasets, and have not explicitly encountered them in our research.

	Target	Test	Auxiliary	Source Distribution
ImageNet	-	-	Train Set	Class labels
CIFAR10	Train Set (1000)	Test Set (1000)	Train Set	Combined
Oxford-IIIT	All but test set	Random 250 per class	All but test set	Class labels
STL10	Train Set (500)	Test Set (800)	Unlabeled data	Clustering
3DB	Validation Set (1000)	Test Set (1000)	Train Set	Attribute labels
SST	Validation set	Test set	Train Set	Sentiment scores
Emoji	Validation set	Test set	Train Set	Emoji labels
Emotion	Validation set	Test set	Train Set	Emotion labels
Yelp	Train set (second half)	Train set (first half)	Test set	Review scores
DailyDialog	Validation set (100)	Test set	Train Set	Emotion labels

Table C.1: We summarize the portions of each dataset used when the given dataset is chosen as the target, test, or auxiliary dataset. For example, when CIFAR10 is the target distribution, we project 1000 images from each class of CIFAR10's training set onto the auxiliary dataset. When testing a model (e.g., one trained on the projected dataset) on CIFAR10, we test on the held-out test set of CIFAR10. When CIFAR10 is used as an auxiliary dataset, we use both the CIFAR10 class labels *and* unsupervised clustering to get source distributions.

Image classification

We standardize all of the image classification tasks into a single unified setting. Specifically, all images are resized to have the same resolution (32×32), and for each scenario, we use classes that are shared between datasets. For example, when projecting Oxford-IIIT onto CIFAR10, we choose the auxiliary dataset to be the the CIFAR10 classes *cat* and *dog*, and project the corresponding classes in Oxford-IIIT onto the cluster-based sources of the cat and dog classes.

ImageNet. ImageNet is the largest vision dataset we consider, and it is substantially larger (both in size and number of classes) than all the other vision datasets. Hence, we use ImageNet primarily as an auxiliary dataset. Specifically, we use the training set of ImageNet.

To get source distributions for a particular class, we find that class in the WordNet hierarchy and use every descendant ImageNet class as a source for the class. For example, for the "dog" class, we find the dog node in the WordNet hierarchy and use all the various breeds of dogs under this node as individual source distributions for the dog class.

CIFAR10. The target and test datasets for CIFAR10 are random subsets of the training and test data respectively. Specifically, we randomly subsample 1000 examples from each class to match the ImageNet dataset.

When CIFAR10 is used as an auxiliary dataset, we split the training data into sources with a combination of class labels and unsupervised clustering. Specifically, we first encode each example into a robust feature representation. We do this with an adversarially robust ImageNet classifier [Eng+19c], as this is known to be more aligned with human visual features [Eng+19a]. We used the open-source pre-trained ℓ_2 robust model for $\epsilon = 3$. For each class, we then cluster the training data into 16 clusters using the robust representations and the `MiniBatchKMeans` function from Scikit-learn. These 16 clusters form the source distributions for the corresponding CIFAR10 class.

Oxford-IIIT Pet. The Oxford-IIIT Pet dataset is originally not split into a training set and test set. Thus, we randomly select 250 cats and 250 dogs for use as a test set, and use the remaining data as the train set.

When Oxford-IIIT Pet is used as auxiliary data, we use the entire Oxford-IIIT Pet dataset, other than the 250 cats and 250 dogs that were set aside as the test set. Specifically, we use the fine-grained labels of dog breeds and cat breeds to divide the dataset into source distributions. In other words, the dog breeds form the source distributions for the dog class, and the cat breeds form the source distributions for the cat class. In total, there are 12 cat breeds and 23 dog breeds, each with (on average) 200 images per cluster.

STL10. The target and test datasets are random subsets of the training and test data, with 500 and 800 examples each, respectively.

We use the unlabeled data in STL10 as the auxiliary data. Similar to CIFAR10, since there are no sources, we use unsupervised clustering to generate the sources. However, in this case since there are no class labels, we rely purely on unsupervised clustering. Specifically, we use the same methodology as for CIFAR10, but instead create 160 clusters total that are not necessarily separated by class (instead of 16 clusters for each of 10 classes).

3DB. 3DB is a synthetic rendering platform that allows the user to generate synthetic image data by specifying a 3D-model, an HDRI background, and a variety of other parameters such as the camera location, camera and object orientation, and the scene

brightness [Lec+21b]. In our case, we generate a "CIFAR10-like" synthetic dataset with this renderer. Specifically, we use this framework with free, Blender-compatible 3D-models that we find online from SKETCHFAB.COM, as well as free HDRI backgrounds from POLYHAVEN.COM.

We first describe how we make the training set for 3DB. For each class that we want to generate (e.g., *airplane* or *dog*), we find 15 high-quality 3D-models and 25 HDRI backgrounds. We also choose 4 distinct camera settings in terms of height and zoom — in particular, we allow the camera to have a zoom factor of either 1.5 or 3.0, and we choose the camera height to be 0 or 1[1]. This gives a total of 1500 unique triplets of (3D-model, HDRI background, camera setting) per class. For each such choice of 3D-model, background, and camera setting, we generate 1000 different images by rotating the camera around the object by a random angle and randomly varying the brightness of the generated image.

Our validation and test sets use similar 3DB parameters, but they are generated independently with different 3D models and HDRI backgrounds. Specifically, we choose 3 new 3D-models per class and find 10 new HDRI backgrounds in total. For each class, we randomly choose 4 out of the 10 backgrounds to be associated with that class, in order to add a realistic background bias. This helps us mimic real image datasets like CIFAR10, where there also exists a background bias. For each class, we use exactly one of the 4 possible height and zoom settings for the camera instead of using all 4. These settings are shown in Table C.2.

Class	Height	Zoom
Airplane	-1	1.5
Automobile	0	3.0
Bird	0	3.0
Cat	0	1.5
Deer	1	1.5
Dog	0	1.5
Frog	1	3.0
Horse	0	1.5
Ship	0	3.0
Truck	0	1.5

Table C.2: Height and zoom values (randomly) chosen for each class in the 3DB validation and test sets.

Using this combination of 3D-models and backgrounds for each class, we generate

[1]We choose camera heights to be 0 or -1 instead in the case of the two classes *airplane* and *bird* because those classes are usually photographed from below rather than from above.

1000 images for each validation set and test set.

Thus, we now associate the train, validation, and test sets of 3DB with the dataset projection setting. When using 3DB as the auxiliary dataset, we use the training set, which has 1500 unique labeled source distributions per class. Each source corresponds to a specific triplet of (3D-model, HDRI background, camera setting). When using 3DB as the target dataset, we use the validation set of 3DB, and when using 3DB as the test set, we use the test set of 3DB.

Sentiment analysis.

Similar to the computer vision setting, we standardize all of the language datasets into a single unified setting. Specifically, all datasets are loaded via the `Datasets` package built by HuggingFace. All sentences are tokenized with the `BertTokenizerFast` tokenizer from the pre-trained `bert-base-case` model on `https://huggingface.co/` with maximum length padding and truncation. All datasets are subsampled to 100 examples total.

Note that, since each sentiment analysis task has a different goal, we cannot canonicalize all scenarios to predict the same set of labels like we could in the computer vision setting. For example, it is unclear at what point a 1 through 5 star rating on a Yelp review translates to positive or negative sentiment in the Stanford Sentiment Treebank.

Stanford Sentiment Treebank (SST). The target and test datasets for SST are random subsets of the corresponding validation and test splits. When SST is used as an auxiliary dataset, we take the SST train set and discretize the sentiment scores into 10 bins of size 0.1. Each bin forms a source distribution for the auxiliary SST dataset. The dataset is available at `https://huggingface.co/datasets/sst`.

Emoji. The Emoji dataset is a subset of the `tweet_eval` dataset on HuggingFace. Specifically, the target and test datasets for Emoji are random subsets of the corresponding validation and test splits. When used as an auxiliary dataset, we use the emoji labels to divide the dataset into 20 source distributions. The dataset is available at `https://huggingface.co/datasets/tweet_eval`.

Emotion. The Emotion dataset is a subset of the `tweet_eval` on HuggingFace. Specifically, the target and test datasets for Emotion are random subsets of the corresponding validation and test splits. When used as an auxiliary dataset, we use

the emotion labels to divide the dataset into 4 source distributions. The dataset is available at `https://huggingface.co/datasets/tweet_eval`.

Yelp. The Yelp dataset does not have a validation set, so we split the training data to get a target dataset. Specifically, the target dataset is the second half of the Yelp training split, and the test dataset is the test split.

The Yelp auxiliary data comes from the first half of the Yelp training split. This subset is further divided into source distributions by using the review scores of each example. Specifically, we divide the subset into 5 source distributions corresponding to 1 star reviews through 5 star reviews. The dataset is available at `https://huggingface.co/datasets/yelp_review_full`.

DailyDialog. The DailyDialog dataset is the `dyda_e` subset of the `sillicone` dataset, a collection of resources designed for spoken language. We make one modification to the dataset: since some of the classes have fewer than 100 examples while others have thousands, we subset the dataset to classes $(3, 4, 6)$ (corresponding to happiness, no emotion, and surprise) which have sufficient representation. We then balance the classes to have no more than 100 examples per class.

The target and test datasets thus come from the corresponding validation and test splits. When used as auxiliary data, we divide the training set into 3 source distributions corresponding to the emotion labels. The dataset is available at `https://huggingface.co/datasets/silicone`.

C.3.2 Dataset projection benchmark

In this section, we provide the full experimental setup for projecting datasets, and training neural networks to either evaluate the projection or to use the projection as dataset augmentation.

Experimental setup

For each experiment we choose two datasets, one auxiliary dataset and one target dataset. For final evaluation, we evaluate on the test set of the target dataset.

At a high level, we first use active set or PGD to project each class of the target dataset onto the auxiliary dataset as described in Appendix C.2. After finding the projected dataset, we validate its effectiveness by training deep learning models on it and reporting the mean and standard deviation of the test accuracy over 5 training

runs. These training runs can be in combination with the target dataset (to measure augmentation performance) or without the target dataset (to validate the projection method).

Baselines

We compare three different methods of choosing a subset of the auxiliary data to determine the most effective one. In the rest of this section, we refer to them as follows.

- **AS-PD:** We find the projected dataset via active set.
- **PGD-PD:** We find the projected dataset via PGD.
- **Random** (Baseline): We use a equally-sized subset of the auxiliary dataset chosen uniformly at random from the sources. This baseline isolates the importance of finding the "right" subset of the auxiliary dataset.

C.3.3 Training specifics for image classification

Our computer vision models are trained with settings that are standardized across all experiments in this paper.

- We use a standard ResNet-18 architecture [He+16].
- We always randomly subsample 100 examples for each dataset we use during model training. In other words, all training datasets (e.g. the target dataset, the projected dataset, the random portion of the auxiliary dataset) have size 100.
- We use SGD to train for 100 epochs. We set the batch size to 32, the learning rate to 0.1 (with learning rate drops every 33 epochs), the momentum parameter to 0.9, and the weight decay to 5e−4.
- For data augmentation of the regular training runs (the ones not labeled as "DA"), we use random crop and random horizontal flip. When using stronger data augmentation (the ones labeled as "DA"), we use the default settings of TrivialAugment [MH21].
- For each experiment, we repeat each training run with 5 different seeds in order to get the mean and standard deviation of the test accuracies.

C.3.4 Training specifics for sentiment analysis

Our language models are trained with settings that are standardized across all experiments in this paper.

- We use a standard BERT architecture [He+16], specifically the pretrained `bert-base-cased` from HuggingFace.
- We use the HuggingFace `Trainer` to fine-tune with the default arguments.
- Data augmentation via backtranslation is done via the French language using the open source translation Opus-MT models `Helsinki-NLP/opus-mt-en-ROMANCE` and `Helsinki-NLP/opus-mt-ROMANCE-en` from HuggingFace [TT20].
- For each experiment, we repeat each training run with 5 different seeds in order to get the mean and standard deviation of the test accuracies.

C.3.5 Projecting dataset specifics

When projecting datasets, we use the same following settings for both AS-PD and PGD-PD, most of which are typical settings from the optimization literature:

- We use $\epsilon \in \{1, 0.1, 0.01, 0.001, 0.0001\}$
- We use a learning rate of $\gamma = 1$
- We use a tolerance level of 10^{-6}. If an iteration does not change by more than this amount, we terminate the algorithm
- We run for a maximum of 1000 iterations
- We use a momentum parameter of $\beta = 0.9$ for the soft active set update
- We use $(\delta, \gamma) = (0.9, 0.9)$ as parameters for the Armijo line search.

C.3.6 Compute requirements

All experiments across all scenarios can in theory be run with a single 1080Ti with 1 CPU core (double-threaded). Most projection and training runs can be completed within a few hours, depending on the number of source distributions. For the projections onto the 3DB auxiliary dataset, we use multi-processing with 4 CPU cores and 2 GPUs to accelerate the projection process due to the large number of source

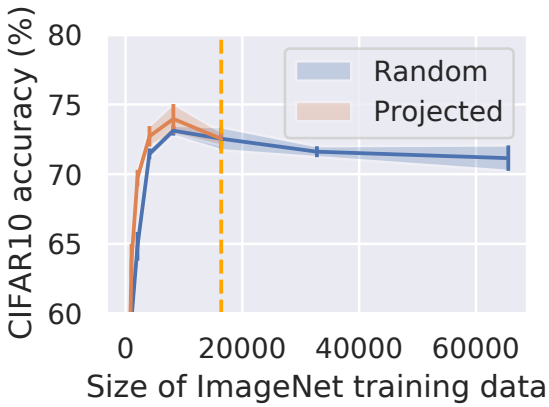


Figure C-3: Comparing test accuracies after projecting CIFAR10 on ImageNet and training on either (1) the projected dataset (2) a random subset of the full auxiliary dataset. Even though the entire projected dataset (marked by the dotted line) is a strict subset of the full auxiliary dataset, training on just the projected dataset gives higher test accuracy than training on the full auxiliary dataset (marked by the datapoint at the far right of the blue line of the graph).

distributions, which makes the numerical gradient estimate expensive. Due to the large number of scenarios and experiments, we run our scenarios in parallel across a cluster consisting of 56 GPUs (1080Ti). In total, there are 36 scenarios, 3 training experiments per scenario, 4-5 methods per experiment, and 5 random seeds, for a total of 2340 models trained.

C.4 Additional experimental results

We include additional experimental results not presented in the main paper here.

Comparing AS-PD with training on the full auxiliary dataset. Figure C-3 compares training on the projected dataset (AS-PD) with training on the full auxiliary dataset. The projected dataset is smaller, and training on an equally-sized random subset of the full auxiliary dataset results in worse performance at all dataset sizes up to the total size of AS-PD. Using even more auxiliary data, including the full auxiliary dataset, only degrades performance further, due to the biases present in the auxiliary dataset. In this case, training only on the smaller projected dataset is beneficial compared to training on all available auxiliary data, even though the full auxiliary dataset has strictly more datapoints than the projected dataset.

Aux. \Target	CIFAR10	Oxford-IIIT Pet	STL10	3DB
ImageNet	2.54	1.64	1.40	−0.10
CIFAR10	1.83	2.84	−1.68	5.49
Oxford-IIIT Pet	2.83	−2.20	0.82	−2.74
STL10	1.73	2.56	2.03	0.90
3DB	1.33	0.88	2.72	2.94

Table C.3: Improvement of active set over PGD in approximating target datasets, measured by the difference in test accuracy between a model trained on AS-PD and a model trained on PGD-PD. Both methods improve approximation over uniformly random sampling from the full auxiliary dataset, but active set is slightly better than PGD (that is, the measured difference is positive) in 16 out of 20 settings.

Comparing AS-PD and PGD-PD. In Table C.3, we examine the differences between the projected datasets found using PGD and active set. Overall, AS-PD performs slightly better, and both methods of solving dataset projection outperform sampling uniformly at random from the auxiliary dataset.

Quantitative comparison of AS-PD and the random baseline via distance metrics. In Figure 4-6, we showed that when projecting Oxford-IIIT onto ImageNet, the projected dataset is much closer in distance to the target dataset than the auxiliary dataset is. We showed this for the MMD distance, which we explicitly use active set and PGD to opimize for, as well as for 3 alternate distance metrics discussed in [ZLB17] — the contrast, luminance, and random filter response (RFR). For completeness, we produce the same plot for every possible choice of auxiliary dataset and target dataset. In the majority of cases, such as in Figure C-4, the quantitative distance metrics add further confirmation that the projected dataset is more effective at approximating the target dataset than the original auxiliary dataset. In a few cases, although dataset projection is able to decrease the MMD distance to the target dataset, it does not always decrease the other distance metrics.

C.4.1 When projected datasets can and can't help

Surprisingly, projected datasets that approximate the target better do not always lead to better results when augmenting the target. When using 3DB as the auxiliary data, projected datasets are better at approximating the target, but the Random baseline is often better at augmenting the target. This may be because the 3DB dataset is generated by pairing random backgrounds with random 3D-models of each class—thus,

training on the baseline dataset encourages invariance to background and different 3D-models, which may actually be more beneficial for generalization. Understanding exactly what properties of a projected dataset make it most useful for augmentation is an interesting future direction.

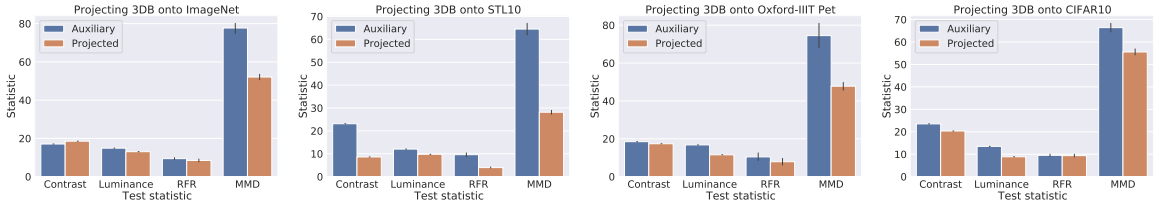


Figure C-4: 1D distance metrics showing how close the projected and auxiliary datasets are to the target dataset (lower is better). 3DB is the target dataset.

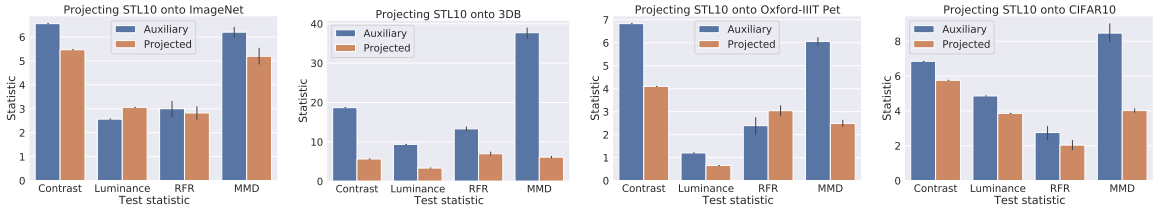


Figure C-5: 1D distance metrics showing how close the projected and auxiliary datasets are to the target dataset (lower is better). STL is the target dataset.

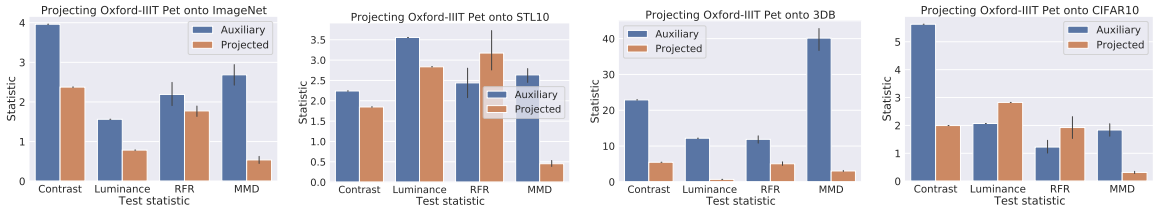


Figure C-6: 1D distance metrics showing how close the projected and auxiliary datasets are to the target dataset (lower is better). Oxford-IIIT Pet is the target dataset.

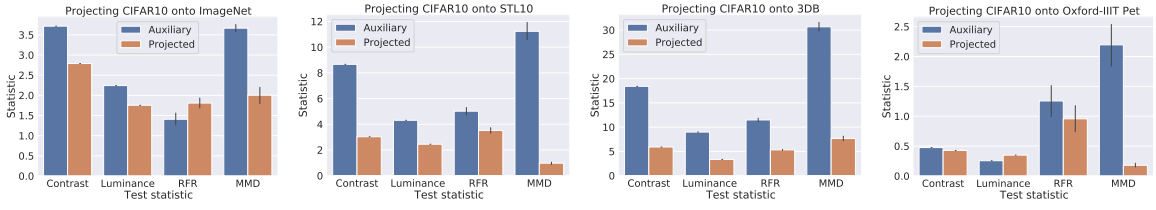


Figure C-7: 1D distance metrics showing how close the projected and auxiliary datasets are to the target dataset (lower is better). CIFAR10 is the target dataset.

Auxiliary	Target	Random	PGD-PD	Not PGD-PD	AS-PD	Not AS-PD
ImageNet	CIFAR10	33.9 ± 0.4	35.5 ± 0.7	27.3 ± 0.7	**37.0** ± 1.0	30.7 ± 0.8
Oxford-IIIT Pet		53.6 ± 0.8	50.5 ± 2.9	**54.6** ± 1.9	53.0 ± 4.1	54.5 ± 1.1
STL10		12.2 ± 1.9	28.3 ± 0.7	10.7 ± 2.5	**33.8** ± 1.2	13.1 ± 1.5
3DB		22.6 ± 1.0	24.3 ± 0.3	22.2 ± 0.8	**25.4** ± 0.8	22.6 ± 0.7
ImageNet	Oxford-IIIT Pet	50.1 ± 2.5	51.8 ± 1.5	48.0 ± 2.1	**55.4** ± 1.6	51.8 ± 2.4
CIFAR10		53.0 ± 1.0	49.4 ± 1.5	49.4 ± 1.7	**55.6** ± 2.1	51.3 ± 1.9
STL10		51.6 ± 2.4	**54.0** ± 1.9	49.2 ± 0.8	53.3 ± 2.1	50.9 ± 2.3
3DB		52.0 ± 2.9	49.0 ± 1.9	48.8 ± 2.3	**53.7** ± 1.6	49.1 ± 1.1
ImageNet	STL10	37.4 ± 1.0	37.7 ± 0.5	37.0 ± 0.5	**38.3** ± 1.1	37.3 ± 0.8
CIFAR10		41.2 ± 1.2	34.8 ± 4.3	35.5 ± 1.1	**38.8** ± 0.5	38.1 ± 1.5
Oxford-IIIT Pet		**55.5** ± 1.5	50.1 ± 3.5	49.7 ± 2.2	55.1 ± 0.9	52.2 ± 1.2
3DB		24.6 ± 1.2	26.3 ± 1.3	21.0 ± 1.1	**30.4** ± 1.5	24.5 ± 1.9
ImageNet	3DB	27.0 ± 5.0	22.2 ± 1.7	16.3 ± 2.4	**29.5** ± 1.7	22.1 ± 1.6
CIFAR10		30.8 ± 3.7	32.7 ± 4.5	19.6 ± 3.1	**35.4** ± 1.0	23.8 ± 0.9
Oxford-IIIT Pet		66.8 ± 8.8	68.4 ± 14.4	55.9 ± 12.8	**71.2** ± 8.0	56.3 ± 12.1
STL10		12.6 ± 4.0	34.8 ± 3.5	5.7 ± 4.0	**37.7** ± 4.0	12.2 ± 3.4

Table C.4: Approximating target datasets with auxiliary data. In this experiment, we train on auxiliary data and test on target data. Higher test accuracy corresponds to better approximation quality. AS-PD typically performs the best, and PGD-PD also typically outperforms Random.

C.4.2 Experiments for vision benchmark

We present the complete set of results for our image classification scenarios. In Table C.4, we record the target-alignment of our projected datasets as measured by training on the projected dataset. In Table C.5, we record the performance of training on the target dataset augmented with projected data. In Table C.6, we record similar results but combined with TrivialAugment data augmentation.

C.4.3 Experiments for language benchmark

We present the complete set of results for our sentiment analysis scenarios. In Table C.7, we record the target-alignment of our projected datasets as measured by training on the projected dataset. In Table C.8, we record the performance of training on the target dataset augmented with projected data. In Table C.9, we record similar results but combined with back-translation data augmentation.

Auxiliary	Target	Target Only	Target + Random	Target + PGD-PD	Target + AS-PD
ImageNet	CIFAR10	43.6 ± 0.4	54.8 ± 2.1	55.3 ± 1.8	57.0 ± 3.0
Oxford-IIIT Pet		51.9 ± 2.1	54.2 ± 2.6	54.8 ± 2.0	**59.3** ± 0.2
STL10		41.0 ± 1.6	37.2 ± 1.8	44.6 ± 0.4	44.3 ± 1.1
3DB		43.6 ± 1.1	50.0 ± 2.2	43.2 ± 0.9	49.4 ± 0.6
ImageNet	Oxford-IIIT Pet	49.3 ± 1.3	58.2 ± 2.7	55.1 ± 2.2	57.4 ± 3.7
CIFAR10		52.1 ± 1.9	54.2 ± 2.5	55.4 ± 2.0	**59.5** ± 2.9
STL10		47.6 ± 0.9	54.2 ± 2.5	55.7 ± 1.9	55.0 ± 2.3
3DB		51.1 ± 1.9	53.8 ± 3.8	51.8 ± 2.7	55.7 ± 1.0
ImageNet	STL10	41.7 ± 2.3	55.9 ± 1.0	53.2 ± 2.1	55.1 ± 1.3
CIFAR10		39.5 ± 0.9	58.6 ± 1.8	53.2 ± 0.8	58.4 ± 1.6
Oxford-IIIT Pet		54.2 ± 2.8	54.3 ± 3.1	**65.1** ± 1.4	61.6 ± 1.1
3DB		42.8 ± 2.4	52.2 ± 1.5	46.9 ± 0.8	51.0 ± 1.4
ImageNet	3DB	84.0 ± 2.1	89.4 ± 1.6	83.2 ± 0.6	87.7 ± 1.4
CIFAR10		80.5 ± 2.0	89.0 ± 3.3	86.3 ± 0.4	**89.5** ± 0.5
Oxford-IIIT Pet		69.8 ± 1.6	73.4 ± 7.2	82.2 ± 3.3	82.3 ± 10.2
STL10		68.5 ± 5.8	76.5 ± 6.9	77.8 ± 1.6	79.6 ± 1.7

Table C.5: Augmenting a target dataset with auxiliary data for vision scenarios. In this experiment, we add auxiliary data to a target dataset for training, and test on target data. For all target datasets, we can find an auxiliary dataset in which augmenting with AS-PD or PGD-PD performs the best. For each target dataset, we highlight the choice of auxiliary dataset where either AS-PD or PGD-PD has the largest benefit in augmenting the target dataset.

C.5 Visualizations of projected datasets

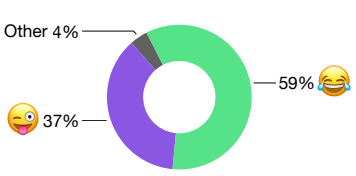
In this section, we present additional visualizations of projected datasets. These visualizations provide a way to analyze the composition of a target dataset via inspection of the resulting source distributions proportions.

Figure C-8 shows the projection of the Angry and Joy classes of the Emotion dataset onto the Emoji dataset. Here, we see that the Angry projection consists of an Emoji with tears and an Emoji with a tongue sticking out. We suspect this is due to the usage of these emojis in sarcastic or cynical expressions that are most closely aligned with the Angry class, since the Emoji dataset does not contain any clearly angry emojis. The Joy projection contains generally happy emojis, including a heart emoji that none of the other Emotion classes had.

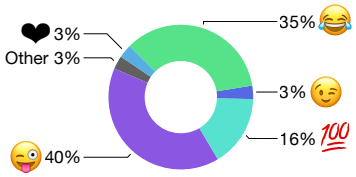
Figure C-9 shows the projection Oxford-IIIT Pet and 3DB onto ImageNet cats. The Oxford-IIIT Pet projection has largely household cats, except for a small proportion of lions. The 3DB projection consists primarily of two types of cats, suggesting that the 3DB cats do not contain very much variation.

Auxiliary	Target	DA	Random + DA	PGD-PD + DA	AS-PD + DA
ImageNet	CIFAR10	45.5 ± 1.0	52.6 ± 0.9	55.2 ± 1.5	52.3 ± 3.3
Oxford-IIIT Pet		53.1 ± 2.2	52.6 ± 1.7	53.5 ± 2.7	**57.8** ± 3.3
STL10		47.5 ± 1.8	37.1 ± 0.9	44.8 ± 1.3	44.9 ± 1.2
3DB		46.2 ± 2.1	48.2 ± 0.6	47.0 ± 1.9	48.0 ± 1.4
ImageNet	Oxford-IIIT Pet	47.0 ± 2.2	50.6 ± 4.4	57.0 ± 2.8	50.9 ± 3.5
CIFAR10		54.0 ± 3.1	53.5 ± 2.8	56.4 ± 2.0	**58.1** ± 2.0
STL10		48.9 ± 2.0	48.9 ± 1.4	55.8 ± 3.1	53.2 ± 1.3
3DB		52.4 ± 3.5	50.7 ± 2.9	50.7 ± 2.3	50.4 ± 2.5
ImageNet	STL10	48.9 ± 1.6	54.0 ± 1.1	53.8 ± 1.5	52.9 ± 1.0
CIFAR10		49.7 ± 0.9	57.8 ± 1.5	57.3 ± 1.5	56.1 ± 3.1
Oxford-IIIT Pet		54.5 ± 4.3	51.9 ± 1.7	**63.0** ± 1.1	58.7 ± 2.5
3DB		48.6 ± 1.1	48.5 ± 0.5	49.6 ± 0.3	49.0 ± 1.2
ImageNet	3DB	92.1 ± 1.9	93.6 ± 1.4	92.4 ± 0.8	94.3 ± 0.5
CIFAR10		91.4 ± 1.8	94.4 ± 0.6	92.4 ± 1.3	94.5 ± 1.0
Oxford-IIIT Pet		73.0 ± 7.6	59.5 ± 2.9	**91.5** ± 0.9	86.3 ± 13.3
STL10		86.9 ± 2.3	91.3 ± 0.8	89.1 ± 1.0	88.2 ± 0.6

Table C.6: Augmenting a target dataset with auxiliary data, combined with TrivialAugment data augmentation. In this experiment, we add auxiliary data and TrivialAugment to a target dataset for training, and test on target data. For all target datasets, we can find an auxiliary dataset in which augmenting with AS-PD or PGD-PD performs the best. For each target dataset, we highlight the choice of auxiliary dataset where either AS-PD or PGD-PD has the largest benefit in augmenting the target dataset.



(a) Angry

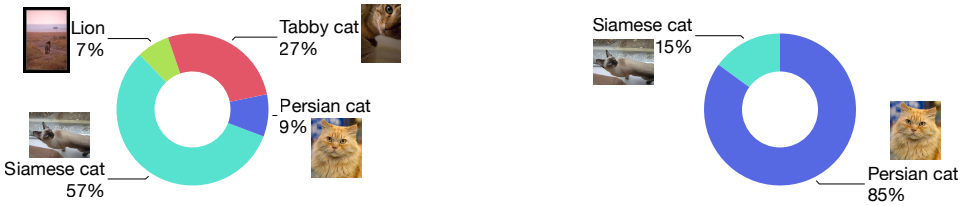


(b) Joy

Figure C-8: A visualization depicting the result of projecting various classes of the Emotion dataset (angry and joy) onto the Emoji dataset. Each doughnut corresponds to the subset of the Emoji dataset that is closest to the target Emotion class as calculated by our framework.

Auxiliary	Target	Random	PGD-PD	Not PGD-PD	AS-PD	Not AS-PD
Emoji	SST	**57.6** ± 10.1	54.8 ± 7.5	56.2 ± 6.6	57.0 ± 0.0	54.0 ± 4.1
Emotion		52.2 ± 4.7	50.6 ± 3.9	57.0 ± 7.8	55.8 ± 3.5	**57.4** ± 0.5
Yelp		50.4 ± 7.7	49.4 ± 5.2	**57.8** ± 6.0	55.8 ± 6.7	50.6 ± 9.1
DailyDialog		51.6 ± 4.8	**61.2** ± 9.8	51.4 ± 4.5	56.6 ± 11.4	55.6 ± 1.9
SST	Emoji	3.2 ± 1.0	4.0 ± 1.5	**5.4** ± 2.6	**5.4** ± 2.1	4.0 ± 1.5
Emotion		3.8 ± 1.5	4.2 ± 1.5	**5.2** ± 1.3	4.2 ± 1.2	3.6 ± 0.8
Yelp		4.2 ± 1.8	4.8 ± 1.7	4.8 ± 1.7	**6.4** ± 3.1	3.2 ± 0.4
DailyDialog		2.6 ± 0.5	**6.2** ± 3.1	3.6 ± 1.2	5.0 ± 2.8	3.0 ± 0.0
SST	Emotion	16.6 ± 7.0	27.0 ± 6.3	13.4 ± 4.1	29.2 ± 2.8	**31.0** ± 0.0
Emoji		16.2 ± 5.3	20.4 ± 3.3	15.0 ± 1.8	**28.6** ± 2.3	**28.6** ± 3.5
Yelp		16.4 ± 5.7	20.4 ± 8.8	22.8 ± 10.0	**29.0** ± 2.7	24.6 ± 6.7
DailyDialog		15.0 ± 6.7	23.8 ± 7.6	20.2 ± 6.8	29.8 ± 2.9	**30.4** ± 1.4
SST	Yelp	21.8 ± 1.6	22.6 ± 3.2	19.4 ± 2.7	22.2 ± 2.4	**24.4** ± 0.8
Emoji		21.0 ± 0.0	19.0 ± 2.6	21.0 ± 0.0	**23.2** ± 4.1	16.8 ± 2.9
Emotion		21.0 ± 0.0	**22.8** ± 2.2	20.8 ± 0.4	21.2 ± 4.3	19.6 ± 4.1
DailyDialog		21.0 ± 0.0	21.8 ± 1.3	21.6 ± 1.4	**23.2** ± 2.6	20.4 ± 6.6
SST	DailyDialog	14.6 ± 3.1	16.8 ± 1.0	15.8 ± 2.0	**17.4** ± 3.4	14.2 ± 2.4
Emoji		13.6 ± 2.2	13.6 ± 1.5	14.4 ± 2.9	**23.6** ± 2.5	19.6 ± 4.2
Emotion		14.6 ± 2.4	19.4 ± 5.9	15.0 ± 4.3	**23.4** ± 5.8	12.0 ± 2.7
Yelp		**16.6** ± 3.3	13.6 ± 0.5	14.2 ± 1.5	**16.6** ± 5.0	13.2 ± 3.1

Table C.7: Approximating target datasets with auxiliary data. In this experiment, we train on auxiliary data and test on target data. Higher test accuracy corresponds to better approximation quality. AS-PD typically performs the best, and PGD-PD also typically outperforms Random.



(a) Oxford-IIIT Pet

(b) 3DB

Figure C-9: A visualization depicting the result of projecting cat classes from various target datasets (Oxford-IIIT Pet and 3DB) onto the ImageNet cats. Each doughnut represents the subset of the ImageNet cats that is closest to the cat class from the target dataset as calculated by our framework.

Auxiliary	Target	Target Only	Target + Random	Target + PGD-PD	Target + AS-PD
Emoji	SST	74.2 ± 6.0	66.2 ± 8.8	71.4 ± 6.1	77.0 ± 6.7
Emotion			61.2 ± 7.4	64.8 ± 9.9	75.8 ± 9.0
Yelp			60.8 ± 8.9	65.6 ± 12.0	**78.6** ± 2.9
DailyDialog			63.6 ± 12.1	76.4 ± 5.4	76.4 ± 6.1
SST	Emoji	12.4 ± 3.6	12.4 ± 3.8	14.6 ± 3.4	24.0 ± 2.7
Emotion			15.0 ± 4.1	13.8 ± 4.1	23.6 ± 2.2
Yelp			13.0 ± 3.8	13.6 ± 3.1	22.6 ± 2.2
DailyDialog			14.0 ± 3.3	12.8 ± 2.9	**24.4** ± 1.0
SST	Emotion	37.6 ± 4.3	39.4 ± 4.1	42.8 ± 7.8	**45.2** ± 4.7
Emoji			39.2 ± 1.3	39.4 ± 4.7	44.0 ± 4.3
Yelp			41.6 ± 3.2	40.4 ± 4.9	44.8 ± 4.2
DailyDialog			43.2 ± 4.1	42.6 ± 8.0	42.6 ± 4.5
SST	Yelp	28.6 ± 1.9	27.4 ± 2.4	29.4 ± 4.8	30.0 ± 4.1
Emoji			29.2 ± 3.0	27.0 ± 5.4	**34.4** ± 5.7
Emotion			27.8 ± 6.9	28.6 ± 4.9	33.6 ± 4.3
DailyDialog			29.0 ± 6.4	33.2 ± 4.8	32.0 ± 5.2
SST	DailyDialog	37.4 ± 6.3	37.0 ± 5.7	42.6 ± 5.4	61.2 ± 3.4
Emoji			36.0 ± 4.9	42.6 ± 5.9	66.2 ± 6.1
Emotion			37.6 ± 5.0	35.0 ± 8.0	**67.4** ± 6.0
Yelp			38.2 ± 5.6	36.0 ± 7.3	64.6 ± 7.5

Table C.8: Augmenting a target dataset with auxiliary data for language scenarios. In this experiment, we add auxiliary data to a target dataset for training, and test on target data. For each target dataset, we highlight the choice of auxiliary dataset where either AS-PD or PGD-PD has the largest benefit in augmenting the target dataset.

Auxiliary	Target	DA	Random + DA	PGD-PD + DA	AS-PD + DA
Emoji	SST	78.4 ± 3.5	78.4 ± 3.5	78.2 ± 2.1	78.4 ± 2.1
Emotion			76.8 ± 6.5	76.0 ± 4.0	76.2 ± 5.7
Yelp			77.8 ± 1.9	78.0 ± 3.4	80.2 ± 2.7
DailyDialog			76.2 ± 4.7	76.0 ± 2.6	**80.8** ± 2.7
SST	Emoji	22.8 ± 3.2	22.8 ± 3.2	22.2 ± 1.7	24.2 ± 1.9
Emotion			23.2 ± 1.3	22.8 ± 1.7	24.0 ± 0.6
Yelp			22.8 ± 1.3	23.6 ± 2.7	24.2 ± 4.0
DailyDialog			23.8 ± 2.0	23.6 ± 1.9	**25.0** ± 2.4
SST	Emotion	50.6 ± 9.0	50.6 ± 9.0	56.0 ± 4.7	59.2 ± 7.4
Emoji			52.6 ± 6.7	52.6 ± 5.1	**59.6** ± 4.4
Yelp			53.4 ± 6.7	53.6 ± 3.3	58.2 ± 3.8
DailyDialog			51.8 ± 8.0	51.2 ± 4.2	57.6 ± 9.0
SST	Yelp	32.4 ± 4.3	32.4 ± 4.3	32.2 ± 6.5	35.8 ± 3.7
Emoji			33.8 ± 6.6	33.6 ± 6.6	**36.8** ± 4.2
Emotion			30.2 ± 3.2	33.2 ± 5.3	36.6 ± 5.4
DailyDialog			29.4 ± 1.9	32.8 ± 5.6	35.8 ± 4.4
SST	DailyDialog	76.2 ± 2.0	76.2 ± 2.0	75.6 ± 1.2	75.4 ± 2.3
Emoji			75.4 ± 1.7	75.2 ± 1.9	76.2 ± 1.5
Emotion			76.0 ± 2.7	76.0 ± 1.1	75.0 ± 2.5
Yelp			74.8 ± 2.9	**76.8** ± 1.5	75.0 ± 1.8

Table C.9: Augmenting a target dataset with auxiliary data, combined with back-translation augmentation via the French language. In this experiment, we add auxiliary data and back-translated data to a target dataset for training, and test on target data. For each target dataset, we highlight the choice of auxiliary dataset where either AS-PD or PGD-PD has the largest benefit in augmenting the target dataset.

Appendix D

Additional details for Chapter 5

D.1 Formulating the MILP model

D.1.1 Formulating ReLU in an MILP Model

We reproduce our formulation for the ReLU below.

$$y \leq x - l(1 - a) \tag{D.1}$$
$$y \geq x \tag{D.2}$$
$$y \leq u \cdot a \tag{D.3}$$
$$y \geq 0 \tag{D.4}$$
$$a \in \{0, 1\} \tag{D.5}$$

We consider two cases.

Recall that a is the indicator variable $a = \mathbb{1}_{x \geq 0}$.

When $a = 0$, the constraints in Equation D.3 and D.4 are binding, and together imply that $y = 0$. The other two constraints are not binding, since Equation D.2 is no stricter than Equation D.4 when $x < 0$, while Equation D.1 is no stricter than Equation D.3 since $x - l \geq 0$. We thus have $a = 0 \implies y = 0$.

When $a = 1$, the constraints in Equation D.1 and D.2 are binding, and together imply that $y = x$. The other two constraints are not binding, since Equation D.4 is no stricter than Equation D.2 when $x > 0$, while Equation D.3 is no stricter than Equation D.1 since $x \leq u$. We thus have $a = 1 \implies y = x$.

This formulation for rectified linearities is sharp [Vie15] if we have no further information about x. This is the case since relaxing the integrality constraint on a

leads to (x, y) being restricted to an area that is the convex hull of $y = \max(x, 0)$. However, if x is an affine expression $x = w^T z + b$, the formulation is no longer sharp, and we can add more constraints using bounds we have on z to improve the problem formulation.

D.1.2 Formulating the maximum function in an MILP model

We reproduce our formulation for the maximum function below.

$$y \leq x_i + (1 - a_i)(u_{max,-i} - l_i) \ \forall i \tag{D.6}$$

$$y \geq x_i \ \forall i \tag{D.7}$$

$$\sum_{i=1}^{m} a_i = 1 \tag{D.8}$$

$$a_i \in \{0, 1\} \tag{D.9}$$

Equation D.8 ensures that exactly one of the a_i is 1. It thus suffices to consider the value of a_i for a single variable.

When $a_i = 1$, Equations D.6 and D.7 are binding, and together imply that $y = x_i$. We thus have $a_i = 1 \implies y = x_i$.

When $a_i = 0$, we simply need to show that the constraints involving x_i are never binding regardless of the values of $x_1, x_2, \ldots, x_m$. Equation D.7 is not binding since $a_i = 0$ implies x_i is not the (unique) maximum value. Furthermore, we have chosen the coefficient of a_i such that Equation D.6 is not binding, since $x_i + u_{max,-i} - l_i \geq u_{max,-i} \geq y$. This completes our proof.

D.1.3 Expressing l_p norms as the objective of an MIP model

l_1

When $d(x', x) = \|x' - x\|_1$, we introduce the auxiliary variable δ, which bounds the elementwise absolute value from above: $\delta_j \geq x'_j - x_j, \delta_j \geq x_j - x'_j$. The optimization

in Equation 5.3-5.5 is equivalent to

$$\min_{x'} \sum_j \delta_j \tag{D.10}$$

$$\text{subject to} \quad \arg\max_i(f_i(x')) \neq \lambda(x) \tag{D.11}$$

$$x' \in \mathcal{X}_{valid} \tag{D.12}$$

$$\delta_j \geq x'_j - x_j \tag{D.13}$$

$$\delta_j \geq x_j - x'_j \tag{D.14}$$

l_∞

When $d(x', x) = \|x' - x\|_\infty$, we introduce the auxiliary variable ϵ, which bounds the l_∞ norm from above: $\epsilon \geq x'_j - x_j, \epsilon \geq x_j - x'_j$. The optimization in Equation 5.3-5.5 is equivalent to

$$\min_{x'} \epsilon \tag{D.15}$$

$$\text{subject to} \quad \arg\max_i(f_i(x')) \neq \lambda(x) \tag{D.16}$$

$$x' \in \mathcal{X}_{valid} \tag{D.17}$$

$$\epsilon \geq x'_j - x_j \tag{D.18}$$

$$\epsilon \geq x_j - x'_j \tag{D.19}$$

l_2

When $d(x', x) = \|x' - x\|_2$, the objective becomes quadratic, and we have to use a Mixed Integer Quadratic Program (MIQP) solver. However, no auxiliary variables are required: the optimization in Equation 5.3-5.5 is simply equivalent to

$$\min_{x'} \sum_j (x'_j - x_j)^2 \tag{D.20}$$

$$\text{subject to} \quad \arg\max_i(f_i(x')) \neq \lambda(x) \tag{D.21}$$

$$x' \in \mathcal{X}_{valid} \tag{D.22}$$

D.2 Determining tight bounds on decision variables

Our framework for determining bounds on decision variables is to view the neural network as a computation graph G. Directed edges point from function input to

output, and vertices represent variables. Source vertices in G correspond to the input of the network, and sink vertices in G correspond to the output of the network. The computation graph begins with defined bounds on the input variables (based on the input domain $(\mathcal{G}(x) \cap \mathcal{X}_{valid})$), and is augmented with bounds on intermediate variables as we determine them. The computation graph is acyclic for the feed-forward networks we consider.

Since the networks we consider are piecewise-linear, any subgraph of G can be expressed as an MILP, with constraints derived from 1) input-output relationships along edges and 2) bounds on the values of the source nodes in the subgraph. Integer constraints are added whenever edges describe a non-linear relationship.

We focus on computing an upper bound on some variable v; computing lower bounds follows a similar process. All the information required to determine the best possible bounds on v is contained in the subtree of G rooted at v, G_v. (Other variables that are not ancestors of v in the computation graph cannot affect its value.) Maximizing the value of v in the MILP M_v corresponding to G_v gives the optimal upper bound on v.

We can reduce computation time in two ways. Firstly, we can prune some edges and vertices of G_v. Specifically, we select a set of variables with existing bounds V_I that we assume to be independent (that is, we assume that they each can take on any value independent of the value of the other variables in V_I). We remove all in-edges to vertices in V_I, and eliminate variables without children, resulting in the smaller computation graph G_{v,V_I}. Maximizing the value of v in the MILP M_{v,V_I} corresponding to G_{v,V_I} gives a valid upper bound on v that is optimal if the independence assumption holds.

We can also reduce computation time by relaxing some of the integer constraints in M_v to obtain a MILP with fewer integer variables M'_v. Relaxing an integer constraint corresponds to replacing the relevant non-linear relationship with its convex relaxation. Again, the objective value returned by maximizing the value of v over M'_v may not be the optimal upper bound, but is guaranteed to be a valid bound.

D.2.1 FULL

FULL considers the full subtree G_v and does not relax any integer constraints. The upper and lower bound on v is determined by maximizing and minimizing the value of v in M_v respectively. FULL is also used in [CNR17] and [FJ18].

If solves proceed to optimality, FULL is guaranteed to find the optimal bounds on

the value of a single variable v. The trade-off is that, for deeper layers, using FULL can be relatively inefficient, since solve times in the worst case are exponential in the number of binary variables in M_v.

Nevertheless, contrary to what is asserted in [CNR17], we *can* terminate solves early and still obtain useful bounds. For example, to determine an upper bound on v, we set the objective of M_v to be to maximize the value of v. As the solve process proceeds, we obtain progressively better certified upper bounds on the maximum value of v. We can thus terminate the solve process and extract the best upper bound found at any time, using this upper bound as a valid (but possibly loose) bound on the value of v.

D.2.2 Linear Programming (LP)

LP considers the full subtree G_v but relaxes all integer constraints. This results in the optimization problem becoming a linear program that can be solved more efficiently. LP represents a good middle ground between the optimality of FULL and the performance of IA.

D.2.3 Interval Arithmetic (IA)

IA selects V_I to be the parents of v. In other words, bounds on v are determined solely by considering the bounds on the variables in the previous layer. We note that this is simply interval arithmetic [MKC09].

Consider the example of computing bounds on the variable $\hat{z}_i = W_i z_{i-1} + b_i$, where $l_{z_{i-1}} \leq z_{i-1} \leq u_{z_{i-1}}$. We have

$$\hat{z}_i \geq W_i^- u_{z_{i-1}} + W_i^+ l_{z_{i-1}} \tag{D.23}$$

$$\hat{z}_i \leq W_i^+ u_{z_{i-1}} + W_i^- l_{z_{i-1}} \tag{D.24}$$

$$W_i^+ \triangleq \max(W_i, 0) \tag{D.25}$$

$$W_i^- \triangleq \min(W_i, 0) \tag{D.26}$$

IA is efficient (since it only involves matrix operations for our applications). However, for deeper layers, using interval arithmetic can lead to overly conservative bounds.

D.3 Progressive bounds tightening

GetBoundsForMax finds the tightest bounds required for specifying the constraint $y = \max(xs)$. Using the observation in Proposition 1, we stop tightening the bounds on a variable if its maximum possible value is lower than the minimum value of some other variable. GetBoundsForMax returns a tuple containing the set of elements in xs that can still take on the maximum value, as well as a dictionary of upper and lower bounds.

GetBoundsForMax(xs, fs)

```
1  ▷ fs are the procedures to determine bounds
2  ▷ fs sorted in increasing computational complexity.
3  d_l = {x : −∞ for x in xs}
4  d_u = {x : ∞ for x in xs}
5  ▷ initialize dictionaries containing best known upper and lower bounds on xs
6  l_max = −∞      ▷ l_max is the maximum known lower bound on any of the xs
7  a = {xs}
8  ▷ a is a set of active elements in xs that can still take on the maximum value.
9  for f in fs:    ▷ carrying out progressive bounds tightening
10       do for x in xs:
11            if d_u[x] < l_max
12               then a.remove(x)      ▷ x cannot take on the maximum value
13               else  u = f(x, boundType = upper)
14                     d_u[x] = min(d_u[x], u)
15                     l = f(x, boundType = lower)
16                     d_l[x] = max(d_l[x], l)
17                     l_max = max(l_max, l)
18 return (a, d_l, d_u)
```

D.4 Additional experimental details

D.4.1 Networks used

The source of the weights for each of the networks we present results for in the paper are provided below.

- MNIST classifiers not designed to be robust:

  - MLP-2×[20] and MLP-3×[20] are the MNIST classifiers in [Wen+18], and can be found at `https://github.com/huanzhang12/CertifiedReLURobustness`.

- MNIST classifiers designed for robustness to perturbations with l_∞ norm-bound $\epsilon = 0.1$:

  - LP_d-CNN is the MNIST classifier in [WK17], and can be found at `https://github.com/locuslab/convex_adversarial`.

  - Adv-CNN was trained with adversarial examples generated by PGD. PGD attacks were carried out with l_∞ norm-bound $\epsilon = 0.1$, 8 steps per sample, and a step size of 0.334. An l_1 regularization term was added to the objective with a weight of 0.0015625 on the first convolution layer and 0.003125 for the remaining layers.

  - Adv-MLP-2×[200] was trained with adversarial examples generated by PGD. PGD attacks were carried out with with l_∞ norm-bound $\epsilon = 0.15$, 200 steps per sample, and a step size of 0.1. An l_1 regularization term was added to the objective with a weight of 0.003 on the first layer and 0 for the remaining layers.

  - SDP_d-MLP-1×[500] is the classifier in [RSL18].

- MNIST classifiers designed for robustness to perturbations with l_∞ norm-bound $\epsilon = 0.2, 0.3, 0.4$:

  - LP_d-CNN was trained with the code available at `https://github.com/locuslab/convex_adversarial` at commit `4e9377f`. Parameters selected were `batch_size=20, starting_epsilon=0.01, epochs=200, seed=0`.

- CIFAR-10 classifiers designed for robustness to perturbations with l_∞ norm-bound $\epsilon = \frac{8}{255}$

  - LP_d-RES courtesy of the authors of [Won+18].

D.4.2 Computational environment

We construct the MILP models in Julia [Bez+17] using JuMP [DHL17], with the model solved by the commercial solver Gurobi 7.5.2 [Gur17b]. All experiments were run on a KVM virtual machine with 8 virtual CPUs running on shared hardware, with Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processors, and 8GB of RAM.

D.5 Additional solve statistics

Table D.1 presents additional solve statistics to supplement the results reported in Table 5.2. If the solver explores zero nodes for a particular sample, it proved that the sample was robust (or found an adversarial example) without branching on any binary variables. This occurs when the bounds we find during the presolve step are sufficiently tight. We note that this occurs for over 95% of samples for LP_d-CNN for $\epsilon = 0.1$.

Table D.1: Additional solve statistics when determining adversarial accuracy of MNIST and CIFAR-10 classifiers to perturbations with l_∞ norm-bound ϵ. We solve a linear program for each of the *nodes explored* in the MILP search tree.

Dataset	Network	ϵ	Mean Time /s	Nodes Explored						
				Mean	Median	Percentile				Max
						90	95	99	99.9	
MNIST	LP_d-CNN	0.1	3.52	2.05	0	0	0	1	701	1387
	Adv-CNN	0.1	135.74	754.51	0	205	3535	20490	31767	50360
	Adv-MLP_B	0.1	3.69	97.51	0	1	3	2221	11866	103481
	SDP_d-MLP_A	0.1	312.43	4661.51	39	17614	21769	27335	29875	29887
	LP_d-CNN	0.2	7.32	16.42	0	1	1	549	2205	7105
	LP_d-CNN	0.3	5.13	33.54	0	5	121	789	2213	19650
	LP_d-CNN	0.4	5.07	61.65	1	79	321	934	3681	43274
CIFAR-10	LP_d-RES	$\frac{8}{255}$	15.23	47.31	0	1	3	1808	4448	5022

D.6 Which adversarial examples are missed by PGD?

PGD succeeds in finding an adversarial example if and only if the starting point for the gradient descent is in the basin of attraction of some adversarial example. Since PGD initializes the gradient descent with a randomly chosen starting point within $\mathcal{G}(x) \cap \mathcal{X}_{valid}$, the success rate (with a single random start) corresponds to the fraction of $\mathcal{G}(x) \cap \mathcal{X}_{valid}$ that is in the basin of attraction of some adversarial example.

Intuitively, the success rate of PGD should be inversely related to the magnitude of the minimum adversarial distortion $\hat{\delta}$: if $\hat{\delta}$ is small, we expect more of $\mathcal{G}(x) \cap \mathcal{X}_{valid}$ to correspond to adversarial examples, and thus the union of the basins of attraction of the adversarial examples is likely to be larger. We investigate here whether our intuition is substantiated.

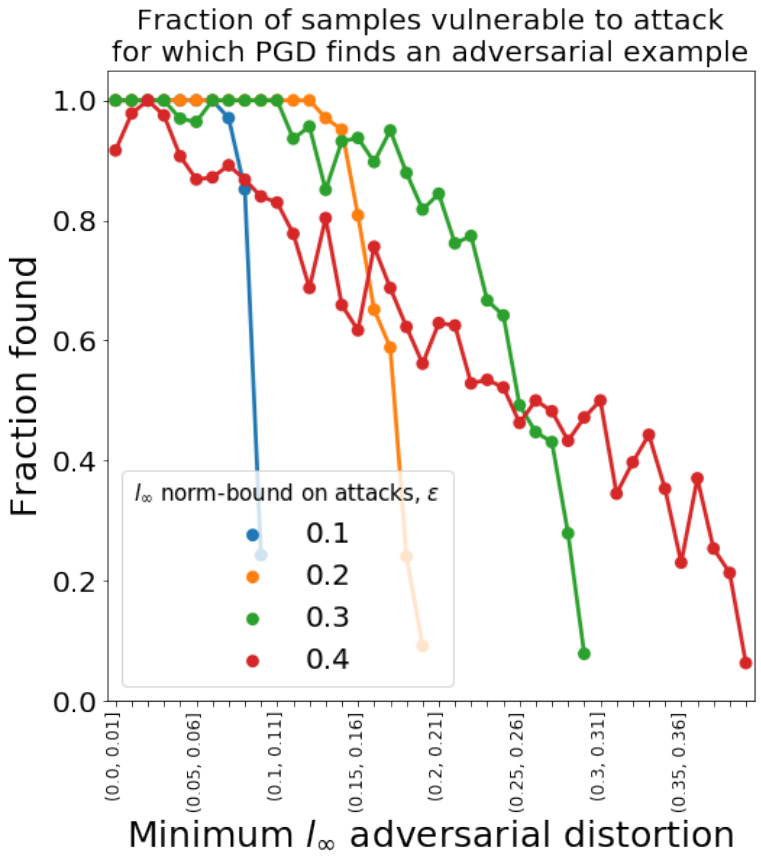


Figure D-1: Fraction of samples in the MNIST test set vulnerable to attack for which PGD succeeds at finding an adversarial example. Samples are binned by their minimum adversarial distortion (as measured under the l_∞ norm), with bins of size 0.01. Each of these are LP_d-CNN networks, and were trained to optimize for robustness to attacks with l_∞ norm-bound ϵ. For any given network, the success rate of PGD declines as the minimum adversarial distortion increases. Comparing networks, success rate declines for networks with larger ϵ even at the same minimum adversarial distortion.

To obtain the best possible empirical estimate of the success rate of PGD for each sample, we would need to re-run PGD initialized with *multiple* different randomly chosen starting points within $\mathcal{G}(x) \cap \mathcal{X}_{valid}$.

However, since we are simply interested in the relationship between success rate and minimum adversarial distortion, we obtained a coarser estimate by binning the samples based on their minimum adversarial distortion, and then calculating the fraction of samples in each bin for which PGD with a *single* randomly chosen starting point succeeds at finding an adversarial example.

Figure D-1 plots this relationship for four networks using the CNN architecture and trained with the same training method LP_d but optimized for attacks of different

size. Three features are clearly discernible:

- PGD is very successful at finding adversarial examples when the magnitude of the minimum adversarial distortion, $\hat{\delta}$, is small.
- The success rate of PGD declines significantly for all networks as $\hat{\delta}$ approaches ϵ.
- For a given value of $\hat{\delta}$, and two networks a and b trained to be robust to attacks with l_∞ norm-bound ϵ_a and ϵ_b respectively (where $\epsilon_a < \epsilon_b$), PGD is consistently more successful at attacking the network trained to be robust to smaller attacks, a, as long as $\hat{\delta} \ll \epsilon_a$.

The sharp decline in the success rate of PGD as $\hat{\delta}$ approaches ϵ is particularly interesting, especially since it is suggests a pathway to generating networks that *appear* robust when subject to PGD attacks of bounded l_∞ norm but are in fact vulnerable to such bounded attacks: we simply train the network to maximize the total number of adversarial examples with minimum adversarial distortion close to ϵ.

D.7 Sparsification and verifiability

When verifying the robustness of SDP_d-MLP_A, we observed that a significant proportion of kernel weights were close to zero. Many of these tiny weights are unlikely to be contributing significantly to the final classification of any input image. Having said that, setting these tiny weights to zero *could* potentially reduce verification time, by 1) reducing the size of the MILP formulation, and by 2) ameliorating numerical issues caused by the large range of numerical coefficients in the network [Gur17a].

We generated sparse versions of the original network to study the impact of sparseness on solve times. Our heuristic sparsification algorithm is as follows: for each fully-connected layer i, we set a fraction f_i of the weights with smallest absolute value in the kernel to 0, and rescale the rest of the weights such that the l_1 norm of the kernel remains the same.[1] Note that MLP_A consists of only two layers: one hidden layer (layer 1) and one output layer (layer 2).

Table D.2 summarizes the results of verifying sparse versions of SDP_d-MLP_A; the first row presents results for the original network, and the subsequent rows present results when more and more of the kernel weights are set to zero.

When comparing the first and last rows, we observe an improvement in both mean time and fraction timed out by an order of magnitude. As expected, sparsifying

[1] Skipping the rescaling step did not appreciably affect verification times or test errors.

Table D.2: Effect of sparsification of $\text{SDP}_{\text{d}}\text{-MLP}_{\text{A}}$ on verifiability. Test error increases slightly as larger fractions of kernel weights are set to zero, but the certified upper bound on adversarial error decreases significantly as the solver reaches the time limit for fewer samples.

Fraction zeroed		Test Error	Certified Bounds on Adversarial Error		Mean Time / s	Fraction Timed Out
f_1	f_2		Lower Bound	Upper Bound		
0.0	0.00	**4.18%**	14.36%	30.81%	312.4	0.1645
0.5	0.25	4.22%	14.60%	25.25%	196.0	0.1065
0.8	0.25	4.22%	15.03%	**18.26%**	69.7	0.0323
0.9	0.25	4.93%	17.97%	18.76%	22.2	0.0079

weights increases the test error, but the impact is not significant until f_1 exceeds 0.8. We also find that sparsification significantly improves our upper bound on adversarial error — to a point: the upper bound on adversarial error for $f_1 = 0.9$ is higher than that for $f_1 = 0.8$, likely because the true adversarial error has increased significantly.

Starting with a network that is robust, we have demonstrated that a simple sparsification approach can already generate a sparsified network with an upper bound on adversarial error significantly lower than the best upper bound that can be determined for the original network. Adopting a more principled sparsification approach could achieve the same improvement in verifiability but without compromising on the true adversarial error as much.

D.8 Robust training and ReLU stability

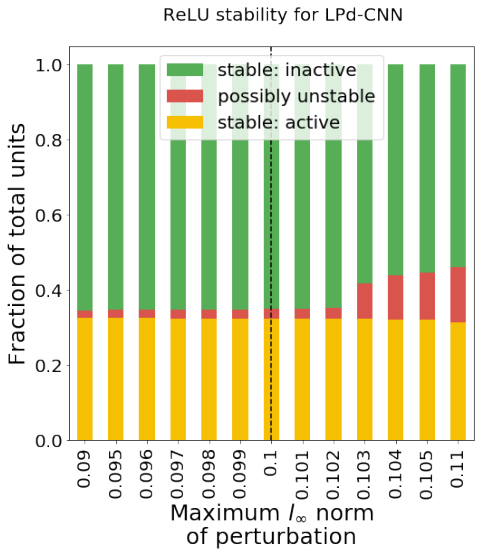
Networks that are designed to be robust need to balance two competing objectives. Locally, they need to be robust to small perturbations to the input. However, they also need to retain sufficient global expressiveness to maintain a low test error.

For the networks in Table 5.3, even though each robust training approach estimates the worst-case error very differently, all approaches lead to a significant fraction of the ReLUs in the network being provably stable with respect to perturbations with bounded l_∞ norm. In other words, for the input domain $\mathcal{G}(x)$ consisting of all bounded perturbations of the sample x, we can show for many ReLUs that the input to the unit is always positive (and thus the output is linear in the input) or always negative (and thus the output is always zero). As discussed in the main text, we believe that

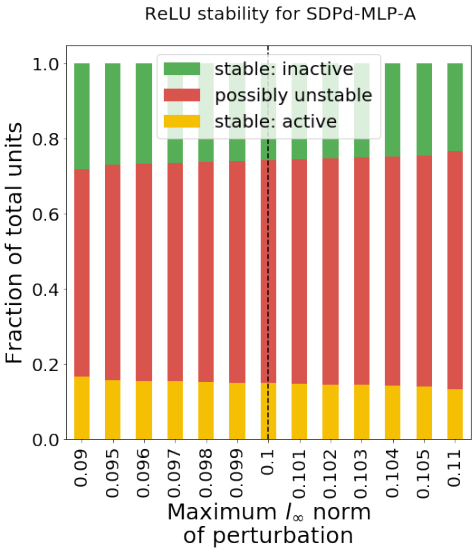
the need for the network to be robust to perturbations in $\mathcal{G}$ drives more ReLUs to be provably stable with respect to $\mathcal{G}$.

To better understand how networks can retain global expressiveness even as many ReLUs are provably stable with respect to perturbations with bounded l_∞ norm ϵ, we study how the number of ReLUs that are provably stable changes as we vary the size of $\mathcal{G}(x)$ by changing the maximum allowable l_∞ norm of perturbations. The results are presented in Figure D-2.

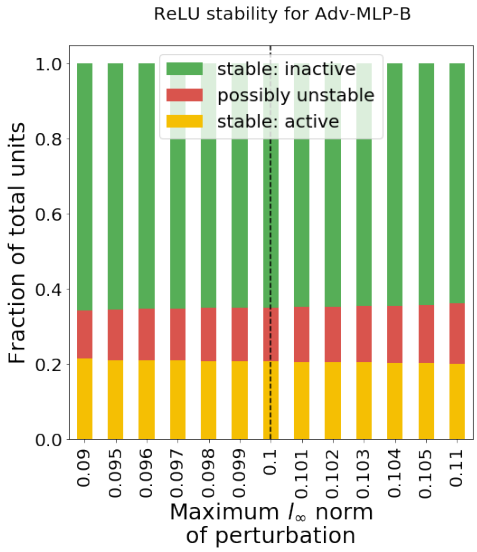
As expected, the number of ReLUs that cannot be proven to be stable increases as the maximum allowable l_∞ norm of perturbations increases. More interestingly, LP_d-CNN is very sensitive to the $\epsilon = 0.1$ threshold, with a sharp increase in the number of ReLUs that cannot be proven to be stable when the maximum allowable l_∞ norm of perturbations increases beyond 0.102. An increase of the same abruptness is not seen for the other two networks.



(a) LP_d-CNN. Note the sharp increase in the number of ReLUs that cannot be proven to be stable when the maximum l_∞ norm increases beyond 0.102.



(b) SDP_d-MLP_A.



(c) Adv-MLP_B. Adversarial training alone is sufficient to significantly increase the number of ReLUs that are provably stable.

Figure D-2: Comparison of provably ReLU stability for networks trained via different robust training procedures to be robust at $\epsilon = 0.1$, when varying the maximum allowable l_∞ norm of the perturbation. The results reported in Table 5.3 are marked by a dotted line. As we increase the maximum allowable l_∞ norm of perturbations, the number of ReLUs that cannot be proven to be stable increases across all networks (as expected), but LP_d-CNN is far more sensitive to the $\epsilon = 0.1$ threshold.

Appendix E

Additional details for Chapter 6

E.1 Natural improvements

E.1.1 Natural regularization for inducing weight sparsity

All of the control and "+RS" networks in our paper contain natural improvements that improve weight sparsity, which reduce the number of variables in the LPs solved by the verifier. We observed that the two techniques we used for weight sparsity (ℓ_1-regularization and small weight pruning) don't hurt test set accuracy but they dramatically improve provable adversarial accuracy and verification speed.

1. ℓ_1-regularization: We use a weight of 2e−5 on MNIST and a weight of 1e−5 on CIFAR. We chose these weights via line search by finding the highest weight that would not hurt test set accuracy.

2. Small weight pruning: Zeroing out weights in a network that are very close to zero. We choose to prune weights less than 1e−3.

E.1.2 A basic improvement for inducing ReLU stability: ReLU pruning

We also use a basic idea to improve ReLU stability, which we call ReLU pruning. The main idea is to prune away ReLUs that are not necessary.

We use a heuristic to test whether a ReLU in a network is necessary. Our heuristic is to count how many training inputs cause the ReLU to be active or inactive. If a ReLU is active (the pre-activation satisfies $\hat{z}_{ij}(x) > 0$) for every input image in the training set, then we can replace that ReLU with the identity function and the

network would behave in exactly the same way for all of those images. Similarly, if a ReLU is inactive ($\hat{z}_{ij}(x) < 0$) for every training image, that ReLU can be replaced by the zero function.

Extending this idea further, we expect that ReLUs that are *rarely* used can also be removed without significantly changing the behavior of the network. If only a small fraction (say, 10%) of the input images activate a ReLU, then replacing the ReLU with the zero function will only slightly change the network's behavior and will not affect the accuracy too much. We provide experimental evidence of this phenomenon on an adversarially trained ($\epsilon = 0.1$) MNIST model. Conservatively, we decided that pruning away ReLUs that are active on less than 10% of the training set or inactive on less than 10% of the training set was reasonable.

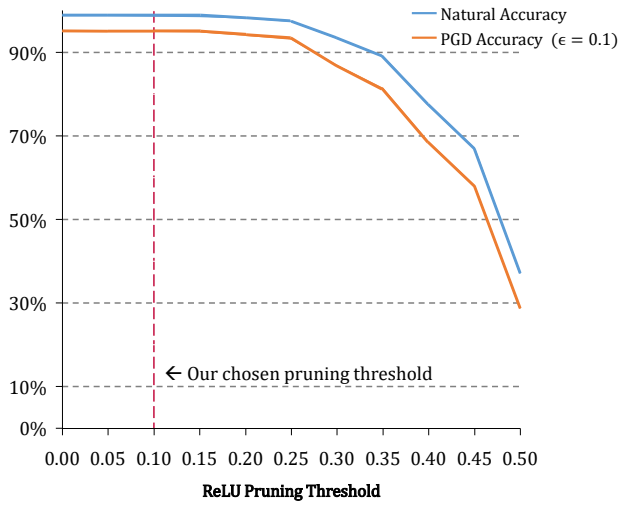


Figure E-1: Removing some ReLUs does not hurt test set accuracy or accuracy against a PGD adversary

E.2 Adversarial training and weight sparsity

It is worth noticing that adversarial training against ℓ_∞ norm-bound adversaries alone already makes networks easier to verify by implicitly improving weight sparsity. Indeed, this can be shown clearly in the case of linear networks. Recall that a linear network can be expressed as $f(x) = Wx + b$. Thus, an ℓ_∞ norm-bound perturbation of the input x will produce the output

$$\begin{aligned} f(x') &= x'W + b \\ &= xW + b + (x' - x)W \\ &\leq f(x) + \epsilon||W||_1 \end{aligned}$$

where the last inequality is just Hölder's inequality. In order to limit the adversary's ability to perturb the output, adversarial training needs to minimize the $||W||_1$ term, which is equivalent to ℓ_1-regularization and is known to promote weight sparsity [Tib94]. Relatedly, [GSS15] already pointed out that adversarial attacks against linear networks will be stronger when the ℓ_1-norm of the weight matrices is higher.

Even in the case of nonlinear networks, adversarial training has experimentally been shown to improve weight sparsity. For example, models trained according to [Mad+18] and [WK17] often learn many weight-sparse layers, and we observed similar trends in the models we trained. However, it is important to note that while adversarial training alone does improve weight sparsity, it is not sufficient by itself for efficient exact verification. Additional regularization like ℓ_1-regularization and small weight pruning further promotes weight sparsity and gives rise to networks that are much easier to verify.

E.3 Interval arithmetic

E.3.1 Naive interval arithmetic

Naive IA determines upper and lower bounds for a layer based solely on the upper and lower bounds of the previous layer.

Define $W^+ = \max(W, 0)$, $W^- = \min(W, 0)$, $u = \max(\hat{u}, 0)$, and $l = \max(\hat{l}, 0)$.

Then the bounds on the pre-activations of layer i can be computed as follows:

$$\hat{u}_i = u_{i-1}W_i^+ + l_{i-1}W_i^- + b_i \tag{E.1}$$

$$\hat{l}_i = l_{i-1}W_i^+ + u_{i-1}W_i^- + b_i \tag{E.2}$$

As noted in [TXT19] and also [Dvi+18b], this method is efficient but can lead to relatively conservative bounds for deeper networks.

E.3.2 Improved interval arithmetic

We improve upon naive IA by exploiting ReLUs that we can determine to always be active. This allows us to cancel symbols that are equivalent that come from earlier layers of a network.

We will use a basic example of a neural network with one hidden layer to illustrate this idea. Suppose that we have the scalar input z_0 with $l_0 = 0, u_0 = 1$, and the network has the following weights and biases:

$$W_1 = \begin{bmatrix} 1 & -1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 2 & 2 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b_2 = 0$$

Naive IA for the first layer gives $\hat{l}_1 = l_1 = [2 \quad 1], \hat{u}_1 = u_1 = [3 \quad 2]$, and applying naive IA to the output $\hat{z}_2$ gives $\hat{l}_2 = 3, \hat{u}_2 = 5$. However, because $\hat{l}_1 > 0$, we know that the two ReLUs in the hidden layer are always active and thus equivalent to the identity function. Then, the output is

$$\hat{z}_2 = z_{11} + z_{12} = \hat{z}_{11} + \hat{z}_{12} = (z_0 + 2) + (-z_0 + 2) = 4$$

Thus, we can obtain the tighter bounds $\hat{l}_2 = \hat{u}_2 = 4$, as we are able to cancel out the z_0 terms.

We can write this improved version of IA as follows. First, letting W_k denote row k of matrix W, we can define the "active" part of W as the matrix W_A, where

$$(W_A)_k = \begin{cases} W_k & \text{if } \hat{l}_{i-1} > 0 \\ 0 & \text{if } \hat{l}_{i-1} \leq 0 \end{cases}$$

Define the "non-active" part of W as

$$W_N = W - W_A$$

Then, using the same definitions for the notation W^+, W^-, u, l as before, we can write down the following improved version of IA which uses information from the previous 2 layers.

$$\begin{aligned}
\hat{u}_i &= u_{i-1}W_{iN}^+ + l_{i-1}W_{iN}^- + b_i \\
&\quad + u_{i-2}(W_{i-1}W_{iA})^+ + l_{i-2}(W_{i-1}W_{iA})^- + b_{i-1}W_{iA} \\
\hat{l}_i &= l_{i-1}W_{iN}^+ + u_{i-1}W_{iN}^- + b_i \\
&\quad + l_{i-2}(W_{i-1}W_{iA})^+ + u_{i-2}(W_{i-1}W_{iA})^- + b_{i-1}W_{iA}
\end{aligned}$$

We are forced to to use $l_{i-1,j}$ and $u_{i-1,j}$ if we can not determine whether or not the ReLU corresponding to the activation $z_{i-1,j}$ is active, but we use l_{i-2} and u_{i-2} whenever possible.

We now define some additional notation to help us extend this method to any number of layers. We now seek to define f_n, which is a function which takes in four sequences of length n – upper bounds, lower bounds, weights, and biases – and outputs the current layer's upper and lower bounds.

What we have derived so far from (E.1) and (E.2) is the following

$$f_1(u_{i-1}, l_{i-1}, W_i, b_i) = (u_{i-1}W_i^+ + l_{i-1}W_i^- + b_i, l_{i-1}W_i^+ + u_{i-1}W_i^- + b_i)$$

Let $\mathbf{u}$ denote a sequence of upper bounds. Let $\mathbf{u}_z$ denote element z of the sequence, and let $\mathbf{u}_{[z:]}$ denote the sequence without the first z elements. Define notation for $\mathbf{l}$, $\mathbf{W}$, and $\mathbf{b}$ similarly.

Then, using the fact that $W_N Z = (WZ)_N$ and $W_A Z = (WZ)_A$, we can show that the following recurrence holds:

$$\begin{aligned}
f_{n+1}(\mathbf{u}, \mathbf{l}, \mathbf{W}, \mathbf{b}) &= f_1(\mathbf{u}_1, \mathbf{l}_1, \mathbf{W}_{1N}, \mathbf{b}_1) \\
&\quad + f_n(\mathbf{u}_{[1:]}, \mathbf{l}_{[1:]}, (\mathbf{W}_2\mathbf{W}_{1A}, \mathbf{W}_{[2:]}), (\mathbf{b}_2\mathbf{W}_{1A}, \mathbf{b}_{[2:]}))
\end{aligned} \tag{E.3}$$

Let $\mathbf{u}_{(\mathbf{x},\mathbf{y})}$ denote the sequence $(u_x, u_{x-1}, \cdots, u_y)$, and define $\mathbf{l}_{(\mathbf{x},\mathbf{y})}$, $\mathbf{W}_{(\mathbf{x},\mathbf{y})}$, and $\mathbf{b}_{(\mathbf{x},\mathbf{y})}$ similarly. Then, if we want to compute the bounds on layer k using all information from the previous k layers, we simply have to compute $f_k(\mathbf{u}_{(\mathbf{k-1},\mathbf{0})}, \mathbf{l}_{(\mathbf{k-1},\mathbf{0})}, \mathbf{W}_{(\mathbf{k},\mathbf{1})}, \mathbf{b}_{(\mathbf{k},\mathbf{1})})$.

From the recurrence E.3, we see that using information from all previous layers to compute bounds for layer k takes $O(k)$ matrix-matrix multiplications. Thus, using information from all previous layers to compute bounds for all layers of a d layer neural network only involves $O(d^2)$ additional matrix multiplications, which is still reasonable

for most DNNs. This method is still relatively efficient because it only involves matrix multiplications – however, needing to perform matrix-matrix multiplications as opposed to just matrix-vector multiplications results in a slowdown and higher memory usage when compared to naive IA. We believe the improvement in the estimate of ReLU upper and lower bounds is worth the time trade-off for most networks.

E.3.3 Experimental results on improved IA and naive IA

In Table E.1, we show empirical evidence that the number of unstable ReLUs in each layer of a MNIST network, as estimated by improved IA, tracks the number of unstable ReLUs determined by the exact verifier quite well. We also present estimates determined via naive IA for comparison.

Dataset	Epsilon	Training Method	Estimation Method	Unstable ReLUs in 1st Layer	Unstable ReLUs in 2nd Layer	Unstable ReLUs in 3rd Layer
MNIST	$\epsilon = 0.1$	Control	Exact	61.14	185.30	31.73
			Improved IA	61.14	185.96 (+0.4%)	43.40 (+36.8%)
			Naive IA	61.14	188.44 (+1.7%)	69.96 (+120.5%)
		+RS	Exact	21.64	64.73	14.67
			Improved IA	21.64	64.80 (+0.1%)	18.97 (+29.4%)
			Naive IA	21.64	65.34 (+0.9%)	33.51 (+128.5%)
MNIST	$\epsilon = 0.2$	Control	Exact	17.47	142.95	37.92
			Improved IA	17.47	142.95	48.88 (+28.9%)
			Naive IA	17.47	142.95	69.75 (+84.0%)
		+RS	Exact	29.91	54.47	24.05
			Improved IA	29.91	54.47	28.40 (+18.1%)
			Naive IA	29.91	54.47	40.47 (+68.3%)
MNIST	$\epsilon = 0.3$	Control	Exact	36.76	83.42	40.74
			Improved IA	36.76	83.44 (+0.02%)	46.00 (+12.9%)
			Naive IA	36.76	83.52 (+0.1%)	48.27 (+18.5%)
		+RS	Exact	24.43	48.47	28.64
			Improved IA	24.43	48.47	31.19 (+8.9%)
			Naive IA	24.43	48.47	32.13 (+12.2%)

Table E.1: Comparison between the average number of unstable ReLUs as found by the exact verifier of [TXT19] and the estimated average number of unstable ReLUs found by improved IA and naive IA. We compare these estimation methods on the control and "+RS" networks for MNIST that we described in Section 6.3.3

E.3.4 On the conservative nature of IA bounds

The upper and lower bounds we compute on each ReLU via either naive IA or improved IA are conservative. Thus, every unstable ReLU will always be correctly labeled as unstable, while stable ReLUs can be labeled as either stable or unstable. Importantly, every unstable ReLU, as estimated by IA bounds, is correctly labeled and penalized by RS Loss. The trade-off is that stable ReLUs mislabeled as unstable will also be penalized, which can be an unnecessary regularization of the model.

In Table E.2 we show empirically that we can achieve the following two objectives at once when using RS Loss combined with IA bounds.

1. Reduce the number of ReLUs *labeled* as unstable by IA, which is an upper bound on the true number of unstable ReLUs as determined by the exact verifier.

2. Achieve similar test set accuracy and PGD adversarial accuracy as a model trained without RS Loss.

Dataset	Epsilon	Training Method	Estimation Method	Total Labeled Unstable ReLUs	Test Set Accuracy		PGD Adversarial Accuracy	
MNIST	$\epsilon = 0.1$	Control	Improved IA	290.5	98.94%		95.12%	
		+RS	Improved IA	105.4	98.68%	(-0.26%)	95.13%	(+0.01%)
		Control (Large)	Naive IA	835.8	99.04%		96.32%	
		+RS (Large)	Naive IA	150.3	98.95%	(-0.09%)	96.58%	(+0.26%)

Table E.2: The addition of RS Loss results in far fewer ReLUs labeled as unstable for both 3-layer and 6-layer (Large) networks. The decrease in test set accuracy as a result of this regularization is small.

Even though IA bounds are conservative, these results show that it is still possible to decrease the number of ReLUs labeled as unstable by IA without significantly degrading test set accuracy. When comparing the Control and "+RS" networks for MNIST and $\epsilon = 0.1$, adding RS Loss decreased the average number of ReLUs labeled as unstable (using bounds from Improved IA) from 290.5 to 105.4 with just a 0.26% loss in test set accuracy. The same trend held for deeper, 6-layer networks, even when the estimation method for upper and lower bounds was the more conservative Naive IA.

E.4 Full experimental setup

E.4.1 Network training details

In our experiments, we use robust adversarial training [GSS15] against a strong adversary as done in [Mad+18] to train various DNN classifiers. Following the prior examples of [WK17] and [Dvi+18b], we introduced a small tweak where we increased the adversary strength linearly from 0.01 to ϵ over first half of training and kept it at ϵ for the second half. We used this training schedule to improve convergence of the training process.

For MNIST, we trained for 70 epochs using the Adam optimizer [KB15] with a learning rate of 1e−4 and a batch size of 32. For CIFAR, we trained for 250 epochs using the Adam optimizer with a learning rate of 1e−4. When using naive IA, we used a batch size of 128, and when using improved IA, we used a batch size of 16. We used a smaller batch size in the latter case because improved IA incurs high RAM usage during training. To speed up training on CIFAR, we only added in RS Loss regularization in the last 20% of the training process. Using this same sped-up training method on MNIST did not significantly affect the results.

Dataset	Epsilon	ℓ_1 weight	RS Loss weight
MNIST	0.1	2e−5	12e−5
MNIST	0.2	2e−5	1e−4
MNIST	0.3	2e−5	12e−5
CIFAR	2/255	1e−5	1e−3
CIFAR	8/255	1e−5	2e−3

Table E.3: Weights chosen using line search for ℓ_1 regularization and RS Loss in each setting

For each setting, we find a suitable weight on RS Loss via line search. The same weight is used for each ReLU. The five weights we chose are displayed above in Table E.3, along with weights chosen for ℓ_1-regularization.

We also train "+RS" models using naive IA to show that our technique for inducing ReLU stability can work while having small training time overhead – full details on "*+RS (Naive IA)*" networks are in Appendix E.5.

E.4.2 Verifier overview

The MILP-based exact verifier of [TXT19], which we use, proceeds in two steps for every input. They are the model-build step and the solve step.

First, the verifier builds a MILP model based on the neural network and the input. In particular, the verifier will compute upper and lower bounds on each ReLU using a specific bound computation algorithm. We chose the default bound computation algorithm in the code, which uses LP to compute bounds. LP bounds are tighter than the bounds computed via IA, which is another option available in the verifier. The model-build step's speed appeared to depend primarily on the tightening algorithm (IA was faster than LP) and the number of variables in the MILP (which, in turn, depends on the sparsity of the weights of the neural network). The verifier takes advantage of these bounds by not introducing a binary variables into the MILP formulation if it can determine that a particular ReLU is stable. Thus, using LP as the tightening algorithm resulted in higher build times, but led to easier MILP formulations.

Next, the verifier solves the MILP using an off-the-shelf MILP solver. The solver we chose was the commercial Gurobi Solver, which uses a branch-and-bound method for solving MILPs. The solver's speed appeared to depend primarily on the number of binary variables in the MILP (which corresponds to the number of unstable ReLUs) as well as the total number of variables in the MILP (which is related to the sparsity of the weight matrices). While these two numbers are strongly correlated with solve times, some solves would still take a long time despite having few binary variables. Thus, understanding what other properties of neural networks correspond to MILPs that are easy or hard to solve is an important area to explore further.

E.4.3 Verifier details

We used the most up-to-date version of the exact verifier from [TXT19] using the default settings of the code. We allotted 120 seconds for verification of each input datapoint using the default model build settings. We ran our experiments using the commercial Gurobi Solver (version 7.5.2), and model solves were parallelized over 8 CPU cores with Intel Xeon CPUs @ 2.20GHz processors. We used computers with 8–32GB of RAM, depending on the size of the model being verified. All computers used are part of an OpenStack network.

E.5 Full experimental verification results

Dataset	Epsilon	Training Method	Test Set Accuracy	PGD Adversarial Accuracy	Verifier Upper Bound	Provable Adversarial Accuracy	Total Unstable ReLUs	Avg Solve Time (s)	Avg Build Time (s)
MNIST	$\epsilon = 0.1$	Adversarial Training*	99.17%	95.04%	96.00%	19.00%	1517.9	2970.43	650.93
		+ℓ_1-regularization	99.00%	95.25%	95.98%	82.17%	505.3	21.99	79.13
		+Small Weight Pruning	98.99%	95.38%	94.93%	89.13%	502.7	11.71	19.30
		+ReLU Pruning (Control)	98.94%	95.12%	94.45%	91.58%	278.2	6.43	9.61
		+RS	98.68%	95.13%	94.38%	94.33%	101.0	0.49	4.98
		+RS (Naive IA)	98.53%	94.86%	94.54%	94.32%	158.3	0.96	4.82
		+RS (Large)**	98.95%	96.58%	95.60%	95.60%	119.5	0.27	156.74
MNIST	$\epsilon = 0.2$	Control	98.40%	93.14%	90.71%	86.45%	198.3	9.41	7.15
		+RS	98.10%	93.14%	89.98%	89.79%	108.4	1.13	4.43
		+RS (Naive IA)	98.08%	91.68%	88.87%	85.54%	217.2	8.50	4.67
		+RS (Large)**	98.21%	94.19%	90.40%	89.10%	133.0	2.93	171.10
		[Won+18]***	95.06%	89.03%	-	80.29%	-	-	-
MNIST	$\epsilon = 0.3$	Control	97.75%	91.64%	83.83%	77.99%	160.9	11.80	5.14
		+RS	97.33%	92.05%	81.70%	80.68%	101.5	2.78	4.34
		+RS (Naive IA)	97.06%	89.19%	79.12%	76.70%	179.0	6.43	4.00
		+RS (Large)**	97.54%	93.25%	83.70%	59.60%	251.2	37.45	166.39
CIFAR	$\epsilon = 2/255$	Control	64.64%	51.58%	50.23%	45.53%	360.0	21.75	66.42
		+RS	61.12%	49.92%	47.79%	45.93%	234.3	13.50	52.58
		+RS (Naive IA)	57.83%	47.03%	45.33%	44.44%	170.1	6.30	47.11
		+RS (Large)**	61.41%	50.61%	51.00%	41.40%	196.7	29.88	335.97
CIFAR	$\epsilon = 8/255$	Control	50.69%	31.28%	33.46%	7.09%	665.9	82.91	73.28
		+RS	40.45%	26.78%	22.74%	20.27%	54.2	22.33	38.84
		+RS (Naive IA)	46.19%	29.66%	26.07%	18.90%	277.8	33.63	23.66
		+RS (Large)**	42.81%	28.69%	25.20%	19.80%	246.5	20.14	401.72

Table E.4: Full results on natural improvements from Table 6.1, control networks (which use all of the natural improvements and ReLU pruning), and "+RS" networks from Tables 6.2 and 6.3. While we are unable to determine the true adversarial accuracy, we provide two upper bounds and a lower bound. Evaluations of robustness against a strong 40-step PGD adversary (PGD adversarial accuracy) gives one upper bound, and the verifier itself gives another upper bound because it can also prove that the network is *not robust* to perturbations on certain inputs by finding adversarial examples. The verifier simultaneously finds the provable adversarial accuracy, which is a lower bound on the true adversarial accuracy. Build times and solve times are reported in seconds. Finally, average solve time includes timeouts. In other words, verification solves that time out contribute 120 seconds to the total solve time.
* The "Adversarial Training" network uses a 3600 instead of 120 second timeout and is only verified for the first 100 images because verifying it took too long.
** The "+RS (Large)" networks are only verified for the first 1000 images
*** [Won+18; Dvi+18b], and [MGV18], which we compare to in Table 6.3, do not report results on MNIST, $\epsilon = 0.2$ in their papers. We ran the publicly available code of [Won+18] on MNIST, $\epsilon = 0.2$ to generate these results for comparison.

E.6 Discussion on verification and certification

Exact verification and certification are two related approaches to formally verifying properties of neural networks, such as adversarial robustness. In both cases, the end goal is formal verification. Certification methods, which solve an easier-to-solve relaxation of the exact verification problem, are important developments because exact verification previously appeared computationally intractable for all but the smallest models.

For the case of adversarial robustness, certification methods exploit a trade-off between provable robustness and speed. They can fail to provide certificates of robustness for some inputs that are actually robust, but they will either find or fail to find certificates of robustness quickly. On the other hand, exact verifiers will always give the correct answer if given enough time, but exact verifiers can sometimes take many hours to formally verify robustness on even a single input.

In general, the process of training a robust neural network and then formally verifying its robustness happens in two steps.

- Step 1: Training
- Step 2: Verification or Certification

Most papers on certification, including [WK17; Won+18; Dvi+18b; RSL18] and [MGV18], propose a method for step 2 (the certification step), and then propose a training objective in step 1 that is directly related to their method for step 2. We call this paradigm "co-training." In [RSL18], they found that using their step 2 on a model trained using [WK17]'s step 1 resulted in extremely poor provable robustness (less than 10%), and the same was true when using [WK17]'s step 2 on a model trained using their step 1.

We focus on MILP-based exact verification as our step 2, which encompasses the best current exact verification methods. The advantage of using exact verification for step 2 is that it will be accurate, regardless of what method is used in step 1. The disadvantage of using exact verification for step 2 is that it could be extremely slow. For our step 1, we used standard robust adversarial training. In order to significantly speed up exact verification as step 2, we proposed techniques that could be added to step 1 to induce weight sparsity and ReLU stability.

In general, we believe it is important to develop effective methods for step 1, given that step 2 is exact verification. However, ReLU stability can also be beneficial for tightening the relaxation of certification approaches like that of [Won+18] and

[Dvi+18b], as unstable ReLUs are the primary cause of the overapproximation that occurs in the relaxation step. Thus, our techniques for inducing ReLU stability can be useful for certification as well.

Finally, in recent literature on verification and certification, most works have focused on formally verifying the property of adversarial robustness of neural networks. However, verification of other properties could be useful, and our techniques to induce weight sparsity and ReLU stability would still be useful for verification of other properties for the exact same reasons that they are useful in the case of adversarial robustness.