Testing Deep Image Classifiers using Generative Machine Learning

Isaac Dunn

Balliol College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Prof. Tom Melham and Prof. Daniel Kroening, for their generous support over the past four years. Their guidance and input on research directions, writing, publication, conferences, careers, motivation, and many other important matters has been invaluable. Thank you.

I am also extremely grateful to Hadrien Pouget for the many stimulating discussions, words written, and hours programming that we shared – your contributions and encouragement were much appreciated. Laura Hanu proved another excellent collaborator, providing reliable code and insightful suggestions.

I have been fortunate to have been funded by the Engineering and Physical Sciences Research Council, which has made my research possible.

I am grateful to all of my friends who have played a role in supporting me through a degree that has, at times, been challenging. Particular thanks go to Natasha Jeppu, Sophie Gullino, Holly Hunt, Will Payne, Alan Taylor, Leonor Aidos, and Linda Ma – thank you.

Last, I want to express thanks to my parents and siblings. I feel very lucky to be able to rely on your kindness and camaraderie, even when the world completely shuts down.

Abstract

Although deep neural networks (DNNs) attain excellent performance on the specific tasks they are trained for, this often seems to be obtained using easier-to-learn proxies for the truly relevant concepts. The problem with proxies is that they cannot be relied on in new situations – the proxy departs from the true concept. And DNNs will very likely be deployed in new situations because the world is always changing, and training data are never exhaustive.

Unfortunately, existing approaches are limited in their ability to diagnose the bad proxies being relied on by a DNN. We also cannot accurately characterise a model's generalisation performance on different kinds of data beyond its training task. These limitations additionally prevent us from developing systems that do not rely on bad proxies.

To improve this situation, this thesis introduces two new test generation procedures for image classification DNNs. These improve on existing approaches by identifying more kinds of inputs for which a particular DNN gives incorrect outputs. This is achieved by exploiting generative machine learning to solve the test oracle problem in new ways. The first new procedure trains a generative network to directly output test cases that identify failures. The second dynamically perturbs the activation values of a pretrained generative network as it generates new examples – the perturbations adjust the features of the generated data so that they also induce failures in the DNN being evaluated.

Besides the primary contribution of these algorithms, this thesis also presents an empirical finding: standard adversarial training that aims to increase model robustness surprisingly *decreases* DNNs' ability to generalise correctly to changes in high-level features such as object position, orientation, shape or colour.

Contents

List of Figures

List of Tables

Contributions to Co-Authored Works

This thesis draws heavily from the three papers listed below, including some sections taken almost verbatim from the original paper texts. In short, I did almost all of the ideation, setting of research direction, implementation and writing work for all three papers.

Throughout, my supervisors Prof. Tom Melham and Prof. Daniel Kroening continuously provided helpful high-level guidance on all matters relating to these papers: research direction, experiments, and writing. They did not contribute directly to any implementation or experiments, and made some helpful but modest fine-grained edits to the texts of the papers. I am grateful to them for allowing me to work with two excellent research assistants, Hadrien Pouget and Laura Hanu, whose contribution is disentangled from my own below.

My first paper, 'Adaptive Generation of Unrestricted Adversarial Inputs' [1], forms the basis of Chapter 4. I contributed the ideas, wrote the code, and ran all the experiments except for the evaluation of the performance of the existing approach Song et al. [2] in the presence of adversarial training, as described in section 4.4.4, which was implemented by Hadrien Pouget. I also wrote most of the paper, with help from Hadrien (especially in revising and improving later versions).

My second paper, 'Evaluating Robustness to Context-Sensitive Feature Perturbations of Different Granularities' [3], forms the basis of Chapters 5 and 7. I was the source of the original ideas, set the research direction, implemented the software tool, and led on running experiments. Laura Hanu integrated the BigGAN into the codebase so that it was in a suitable format to be used by the tool. Hadrien Pouget helpfully assisted with experiments and contributed to high-level discussions about the paper. I did the large majority of the writing, with the remaining contributions from Hadrien.

My third paper, 'Exposing Previously Undetectable Faults in Deep Neural Networks' [4], forms the basis of Chapter 6. Although I also led on this paper, it was a more equal collaboration with Hadrien Pouget. The software implementation was split fairly evenly between me and Hadrien, and Hadrien also contributed to the writing.

A note on pronouns

For reasons of style only, I have avoided using singular first-person pronouns ('I', 'my', etc.) in this dissertation, preferring the plural 'we' etc. This should not be used to infer anything about the division of responsibility of any work. Often, 'we' refers to my sole contribution; rely on the above descriptions to determine the division of credit, rather than assuming that 'we' means the contribution of collaborators.

1 Introduction

Contents

Can deep neural networks be relied upon? Despite the rise of deep learning, there is far to go in identifying, understanding and addressing limitations in their ability to generalise beyond the distribution of data they were trained on. This thesis contributes to progress in this area by developing and applying two new test generation procedures that are better able to detect failures and faults in deep neural networks.

1.1 Motivation

1.1.1 Deep learning models learn to use shortcut features

Shortcut learning is undesirable

Suppose we wish to train a classifier to distinguish between images of dogs and wolves, and that the training data includes dogs only on grassy backgrounds and wolves only on snowy backgrounds. It would be possible to obtain high

performance on this data by looking only at the backgrounds, ignoring the animals entirely. If a classifier did this, it would have learned a 'shortcut': although performance on the training task would be good (including on a hold-out test set, so the problem is not overfitting), it would be relying on incorrect features. Incorrect features in what sense? Incorrect in the sense that reliance on them will fail to generalise to new situations – in this example, a dog in the snow would be classified incorrectly. So shortcut learning is when a model learns features that perform well on data drawn from the same distribution of the training distribution, but that fail to generalise "out of distribution" to data that differs from the training distribution.

At first blush, such failure to generalise well to new data "out of distribution" does not seem worrying. Out-of-distribution generalisation is asking for more than we might reasonably demand – we optimise our models solely for performance on training data, providing no guarantees about their performance on entirely new data.

But the 'i.i.d.' assumption – that data will be independently drawn from an identical distribution – has been called the 'the big lie in machine learning' [5, 37:44]. The world is complex and changes by time and place. For example, the COVID-19 pandemic caused 2020 to be very different to prior years in many application domains. But rare and dramatic events are not the only source of change: technological and cultural development drive constant change in almost every area of life and industry. In addition to data changing by time and place, it is challenging to gather training data that captures the full distribution of interest; selection biases are often present.

Given that the 'i.i.d.' assumption cannot be depended on, we must rely on our models' ability to generalise out of distribution. So learning shortcuts should be avoided.

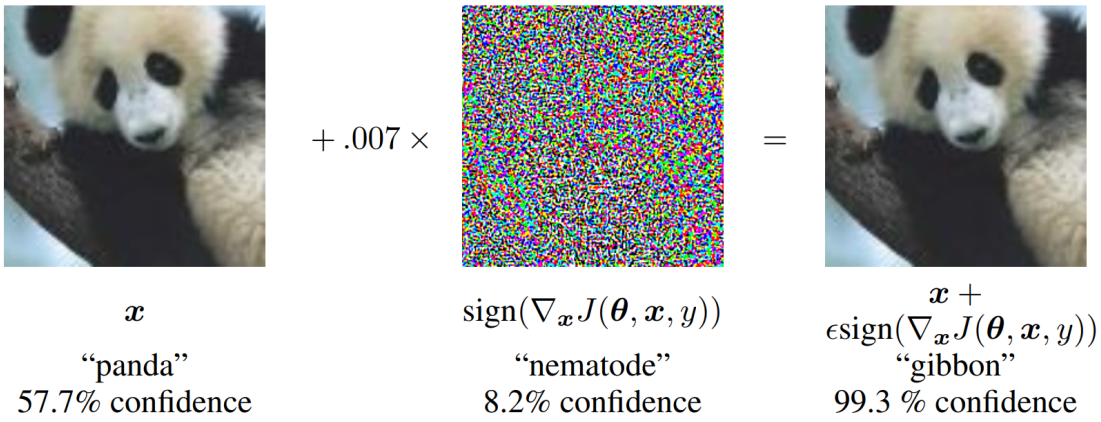


Figure 1.1: An 'adversarial example' from Goodfellow, Shlens, and Szegedy [7], created by slightly perturbing the values of each pixel, so that the perturbed image is incorrectly classified.

Deep neural networks do learn shortcuts in practice

Given redundancy in a dataset, there are many possible features that could each be relied on to perform well on the training task. Because it intuitively seems that most of these are shortcut features that cannot be relied upon in all situations, we might intuitively expect models trained only to maximise training performance to rely on shortcut features by default.

There is considerable empirical evidence that this intuition is correct. That is, deep neural networks do learn to rely on shortcut features in practice [6].

The well-known so-called 'adversarial examples' phenomenon (introduced fully in Section 2.3) is, in short, that models with excellent performance can be shown to be consistently reliant on fragile shortcut features. It is possible to take a data point drawn from the training distribution and make almost imperceptible changes to it, thereby changing a model's output from correct to completely incorrect. An example of such a pixel perturbation from the image classification domain is shown in Figure 1.1. We might hope that our systems might reliably generalise to imperceptibly different data, even if those data were not present during training. In response to Ilyas et al.'s [11] cogent claim that reliance on non-robust shortcut features is a primary cause of robustness to adversarial perturbations, Gilmer and Hendrycks [12] comment that a broader understanding

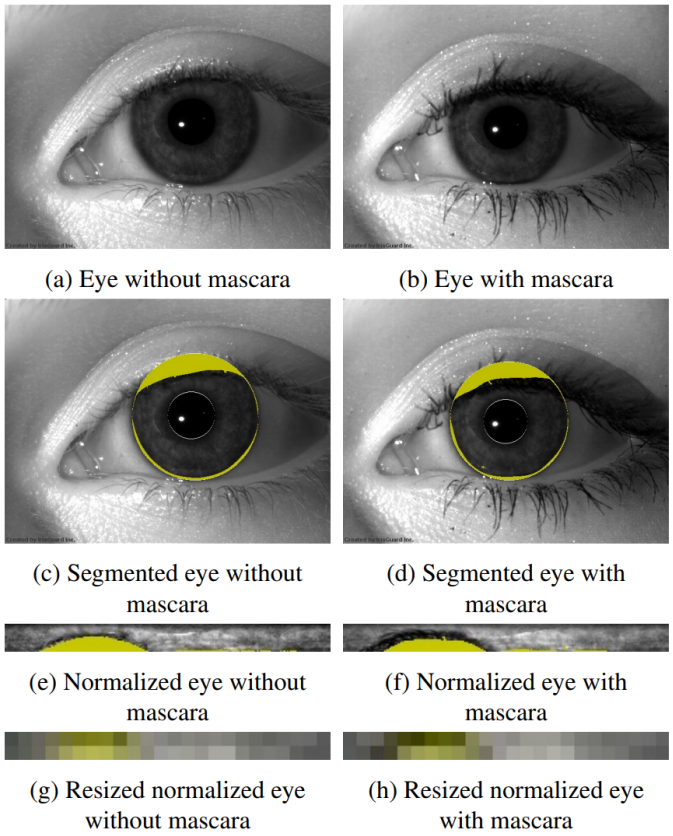


Figure 1.2: Illustration from Kuehlkamp, Becker, and Bowyer [8] of the difference between eyes with and without make-up. The mascara, still noticeable after segmentation and normalisation, was used as a 'shortcut' feature by gender classification models. Reproduced with permission. ©2017 IEEE.

of distributional robustness is required: in general, there is no reason to expect a model to perform well when given an entirely different distribution of inputs.

Models that hoped to predict gender from iris texture were shown to be relying on the presence of mascara as a proxy [8]. This proxy works on the training dataset, but fails to generalise to a similar dataset with no eye makeup. See Figure 1.2.

Models trained to identify animals struggle to generalise correctly to locations different from those included in the training data [9], suggesting that location-specific shortcut cues are being relied on. See Figure 1.3. The WILDS benchmark [13] includes the heat- and motion-activated wildlife camera images from different locations as one of its ten datasets for the evaluation of models under distribution shift. In all cases, out-of-distribution performance (for instance, on

(A) **Cow: 0.99**, Pasture: 0.99, Grass: 0.99, No Person: 0.98, Mammal: 0.98

(B) No Person: 0.99, Water: 0.98, Beach: 0.97, Outdoors: 0.97, Seashore: 0.97

(C) No Person: 0.97, **Mammal: 0.96**, Water: 0.94, Beach: 0.94, Two: 0.94

Figure 1.3: Figure showing that image recognition algorithms generalise poorly to new environments. In this example, cows in their expected context are correctly identified (A), but in the uncommon setting of a beach are either not detected (B) or classified poorly (C). Reproduced from Beery, Horn, and Perona [9] by permission from Springer Nature, ©2018. Note that these examples use a third-party (ClarifAI.com) model, rather than the models trained by Beery, Horn, and Perona [9].

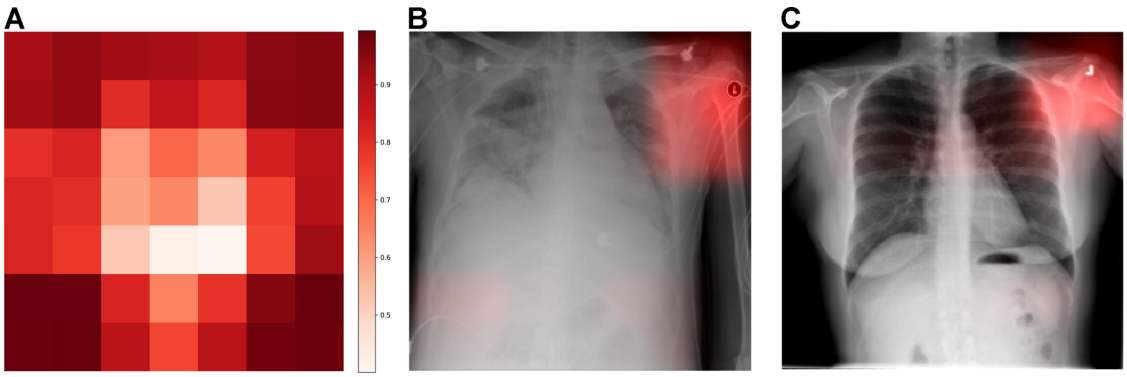


Figure 1.4: Figure from Zech et al. [10] showing which regions contributed most to classification decisions. Radiology workers put location-specific metal tokens in the corners of the scans: (A) shows that over all scans, the corner regions, containing these tokens rather than lungs, make an especially big contribution to decisions; (B) and (C) show the contributions for particular scans, focusing on the corner with the token. Reproduced under the paper's creative commons licence.

images from a different location) is substantially worse than performance on (unseen) data from the training distribution.

Efforts to train a convolutional neural network to detect pneumonia from radiological scans were found to suffer from shortcut learning [10]. Despite good performance on new data from the same sites used during training, performance on scans from different hospitals was much worse. Investigation suggests that the models were able to predict exactly where the scan was taken – in particular by looking at metal tokens placed by technicians in the corners of images – and that scan origin was relied upon as a proxy for the likelihood of pneumonia. See Figure 1.4.

The problem is not specific to vision models. Over-reliance on superficial patterns and associations in limited evaluation sets led state-of-the-art common-sense reasoning models to perform much worse when these spurious correlations were removed [14, 15]. Miller et al. [16] produce new test datasets for the Stanford Question Answering Dataset (SQuAD) and find that even though state-of-the-art models have not overfitted in any sense (because they are able to generalise equally well to a new test set from the training distribution), shifts from that Wikipedia domain to New York Times articles, Reddit posts and Amazon product reviews result in significant performance drops.

Besides these specific examples of shortcut learning, refer to Geirhos et al.'s excellent overview of shortcut learning [6] for many more examples. Another framing of shortcut learning is that the training process is *underspecified*: many more hypotheses have equivalently good performance on the training task, but different performance in various deployment domains [17]. That we end up with shortcut features is therefore a failure of specification, rather than a failure of learning.

1.1.2 Problem: limited diagnosis of learned shortcuts

We know that our models can learn shortcut proxies that do not generalise to out-of-distribution data as we would wish. But there are several ways in which

our present abilities and tools are limited.

Given a particular model, we cannot reliably identify the shortcuts it has learned. We are largely ignorant of how any model will behave on out-of-distribution data of various kinds. So decisions about whether deep neural networks can be relied upon in deployment must be taken in the dark. Decision makers must choose between deploying and risking poor out-of-distribution performance causing problems, or not deploying and missing out on the benefits that the system could offer. It would be much better if such decisions could be informed by a characterisation of the DNN's limitations and weak spots.

In general, we do not understand why shortcut learning arises. We do not have theory relating the training data, training task and model architecture to the shortcuts learned and out-of-distribution performance of a model. We therefore do not understand how to design models and training processes so as to create models that have the out-of-distribution generalisation properties we want. It is quite possible that, until this is achieved, all models that we train will learn shortcuts that make them fundamentally unsuitable for deployment in important contexts in which reasonable out-of-distribution behaviour is necessary.

In addition, we do not yet have conceptual clarity about the properties we might require from our systems. We cannot say which shortcuts are harmless and which are problematic because we cannot specify the set of out-of-distribution data that we require reasonable behaviour on, and using the full set of possible inputs as an upper bound is infeasible.

1.2 Aims

The core contribution presented in this thesis is two new testing algorithms that represent progress in our ability to identify different out-of-distribution inputs that a given DNN fails to generalise to. Before these are described in Sections 1.3.1 and 1.3.2, this section explains how such testing algorithms address the problems we have identified and describes the specific requirements that such testing algorithms must fulfil in order to make a significant contribution.

1.2.1 Key idea: improved testing of DNNs

Suppose that we had an algorithm for testing DNNs that, given a model, could identify all the inputs for which the model's output was incorrect. Such an algorithm would better inform decisions about whether to deploy models. By comparing the identified set of all inputs for which a model misbehaves to the set of inputs that could plausibly occur during deployment, we gain a better sense of how well the model is likely to perform.

But in practice, the set of all inputs for which the model gives incorrect outputs will be far too large to comprehensively compare to the set of plausible deploy-time inputs. Instead, through inspection and analysis of a non-exhaustive subset, we might draw inferences about the shortcut proxies being used by that particular model. So we could make more accurate predictions about the kinds of data that the model would correctly generalise to, and therefore make better decisions about whether the model can be relied upon.

By analysing multiple models in this way, we might learn general lessons about the kinds of shortcut proxies that DNNs learn, and therefore the kinds of situations in which these systems can and cannot be trusted. By comparing the out-of-distribution generalisation of models that differ in their architecture or training, we might also develop insights into the causal relationship between these aspects of model development and the shortcut proxies learned.

These insights into shortcut learning would enable us to engineer systems with improved out-of-distribution generalisation. Even without those insights, the algorithm's ability to evaluate out-of-distribution generalisation would facilitate a trial-and-error approach to development of improved models.

With experience, we may also develop greater conceptual clarity, allowing us to better characterise the kinds of out-of-distribution generalisation behaviours that are and are not problematic in practice.

It is not feasible that we could soon develop a practical algorithm that could identify *all* of the inputs for which a given model gives the wrong output. But developing a new algorithm that is able to identify many more such inputs

than the limited current testing approaches would be a significant research contribution, because all of the benefits listed above would still apply. The better the tools we have to probe and analyse out-of-distribution generalisation, the better our ability to identify problems in our systems, and the better our ability to understand and address them. The main work described in this thesis is the development and evaluation of two such new testing algorithms.

1.2.2 Application domain: image classification

Although shortcut learning is a concern with DNNs in general, we will restrict our attention in this thesis to one particular application domain: image classification. For our purposes, an image is a matrix of real numbers, where each element represents the colour of a pixel – that is, a member of $\mathbb{R}^{m \times n}$, or $\mathbb{R}^{m \times n \times 3}$ in the case of three colour channels.

A classification task is determined by an *oracle* partial function. The oracle $o : \mathbb{X} \rightharpoonup \mathbb{Y}$ for a classification task maps some elements of $\mathbb{X}$ to the correct corresponding 'classes' or 'labels' from the fixed set $\mathbb{Y}$. Typically, this oracle is human judgement about a task that we are trying to approximate. The oracle function is only partial because some possible inputs do not belong to any class, perhaps because they are ambiguous, ill-formed, meaningless or irrelevant. For instance, most of the set of possible images (2D pixel arrays) are meaningless noise, like the static on an analogue television. The ImageNet Large Scale Visual Recognition Challenge [18] is a representative image classification task that will be used in this thesis: photographs of creatures and objects are mapped to one of 1,000 possible classes.

There are several advantages to an exclusive focus on image classification. Significant research attention has been devoted to image classification, so there are many useful datasets and pretrained models available. Images are easy to display on screens and in print, allowing for faster interpretation of results, simpler experiments, and better communication to readers of papers. There are also benefits to focus, allowing research efforts to be concentrated on the most

important questions rather than tedious engineering. Last, although lessons seem likely to generalise beyond image classification, this domain is significant enough in itself that investigating this generalisation is not necessary.

1.2.3 Our requirements for successful test generation

In this section, we will spell out requirements that are sufficient for a new testing algorithm to constitute a significant and valuable contribution.

Summary

In short, we would like test generation algorithms that:

1. Produce well-formed test inputs that are assigned meaning by the task oracle.
2. Produce test inputs for which the classifier being tested makes incorrect label predictions.
3. Can detect as wide a range of problems as possible.
4. Are efficient and practical.

These requirements are clarified and justified in turn below.

1. Production of meaningful test cases

We would like algorithms that produce test cases that are well-formed inputs that are assigned meaning by the oracle for the classification task. Otherwise, the model cannot be said to be incorrect for that case.

For instance, if the two class labels allowed are "bird" or "bicycle", then test cases that include both a bird and a bicycle, or that include neither a bird nor a bicycle, or that are otherwise unclear are not useful, as in Figure 1.5. Note also that almost all possible images (2D pixel arrays) are meaningless noise, and so would not be suitable test inputs.

Another relevance of this requirement is as a proxy for inputs we care about in practice. We cannot assume that the training distribution is representative of the data that may be encountered during deployment. But we also do not know which data may be encountered during deployment – if we did, then we could simply include these during training. So, at this early stage of investigation, we will assume that any input that is meaningful is one for which we could plausibly require good performance during deployment. In addition, allowing an exploration of all meaningful inputs rather than prematurely restricting our focus is useful for better understanding how models tend to generalise.

2. Causing the classifier to output incorrect predictions

An often-cited useful perspective attributed to Dijkstra is that "testing shows the presence, not the absence of bugs" [20]. A test that finds no problem gives almost no information about whether any problems actually exist; a test for which the tested model diverges from the oracle is a concrete failure. So we will require that generated test cases identify failures in the following sense:

Definition 1.1 (Failure-identifying test case)**.** A test case $(x, y) \in \mathbb{X} \times \mathbb{Y}$ for oracle $o : \mathbb{X} \rightharpoonup \mathbb{Y}$ and model $f : \mathbb{X} \to \mathbb{Y}$ identifies a failure if $o(x)$ is defined, $y = o(x)$, and $f(x) \neq y$.

In order to accurately identify a failure, we need not only a test input x that causes an incorrect classifier prediction, but we also need to know the correct

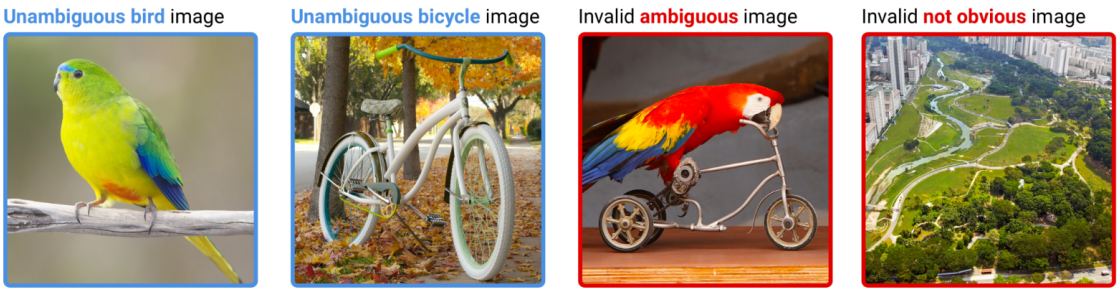


Figure 1.5: Examples from Brown et al.'s "unrestricted adversarial examples challenge" [19] showing images that would be unsuitable test input because they do not clearly have a meaning that belongs to one of the relevant classes ("bird" or "bicycle").

expected output $y = o(x)$. Because we do not have cheap access to the task oracle o, we need to have a way of automatically obtaining the correct label $y = o(x)$. This 'test oracle problem' is a significant challenge.

3. Detection of a wide range of problems

We would like test generation algorithms that can catch as wide a range of problems as possible. There are two possible senses of the word 'problem' that could apply here: fault (underlying problem), or failure (specific outcome that is symptomatic of a fault), as distinguished by the IEEE standard glossary [21].

We want to detect as many faults (underlying problems) as possible, because detection is necessary for understanding, which in turn is necessary to address the causes of the problems. But it is subjective what exactly constitutes a fault in a deep neural network: unlike conventional software, it is less likely that the fault is a human programmer's mistake. Instead, it is likely the problem has arisen through the training process, even if implemented as intended.

So we will largely focus on failures, because they are easier to define and measure. That is, we want our test generation algorithms to detect as wide a range of specific instances of incorrect behaviour as possible. Since each failure is caused by an underlying fault, the detection of failures is helpful in the identification of faults.

So we want our test generation procedures to be able to output as many distinct failure-identifying test cases as possible. In particular, a procedure that could output classes of failures that were undetectable by previous approaches would be significantly more useful than one that could only test cases that could have been part of a hold-out test set from the training distribution, or only cases that are within an ℓ_∞-constrained pixel perturbation of such examples.

4. Efficiency and practicality

We would like test generation algorithms that are efficient and practical. Even if an algorithm is only intended to be used for scientific ends only, being cheaper

and more convenient increases the number of experiments that can be feasibly run. Relevant costs include overhead computation (such as training models), marginal computation for each extra test generated, necessary hardware, soft assets such as labelled data or pretrained models, and labour.

Typically, testing accounts for very roughly half of the resources and efforts in developing software [22]. So we should aim for the overall testing costs to be roughly the same order of magnitude as the cost of developing the system being tested, in order not to be prohibitively expensive. Of course, being cheaper than this would be better.

1.3 Contributions

The main research contributions presented in this dissertation are two new procedures for the testing of deep neural networks. In particular, these two algorithms meet the requirements established in Section 1.2.3. In different ways, both of the new algorithms depend centrally on generative neural networks (networks that model the full distribution of training data, as opposed to discriminative models that model only the conditional probability of a label given a data point). The first, described in Section 1.3.1, introduces a new way of training generative networks so that they learn to generate suitable test cases for a given model. The second, described in Section 1.3.2, dynamically perturbs the activation values of pretrained generative networks as they generate new examples, so that the generated data are optimised to probe the behaviour of a given model.

The thesis of this work is that generative models can be exploited in this way to generate tests in such a way that meets our specified requirements, and therefore improve our ability to identify failures and faults caused by shortcut learning in deep neural networks.

As well as the contribution of the two new test generation algorithms, this dissertation also presents a surprising empirical finding about the relationship between the training of deep neural networks and their out-of-distribution generalisation properties. This is summarised in Section 1.3.3 and demonstrates

that, as hoped, improved test generation algorithms can facilitate useful insights into shortcut learning.

1.3.1 Training generative networks to output test cases

The first research contribution of this dissertation, presented in Chapter 4, is one of two novel algorithms that generate tests to identify failures of generalisation in deep neural networks.

Test generation procedure

In short, the training scheme for generative adversarial networks (GANs) is modified so that the generator network of the GAN pair learns to generate tests for a given image classifier. The generator network in a standard GAN is incentivised during training to generate examples that the discriminator network is unable to distinguish from examples drawn from the training distribution. To train a generator network to generate OOD tests, we introduce a new loss term that incentivises the generation of test inputs that reveals weaknesses in the system being evaluated. This additional loss term is optimised simultaneously with the standard generator training loss. With the help of some new techniques to make this training process converge as desired, the end result is a generator network whose generated outputs are good out-of-distribution tests for the model being evaluated.

Performance on requirements

As detailed in Section 4.9, this test generation algorithm meets all four of the requirements introduced in Section 1.2.3. Studies with human judges acting as the oracle found that not only are the generated tests successful in causing classification failures in the tested models, but they often cannot be identified as being artificial from a line-up of dataset examples.

By virtue of solving the test oracle problem using a conditional generative network rather than a constrained perturbation, this approach is able to generate

(a) Original image. **(b)** Diff from perturbation. **(c)** Perturbed image.

Figure 1.6: An example of changing the computed classification from 'volcano' to target label 'goldfish' using a context-sensitive feature perturbation. Coarser-grained changes include darkening the sky, causing an eruption of lava, and adding a rocky outcrop in the foreground; finer-grained changes include slightly flattening the curve of the volcano, and adjustments to the texture of the trees, rocks and cloud.

test cases and so identify problems that are not possible using existing approaches. This is verified with a range of empirical experiments.

After the overhead cost of training a GAN as required, each marginal generated test is very cheap, costing only a single forward pass through the generator for each batch.

1.3.2 Generating tests by perturbing generative network activations

The second research contribution of this dissertation is also a novel algorithm that generates tests for DNNs, exploiting generative networks in quite a different way. Presented in Chapter 5, it introduces a new way of making perturbations to data as a way of producing test inputs.

Test generation procedure

By making perturbations to data, we mean that it takes known test inputs and makes changes that maintain their oracle-assigned semantics while altering the tested model's outputs. In particular, this procedure exploits the latent representations learned by a fixed, already trained generative network. By performing constrained perturbations to the activation values during a forward

pass of such a network, this procedure is able to effect a *context-sensitive* change to the generated test instance. As standard, the direction and magnitude of the perturbation are computed using a gradient-walking optimisation procedure with the goal of inducing a failure in the tested model without making too large a change. Figure 1.6 demonstrates the effect of a perturbation identified by this procedure.

Perturbing activation values in earlier layers in the generator network results in changes to high-level features such as object shape, location, colour or orientation, while perturbations in later layers are more localised and affect features at a finer level of granularity, such as texture. By default, our algorithm performs perturbations at all layers of the generator network, although for some experiments and purposes it makes sense to restrict these to only affect some layers.

Performance on requirements

Section 5.5 examines the evidence that this test generation algorithm meets the necessary requirements. The perturbations are easily able to produce test inputs that cause the tested classification models to give incorrect outputs. Experiments with human judges verified that the perturbed images maintain their original meanings as necessary.

By exploiting the representations learned in all layers of a generative model, the perturbations made by this procedure are context sensitive (as opposed to pixel perturbations that ignore the features present in each image), and can cause changes at all levels of granularity from very local (fine) to global (coarse). Chapter 6 is an empirical investigation of whether this algorithm is able to identify faults that could not be identified by existing approaches.

Although this can be increased or decreased as necessary, the optimisation process for a single high-resolution ImageNet perturbation takes on the order of one minute. Experiments with a range of datasets verify the ability to scale in practice.

1.3.3 Empirical finding relating adversarial training to generalisation behaviour

This dissertation also presents a striking empirical finding, concerning the relationship between how a model is trained and its out-of-distribution generalisation behaviours. This finding was made using the new perturbation-based algorithm described in the previous section, and is a promising example of how such improved testing procedures being used to test DNNs can lead to useful insights that may be useful in the development of trustworthy systems.

There has been much focus on a particular perturbation approach in computer vision, which we will call the 'pixel perturbation' approach. This involves making a context-insensitive adjustment to the value of each pixel, with an ℓ_p norm ($\|x\|_p = (\Sigma_i |x_i|^p)^{\frac{1}{p}}$) constraining the magnitude of the overall perturbation so as to preserve image semantics. By default, deep neural network image classifiers are not robust to such perturbations; they fail to generalise to deliberately perturbed images. The most promising approach to improving models' generalisation to these kinds of inputs is 'adversarial training': the inclusion of current worst-case perturbations during training.

Optimistically, we might hope that improving a model's ability to generalise to a particular set of inputs that are not part of the training dataset might reliably improve its ability to generalise outside this training distribution in general. Chapter 7 explores whether this is the case.

Disappointingly, it seems that this is not the case. When it comes to generalisation to changes to high-level features (e.g. shape, orientation, position, colour), we have found that adversarial training against pixel-space perturbations is not just unhelpful: it is *counterproductive*.

Empirical evidence that adversarial training against pixel perturbations decreases models' robustness to changes of coarser granularity is presented in Chapter 7. This finding was made using the new context-sensitive perturbation algorithm presented in Chapter 5. The evidence comes from experiments comparing perturbation magnitudes for image classification networks with

standard training to those that have undergone adversarial training against pixel perturbations, for different kinds of perturbation. By measuring the magnitudes of the perturbations to latent activation values required to find a test input that a model classifies incorrectly, we are measuring the robustness of the model. If it is often easy to find a small perturbation, the model generalises poorly to even small changes of the kind being applied; if larger perturbations are required, the model is more robust. While adversarially trained models indeed perform better (smaller magnitudes) under perturbations at later layers in the network as we might expect, they perform no better under mid-generator perturbations, and significantly *worse* when perturbations were performed in the early layers only. These early-layer perturbations are shown to correspond to high-level, coarse-grained feature changes.

2
Background

Contents

This chapter introduces background material that is useful to understand the rest of this dissertation. Readers may wish to skip topics they are already familiar with. Appendix A provides background to the background: an introduction to deep neural networks.

Section 2.1 describes approaches to generative machine learning in particular, with a focus on deep neural networks. The following section, 2.2, presents some existing work suggesting the kinds of meanings and representations encoded by individual neuron units in different kinds of deep neural networks. The last section, 2.3, introduces the phenomenon of so-called adversarial examples: by making small worst-case perturbations to the values of the pixels of an image, it is by default easy to cause an image classification model to perform very poorly. The section also introduces techniques intended to create models that do not suffer from this problem, including adversarial training.

2.1 Generative machine learning

Let us draw a distinction between *discriminative* models and *generative* models. Note that this is not a strict, formal dichotomy that all models are cleanly divided into, but a helpful distinction that will be useful for our ends. In particular, the work presented in this dissertation largely concerns unambiguously discriminative models being evaluated using approaches that exploit unambiguously generative models.

For our purposes, a discriminative model is one that models the distribution of some unobserved variable y *conditional* on a particular observed variable x. Classification is an archetypical discriminative task: a classification model predicts which of a finite number of classes each input belongs to. Regression – modelling of a continuous value associated with each input – is also discriminative.

A generative model, in contrast, simply models the distribution of samples drawn from some training source. That is, rather than modelling some unseen variable associated with each particular instance drawn from the training distribution ($p(x \mid y)$), a generative model models that training distribution entirely ($p(x)$, or $p(x, y)$ if each training instance includes additional variables y). This model may be an explicit estimate of the target distribution, or may be only implicit, as in the case of models that are only able to generate individual samples drawn from (an approximation of) the training distribution. While there are many discriminative and generative models that are not deep neural networks, deep neural networks will be the almost exclusive focus of this dissertation.

2.1.1 Generative adversarial networks (GANs)

Generative adversarial networks (GANs) [23] are a class of generative machine learning models involving the simultaneous training of two deep neural networks: a generator g and a discriminator d. Specifically, given a dataset $D \subseteq \mathbb{X}$ of samples drawn from a probability distribution p_D, the generator $g\colon \mathbb{Z} \to \mathbb{X}$ learns to transform random noise z drawn from a standard distribution p_z (typically

Gaussian) into an approximation of p_D. The discriminator network $d\colon \mathbb{X} \to \mathbb{R}$ learns to predict whether a given example x is drawn from the data distribution p_D or was generated by g. The generator and the discriminator are *adversarial* because they train simultaneously, with each being rewarded for outperforming the other. That is, while the outputs of both are initially random, the discriminator over time learns to identify features that differ between the trained and generated data, which then allows the generator to improve by adjusting that feature of its generated data to match the training distribution.

GANs' training behaviours are notoriously temperamental, and many modifications to the original algorithm have been proposed [24]. Although Lucic et al. [25] have argued that performance improvements purported to be contributed by such variants may in fact be due to differences in computational budget or hyperparameter tuning, there have been certain algorithmic innovations which have been notably influential. A short survey of these follows; refer to detailed reviews of the field for more detail [26, 27].

Radford et al. [28] introduce Deep Convolutional GANs, showing that convolutional layers remain well-suited to processing image data when adapted from the discriminative to the generative context.

The Wasserstein GAN variant [29] aims to provide a more reliable gradient by designing the discriminator (renamed 'critic') to approximate the Wasserstein distance between the distribution generated by g_θ and the data distribution p_D. An additional 'gradient penalty' loss term L_{gp} can be added to implement the constraint that the function is 1-Lipschitz continuous [30]. The loss functions for this Wasserstein GAN with gradient penalty (WGAN-GP) are: $L_g = \mathbb{E}_{z\sim p_z}[-d(g(z))]$ and $L_d = -L_g + \mathbb{E}_{x\sim p_D}[-d(x)] + \lambda L_{gp}$; where the gradient penalty $L_{gp} = \mathbb{E}_{\tilde{x}\sim p_I}[(\|\nabla_{\tilde{x}} d_\phi(\tilde{x})\|_2 - 1)^2]$, where p_I denotes the distribution sampling uniformly from the linear interpolations between generated samples and examples from p_D.

Much work has focused on the ability of GANs to scale up to large, high-resolution datasets. Karras et al. [31] achieved surpassed the state of the art by

dynamically adding layers corresponding to progressively higher resolutions during training. Self-Attention GANs [32] apply the popular attention mechanism [33] in the generative context, along with innovations in normalisation, to achieve yet higher-quality synthetic images. The current state of the art, BigGAN [34], does not contribute a significant change to architecture or training procedure, but instead dramatically scales up existing techniques; that this is so effective suggests that Lucic et al.'s [25] claim that computational resources are the most important determinant of generated image quality holds.

2.1.2 Conditional GANs

A conditional GAN [35] is a variant that learns to generate samples from a conditional distribution by simply passing the intended label y for the generated image to both the generator and the discriminator, and training the generator to maximise the log-likelihood of the correct label in addition to optimising its usual objective. That is, a *labelled* dataset $D \subseteq \mathbb{X} \times \mathbb{Y}$ must be used during training, the discriminator $d\colon \mathbb{X} \times \mathbb{Y} \to \mathbb{R}$ learns to discriminate between labelled dataset and generated examples, and the generator $g\colon \mathbb{Z} \times \mathbb{Y} \to \mathbb{X}$ learns to generate images with the specified label $y \in \mathbb{Y}$.

An extension of this approach is the auxiliary classifier generative adversarial network (ACGAN) [36], in which the discriminator is modified to also predict the label y for the input data. The loss is adjusted to maximise the log-likelihood of the correct label in addition to the standard objective.

2.1.3 Beyond GANs

Generative adversarial networks have one important property which makes them especially suitable for test generation for image classifiers: they are able to learn to generate crisp high-quality examples as though sampled from a relatively complex training distribution [34]. However, Goodfellow [24] sets out a taxonomy of generative models, elucidated below, clarifying that other kinds of approaches with different strengths exist. Most notably, GANs' learned representation of

the training distribution is *implicit*: although samples can be drawn from it, no probability density value can be given, unlike the more common class of explicit-density models.

Fully-visible belief networks [37] are an explicit-density model, and make use of the chain rule to decompose a multidimensional probability distribution (which can be considered to be a joint distribution over the dimensions) into a product of single-dimensional distributions, each conditional on those that have gone before. While this is effective enough to accurately generate realistic human speech [38] and CIFAR10 images [39], the unfortunate effect of the decomposition into conditional distributions is that the computation must be performed sequentially; GANs are able to parallelise such computations, preventing too strong an efficiency dependence on the data dimensionality [24].

Other explicit-density models include: flow models such as nonlinear independent components analysis [40], or neural approximations to it [41, 42], which require the generator to be invertible and so unfortunately require the dimensionality of the latent input to match the dimensionality of the generated data; models such as Boltzmann machines [43] which rely on Markov chain Monte Carlo methods which scale poorly and so are now considered obsolete; and variational autoencoders (VAEs).

An autoencoder consists of two neural networks: an encoder and a decoder. The encoder learns to map training data to a low-dimensional latent encoding; the decoder learns to reconstruct the original data from its latent encoding with as high a fidelity as possible. A *variational* autoencoder [44] imposes the additional constraint that the latent encodings must conform to a standard probability distribution such as a Gaussian. This is enforced by introducing an extra loss term: the Kullback-Leibler divergence between the encoding distribution and the standard Gaussian. Unlike GANs, VAEs can give an approximation of the probability of a generated datum under the model. GANs, however, do not rely on any approximations so reaching a global optimum during training implies that the generator has accurately learnt the training distribution. There is also

consensus that the data generated by GANs are more *realistic*; they appear to be able to generate more plausible samples from the training distribution [24]. Note that any non-GAN network would be perfectly suitable replacements for a GAN-trained generator, if its performance were satisfactory.

2.2 Semantic representations in DNNs

One key motivation for deep learning is the automatic learning of suitable feature representations. Typically, the raw, original data is in a format such as the individual pixel values of an image where each individual feature value (e.g. pixel) contains little relevant information in itself – it is higher-level patterns and relationships between these features that directly encode the relevant information. So for many approaches to machine learning, success depends on whether the representations of the data encode the relevant meanings directly enough. This means that before applying the learning algorithm (for example, a support vector machine), a dimensionality-reducing feature extraction algorithm is used to process the raw data into a form that more directly encodes the information useful for learning. Kumar and Bhatia go as far as suggesting that this stage is "the single most important factor in achieving high recognition performance" [45]. Before deep learning, feature extraction algorithms such as the popular scale-invariant feature transform (SIFT) [46] were written by hand.

But the promise of deep learning is that each layer in the neural network learns its own simple feature extraction algorithm, improving the representation of the features, until the input data are encoded in a form so relevant to the task at hand that it can be performed trivially by the final layer. This ability to simply feed in raw features and have the neural network automatically extract relevant features is part of the reason for the success of deep learning.

This raises an interesting question: what features do neural networks learn? Do different neurons at different stages in a network encode different features that can be understood by humans? Some work, summarised below, has attempted to answer this. The motivation is often (indirectly) trustworthiness –

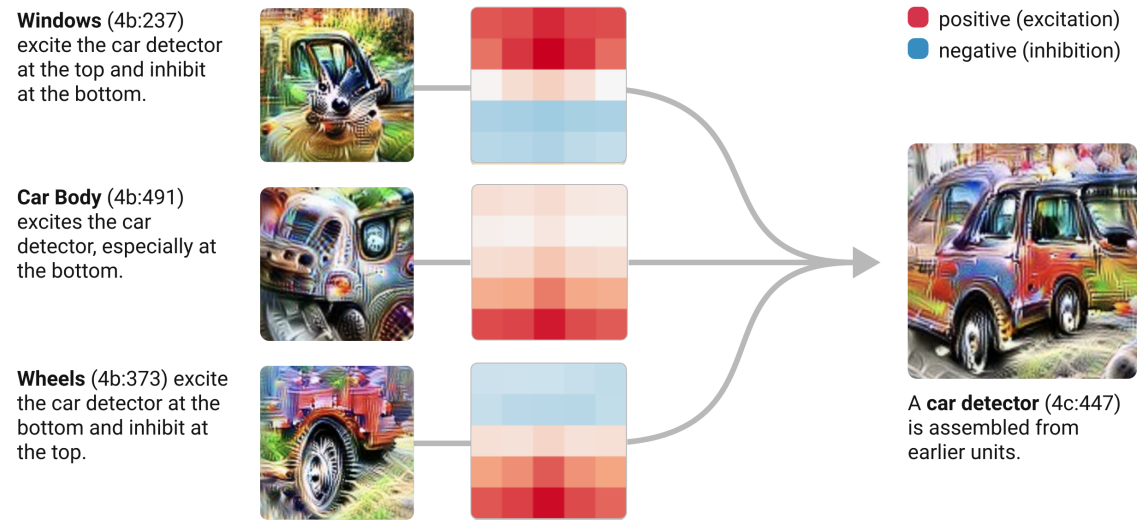


Figure 2.1: Figure reproduced from Olah et al. [47] illustrating the composition of lower-level features into a higher-level neuron.

if we can understand how a model is arriving at its conclusion, then we can understand how much to trust in that conclusion. Most relevant to the rest of this dissertation is the nature of the representations learned by different parts of neural networks, rather than our ability to interpret their overall decision-making, so this will be our focus.

2.2.1 Discriminative DNNs

Consider a discriminative neural network. It takes low-level raw features as input, uses a number of layers to process these data, and outputs a low-dimensional output. How might we expect the representation to change as we progress further through the network? The features at the start are fine-grained, at a low level of abstraction (e.g. a pixel value). The features that are most relevant for the learning task are likely to be coarse-grained and at a high level of abstraction – for instance, discriminating between images of cats and dogs might depend on high-level features like ear shape, tail type, fur type and pose. So we might expect that each layer in the network outputs a representation that is at a slightly higher level of abstraction than the layer before.

Cammarata et al. [48] investigate in a series of articles whether this assumption is accurate in the context of the InceptionV1 image classification model [49], which

was the state of the art for classification of the ImageNet dataset [50] in 2014, and which contains 10,000 individual neurons. Feature visualisation [51] is an important tool used in this work: optimising the input to the network so as to maximally stimulate the given neuron, with a few tricks to ensure meaningful convergence. Among things, they find that:

- Individual neurons *do* encode human-interpretable meanings, such as boundaries between high and low-frequency textures, dog head detectors, or car wheel detectors [47].

- Furthermore, it is possible to understand how features from earlier layers are composed to create higher-level features in a later layer. For instance, separate neurons detecting car windows, bodies and wheels can be combined to create a car detection neuron [52]. See figure 2.1 for an illustration.

- There are neurons in the first few layers that each detect a curve at a certain orientation, together spanning all orientations – and they are actually detecting curves, rather than a feature merely correlated with curves [53]. A range of evidence is presented in favour, including feature visualisation, correlation with human judgement of curves in dataset images, measurement of activations at different rotations of dataset and synthetic curves, and a somewhat successful attempt to replace the learned neurons with hand-coded curve detectors [54].

Later work by the same team – on the discriminative model CLIP [55] that models the relationship between images and arbitrary text captions (rather than fixed image class labels) – asserts that its neurons not only learn human concepts, but that these neurons generalise these concepts across 'modes' within the images such as photos, drawings, and images of text [56].

Mu and Andreas [57] share the view that individual units can learn human-interpretable features, and these can be composed to represent higher-level concepts in later layers.

Bau et al. [58] leverage pixel-wise semantic annotation (image segmentation) datasets to identify particular neurons associated with particular human-created labels at different levels of granularity (for example texture, material, part, object and scene levels). They find that individual neurons *do* tend to learn such human-interpretable features, to a significantly greater extent than random directions in the feature spaces (random linear combinations of neurons).

There is some evidence that neurons in networks that process natural language also specialise and may learn human-interpretable concepts. Dalvi et al. [59] extract salient neurons using correlational analyses, where the correlations are either with certain chosen properties predicted by a property classifier, or with other models under the assumption that individual neurons in different networks will learn the same important feature. They find that individual neurons learn concepts such as verb tense or the position of a verb in a sentence. This is consistent with Mu and Andreas's [57] finding that natural language inference models do have neurons that learn human-interpretable features, but that higher-performance models have fewer of these.

The overall picture from the literature is that the promise of deep learning is true, in the sense that later layers of neural networks combine the lower-level features learned by earlier layers into features at higher levels of abstraction. There is some evidence that individual neurons can represent concepts that seem to be interpretable by humans – but this evidence is most convincing in the earlier layers of networks. Caution should be used when asserting that the behaviour of a neuron precisely corresponds to some human concept. Polysemantic neurons [47], which seem to encode multiple unrelated concepts, and neurons that are more difficult to interpret should give pause for thought. It is much easier to show that a neuron is associated with a human concept than it is to show that this is precisely the only function of that neuron.

2.2.2 Generative DNNs

Consider a generative neural network that generates examples of images drawn from an approximation of the training distribution. The input to such a network is a random sample from a standard (typically Gaussian) distribution, perhaps with an additional encoded class label if the generator is conditional. The output from such a network is the raw pixel values. So we might expect that such a network behaves in one sense like a reversed discriminative network: earlier layers are more abstract, which are decomposed into fine-grained and lower-level features in subsequent layers.

This is borne out by a small literature. Earlier layers tend to encode higher-level information about objects in the image, whereas later layers deal more with "low-level materials, edges, and colours" [60, p.7]. In addition, by performing linear motion in the random input space of a generator, features such as zoom and object position and rotation can vary in the image generated as its output [61], demonstrating that this earliest-layer representation encodes high-level human concepts. State-of-the-art GANs are particularly able to smoothly and convincingly interpolate between different images by so adjusting the input to the generator [34].

This theme will be returned to and exploited in Chapter 5.

2.3 Robustness to pixel perturbations

The standard process for the evaluation of a neural network classifier (or other model) is to measure its accuracy (or other standard metric) on a *test set*: a dataset of inputs not used in any way during training [62]. Typically this is obtained by dividing the full original dataset into two partitions, one for training and one for evaluation. However, in some contexts, simply computing the mean performance on data drawn from the same distribution as the training data is not sufficient. If we suspect that our model may encounter inputs drawn from a different distribution, we would like to have a way of evaluating its likely

performance. This might be because of inadequacies in the original dataset collation, because in reality distributions in the world shift, or perhaps because it is suspected that an agent with influence over the inputs may be motivated to attempt to degrade the performance of the system. This latter assumption can also be useful as an indication of worst-case performance, even if no adversary is expected to exist during deployment.

2.3.1 'Adversarial' pixel perturbations

Well before the success of deep neural networks brought them to the wide attention of the research community, progress has been made in the field of adversarial machine learning: the study and improvement of the worst-case performance of a machine learning system such as a spam filter or biometric identity recognition system under the restricted influence of a malicious actor (an *adversary*) [63].

However, work in this area was galvanised by the discovery in 2013 by Szegedy et al. [64] that state-of-the-art deep neural networks could easily be fooled by an adversary whose power was limited to performing a small perturbation to the given input. In particular, given a classifier network $f\colon X \to Y$, where $X = \mathbb{R}^d$ for some input size d and Y is a set of discrete labels, and given a particular input $x \in X$, Szegedy et al. proposed an algorithm to find an *adversarial example* $\hat{x}$, such that $\|x - \hat{x}\|_2$ is small yet $f(\hat{x}) = l$ for some target label $l \neq f(x)$. By using a box-constrained variant of the limited-memory BFGS optimisation algorithm [65] to minimise a linear combination of the perturbation magnitude $\|x - \hat{x}\|_2$ and the ordinary classifier loss using the target label l, this approach was able to find *imperceptibly small* adversarial perturbations to images: although each $\hat{x}$ was classified as target label l, it was visually similar enough to original image x that not only should $\hat{x}$ have been classified the same as x, but it was difficult for humans to notice the difference between the two. This was framed as a failure of generalisation: since deep neural networks usually had excellent performance when generalising to unseen examples drawn from the

same distribution, it seemed surprising that they failed to generalise to something which was visually indistinguishable from an example drawn from the training and test distribution. However, this failed to account for the fact that two images which may appear similar to a human visual system may in fact be very easy to distinguish mathematically: consider the field of steganography [66].

Goodfellow, Shlens, and Szegedy [7] made an attempt to explain how imperceptibly small perturbations could make such a big difference to the classifier's output. Noting that popular activation functions such as the ReLU [67] result in quite linear behaviour, that an image vector has fairly high dimensionality, and that the maximal change in the output of a linear classifier that can be induced by a fixed-magnitude perturbation to each of its inputs is proportional to the dimensionality of the input, Goodfellow, Shlens, and Szegedy suggest that the linear behaviour of neural network classifiers may be the cause of the adversarial example phenomenon. They corroborate this story by demonstrating that their "fast gradient sign method" (FGSM), in which each pixel is adjusted by a fixed magnitude of ϵ upwards or downwards depending on the sign of the relevant derivative, was able to easily fool state-of-the-art classifiers. But this only tells part of the story. Even if locally-linear behaviour explains the vulnerability of neural networks to small l_∞ perturbations if the input has many dimensions, it does not explain *why* the trained networks exhibit this behaviour (to the extent that they do), given their ability to approximate any continuous function in principle [68]. In addition, Goodfellow, Shlens, and Szegedy's explanation does not say anything about neural networks which do not display this linear behaviour; perhaps there are other causes of adversarial vulnerability neglected by this story.

It is worth noting that there have been many proposed adversarial-perturbation algorithms [69]. Popular attacks include: the 'basic iterative method' [70], in which constrained gradient steps are repeatedly taken; the Carlini & Wagner attack [71], which minimises a linear combination of the perturbation magnitude (under the chosen l_p norm) and some function that is positive if and only if the perturbed example is classified correctly. Of the various candidates for the

latter function, the standard choice is the maximum of $-\kappa$ for some positive confidence hyperparameter κ and the difference between the logit (penultimate layer activation) corresponding to the target class and the highest other logit; and the DeepFool attack [72] is designed to spend more computational resources in order to find particularly low-magnitude perturbations that still fool the target classifier, relative to other attacks.

Leveraging the surprising property that adversarial examples crafted to fool one classifier will somewhat often be misclassified by another trained on the same dataset [64], it is possible to attack a network even with only black-box access by crafting examples to fool such a proxy model. This approach can be generalised to situations without access to the training dataset: Papernot et al. [73] showed that a proxy model can be trained with labelled data obtained by treating the target model as an oracle. Development of this work demonstrated that this approach is not limited to attacking deep neural networks but many kinds of machine learning models, and improved its efficiency, successfully attacking commercial systems with only 800 queries [74]. Moosavi-Dezfooli et al. [75] take transferability one step further by crafting adversarial perturbations which not only generalise across classifiers but also generalise across classes: when added to an image of any class, these 'universal' perturbations are likely to induce a classification, in contrast to traditional perturbations, which are specific to the image being targeted.

Although the literature focuses primarily on image classification as its application domain, this is in general due to convenience rather than necessity. The principles and approaches can be applied to other domains such as speech recognition [76] and natural language processing [77, 78]. It is also possible to consider adversarial attacks on reinforcement learning, where a variety of threat models can be considered [79–81].

Since Goodfellow, Shlens, and Szegedy's [7] initial suggestion that the linear behaviour of neural networks is the cause of adversarial vulnerability, a number of other explanations have been put forward. Shamir et al. [82] show that

vulnerability to l_0 perturbations roughly of the size of the number of possible labels is a consequence of networks which exhibit piecewise linearity, such as those with ReLU activations. It is worth noting, however, that this result depends heavily both on piecewise linearity and on the use of the l_0 metric (i.e., the number of inputs changed, regardless of magnitude), which for many domains is particularly unrealistic as a measure of similarity. Shafahi et al. [83] argue that there are fundamental limits to the adversarial robustness that can be achieved which depend on the dataset, metric, and perturbation magnitude bound used. Jetley, Lord, and Torr [84] frame adversarial vulnerability as inevitable given that neural networks process visual data differently to humans, resulting in the existence of directions in pixel space which affect a network's output while being insignificant to human perception. Ilyas et al. [11] present a compelling case which builds on this, arguing that adversarial vulnerability is a consequence of neural networks' reliance on 'non-robust' features (which correlate with the training labels yet can be altered by adversarial perturbations) which humans cannot perceive. This story is corroborated by two experiments: they attempt to modify a dataset to remove its non-robust features, and show that standard training on this dataset results in a somewhat robust classifier; and they create a dataset for which only non-robust features match the labels, not the robust features, and show that training on this dataset results in a somewhat accurate classifier. Schmidt et al. [85] give an information-theoretic result that training an adversarially-robust model requires strictly more data: robust learning needs $O(\sqrt{d})$ samples, where d is the input dimensionality, rather than $O(1)$ (or arguably just one sample). This tallies with Ilyas et al.'s [11] claim: more data is required since only the robust features can be relied upon in the robust setting.

2.3.2 Improving robustness to pixel perturbations

The existence of adversarial examples raises a natural question: can we develop neural networks that do not perform poorly on such inputs? Some techniques,

introduced below, will prove relevant at various points in this thesis.

Szegedy et al. [64] not only reported the first adversarial perturbation algorithm, but suggested the first instance of what has become known as *adversarial training*. Their preliminary experiments found that by including adversarially-perturbed data in a classifier's training procedure, its test error on the ordinary test set could be improved; adversarial training regularised the network. Goodfellow, Shlens, and Szegedy [7] introduced the fast gradient sign method (FGSM), making adversarial training more feasible. Minimising a linear combination of the loss function on test data and adversarially-perturbed test data, they were able to train a classifier which reduced the success rate of FGSM attacks from 89% to 18%. They also introduced a valuable perspective: adversarial training is equivalent to minimising the worst-case error in the presence of the adversarial attack being trained upon.

Progress was subsequently made in developing and scaling adversarial training to CIFAR10 [86] and ImageNet [87], but only in 2017 did Madry et al. [88] successfully use adversarial training to achieve something that might reasonably be called a defence against adversarial perturbation attacks: rather than only mitigating attacks from the specific attack used during training, the defended network appears to be somewhat robust against all similarly-constrained adversarial perturbation attacks making use of first-order gradient information only. This is suggested to be related to Danskin's theorem [89]: the worst-case error against a class of perturbations of a particular point can be reduced by reducing the error at the perturbed point in that class with the highest error. Although the iterated FGSM attack with random starting perturbations (also known simply as projected gradient descent (PGD)) is not guaranteed to find the *worst* perturbation of a given point, Madry et al. argue that it is able to find a "bad enough" point that the effect is similar. This intuition together with impressive empirical results (as of October 2019, no l_∞-perturbation attack with $\epsilon = 0.3$ has reduced the accuracy of their MNIST network below 88.3%) positions adversarial training as one of the most promising approaches to defence against perturbation attacks.

As attention has turned to moving beyond robustness against a single l_p-norm perturbation attack, a natural first step is to attempt to use adversarial training to achieve robustness against multiple l_p-norm perturbation attacks simultaneously. Although there may be some trade-offs to be made between robustness against different such attacks [90], Maini, Wong, and Kolter [91] have had recent success in simultaneous robustness against l_∞-, l_2- and l_1-perturbations by considering the worst-case direction in the union of these threats at *every* iteration of projected gradient descent.

Since precisely evaluating robustness to l_p-constrained perturbations is NP-hard [92], evaluating the efficacy of a defence technique is difficult. For instance, Papernot et al. [93] proposed leveraging the distillation of neural networks (i.e., training a model to imitate the output levels of another) to smooth the models, making gradient-based attacks very difficult. While this did indeed reduce the model's gradients (by a factor of 10^{30}), and prevent a certain class of existing attacks, Carlini and Wagner [94] showed that defensive distillation does not result in a network that is any more robust than an undefended network. This trend of insufficient evaluation of defence methods being broken by a more rigorous analysis has continued, with many papers published in top conferences being shown to be useless [95, 96]. Best practice is to ensure that any defence evaluation considers adaptive attacks which take into account the particulars of the defence method being used [97].

The collection of broken defences has led to increased interest in symbolic defence methods which provide guarantees about the robustness of the resulting model [98]. Mirman, Gehr, and Vechev [99] go about this by computing (an overapproximation of) the set of activation vectors at each layer which could be induced by an l_p-norm ball around a particular input. For verification, the set of possible output vectors can then be checked to ensure that all points in the input ball must be given the same classification as the original input point. If not, the extent to which the output vectors may be misclassified can be summarised as a loss value, which can then be backpropagated through the network to allow

for the training of a provably-robust model. Approaches which are similar at a high level are introduced by Wong and Kolter [100] and Croce, Andriushchenko, and Hein [101]. Wang et al. [102] improve the efficiency of these symbolic robust training techniques by thoughtfully using fewer training points at once, and by adaptively balancing the loss terms corresponding to accuracy in the presence of an adversary and accuracy on unperturbed data. Croce and Hein [103] are able to use symbolic techniques to train a network provably robust against multiple perturbation types simultaneously.

3 Related Work

Contents

This chapter places our research contributions in the context of the existing literature. In particular, it attempts to comprehensively identify relevant existing work and articulates how the research presented in this thesis differs from previous approaches.

3.1 Method of literature review

To maximise the likelihood of identifying relevant articles, I took a systematic approach to searching the literature in addition to the usual organic process of finding related work.

By creating a long list of search terms that might identify relevant work, and searching the literature with each using Google Scholar, it is less likely that any

related papers will be missed in this literature review. The goal in creating the list of search terms was to include all terms I could reasonably identify that had some chance of surfacing relevant papers. To create the list, I simply listed all search terms I thought might be relevant, then skimmed all my papers and added any additional search terms that became apparent, skimmed Geirhos et al. [6] and added additional relevant terms, and lastly added search terms that arose when reading related work. I repeated this process twice, resulting in lists of lengths 61 and 43 (with this second list including more disjunctive searches). These were then merged into one final list of search terms.

For each search term entered into Google Scholar, reviewed the title and abstract of each result until reaching two pages of ten results that were all not relevant to be included. All relevant papers are included in this chapter, the only exception being when certain sections of the literature (such as adversarial pixel perturbations) are dealt with as a whole, using specific representative references rather than exhaustively enumerating papers in that field. The articles citing and cited by each paper identified by this systematic search were also subject to the usual organic exploration described below.

As well as identifying relevant papers through the above systematic search, others were identified in the usual, more organic way. There are several mechanisms that complement one another: alerts from services providing links to possibly relevant new papers, browsing the articles cited by relevant papers, browsing the articles that cite relevant papers, conference proceedings, recommendations from supervisors, keeping up with the outputs of researchers with relevant interests, and reading survey papers.

3.1.1 Search terms

Below follows the complete list of search terms used as described above.

Note that in Google Scholar, OR works only with quoted phrases or individual words – it ignores parentheses. So it is impossible to search for `DNN (out of distribution OR distribution shift)`, because it is equivalent to `DNN out of`

`(distribution OR distribution) shift`. This is not that important for the reader to understand, except to explain some of the strange-looking queries below, and to acknowledge that the parentheses in the listings below are for readability only.

Note also that for readability in the following listing, the variable **DNN** represents the following string: `DNN OR "neural network" OR "neural networks" OR AI OR ML OR "machine learning" OR ImageNet OR MNIST OR "image classifier" OR "image classification"`, the variable **test** is short for: `test OR testing OR evaluate`, and the variable **generalisation** is short for: `generalisation OR generalization`,

---

DNN **test**

DNN `(shortcut OR proxy) learning`

DNN `(shortcut OR proxy)` **test**

DNN **generalisation** (**test** `OR ability`)

DNN `out of distribution`

DNN `distribution shift`

DNN `out of distribution` **generalisation**

DNN `distribution shift` **generalisation**

DNN `out of distribution` **test**

DNN `distribution shift` **test**

DNN `domain transfer` **test**

DNN `perturbation` **test**

DNN `adversarial` **test**

`(generative OR GAN)` **DNN** **test**

DNN `data augmentation`

`(generative OR GAN OR generate) data augmentation` **DNN**

DNN `data augmentation` **test**

DNN `generative data augmentation` **test**

DNN `data augmentation out of distribution` **generalisation**

DNN data augmentation distribution shift

adversarial training out of distribution **generalisation**

adversarial training **test**

DNN robustness

DNN robustness **test**

DNN robustness benchmark

DNN robustness out of distribution **generalisation**

DNN robustness **generalisation**

DNN robustness unhelpful

DNN robustness transfer

DNN robustness distribution shift

DNN robustness data augmentation

DNN robustness (generative OR GAN)

domain **generalisation**

domain **generalisation** evaluation OR testing

DNN (semantic OR meaningful OR unrestricted) robustness

(semantic OR meaningful OR unrestricted) adversarial example **DNN**

(semantic OR meaningful OR unrestricted) perturbation **DNN**

(semantic OR meaningful OR unrestricted) feature change **DNN**

(semantic OR meaningful OR unrestricted) **test DNN**

(semantic OR meaningful OR unrestricted) counterfactual explanation

DNN

context sensitive perturbation **DNN**

perturb activations (**DNN** OR (generative OR GAN))

perturb latent representations (**DNN** OR (generative OR GAN))

(two OR dual OR multiple) objective training (generative OR GAN)

(generative OR GAN) (multiple OR two) loss

train (generative OR GAN) **test DNN**

(**DNN** OR generative OR GAN) test oracle problem

DNN (failure OR fault) (**test** OR detect)

DNN `fuzzing`

DNN **test** `beyond pixel perturbations`

DNN `beyond adversarial perturbations`

DNN `security`

DNN `safety`

DNN `bias` **test**

DNN `clever hans` **test**

DNN `iid failure` **test**

DNN `overfitting` **test**

`reinforcement learning reality gap` **test**

DNN `anthropomorphism`

DNN `feature learning` **test**

DNN `(interpretability OR explainability) generative`

DNN `shortcut` **test**

DNN `shortcut learning`

3.2 Constrained pixel perturbations

3.2.1 Adversarial pixel perturbations

A vast amount of work has been focused on so-called 'adversarial examples': inputs deliberately made to fool a classifier. This security motivation – that there may be an attacker who actively works to craft inputs for which the model fails – seems to have captured the imagination of the research community, in contrast to the more mundane safety motivation, in which there is no adversary deliberately trying to cause failure.

By far most popular method for crafting these adversarial examples, dubbed 'pixel perturbations', takes after seminal papers by Szegedy et al. [64] and Goodfellow, Shlens, and Szegedy [7], and involves fooling the classifier by individually changing the pixel values of an input image; both attacking with and defending against these perturbations by improving models has been extensively

explored [104–106]. Carlini has (partly automatically) identified over five *thousand* relevant arXiv papers written since 2014, with 90% of these since 2018 and a large majority since 2020 [107].

Typically, pixel perturbations allow for arbitrary changes, independent of the content of the image, as long as they are almost unnoticeable to the human eye. In practice, this is done by limiting the magnitude of the perturbation in pixel space as measured by an ℓ_p norm (where typically $p \in \{0, 1, 2, \infty\}$), thereby bypassing the oracle problem by assuming that the true class of the perturbed image is unchanged from the original image.

Unfortunately, as Gilmer et al. [108] lucidly clarify, the relationship between system security and imperceptible pixel perturbations is not as straightforward as is often assumed in the literature. In security, a threat model of the attacker is essential, and Gilmer et al. [108] point out that the relevance of ℓ_p-constrained adversarial perturbations to the secure deployment of models is limited. Gilmer et al. [108] enumerate five suggested applications of the imperceptible-perturbation threat model. For each, they point out that even if the threat model does apply, the attacker is likely either to have an easier way of achieving their goal (for instance, by performing a physical attack simply presenting some unperturbed input which the system misclassifies, since its generalisation is not perfect) or to achieve nothing of importance even if the machine learning model is fooled.

Although this does not preclude the existence of a real-world security scenario for which imperceptible perturbations are precisely the main concern, it does lend weight to Gilmer et al.'s main conclusion: if adversarial example research is to be motivated by security concerns, then a plausible and precise threat model should be stated and its relevance carefully justified. Conversely, if adversarial robustness is not motivated by security, then its motivation and relevance should be clearly articulated. It does seem likely that improving our understanding of worst-case behaviour in the presence of a theoretical adversary could improve our understanding of models' generalisation in general; recent work has also suggested that adversarially-robust models obtain better high-level

feature representations of the training data [109] and can be leveraged to perform a variety of image-manipulation tasks [110].

The other motivation for ℓ_p-norm constraints is viewed as a technique for confidently knowing the correct label for a perturbed example. but the poor correspondence between human perception and ℓ_p norms [111] poses a severe limitation. Being within the ℓ_p-norm ball is far from being necessary and, depending on the radius of the ball, may not be sufficient either.

Fundamental limitation of constrained pixel perturbations

So what does the research in this thesis contribute beyond the very well-studied ℓ_p-constrained pixel perturbation methods? The key insight is that use of the ℓ_p pixel space metric force any test generation procedure to constrain its outputs to a vanishingly small subset of the relevant possibilities.

Consider the two kinds of perturbation included in Figure 3.1. Almost all possible pixel perturbations are meaningless, as in column (c). A randomly selected perturbation is likely to have each pixel changing arbitrarily in a way that is independent of its neighbours and from the meaningful features in the image; it is the addition of random noise. A vanishingly small proportion of possible perturbations affect meaningful, higher-level features as in column (b). Of these meaningful perturbations, some will alter the oracle-assigned meaning of the images, Figure 3.1 (b), but others such as adjustments to the position, orientation, pose, colour, texture, or other non-essential characteristics of the object (or any changes to the background) will not alter the correct class label.

If we are to impose an ℓ_p constraint on a perturbation to ensure that the perturbed image retains the same class as the original, then we must ensure to exclude all meaningful perturbations that could change (or remove) its class. But as Figure 3.1 demonstrates, this implies that the ℓ_p magnitude threshold must be set low enough that a very large number of noise-type perturbations are also excluded. This fundamental inability to discriminate between perturbations that do and do not affect the meaning of an image is a severe limitation on the

usefulness of the ℓ_p norm constraints as a solution to the test oracle problem. Furthermore, their inability to distinguish changes to high-level features that do and do not affect the oracle-assigned meaning means that the latter kind must be excluded to avoid the former.

So tests that are constrained by an ℓ_p norm in pixel space can only output a vanishingly small proportion of the possible meaningful-preserving perturbations to a given test input.

Comparison to the present work

The problem of developing models that perform well even in the presence of worst-case constrained pixel-space perturbations is a valuable one that deserves attention. If we cannot solve this simple case, with its limited and well-specified scope, we do not have much hope for developing models that reliably generalise. That said, solving pixel-space robustness is far from sufficient, and the limitations of this framework may have been neglected. The most important of those is addressed by this research: ℓ_p constrained perturbations comprise a vanishingly small proportion of the test cases we may care about. The two new test generation algorithms presented in this thesis are able to output a much larger volume of possible test inputs, and so detect a wider range of failures and faults.

The first boost to the number of reachable test cases comes directly from the use of a generative model. Whereas traditional perturbation-based approaches can only perturb the fixed number of reserved test seeds, use of a generative model immediately expands the range of possible test seeds: rather than using a held-back test seed, the generator can now generate a fresh one. Assuming it has learned the training distribution well, it is essentially "filling in the gaps", giving access to a large contiguous space of possible test seeds, rather than tiny isolated pockets in input space.

But both algorithms have their own reasons to expect a much larger increase in the number of reachable test cases. The first algorithm, introduced in Chapter 4 and involving the training of a generator network to directly generate suitable

test cases, is not perturbation based. That is, rather than making small changes to a test seed, it is free to output any test case identified as suitable through the learning process, which does not impose any explicit constraints but rather searches for test cases with certain properties, as encoded in the loss function. Therefore, it typically outputs test cases that are far beyond the constraints required under a pixel perturbation approach. See Section 4.4 for relevant empirical evidence.

The second algorithm, introduced in Chapter 5, perturbs the latent activations of a generative network so as to make *context-sensitive* changes to a test seed. In this way, the fundamental limitation of pixel-space ℓ_p constraints is escaped. Chapter 6 presents much evidence that this algorithm can identify not only failures, but indeed faults that pixel perturbation approaches cannot.

Another difference is that adversarial pixel-space perturbations are usually described as being motivated by security. Gilmer et al. [108] have cast serious doubt on the plausibility of the existence of a realistic relevant threat model. Our work is motivated by safety, not security; we are not modelling the threat from a potential adversary, but trying to understand the limitations that may cause our models to perform poorly during deployment simply because the distribution of the data has changed. We can reframe the vast adversarial pixel perturbation literature as a test generation literature: any so-called adversarial example can be viewed as a test case that reveals that the model has failed to appropriately generalise. It is in this light that the above comparisons are made.

Discussion of (possibly 'adversarial') perturbations that are not constrained by an ℓ_p norm is below, in sections 3.3 and 3.4.

Generated constrained pixel perturbations

One idea in the literature is to train a generative model to output adversarial pixel perturbations.

Hayes and Danezis [113], Baluja and Fischer [114], Xiao et al. [115], and Poursaeed et al. [116] are all typical examples of this kind of work. While their

Figure 3.1: Illustration of the limitations of ℓ_p constraints in measuring similarity. Each row contains (a) an unperturbed image from the ImageNet validation set, (b) a manual perturbation of ℓ_2 magnitude 19 and 22 respectively that completely changes the class of the image (from 'goldfinch' to 'pineapple' and 'hermit crab' to 'strawberry'), and (c) a random perturbation of the same magnitude as in (b). Note that (b) is identical in meaning to (a). This implies that any ℓ_2 norm threshold that excludes perturbations like (b) must also exclude a vast number of perturbations as in (c) that do not change the meaning of the image. Figure reproduced from Tramèr et al. [112] with permission.

details differ, they share the same essential idea. In short, rather than identifying an 'adversarial' perturbation using a dynamic optimisation procedure, they instead train a generative network to output an adversarial perturbation for whichever image is given as its input. In this way, there is computation required upfront for the training of the generator, but each additional perturbation requires only a single forward pass – no backpropagation.

Although these approaches are demonstrably successful in generating ℓ_p-constrained pixel perturbations, they nevertheless suffer from the same problems as pixel perturbations identified in the usual way: the ℓ_p constraints are much too restrictive. So while these techniques do exploit generative learning, they are fundamentally different from the algorithms presented in this thesis, which escape the limitations of constrained pixel perturbations as described above. Our new approaches therefore are able to identify failures where pixel perturbations would not find any problems.

3.2.2 Techniques explicitly aiming to test DNNs

Another school of thought regarding the evaluation of neural networks does not consider worst-case performance in the presence of an adversary, but rather draws inspiration from the world of software testing.

One concept adapted for use with DNNs is test coverage [117, 118]. Test coverage criteria are measures of a test suite that are intended to correlate with how well the suite explores the possible behaviours of the system under test, and therefore with the likelihood of discovering any bugs present. Popular metrics include branch coverage [119] and modified condition/decision coverage (MC/DC) [120]. Pei et al. [121] introduced the concept of neuron coverage by analogy to statement coverage: rather than ensuring that every line of code in a program is executed by at least one test case, for deep neural networks, the goal is that every neuron (i.e., dimension in a hidden layer) is activated (i.e., has positive value) by at least one test input. Although this criterion is almost trivially easy to achieve in practice, and is therefore too weak, it inspired a range of other coverage criteria: neuron boundary coverage [122] extends neuron coverage by considering a set of activation bounds to be covered, and Sun, Huang, and Kroening [123] introduce a neural-network analogy of MC/DC coverage criteria, for instance.

Symbolic execution is another concept that has been usefully applied. In short, in contrast to the concrete execution of a program with one input leading to one output, symbolic execution represents values as symbols, allowing the conditions for (say) a particular branch to be taken to be represented symbolically. Once computed, these can be used to check for violations of specified properties. Introduced in the 1970s [124], there is now a flourishing set of mature tools implementing symbolic execution: KLEE [125], built on the LLVM compiler, is perhaps the most prominent, but there are engines for most targets, including S^2E [126] for binary files and angr [127] for Python. Many of these tools also allow "concolic" execution, which is the use of symbolic execution techniques to identify new concrete inputs that can (for instance) maximise code coverage. Gopinath et al. [128] were among the first to apply these ideas to neural networks,

introducing techniques that make the application of symbolic execution feasible – a naive attempt would quickly run into the problem of the large scale of deep neural networks, in addition to their non-linearity. Sun et al. [129] also introduce a concolic method to construct test suites maximising a given coverage criterion. Li et al. [130], however, cast doubt as to the utility of test sets designed to maximise such structural coverage criteria; perhaps more effort ought to be devoted to clarifying the properties we desire from our models before devoting resources to evaluating them.

Fuzzing, introduced around 1990 by Miller, Fredriksen, and So [131], is the simple idea that software can be stress-tested using randomly generated or mutated test inputs. TensorFuzz [132] applies this approach to DNNs. Just as fuzzing has been surprisingly successful in testing software by randomly mutating given inputs, this work randomly mutates inputs from a given test set, keeping those which improve coverage, defined in this case to be of sufficient distance in activation space from the activations induced by any existing test input. The mutations of the test seeds are constrained using an ℓ_∞ distance.

Comparison to present work

Although this pocket of the literature suggests interesting new properties that are desirable for a set of tests to have, the algorithms to create these sets of tests always solve the test oracle problem using ℓ_p-constrained pixel perturbations. Therefore, the test algorithms introduced in this thesis are able to identify more failures than any of these approaches, for the reasons described in the above discussion of adversarial pixel perturbations. There is scope for our new test algorithms to be combined with (for instance) an aim to maximise coverage criteria proposed in this literature, but this is a new project left for future work.

3.3 Perturbations using generative models

While an ℓ_p pixel-space constraint makes perturbations simple to understand and implement, we have seen that is also very limiting, allowing for testing

on only a tiny fraction of interesting cases, due to its inability to distinguish between the very different possible perturbations at the same ℓ_p magnitude. The second new test generation algorithm introduced in this thesis perturbs the latent activation values in a generative network as a way to address this problem. No existing work has taken this approach, but there *are* prior works that in other ways try to exploit generative machine learning to perform perturbations in a way that overcomes the limitations of ℓ_p-constrained pixel perturbations. These are discussed in this section.

3.3.1 Perturbing hand-selected disentangled attributes

Some techniques attempt to allow fine-grained control over the changes made by sacrificing flexibility. These methods select the features they will modify (for instance, whether the output face is smiling or not), and then use a generative model to specifically perturb those features. Typically, these exploit generative models with disentangled latent spaces – that is, generative networks that explicitly represent chosen features as distinct, controllable inputs, like a conditional GAN with multiple inputs. Given a disentangled generative model $g : \mathbb{A} \times \ldots \to \mathbb{X}$ that takes an attribute value $a \in \mathbb{A}$ as one of its inputs, that attribute value can be optimised so as to (for instance) make the generated output $g(a, \ldots) \in \mathbb{X}$ an adversarial input for a classification model.

Gowal et al. [133] use a *StyleGAN* and partition the latent space according to whether or not it should influence the label. The (disentangled) representations of different inputs are adversarially composed. Selecting the features to perturb like this allows for precise control over these features, but like hand-crafted perturbations, results in narrow kinds of changes to images. *DeepRoad* [134] uses *UNIT* [135], a different image-to-image translation technique, to produce images of the same road in sunny, rainy and snowy conditions.

Joshi et al. [136] use *Fader networks* that are able to control attributes such as spectacles, smile, eye shape and hair colour, manipulating these features to create 'adversarial' inputs for face classifiers. Sharif et al. [137] focus primarily

on eyeglasses, training a network to generate patterned spectacles, which, when added to an image of a face, cause misclassification. This approach is adapted to MNIST in an approach similar to Chapter 4. However, this only achieves a success rate of 0.83% after filtering to "only the digits that were likely to be comprehensible by humans" in contrast with Chapter 4's 80%. This difference may be because the GAN training is difficult, requiring the use of the techniques such as those presented in Section 4.1.3.

Bhattad et al. [138] leverage pre-trained colourisation and texture-transfer models to adversarially change the colours and textures of an image, provoking poor performance in image classification models and image captioning models; the changes made are constrained only to colourising and texture transfer.

Rather than interpolating on the labelled attributes a directly, Qiu et al. [139] use a disentangled generator to generate an unperturbed image $g(a = 0, \ldots)$, and an image with the relevant attribute changed $g(a = 1, \ldots)$, and interpolates between the latent activations for these two examples at some fixed layer in the generator.

Comparison to present work

Unlike the techniques presented in this thesis, this set of approaches has the benefit of being able to evaluate how a model performs when certain chosen attributes are adjusted. This is valuable to the extent that we are able to predict the kinds of change that are of interest, either because they may be encountered in deployment or because they might yield new insights into our models.

But this is also a limitation – perturbing known attributes does not give any information about behaviour when any other kinds of changes are made. The set of features that could possibly change is very large, especially when we consider that most of these will not be nearly encapsulated in an interpretable concept such as "hair colour". So any such attributes explicitly specified and tested will remain a vanishingly small proportion of the changes of interest. In contrast, the test generation algorithms introduced in this thesis are not restricted to affecting

any particular features. Training a generator to output test cases, as in Chapter 4, generates test inputs that can have any value for any attribute, but with the constraint that the discriminator network must struggle to distinguish generated from dataset data. Perturbing a generator's latent activations, introduced in Chapter 5, allows changes to all features that the generator has learned to be relevant from the training data, rather than just those that are explicitly labelled. This allows for a much wider range of possible perturbation effects, including those that may not be directly interpretable to a human, which is important if the classification model being tested relies on such features.

In addition, the creation of such disentangled generative models depends on the relevant datasets being labelled not only with the primary label, but with secondary labels for each of the attributes to be conditioned upon. Acquiring these labelled datasets is clearly several times more expensive than a standard dataset. The algorithms introduced in this thesis do not require any labelled attributes to be labelled in their training data, because they rely on generative models' ability to *learn* features.

Qiu et al. [139] deserve particular focus, because like our perturbation approach introduced in Chapter 5, theirs operates by adjusting tensors in a latent feature space inside the generator, rather than directly manipulating the attribute at the input to the generator. However, there is a crucial difference. Whereas our technique is free to introduce a latent perturbation of any direction (and magnitude), Qiu et al. are constrained to interpolate only between the feature values of two images that differ only in the specified attribute, such as hair colour. So in practice, this approach is essentially contributing a new way of adjusting the specified attribute, whereas our algorithm allows for any kind of change to any feature learned by the generative model.

3.3.2 Perturbing generators' random seeds

Another category of approaches is not to perturb a specific disentangled feature, but rather to perturb the input to the generator that is sampled from a fixed

probability distribution during training (and usually during testing). Because this input is the sole source of randomness, holding any other inputs constant and varying this input over its possible values should cause the generator's output to range over all of its relevant possible values, too. In short, the idea is that small changes to this input will cause the generator to produce related, but still "in distribution" outputs.

Zhao, Dua, and Singh [140] were likely the first to introduce this idea. They train a generative network $g : \mathbb{Z} \to \mathbb{X}$ that maps from a latent space to an output space, and an inversion network $i : \mathbb{X} \to \mathbb{Z}$ that is trained to invert g using a reconstruction loss. Given a test seed x, they perform a guess-and-check black-box optimisations to identify perturbations p such that $g(i(x) + p)$ is misclassified by the classifier under test. Unfortunately, some of the perturbed examples seem to also change their true class, and the limited evaluation with human participants only asks whether the crude "fast gradient sign method" is worse, rather than checking that perturbed data retain their original semantics.

Song et al. [2] develop the idea, using a white-box gradient-based optimisation to identify perturbations to the random generator input. It is also one of the first papers to focus on *unrestricted* adversarial examples, acknowledging the limitations of the ℓ_p-constrained pixel perturbation approach, resulting in less emphasis on constraining the perturbation to be imperceptibly small. Another innovation is a loss term incentivising the output to be confidently correctly classified by the discriminator's auxiliary classifier (an auxiliary classifier GAN [36] is required to use this procedure). Perhaps because these factors together improve the search for good 'random' inputs to the generator, the so-called adversarial examples identified by this algorithm are more likely to be successful and maintain the correct class, as verified by thorough experiments. This paper also drops the need to perturb a given test example, instead simply randomly sampling a starting seed to be optimised each time.

Like Song et al. [2], Byun et al. [141] also train a standard conditional generative model, and then search in its latent space for examples on which the

model under test is likely to fail. In this case, however, the generative model is a conditional variational autoencoder (VAE), and the goal is explicitly to identify test cases *from the training distribution* (the contribution of the VAE being the ability to interpolate between given examples). Wang et al. [142] take fundamentally the same approach, searching for perturbations to generative models' inputs, but using somewhat more involved model configurations (such as a VAE with various discriminative add-ons) and search algorithms. They also perturb disentangled attributes, discussed above in the previous section. Yang et al. [143] also perturb the latent space of an encoder-decoder generative model; the stated goal is to identify closest counterfactuals, rather than adversarial examples, but is almost identical when all are re-interpreted for our purposes as test generation algorithms. Toledo et al. [144], taking a more formal verification perspective, observe that procedures searching for a test input that violates a desired property specification could benefit from searching only in the input space of a generative model, and set up infrastructure that allows a suite of seven such search algorithms to do so with some success. Wong and Kolter [145] expand on this approach by searching for suitable inputs to a generator that has been trained on already perturbed data.

Comparison to the present work

These approaches that perturb the input seed of a generator network are valuable. By exploiting the distribution learned by the generator, they are able to make perturbations that would escape ℓ_p constraints in pixel space, thereby being able to identify problems that pixel perturbations cannot.

However, there is a key downside to this approach. The better the performance of the generative network used, the better it approximates the distribution of data seen during training. So, on the assumption that the generator performs well, it will output data that are indistinguishable from data drawn from the training distribution. Such data will be output even if the generator input is perturbed. Therefore, these perturbations are at best effectively implementing a

search for test cases in the training distribution. This is still valuable: hold-out test sets drawn from the training distribution are popular for a reason, and such approaches are not constrained to those in a finite test set. But these approaches do not comprise out-of-distribution test generation procedures, and we have seen that evaluating out-of-distribution performance is crucial. Both new algorithms introduced in this work are able to generate out-of-distribution tests.

Training a generator specifically to generate useful tests as introduced in Chapter 4 is clearly able to generate out-of-distribution examples because the generator is no longer incentivised to just imitate the training distribution. Empirical evidence in Section 4.4 confirms this, especially a direct comparison with Song et al. [2], the contemporary state of the art, showing that our procedure is able to generate a wider range of possible tests.

Perturbing the latent activations of generative networks, as introduced in Chapter 5, differs from perturbing only a generator's initial input. As mentioned above, perturbing an input leads to outputs drawn from the training distribution, but perturbing latent activations does not. This can be most clearly seen when perturbations later in the generator are considered, but the same principle applies earlier in the generator: since different parts of the latent spaces control different features, and they can be perturbed separately, they can be adjusted in ways not seen in the training distribution. Comparing the perturbations at different layers provides unambiguous empirical evidence of this. Qualitatively, it is clear from Figure 5.6, for instance, that perturbing at different places allows different feature changes to be effected; quantitatively, the very different classifier behaviours in the presence of perturbations at different layers (see 7.1) implies that these perturbations are having materially different effects. In short, by perturbing a generator's latent activations rather than just the input, more of the generator's representations can be accessed and exploited, to change a wider range of features.

3.3.3 Training generators to output constrained perturbations

There are several papers that train generative networks in ways that seem similar to our algorithm presented in Chapter 4, but on closer inspection have constraints enforced during training that ensure that the generated outputs are equivalent to ℓ_p-constrained pixel-space perturbations. Although using generators in this way may be useful for some purposes, for our purposes, these techniques are no different than other pixel perturbation approaches despite their superficial similarity: they remain constrained to output only a vanishingly small fraction of possible test cases, as described in Section 3.2.

Dola, Dwyer, and Soffa [146] find adversarial examples using pixel perturbations, but with the additional constraint that the reconstruction probability assigned by a VAE must be above a threshold (and so this is included as a term in the perturbation objective). This is a search for explicitly in-distribution inputs, in contrast to our algorithms that seek to probe models' out-of-distribution generalisation. Minderer et al. [147] also use a technique in which a generative network is trained to output images (a) reduce the performance of a classification model and (b) accurately reconstruct the image originally input to the generative network, according to a pixel-wise ℓ_2 loss, albeit for a different purpose (prevention of the learning of some shortcuts). Yang, Song, and Wu [148] train the decoder of an encoder-decoder pair to output "quasi-imperceptible" changes to the original input so that a face recognition model outputs a similar logit distribution (measured using cosine similarity) as for a given target face. As well as a GAN-style discriminator, an ℓ_1 reconstruction loss is used to minimise the difference between perturbed and unperturbed images.

Wang, He, and Hopcroft [149] propose a procedure that is superficially similar to that in Chapter 4, training a GAN to directly generate test cases Instead of using the standard GAN loss to maintain the semantics, they use a new loss term. This term, $\|g_{pretrained}(z) - g(z)\|_p$, penalises the generator g given input z proportional to the deviation caused by finetuning from the original output. But this choice of loss term in effect works by constraining the output to be within an l_p-norm ball of

a realistic input – the generator is essentially only searching for a constrained pixel perturbation. By contrast, our approach allows for truly unrestricted test cases.

For the purpose of data augmentation, a recent paper by Baek et al. [150] trains a generator network to output adversarial masks for an image, which are applied multiplicatively rather than additively. The result is effectively a coarse-grained colour change perturbation that is somewhat sensitive to the structure of the image. Unlike the perturbation algorithm introduced in Chapter 5, this multiplicative perturbation is essentially constrained to only make these broad colour changes; the range of features our algorithm is able to affect is much wider.

3.3.4 Perturbing non-learned generative models

For this section only, the phrase "generative model" will not refer to a machine learning model, but to any software that is able to generate suitable inputs for the classification neural network under test. Rather than *learning* the relevant features to adjust, the idea is that if you can manually write software that generates plausible test inputs, then making adjustments to the generative parameters allows a search for test inputs that identify failures.

One interesting but expensive possibility is writing a differentiable renderer for the desired domain. Because the renderer is differentiable, this allows its parameters to be optimised using standard backpropagation and a gradient walk to identify failures. Liu et al. [151] and Jain et al. [152] both take this approach. Liu et al. adjust lighting and geometrics (i.e., object shape and position) parameters; Jain et al. aim to be more general, introducing a procedure to perturb the parameters of any renderers, demonstrating the approach on a ray tracer and a 3D traffic scene renderer.

If the model cannot provide a gradient allowing its parameters to be the subject of a white-box optimisation algorithm, black-box optimisation (evaluating the effects of random changes) can be used instead. Riccio and Tonella [153] use a non-machine learning model of the inputs of a DNN to make perturbations to known test cases. For instance, representing MNIST digits as Bézier

curves, making random adjustments to the curve parameters, until two examples originating from the same test seed result in different outputs from the system under test. Similarly, Gambi, Mueller, and Fraser [154] procedurally generate test inputs for autonomous cars, and use a genetic algorithm to search for test inputs that meet the desired criteria.

Comparison to present work

Assuming the existence of a suitable model, this approach allows for powerful and interpretable probing of specific situations in which the model under test fails. But the techniques introduced in this thesis have some advantages over the use of a hand-coded generative model.

For most domains, there does not exist a suitable hand-coded generative model, and the creation of such a model is very costly, requiring a great deal of expensive programmer time. In contrast, generative machine learning can automatically learn a suitable model for any new domain given enough data, which is more likely to already exist and cheaper to acquire if not.

Furthermore, differentiable renderers for non-trivial domains often require expensive computations such as ray tracing on each use. There is often a tradeoff between reducing computational cost and increasing the quality of the generated outputs.

Last, hand-coded generative models are necessarily oversimplifications of reality. Any aspect of reality that is not captured in the hand-coded model – textures, variety of objects, backgrounds, subtle distortions, imperfections, etc. – will never be probed. But a well-trained generative model must learn all the relevant features, including those that are subtle and difficult for humans to identify or write down.

3.4 Manually designed perturbations

So far, we have covered ℓ_p-constrained pixel perturbations and perturbations that exploit generative models. In this section, we cover perturbations that are

'unrestricted' in the sense that they are not bound by an ℓ_p constraint in pixel space [19], but are implemented by hand, rather than by exploiting any models. In short, these approaches are valuable because they probe models' abilities to generalise in the presence of specific feature changes of interest, but they are inherently limited by their manual implementation; our use of generative learning in contrast allows a wide range of features of interest to be automatically tested.

Hosseini and Poovendran [155] contributed one of the first procedures for making such an 'unrestricted' change to an image that was independent of the classification task, and so ought not to change the oracle-assigned label. Using the Hue-Saturation-Value representation of image colours (rather than the usual Red-Green-Blue), the values (i.e. brightness levels) of the pixels in each image were held fixed, with the hues and saturations adversarially chosen. This resulted in a high success rate for an untargeted attack against a standard classification network; the targeted attack was less successful. Zhao, Liu, and Larson [156] also perturb image colours, but specifically exploit human biases in perceptual colour distance to make large changes under an ℓ_p pixel norm that nevertheless remain imperceptible.

Engstrom et al. [157] show that performing worst-case small translations and rotations is sufficient to fool image classifiers; this is another example of a slight semantic change to the image which results in a large distance under any l_p norm yet should not affect the correct classification at all. Tian et al. [158] evaluate whether cutting irrelevant areas from images affects classification outcomes. Snoek et al. [159] evaluate the robustness of the calibration of classifiers' *confidences* to rotated and translated images (as well as to out-of-distribution inputs such as not-MNIST [160]).

Several papers combine such transformations with pixel-wise operations such as contrast or brightness adjustments, or Gaussian noise. DeepHunter [161], for instance, uses such a combination of adjustments to perform fuzz-based testing. Gao et al. [162] likewise perform fuzzing with a slightly different set of operations, including zooming and shearing. Hendrycks et al. [163]

introduce a data augmentation technique that 'mixes' (using elementwise convex combinations) several randomly-chosen chains of standard augmentations such as translations and colour adjustments. Mohapatra et al. [164] introduce a framework that parameterises a range of changes, including colour, brightness, rotation and occlusion, using an ℓ_∞ parameter so that ℓ_p-constrained verification techniques can be applied. Tian et al. [165] increase coverage of their autonomous driving tests using a combination of linear, affine and convolutional (e.g., a rain effect) transformations.

Other papers introduce a wider range of hand-selected 'distortions' that can be applied to any image. Hendrycks and Dietterich [166] evaluate the robustness of networks to various types of corruption such as Gaussian noise, motion blurring, artificial fogging, pixelation, and brightness and contrast adjustments. Geirhos et al. [167] compare DNN performance with human performance at image classification in the presence of twelve different such image distortions. Humans turn out better, except that a DNN trained on an image distortion performs very well for that particular distortion (only). Using such distortions, robustness benchmarks have been created for datasets including ImageNet [166] and MNIST [168]. Pei et al. [121] generate test inputs by setting small rectangular regions of the image to black.

Scimeca et al. [169] train classification models using labels that ambiguously could refer to one of three human-interpretable features. By testing these models with inputs where the three features appear in an unseen combination, it can be deduced which of the cues the model has learned to pay attention to. This is useful work that directly obtains evidence about the nature of shortcuts that tend to be learned by classifiers. But the ad hoc approach means that only information about the relative likelihood of the hand-chosen features can be gained; our testing approaches, in contrast, are able to identify a much wider range of shortcuts since they are free to modify any learned feature, not just hand-specified ones.

Another fairly specific instance of ad-hoc, manually chosen perturbations is Nauta et al. [170], who use automated addition and removal of colour calibra-

tion charts on medical scans to evaluate the extent to which a DNN learns to inappropriately rely on these charts when the training dataset naturally contains these charts only for one of the two possible classes.

Comparison to present work

This category of approach is valuable because it allows the examination of model robustness in the presence of precisely known conditions. In particular, it allows comparison between the effects of different perturbation types. But by virtue of being manually chosen and implemented, these techniques are inherently limited. Because the perturbations are insensitive to the semantic structure of the images, they clearly incapable of making the wide range of adaptive changes possible using the approaches introduced in this thesis. They would not, for instance, be able to change the background from snow to grass, or induce a dog to stick out its tongue, as discussed in Chapter 6. By exploiting learned representations, our techniques select the features to change automatically, rather than manually, an entirely complementary approach.

3.5 Generating test cases without perturbations

This section considers papers that primarily solve the test oracle problem by using a conditional generative network, as our technique introduced in Chapter 4, rather than using a perturbation from known data. Works that train a conditional generator but in effect constrain it to only output data a small ℓ_p distance from known data are instead discussed in Section 3.3.3. The novelty of the algorithm introduced in Chapter 4 depends on its dissimilarity from these papers.

Sauer and Geiger [171] train a disentangled generative model in three parts that allows the user to separately specify the foreground object shape, foreground object texture (or contents), and background in the generated images. While the paper aims to generate 'counterfactual images' to be used for data augmentation, a repurposing of this technique could be used to generate test cases by holding relevant parts of the image constant and varying (say) the background or texture.

This would evaluate a model in this fairly specific way; the resulting images are obviously artificial, being roughly equivalent to cutting and pasting one image on top of another, cut in the shape of a third object. Indeed, it is not always clear how the generated images should be classified – what is the most appropriate label for an ostrich-shaped strawberry in a swimming pool, to take the example from their first figure? Experiments with human judges would clarify this. In any case, it is clear that this approach is at best complementary to the techniques introduced in this thesis. The kinds of test cases that are possible to generate are restricted to this hand-specified mask-foreground-background regime, whereas our algorithms are not restricted in this way (and would be unlikely to output such test cases).

Zhou et al. [172] generate what are essentially perturbations of existing training data to provide more training domains to increase the domain generalisation ability of a model. Like our algorithm described in Chapter 4, a conditional generator network is trained using multiple losses. But whereas the loss term we use to preserve the meaning of the generated data operates through a discriminator, thereby constraining the distribution of generated data, the cycle-consistency loss used by Zhou et al. forces each generated image to closely map (e.g., the shape of a digit) onto a particular training dataset example. So our algorithm is freer to generate images that do not directly resemble any specific training examples. Perhaps more importantly, Zhou et al. optimise the generator to generate images that are at as great as Wasserstein ("earth moving" or optimal transport) distance as possible. The result is that it seems to generate images that have different iridescent colourful backgrounds but are otherwise unchanged. While it could be useful to evaluate a model's performance when the background changes in this way (and the paper demonstrates that they are useful for domain generalisation training), the set of plausible test cases this algorithm could create are different from those of our new algorithms – and our test generation algorithms are specifically optimised to identify problems, rather than merely optimised for data diversity.

Like the present work, Liang et al. [173] also use generative networks to perform semantic manipulation. But whereas our work aims to keep the true class of the generated image the same (while changing the classifier prediction), Liang et al. aim for the converse: to *change* the object in the image (say from cat to dog) while preserving details such as image structure and colour that do not determine the image class. This entails a significantly different approach to training GANs to that taken in Chapter 4.

3.6 Effect of adversarial training on generalisation

Chapter 7 presents an empirical result relating adversarial training against pixel-space perturbations to much decreased ability to generalise well to high-level changes. This section discusses relevant findings in the existing literature.

Adversarial training is known to improve model robustness in the presence of the kind of perturbation used during this training, being the only such technique not found wanting by Athalye, Carlini, and Wagner's analysis [95]. In this context, Xie and Yuille [174] find that adversarial training is more effective if the Batch Normalization is removed or appropriately adjusted, and if models with even more layers than usual as used.

However, there is evidence that adversarial training may not provide much benefit beyond improving performance in the presence of the specific kind of perturbation used during training. Gulrajani and Lopez-Paz [175] compared techniques designed to improve out-of-distribution generalisation, and found that a carefully tuned baseline optimised only for performance on the training distribution was not significantly outperformed by any generalisation-specific technique on a range of datasets. Wiles et al. [176] analyse nineteen techniques designed to improve generalisation on six datasets under three categories of distribution shift and conclude (among other things) that it *is* possible to sometimes outperform a carefully tuned standard baseline, but which techniques are useful depends on the kind of dataset and the kind of data shift.

Hendrycks and Dietterich [166] evaluate the robustness of networks to various types of corruption such as Gaussian noise, blurring, fogging, pixelation, and brightness and contrast adjustments. They found that most models generalised surprisingly poorly in the presence of these simple changes, but that networks adversarially trained to be robust in the presence of l_∞-constrained pixel perturbations performed slightly better. Kang et al. [177] built on this work, performing an extensive empirical evaluation of how adversarial training against one kind of adversarial perturbation or corruption affects robustness to other kinds. In short, they warn that trained robustness against one kind of change does *not* necessarily transfer to other kinds of change that were unseen during training, suggesting that adversarial training against one kind of pixel perturbation is unlikely to be sufficient to improve out-of-distribution generalisation performance in general. This is a strengthening of an earlier result, that adversarial training against only one ℓ_p norm does not confer robustness to other values of p [178].

Taori et al. [179] set out to investigate how models generalise in the presence of 'natural' distribution shifts that arise in real data, as opposed to synthetic changes such as corruptions (e.g., Gaussian noise, artificial fog), pixel perturbations, and style transfer. The 'natural' shifts examined include nearby video frames, and different datasets gathered with compatible sets of class labels. Model performance substantially worsens in the presence of these shifts. Moreover, 86 different models designed with out-of-distribution robustness to synthetic shifts in mind – typically with adversarial training or data augmentation – show "little to no consistent improvements" on these shifts.

It has been observed that adversarial training decreases accuracy on a standard hold-out test set [88, 180], at least partly because of a trade-off: reliance on 'non-robust' features can improve accuracy while decreasing the pixel-perturbation robustness demanded by adversarial training [180, 181].

But there is some limited evidence that adversarial training may in fact decrease model out-of-distribution generalisation performance in some situations. While Gilmer et al. [182] saw that adversarial training improved performance on

most ImageNet-C corruptions, it decreased performance from 85% to 55% in the presence of fog and contrast changes. Yin et al. [183] further investigate, identifying that adversarial training against standard pixel perturbations decreases performance in the presence of corruptions with low frequency in the Fourier domain (such as pixel-wise addition of 2D sinusoids with long wavelengths). Tramèr et al. [112] demonstrate that adversarial training decreases robustness to 'invariance attacks' that change the true label but maintain the model's prediction. This attack works by identifying the most similar training-set image with the desired class, and applying a perturbation to make the starting image look like this training image (without changing the classification model output).

Building on this, the findings in Chapter 7 also show that adversarial training using ℓ_p constrained pixel perturbations not only fails to generalise to distribution shifts other than the perturbations used in training, but that adversarial training can significantly *worsen* out-of-distribution generalisation. This notably strengthens the limited existing evidence, described above. By leveraging the latent representations learned by a generative model, the new finding concerns generalisation to context-sensitive changes to high-level features (for instance, object location or pose). While poor generalisation to artificial fog and contrast [182] and low-frequency sinusoids [183] hint that adversarial training can worsen performance in limited circumstances, our demonstration that adversarial training causes poor performance under features changes that are derived from real data cast significant doubt on whether adversarial training is more helpful than harmful in practice. And although 'invariance attacks' [112] are complementary to standard testing, Tramèr et al.'s algorithm can only apply to the MNIST dataset or other datasets simple enough for pixel-space operations such as distances and perturbations to approximate semantic operations such as semantic similarity and semantic interpolation.

3.7 Miscellaneous related work

Another approach to testing a particular classifier that relies on neither perturbations nor generative modelling is to simply gather new data. On ImageNet, Recht et al. [184] repeat the original process used to create ImageNet and CIFAR-10, and find that state-of-the-art classifiers fail to generalize. Hendrycks et al. [185] photograph scenes in the world that are deliberately intended to cause ImageNet classification networks to perform very poorly. These approaches can be an effective way to evaluate generalisation performance, with the benefit that the data is 'real'. But gathering new data is orders of magnitude more costly than using test generation algorithms that exploit generative modelling to avoid the need to collect fresh data, as in this thesis.

Djolonga et al. [186] transplant foreground objects belonging to 62 ImageNet classes from an open-source dataset onto nature landscapes from open-source stock images, and vary the pasted objects' location, size and orientation. This approach reliably solves the test oracle problem since the foreground objects are known, and evaluates generalisation in ways that reveal the presence of learned shortcuts. But the resulting images are typically artificial and unrepresentative of plausible situations, unlike the tests generated by the algorithms presented in this thesis.

D'Amour et al. [17] demonstrate that using different random seeds to initialise training results in quite different out-of-distribution model performance (on ImageNet-C and ObjectNet), although training and hold-out test set performance does not vary. This adds to the evidence that out-of-distribution generalisation performance is simply ignored when standard training is used; unless we have good reason to expect performance to generalise outside the training data and task, we cannot rely on it.

Hu et al. [187] introduce a search for pairs of nearby test cases for which the classification model under test outputs different answers despite their similarity. Unfortunately, this method does not ensure that the selected test cases

have meaningful semantics; if the search starting point is random, they are overwhelmingly likely not to. If instead, it is a known input, the examples reduce to a standard pixel perturbation.

Defense-GAN [188] is not related work, although it does relate to both adversarial examples and generative modelling. Its algorithm (unsuccessfully [95]) attempts to mitigate worst-case pixel perturbations by first projecting a model's input onto the GAN's learnt data manifold. This cannot be repurposed to generate test cases, or to otherwise evaluate a model's generalisation ability.

Some prior work has trained GANs to generate data that are not only realistic, but also have some other property. Guimaraes et al. [189] extend the SeqGAN [190] approach of using reinforcement learning to train GANs on discrete sequence by simply adding an additional reinforcement learning reward signal. This approach was applied to generate molecules which were both realistic and 'drug-like', and musical melodies which were both to some extent realistic and tonal. Cao and Kipf introduced MolGAN [191], which generates drug-like molecules with some success. In addition to the usual GAN loss, the generator was incentivised to maximise a score given to its generated molecules by (a neural network approximation of) a third-party drug-likeness evaluation function. So the procedure introduced in Chapter 4 is not the first to use the idea of multiple simultaneous objectives during GAN training. However, the additional objective used is quite different from that of previous work, and the techniques introduced that improve the likelihood that training is able to well optimise both objectives simultaneously are new.

3.7.1 Domain generalisation datasets

The domain adaptation and domain generalisation literatures attempt to address specific sub-problems of the shortcut learning problem. In this context, "domain" simply means a different data distribution (usually on the same underlying set). Image classification examples would include photographs of the same objects taken in different countries, or the same objects represented in different artistic

styles. The domain generalisation literature [192] assumes that you are given data from some number of such domains, and need to generalise to an entirely unseen new domain; domain adaptation [193] differs only in that you are given some data from the new domain, and typically may be given only one training domain. The majority of these literatures concerns the development of new techniques to create models that will generalise better to the new domains. Besides the data augmentation approaches that are sometimes included as part of this literature, discussed earlier in this chapter, these techniques are not relevant: our focus is on the evaluation of models.

Techniques designed to improve a model's domain generalisation are evaluated using datasets that consist of multiple domains. The technique being evaluated trains the model using the data from all the domains but one, which is then used to measure performance. For example, the PACS dataset [194] contains the same seven object classes in four different types of image: Photo, Art painting, Cartoon and Sketch (hence "PACS"). Another common choice is the use of four different digit classification datasets (domains) [172]. Zhou et al. [192] list over twenty such image classification domain generalisation datasets in their first table The recent NICO++ dataset [195], containing images labelled with both their class labels and their domains, is particularly extensive. Artificial datasets such as Colored MNIST [196] allow the evaluation of whether a specific shortcut is taken or avoided.

How useful are these multi-domain datasets as a way of evaluating out-of-distribution generalisation of the kind we are concerned about? Well, if you have a model trained to perform the same task as one of these datasets, and to the extent that the variation you expect at deploy time matches the variation between domains in a dataset, such an evaluation is helpful. More broadly, testing a model using data from a different dataset gives you *some* information about how it may adapt to other changes in dataset.

But the test generation algorithms introduced in this thesis have two important advantages over these datasets. First, they are task-agnostic. Rather than

needing to gather or find a dataset trained on the relevant task, the algorithms take a dataset of task examples and generate suitable tests. Second, rather than testing the model in one specific new situation, they deliberately construct data to probe and identify failures in your model. Generalising successfully to one new test domain does not necessarily indicate successful generalisation to *other* unseen domains; weaknesses are more clearly identified when deliberately sought out.

4

Training Generative Networks to Output Test Cases

Contents

As motivated in Chapter 1, this research aims to further analysis of the ways that deep neural networks use 'shortcuts' in their learning and thereby generalise poorly outside the original distribution of training data. Better software tools for this purpose would allow both the testing of particular systems that will be deployed in practice, and would improve our fundamental understanding of how deep models work, what their limitations are, and how to address these.

This chapter introduces a new procedure for the testing of deep neural networks. This procedure aims to identify new ways that the model being evaluated fails to generalise outside its training distribution. In short, it works by

training a generative neural network using a new objective function so that its generated outputs:

1. induce an incorrect output from the model being evaluated, while
2. retain the expected semantics from the training distribution.

By solving the test oracle problem in a new way – leveraging the abilities of conditional generative networks – this approach is able to generate a much broader range of test cases than prior works that use a heavily constrained perturbation approach.

Section 4.1 presents the new test generation procedure. As well as the essential approach to training a generative network so as to generate useful test cases, it also describes difficulties encountered in the training process, and new techniques introduced to mitigate these. The rest of the chapter contains an extensive empirical evaluation of this new procedure, including verification that the generated tests have the desired properties (having the intended semantics, while being misclassified by the tested model), experiments evaluating the procedure's ability to find *new* problems in a model, and a set of ablation experiments to determine how useful a contribution each aspect of the procedure makes. Section 4.9 evaluates how well the presented new algorithm performs according to the requirements set out in Chapter 1, in light of our experimental evaluation.

4.1 Procedure for training generative networks

Suppose we have a trained target classifier network $f : \mathbb{X} \to \mathbb{R}^{|\mathbb{Y}|}$ that attempts to approximate an oracle partial function $o : \mathbb{X} \rightharpoonup \mathbb{Y}$ by outputting a confidence $f(x)_c \in \mathbb{R}$ for each class $c \in Y$. Our goal is to identify test inputs x such that the classifier's prediction is incorrect: $\arg\max_c f(x)_c \neq o(x)$.

A key difficulty is the test oracle problem. If we had cheap access to o, we would not need the approximation f – so for a particular test input x, how can we know whether the prediction of f matches the answer from the oracle o? One

approach is to only allow test inputs that are sufficiently similar to inputs for which the correct output is known so that the oracle can be assumed to give the same answer. If we want to take a different approach, this provides a vastly larger space of candidates, but we must solve the test oracle problem another way.

We leverage conditional generative adversarial networks to solve this problem. Recall from Section 2.1.2 that a conditional generator learns to generate realistic examples of each specified label: if g is a well-trained conditional generator, then $g(y, z)$ should be an instance of class y if z is suitably randomly sampled. We can rely on this feature to solve the test oracle problem – we can assume that the correct label for $g(z, y)$ is in fact y. This assumption is experimentally validated in section 4.3. In principle, any conditional generative model would suffice – conditional variational autoencoders (VAEs), for instance. But GANs are a good practical choice. As noted in Section 2.1.3, they are particularly able to generate samples difficult to distinguish from real data, even at the cost of 'mode dropping', or not representing the full training distribution. Their popularity also means that there are model architectures and checkpoints available.

4.1.1 Dual-objective training of generative networks

So we begin by taking any conditional GAN, with generator loss $l_{ordinary}$; the use of a conditional GAN allows us to determine the correct label y of our generated test inputs. We then introduce loss terms which incentivise the generator g to create particularly useful or probing tests for a classifier f. In particular, we want to identify test cases for which the classification model predicts incorrectly – such examples indicate ways that the model fails to generalise out of distribution.

By default, in the 'untargeted' case, we introduce an additional loss term that is minimised for any misclassification of the test input $g(z, y)$:

$$l_{untargeted} = f(g(z, y))_y - \max_{c \neq y} f(g(z, y))_c.$$

This term is the difference between the classifier f's confidence in the correct label and the classifier's confidence in whichever incorrect label it has the greatest

confidence in. Minimising this term is achieved by increasing the confidence in an incorrect label over the confidence in the correct label.

For our purposes, we will sometimes prefer to generate *targeted* tests. The targeted case for true label y and target classification $t \neq y$ should output test inputs that a human (oracle) would consider to have label y yet are predicted as t by the classifier. This is useful in addition to the untargeted case because it allows a more thorough probing of the classification model's behaviour. Generating test cases aimed at a particular target label lets us investigate failures or shortcuts associated with specific decision boundaries; the features and proxies used by the classifier may not be uniform across the input space. To produce such tests, we introduce a loss term, $l_{targeted}$, which is minimised when the conditional generator output $g(z, y)$ is classified in this way:

$$l_{targeted} = \max_{c \neq t} f(g(z, y))_c - f(g(z, y))_t.$$

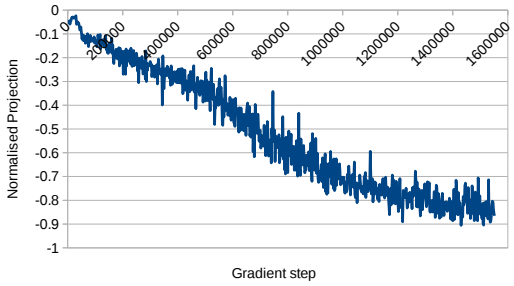
This is the difference between the classifier f's confidence in its current prediction (besides t) and the classifier's confidence in the incorrect target label t. Minimising this term is achieved by increasing the confidence in label t over its otherwise highest-confidence label.

Note that these new terms assume that the true labels of the generated data $g(z, y)$ do indeed match the intended labels y, an assumption empirically validated in Section 4.3.2.

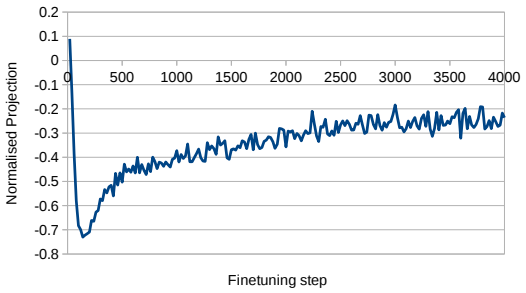
Our procedure is to alter the generator's training objective so as to minimise both $l_{ordinary}$ and $l_{(un)targeted}$ simultaneously, thereby training the generator to generate new test cases which are both realistic enough to maintain their meaning and also reveal regions of input space over which the classifier generalises incorrectly.

4.1.2 The challenge of conflicting gradients

Intuition suggests that the gradient from $l_{ordinary}$ may be pointing in a different direction to the gradient from $l_{(un)targeted}$. Note that *most* changes that can be made to an image would distinguish it from unchanged training examples:



(a) Beginning from a randomly-initialised GAN.



(b) Finetuning a pretrained GAN.

Figure 4.1: Projecting normalised gradient vectors from $l_{ordinary}$ and $l_{(un)targeted}$ onto one another.

the default is that any change will be in conflict with the goal of creating data indistinguishable from training examples. In addition, consider the features used by the classifier to distinguish between classes. These are the features that gradient updates from $l_{(un)targeted}$ will encourage the generator to change. Manipulating these features so that (say) a '7' is classified as an '8' seems likely to be in direct tension with manipulating the features of that image so as to make the most convincing and normal '7' possible.

A simple experiment suffices to verify whether these gradients do in fact conflict. We compute the cosine similarity between the gradients of the two loss terms at each step, i.e. $\frac{\nabla l_{ordinary} \cdot \nabla l_{(un)targeted}}{\|\nabla l_{ordinary}\| \|\nabla l_{(un)targeted}\|}$. Figure 4.1a shows that this projection tends towards -1; for reference, if the gradient vectors were selected uniformly at random, the magnitude of this projection would very rarely exceed 0.001. In other words, as training progresses, the gradients from these terms tend toward pointing in actually *opposite* directions. This makes joint optimisation using a gradient descent approach challenging.

4.1.3 Strategies to overcome training challenges

We empirically evaluate the effect of each technique described in this section in our ablative experiments reported in Section 4.6.

Realistic pretraining It is widely accepted that real image data occupy a relatively low-dimensional and contiguous manifold [197, p. 160] among the

set of all possible image instances. Conversely, we know that misclassified inputs pervade the full input space – the phenomenon of pixel-perturbation 'adversarial examples' show that there is a misclassified example within a small distance of nearly any point in the input space. Therefore, a generator that is pretrained using only $l_{ordinary}$ before *dual-objective finetuning* by introducing our additional loss term is more successful than using both loss terms from a random initialisation. By beginning our search with a generator that has an approximate representation of the training distribution, we are more likely to find test cases that fulfil both criteria.

Besides the generated images being visually closer to the training distribution because the training distribution is now the start point for the finetuning, Figure 4.1b shows that the gradients conflict to a much lesser extent. One possible explanation is that small changes to data that are nearby the training distribution are small enough that they largely result in data that are still nearby the training distribution, so that the changes induced by optimising for incorrect classification do not contradict the objective to stay nearby the training distribution. Another related possible explanation is that the ways that the generator can most easily cause examples to be misclassified entail making arbitrary unrealistic changes when the generator is randomly initialised; but when the generator is pretrained, the ways it can most easily cause examples to be misclassified are by making changes that affect the within-distribution characteristics of the data, such as changing the curve of a shape, rather than adding random noise.

Note that any existing conditional GAN architecture, pretrained checkpoint and training algorithm could be used here, allowing our procedure to leverage the significant advances being made in this area.

Amalgamation of loss terms Rather than naïvely summing $l_{ordinary}$ and $l_{(un)targeted}$, we use the following per-example loss term:

$$l_{finetune} = s(l_{ordinary}) \cdot s(l_{(un)targeted} - \kappa), \text{ where } s(l) = \begin{cases} 1 + \exp(l) & \text{if } l \leq 0, \\ 2 + l & \text{otherwise.} \end{cases}$$

Here, κ is a hyperparameter similar to that in Carlini and Wagner [71]: it controls the overconfidence of the misclassification of the generated test inputs. If the difference between the desired logit and the next-greatest logit is less than κ, the generator is linearly rewarded for improving this gap (gaining confidence); beyond a difference of κ (once an example is 'good enough'), the reward exponentially decreases. $\kappa = 0$ is used for our experiments as strong misclassifications are not required.

Stochastic loss selection The gradients from the two loss terms are in conflict, and in practice the $l_{(un)targeted}$ gradient dominates. The proportion of misclassified generated inputs rises quickly to almost 100%, but the generated images were noticeably visually overly different from the training distribution, meaning their correct label may change. To address this, we introduce a new hyperparameter: 'finetuning rate', μ. During adversarial finetuning, the finetuning dual-objective loss term is used at each step only with probability μ; with probability $1 - \mu$, the pretraining loss ($l_{ordinary}$ only) is used. As desired, this new hyperparameter allows the proportion of generated tests that are misclassified to be traded off with their similarity to the training distribution.

4.2 Experimental evaluation setup

Because the success of the test generation procedure depends on fundamentally empirical questions such as how humans interpret the meaning of the generated examples, it is essential to subject it to a comprehensive empirical evaluation. This section describes the setup for the experiments described in the remainder of this chapter. For context and for reproducibility, this section contains the full details of the set-up used for the experiments in the remainder of this chapter.

The MNIST handwritten digit dataset [198] is the main focus of the experimental evaluation, because this is the most challenging domain to find failures in models. The classification task is particularly easy, so state-of-the-art models perform very well, with around 0.2% test error [199, 200]. Furthermore, attempts

Figure 4.2: Samples from the training dataset.

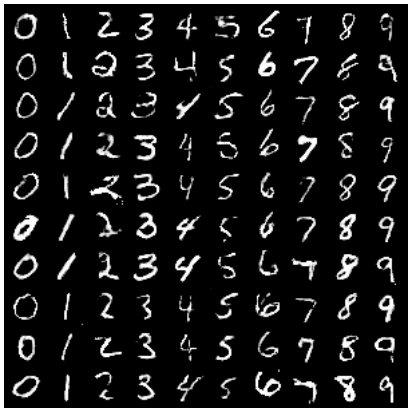


Figure 4.3: Samples from the pretrained generator, trained to model the dataset.

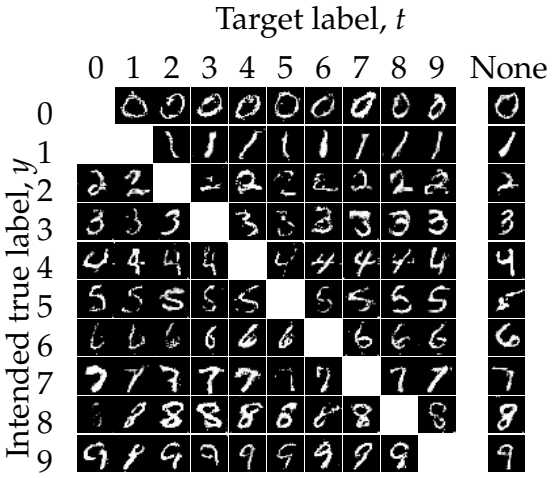


Figure 4.4: Examples of generated targeted (grid) and untargeted (rightmost column) test inputs for a standard MNIST classifier by eleven different finetuned generators.

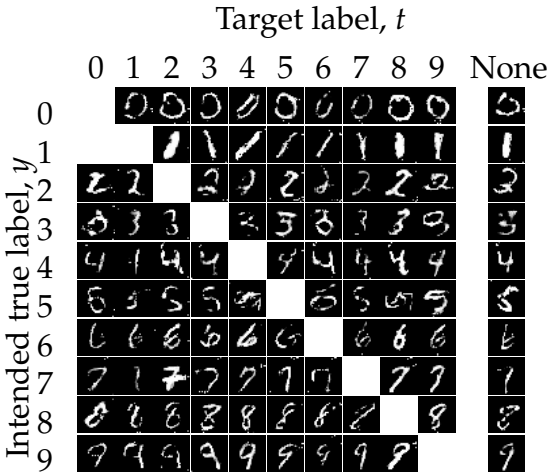


Figure 4.5: Examples of generated targeted (grid) and untargeted (rightmost column) test inputs for Wong and Kolter's [100] robust MNIST classifier by eleven different finetuned generators.

to create classifiers robust to pixel perturbations have also been most successful on this dataset, again due to its simplicity [83]. The experiment targets five pretrained classifiers 'provably robust' to bounded pixel perturbations (plus two non-robust models): there is no misclassified input within a distance ϵ of p% of test inputs under the ℓ_∞ norm. All five are the current state-of-the-art in this domain, trained by Wong and Kolter [100], and Wang et al. [102]. See table 4.1 for full details.

In the experiments, the MNIST GAN architecture is a combination of a

Table 4.1: Descriptions of and references to the classifiers evaluated. For each robust model, there is no misclassified input within a distance ϵ of p% of hold-out test set inputs under the ℓ_∞ norm.

Our Name	Abbrev.	ϵ	p	Architecture
Wong and Kolter [100]	W&K	0.1	94.2	2 convolutional layers followed by 2 dense layers
MixTrain [102] A	MT-A	0.1	97.1	'MNIST_small': 2 convolutional layers followed by 1 dense layer
MixTrain [102] B	MT-B	0.3	60.1	'MNIST_small': 2 convolutional layers followed by 1 dense layer
MixTrain [102] C	MT-C	0.1	96.4	'MNIST_large': 4 convolutional layers followed by 2 dense layers
MixTrain [102] D	MT-D	0.3	58.4	'MNIST_large': 4 convolutional layers followed by 2 dense layers
Fully-Connected	FC	N/A	N/A	Three fully-connected layers of size 256, 128 and 32 with LeakyReLU activations. Standard training.
Standard Convolutional	Conv.	N/A	N/A	Three convolutional layers of size 256, 128 and 32. LeakyReLU activations. Standard training.

Wasserstein GAN with gradient penalty (WGAN-GP) [30], a conditional GAN [35] and an auxiliary classifier GAN [36]. Comprehensive details are given in Appendix B.1.

A GAN was finetuned using dual-objective training for each of the 10 target labels, and for the untargeted case. Once trained, each of these generators was used to produce test cases for all digits y from 0 to 9. These were then filtered so that the classifier output matched the target label for that generator – if the classifier gave the correct output (y), for instance, that particular test

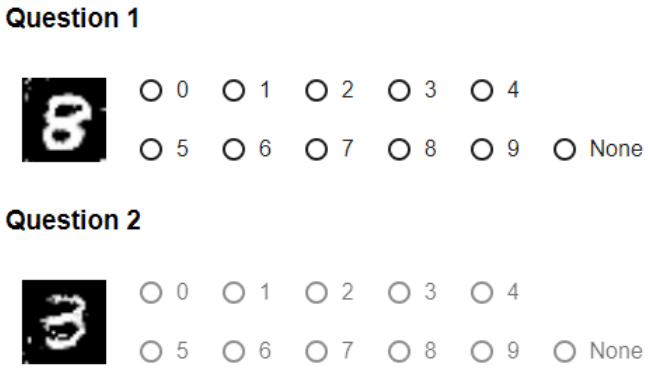
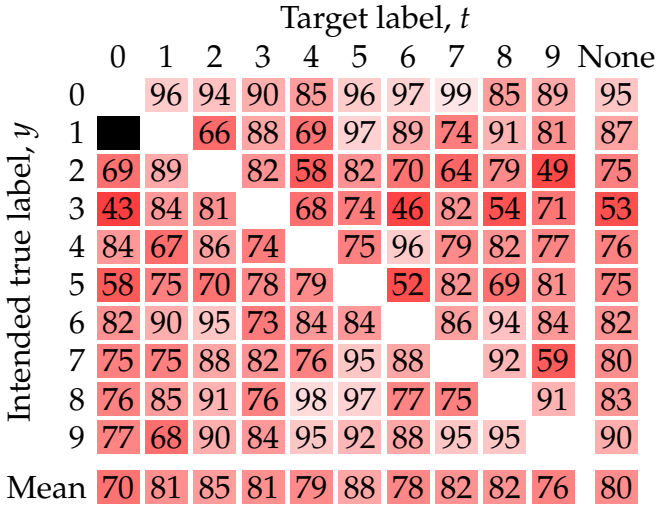


Figure 4.6: Screenshot of the interface used by participants to label generated test inputs, excluding the instructions and later questions.

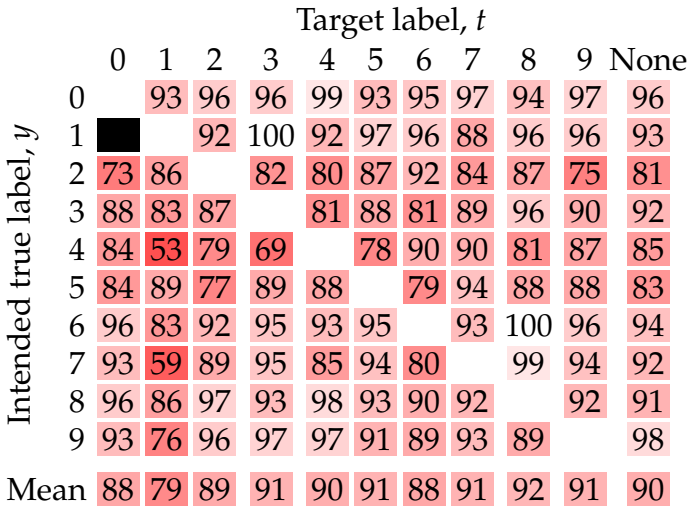
case would not be used.

Test cases were generated until 200 filtered examples were obtained, or until 100 seconds had elapsed. Figure 4.4, Figure 4.5, and Appendix B.1.1 give examples of generated images for which the computed label matches the target classification.

Interestingly, this led to no test cases with true label '0' and target classification output '1', most likely because these classes are exceptionally easy to distinguish (a '0' must always have a dark region at its centre), because the set of manipulations available to the generator for a '1' does not include anything that would lead to a '0' classification, and because the generator can 'get away with' ignoring particular cases that are too difficult if it leads to better overall performance across both objectives and all inputs. So this particular case is omitted from the results since the generator produced no examples of it. If it were of specific interest that this was omitted, one approach would be to finetune a generator to generate only this particular case; another would be to modify the loss term so that there was a high penalty for failing to generate enough misclassified examples of a particular type. But for our analyses, it is not a problem that this one case is missing.



(a) Testing the Wong and Kolter robust network (described in Table 4.1).



(b) Testing the standard convolutional network (described in Table 4.1).

Figure 4.7: The proportion of filtered generated test inputs for which humans judge the correct label for the generated image to be the intended true label, y. The filtered generated inputs, $g(z, y)$, are those generated by finetuned generative networks that have been filtered to remove the under 1% of examples for which the classifier prediction does not match the (incorrect) target label t. The colours represent the data visually.

4.3 Efficacy of test generation

The primary empirical evaluation is whether this approach is able to generate test inputs for which the model under test gives incorrect outputs. There are two separate checks that need to be made. First, that the procedure can generate inputs for which the label output by the classifier under test, $\arg\max_i f(g(z,y))_i$, is not the intended correct label y passed to the conditional generator. Second, that the label that a human would assign to those generated examples, $o(g(z,y))$, matches the intended correct label y passed to the conditional generator. Since equality is transitive, these two results imply that $\arg\max_i f(g(z,y))_i \neq o(g(z,y))$; the classifier output is incorrect on the test input $g(z,y)$.

For targeted cases, this is slightly modified. Rather than requiring classifier output $\arg\max_i f(g(z,y))_i \neq y$, we instead require $\arg\max_i f(g(z,y))_i \neq t$, where $t \neq y$ is the targeted label.

4.3.1 Classifier outputs are as desired

The first check is how often the label output by the classifier under test, $\arg\max_i f(g(z,y))_i$ is as required: either any label except y, or particular target label t in the targeted case. Luckily our procedure is particularly effective in this respect: it is easy for greater than 99% of generated test inputs to be misclassified by the classifier, either as any label but y, or as t in particular if required. And the small fraction that does not satisfy this requirement is simply automatically filtered out.

4.3.2 Semantics of data are as desired

So most of our effort will be directed at the need to check that oracle (human) judgement of the true label matches the intended label y for each example. Without this check, the generator could simply be producing images that in fact should *not* be classified as y. To investigate this, we asked human judges, recruited on Amazon's MTurk platform, to classify the generated images. For cost reasons, we only did this targeting Wong and Kolter's pixel-robust network

[100] and our standard, non-robust, convolutional network. We used a sample size of 100 different judgements for each (intended true label y, target label t) pair for each experiment. The test cases shown to the judges were randomly sampled from the 200 for each (y, t) pair. Figure 4.6 shows the key part of the interface used by participants.

The results are presented in Figure 4.7. Each number shows the proportion of judgements for the particular (y, t) pair for which $o(g(z, y)) = y$. Since the test cases are already filtered so that $\arg\max_i f(g(z, y))_i = t$, this number directly measures the success of the algorithm in its ability to generate new test cases that reveal the classification model's failures to generalise. Testing the robust network, the mean proportion of successful test cases in the untargeted regime is 80%, with comparable numbers for the targeted cases. These numbers increase to around 90% when testing the standard model. This is clear evidence that this test generation algorithm meets our first two requirements: that the generated tests are meaningful inputs according to the oracle, and that they cause failures (misclassifications) in the model being tested.

4.4 Ability of tests to identify *new* problems

The solution to the test oracle problem, used in almost all prior work (exceptions examined in Section 3.5), is to explicitly or implicitly bound the distance of the new test from a known existing input. According to Taori et al. [179, p.1], "all of the distribution shifts [in the literature] are synthetic: the test examples are derived from well-characterized image modifications at the pixel level". But as discussed in Chapter 3, this limits the test inputs to a very small region of the possibilities; because our new procedure instead solves the test oracle problem by simply relying on a conditional generative network to generate examples with suitable semantics, we should expect our new procedure to identify new ways in which the tested models fail to generalise because of its ability to identify test cases that prior methods cannot. In this section, experimental evidence is presented investigating whether this is indeed the case.

Table 4.2: Comparison of the closest distance to the nearest training example from ten particularly realistic generated tests with typical pixel-space perturbation magnitudes found in the literature.

Metric	Nearest neighbour seen	Typical perturbation magnitude
l_0	508	<40 [201]
l_1	22.8	<5 [202]
l_2	3.28	~1.5 [203]
l_∞	0.838	~0.1 [100]

Table 4.3: Ten selected test inputs used for Table 4.2.

4.4.1 Distance from dataset examples

First, we perform a simple check to verify that the generated images are indeed not close to images in the training set, as could be caused by over-fitting. We selected ten generated inputs that are visually similar to the training set and computed the shortest distances between the images and all images in the training set. The selected images are given in Table 4.3. Table 4.2 shows that they are much further from any training example than would be the case with an ℓ_p-bounded perturbation.

So, as expected, the new approach to solving the test oracle problem *does* free the generated test cases from being at all nearby any existing examples.

4.4.2 Transferability

Pixel perturbation-based test cases typically somewhat transfer between models, in the sense that examples optimised to cause one model to give the incorrect prediction tend to cause the same behaviour in other, different models trained on the same task [64, 204].

This has two implications. First, that different models trained on the same classification task are using similar shortcuts, causing them to fail to generalise in similar ways. Second, that the pixel perturbation algorithms in question tend to produce test cases that reveal this shared shortcut.

This section presents an experiment investigating whether the same phenomenon occurs using the test generation algorithm of this chapter. About 20,000 untargeted failing test cases were generated for each of five target classifiers, and each of these tests was used as an input to the other four models. By measuring the proportion of these that were classified correctly, we are measuring how well the generated examples generalise between models in the above sense.

Results

The results, presented in Table 4.4, suggest that in general, these test cases do *not* transfer between models. In most instances, over 80% of generated test cases for one model were *correctly* classified by the others. An exception seems to be classifier MT-D, which often gave incorrect predictions for test cases created for the other classifiers.

Interpretation

The main result is that the test cases do not transfer between models. If they did transfer, we could conclude that both (a) the classification models are relying on similar shortcuts because they are failing to generalise in the same ways, and (b) the test generation algorithm finds examples that reveal this fact. So the negative transferability finding suggests the negation of this: the test generation procedure seems to find *different* ways that different models are taking shortcuts and thereby failing to generalise.

In combination with the results in the following section, these results are evidence that the new procedure of this chapter is able to detect new, different ways that deep neural networks fail to generalise out of distribution. Otherwise, we would see that test cases misclassified by one model were also misclassified by another.

Table 4.4: Percentages of failing test cases generated for each classifier (each row) which are also misclassified by the other classifiers (columns). See table 4.1 for descriptions of the classifiers.

		To					
		W&K	MT-A	MT-B	MT-C	MT-D	FC
From	W&K		20.2	18.4	9.0	60.7	16.8
	MT-A	19.5		14.1	13.3	55.2	4.7
	MT-B	5.2	4.8		1.6	57.8	2.6
	MT-C	25.8	47.6	13.9		67.8	12.1
	MT-D	5.9	7.3	9.4	4.3		1.7
	FC	2.7	2.6	2.6	1.3	48.0	

4.4.3 Adaptivity against adversarially trained classifiers

In our experiments so far, we have evaluated our procedure using certain fixed, pretrained classifiers, and found evidence that the faults detected are not the same as those detected by existing algorithms. Because our finetuning procedure is free from the constraints of perturbation approaches, it is plausible that it is able to detect a wider range of generalisation failures than these approaches. This intuition comes from considering that perturbation approaches can only detect failures of generalisation within their tight constraints, whereas in principle a generative neural network is free to generate any test input it chooses.

To investigate this, we apply standard adversarial training [88] using the output of our test generation algorithm. Recall that adversarial training is the most promising approach to improve the out-of-distribution generalisation of a network: if the network generalises badly to a certain kind of input, adversarial training simply includes these worst-case inputs during training, so that the model learns to perform well on them. Consider using the outputs of a test generation algorithm to adversarially train a model. If the test generation algorithm is then unable to generate further useful tests for the so-trained model, this is an indication that it has only a small pool of possible test cases to draw from; on the contrary, if it continues to be able to identify new problems with the

model, this suggests that it is able to detect a wider range of possible problems than just the problems detected by the initial batch of tests for the original model.

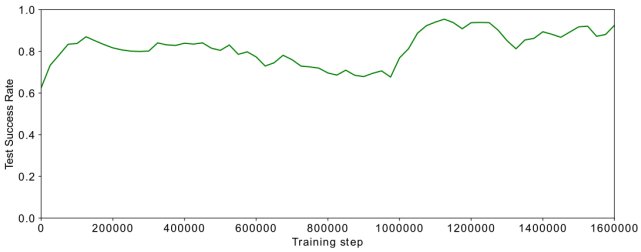
As our test generation algorithm involves a generator network learning the weaknesses of the classifier model over time, it is not immediately clear how to integrate it into the adversarial training framework. The two obvious possibilities are finetuning the generator at each step of training, or alternately training the classifier and finetuning the generator. For completeness, we explore both of these below as 'online' and 'offline' adversarial training, respectively. The classifier used for these experiments (both the architecture training and hyperparameters) is the one used in Madry et al. [88], and in particular from their associated 'MNIST Adversarial Examples Challenge'.

Online Adversarial Training

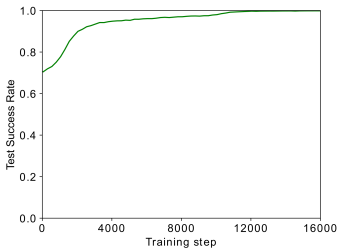
In this experiment, we update both the generator and classifier at each training step. The classifier is updated using a batch of test cases generated by the generator; the generator is updated using the dual-objective finetuning loss. We run adversarial training for over 1.6 million training steps. Figure 4.8a shows that even during training, the generator maintains a roughly 80% success rate in identifying examples for which the classification model's predictions are incorrect. Once we have finished training the classifier, we generate tests using our algorithm that aim to identify failures. Despite the classifier having been adversarially trained against the generated test cases, the generator is immediately able to output test cases that the classifier performs badly on, as shown in Figure 4.8b. After 16,000 gradient steps, over 99% of generated tests identify failures.

Offline Adversarial Training

Starting with a pretrained GAN and classifier, we iterate 'training rounds' consisting of two phases. First, a GAN is finetuned (starting from the initial pretrained GAN each time) for 5,000 gradient steps to generate test cases that induce

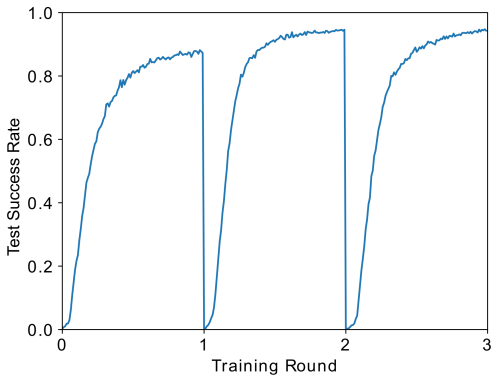


(a) Online adversarial training performance.

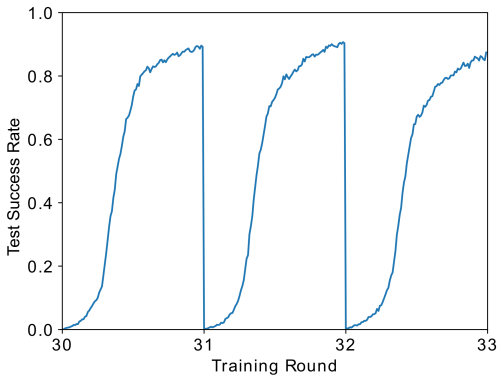


(b) Test case performance after online training.

Figure 4.8: Plots showing the efficacy of the test generation algorithm in the presence of online adversarial training.



(a) Adversarial training against our test generation algorithm, first few training rounds.



(b) Adversarial training against our test generation algorithm, later training rounds.

Figure 4.9: Plots showing the efficacy of the test generation algorithm in the presence of offline adversarial training.

incorrect predictions from the classifier. The hyperparameters are those from Table B.3, but with an finetuning rate of $\mu = 0.4$. Second, 80,000 generated of these test cases are added to the existing training dataset, and the classifier *continues* training on the entirety of the pool of samples generated so far for 30 epochs, with a batch size of 128. This in all cases achieved accuracy close to 100%. Once a training round is completed we start again, resetting the GAN to how it was *before* any finetuning.

Figure 4.9a shows that, in the first training rounds, the finetuning is successful: the proportion of test inputs that the model misclassifies increases to over 80%. Figure 4.9b shows the same story 30 rounds (and hence hundreds of thousands of classifier gradient steps) in. There is no reason to expect this not to continue

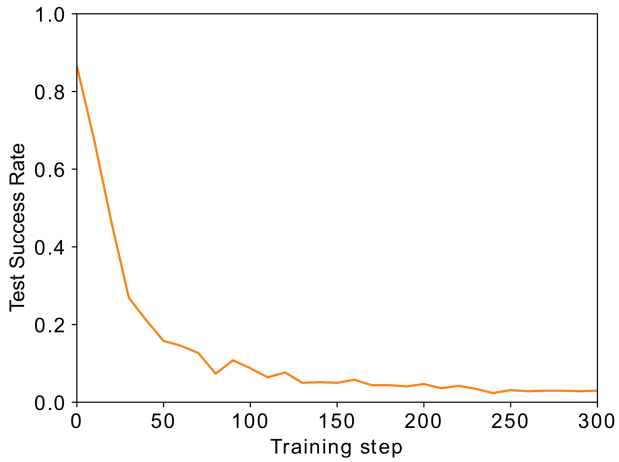


Figure 4.10: Plot showing the effect of adversarial training against Song et al. [2].

in future training rounds.

In short, we find that the adversarially trained classifier is able to correctly classify the kinds of test cases previously produced by the generator. However, the generator's opportunity to finetune again allows it to generate test inputs in a new 'blind spot' of the classifier, suggesting that in contrast to existing perturbation approaches, it is able to identify a wider range of possible problems.

4.4.4 Comparison of adaptivity with prior work

We compare our procedure to that of Song et al. [2], the contemporary state of the art in generating unrestricted adversarial examples. As discussed in Section 3.3.2, this approach, like ours, leverages a pretrained GAN. It differs, however, in how new classifier inputs are produced. Instead of finetuning the generator, it in short searches for an input to the (fixed) generator that both deceives the target network and is confidently correctly classified by the discriminator's auxiliary classifier (an auxiliary classifier GAN [36] is required to use this procedure). The GAN training is therefore entirely independent of the target model using this procedure, in contrast to our procedure that uses dual-objective finetuning to optimise the generator network so as to generate tests for the target model specifically.

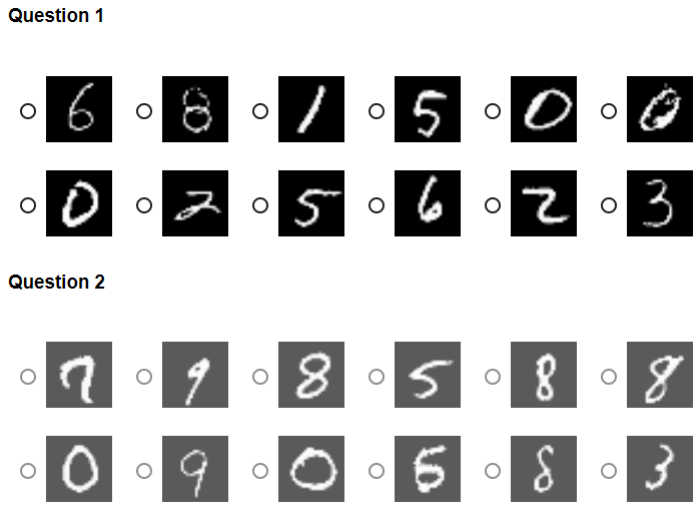
An important difference between the two approaches is their constraints on the range of test cases that can be generated, and therefore their scope to detect different kinds of problems. In Section 4.4.3 we have shown that after further training the classifier using generated test cases, our approach is always able to identify new test cases that induce mispredictions, revealing new regions that the classifiers fails to generalise to. To compare, we repeat this experiment using Song et al.'s [2] test generation algorithm, running 300 training gradient steps for the Madry et al. [88] image classifier with a batch size of 64. At each step, the training data is produced by Song et al.'s model. We use their code and the hyperparameters they provide for untargeted attacks in Table 4 of their appendix.

Figure 4.10 plots the results. We can see that the classifier is able to quickly 'patch' the particular failure initially revealed by Song et al., and their test generation procedure is entirely unable to identify new test cases that the classifier performs poorly on. This difference is likely a result of Song et al.'s approach using a fixed generator and constrained perturbations, whereas the procedure of this chapter seems always able to adapt the generator, free from constraints, to find new weaknesses in the particular model being tested. This is significant because it suggests that the set of possible faults detectable by our new procedure is much larger than that of existing perturbation tests.

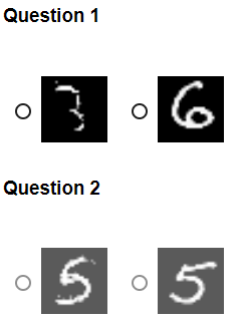
4.5 Similarity of tests to training examples

An important question is whether or not we care about our models failing to generalise to the test inputs generated by a particular procedure. It is of little consequence that a model gives an incorrect output for a particular input if that input is nothing like any data that we care about our model performing well on for practical purposes. (Although it may still be of interest so that we can better understand the internal workings and limitations of models in general.)

A sufficient but not necessary condition for us to care in practice about an input being misclassified is that it is indistinguishable by human judgement from examples in the training dataset. This is because the training dataset by definition



(a) Examples of questions with ten training examples and one generated example. There are twelve images because participants were instructed to ignore the second and third images as a check to ensure that they had read the instructions.



(b) Examples of questions with one training example and one generated example.

Figure 4.11: Examples of the interfaces seen by human judges when trying to pick out which one image is not drawn by a person.

contains examples that we care about the model performing well on, and if humans cannot distinguish two inputs, they must value model performance on both equally, by symmetry.

So, for this reason, this section describes experiments undertaken to determine whether human judges are able to distinguish the generated test cases from dataset examples. Another important motivation of these experiments is to provide more objective evidence about the similarity of the test cases to examples we certainly care about; different readers may have different subjective opinions about whether the generated test cases such as those in Figure 4.4 are similar enough to handwritten digits to care about. By gathering empirical data from a wide range of human judges, the reader need not rely on their own judgement.

Method

To access a wide pool of human judges, we again used MTurk workers. After familiarising themselves with examples from the training dataset, each judge was presented with ten images, nine of which were training examples, and one

of which was a generated test case. Their task was to pick which image out of ten was most likely to have been generated. See Figure 4.11 for cropped screenshots of the web interface used; Appendix B.2 shows full screenshots, including instructions. There was a financial incentive to pick correctly.

This experiment was repeated but with each judge being presented with just two images instead of ten, one of which was a training dataset instance and the other was a generated test case. This was to facilitate a direct comparison with Song et al. [2]. In addition, having these two experimental settings can increase the strength of our evidence, because it should reveal how contingent the results are on the number of images being shown.

Results

The number of judgements out of 100 for which tests for Wong and Kolter's pixel-robust network [100] were *not* identified as the generated 'odd one out' when placed among nine training examples is shown in Figure 4.12. That is, the data show how often the generated tests pass for training examples in this context. If the generated images were completely indistinguishable, then the data would all be expected to be 90% – this is how often a uniformly random guesser would fail to select the odd one out. And if the judges found any way of reliably distinguishing generated examples from training examples, then the data would all be 0%. As it is, for the W&K robust network, around 45% of targeted tests and 50% of untargeted tests were not identified as the odd one out. This increases by about 10 percentage points for tests for the non-robust convolutional network.

Figure 4.13 gives the same results, but in the setting that placed each generated test next to one training example, rather than nine. The bottom line is that the generated untargeted tests were not identifiable 24% or 30% of the time, depending on the classifier, compared with 50% for tests undistinguishable from training examples, or 0% for tests that can always be distinguished.

Intended true label, y	Target label, t										
	0	1	2	3	4	5	6	7	8	9	None
0		40	60	56	34	46	51	40	36	63	51
1			37	52	36	51	81	40	53	35	49
2	30	37		43	40	42	35	37	55	32	54
3	39	39	43		34	40	40	42	45	48	40
4	51	50	34	38		37	46	42	41	43	40
5	32	34	32	36	43		42	36	37	55	51
6	51	39	45	36	57	46		45	57	40	46
7	47	48	53	33	42	58	41		52	44	39
8	29	46	47	55	44	48	36	39		42	60
9	38	34	50	49	54	53	53	69	57		67
Mean	40	41	45	44	43	47	47	43	48	45	50

(a) Tests targeting the Wong and Kolter robust network (described in Table 4.1).

Intended true label, y	Target label, t										
	0	1	2	3	4	5	6	7	8	9	None
0		45	48	46	43	38	61	51	44	59	53
1			64	74	72	62	78	83	73	75	79
2	52	43		53	41	52	35	48	54	40	55
3	64	41	62		43	60	29	51	56	58	50
4	59	45	49	36		45	53	55	49	69	65
5	48	46	44	63	62		60	49	57	61	50
6	76	48	44	46	54	54		38	62	54	58
7	51	32	60	59	54	54	46		61	65	65
8	62	53	62	57	56	56	54	50		67	57
9	47	42	51	60	72	69	54	66	71		67
Mean	57	44	54	55	55	54	52	55	59	61	60

(b) Tests targeting the standard convolutional network (described in Table 4.1).

Figure 4.12: The number of times out of 100 that generated test images are not identified as the 'odd one out' in a set of ten. If the generated images always passed as training examples, the expected result would be 90; if they were always spotted, it would be 0. The colours represent the data visually.

Discussion

The main result is that human judges were often unable to identify which of the presented examples were generated test cases rather than drawn from the training dataset. This implies that at least a large fraction of the generated tests are indistinguishable from training examples under human judgement. Therefore, we can be confident that the tests generated by our procedure identify failures *that we care about*. That is, the failure-inducing test inputs are of real concern, and cannot be dismissed as being too different from the training data.

The side-by-side setting was partly motivated by facilitating a direct comparison with prior work. Generating tests for the W&K network, Song et al. [2] report that participants select the generated image as the more realistic of the two 21.8% of the time, while for our untargeted case, this figure is 24%; completely realistic images would be chosen 50% of the time. This suggests that the procedures are broadly similar in their ability to generate test cases that look like training examples, with our method perhaps representing a small improvement.

4.6 Ablative studies

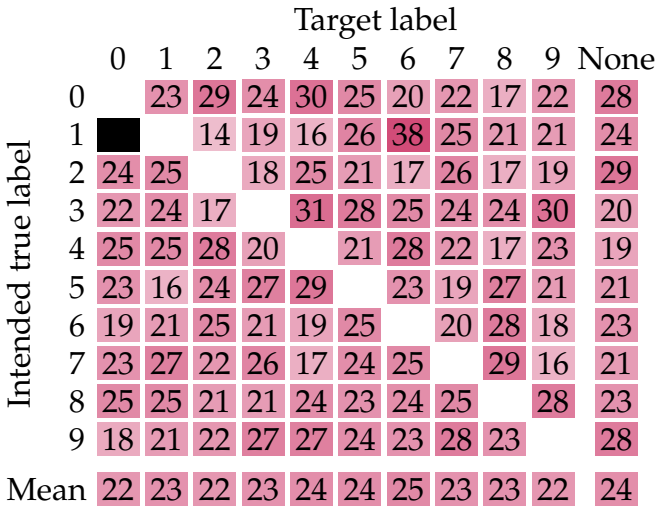
In this section, we investigate the contribution of different aspects of our test generation procedure by removing them one by one, and investigating the effect.

4.6.1 Generative model without finetuning

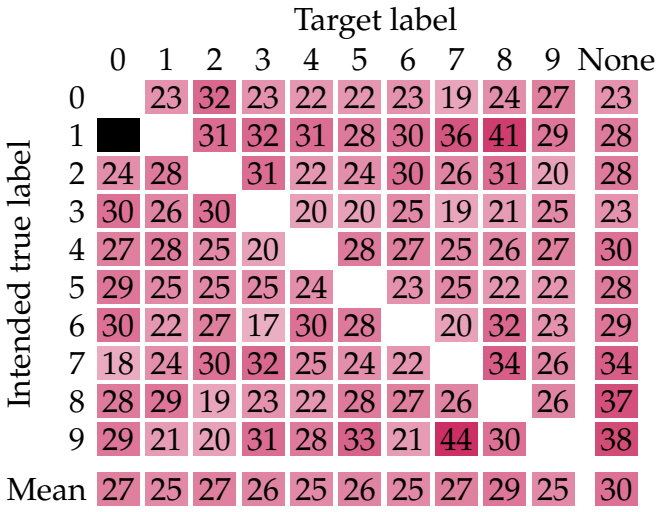
Method

To evaluate the extent to which our dual-objective finetuning procedure is effective, we repeat the main efficacy experiment described in Section 4.3 but using an out-of-the-box GAN that has not been finetuned using our dual-objective procedure at all.

That is, we use the fixed, pretrained generator to generate many 'test inputs', and filter to keep all those which are misclassified (the untargeted case) or misclassified with a particular label (the 'targeted' case, although note that of



(a) Tests targeting the Wong and Kolter robust network (described in Table 4.1).



(b) Tests targeting the standard convolutional network (described in Table 4.1).

Figure 4.13: The number of times out of 100 that generated test images are not identified as the 'odd one out' in a side-by-side comparison If the generated images always passed as training examples, the expected result would be 50; if they were always spotted, it would be 0. The colours represent the data visually.

course the not-finetuned generator has not been trained to target any particular class). We then use human judges to evaluate the proportion of these filtered examples which correctly maintain their semantics, a necessary condition to ensure that the examples really are misclassified. We also report the proportion of these filtered examples which are not correctly identified by human judges as being generated (out of a selection of ten). These results are shown in Figure 4.14.

For an illustrative example of our method here, consider the result for intended true label $y = 9$ and target label $t = 0$. We first use the conditional GAN to produce a set of images that are intended to be 9s. We then filter this set and keep only those that are classified as 0s by the classifier – a much smaller subset. Finally, we report below the percentage of these for which the true label determined by humans is indeed a 9: 55%.

We then use the same filtered generated examples to repeat the experiments from Section 4.5 – asking human judges to identify the one generated example alongside nine or one training dataset example. These results are shown in Figures 4.15 and 4.16.

Results and discussion

As expected, a generator without any of our finetuning is much less able to produce test cases that reveal a failure in the model being tested. Unsurprisingly, the proportion of generated examples that are misclassified broadly matches that of the hold-out test set – around 2% – which is what we should expect, given that the standard generator is optimised to match the training (and therefore test) distribution. In contrast, our dual-objective finetuning can easily increase the proportion of generated examples that potentially expose classification failures to well over 99%.

Furthermore, only 66% of the standard generator's misclassified outputs had maintained their true label in the untargeted case, compared with 80% for a dual-objective finetuned generator. In the targeted case, we see that the generator without finetuning does even worse: an average of 57% across the ten classes,

compared with 80% again in the finetuned case. The results are analogous for tests targeting the non-robust classifier. So in short, our procedure significantly increases the ability of generated outputs to actually expose classification failures.

In general, the outputs of the generator without finetuning similarly or more often pass to humans as training examples than the outputs of finetuned generators. This matches what we might expect because, without our finetuning, a standard generator is optimised solely to generate outputs that are indistinguishable from the training distribution. Curiously, the not-finetuned generator outputs pass as training examples particularly often in the side-by-side case targeting the pixel-robust classifier. This may be because adversarially trained classifiers tend to have decreased performance on the training distribution as a trade-off for their increased robustness; this could result here in more 'normal-looking' generated examples potentially inducing a misclassification.

In summary, although our dual-objective finetuning slightly increases the proportion of tests that humans can identify as being generated rather than training examples, it hugely increases the proportion of generator outputs that identify failures in the model being tested.

Intended true label, y \ Target label, t	0	1	2	3	4	5	6	7	8	9	None
0		73	91	62	62	91	77	88	50	59	75
1			49	51	80	06	31	47	61	77	57
2	13	62		53	32	30	30	52	50	19	60
3	29	69	60		26	60	12	79	22	28	71
4	42	55	66	43		70	80	70	48	73	65
5	18	46	55	61	54		29	59	31	51	60
6	50	60	80	87	74	80		74	54	59	66
7	22	43	82	63	31	64	00		62	38	40
8	70	68	80	75	75	91	63	50		80	79
9	55	66	88	74	88	92	66	87	69		88
Mean	37	60	72	63	58	65	43	67	50	54	66

(a) Testing the Wong and Kolter robust network (described in Table 4.1).

Intended true label, y \ Target label, t	0	1	2	3	4	5	6	7	8	9	None
0			76	44	65	70	77	89	56	75	76
1			82	89	95	99	93	76	98	98	98
2	35	45		94	55	40	55	75	75	48	68
3	64	66	66		46	71	33	73	81	84	80
4	66	41	66	68		56	66	61	59	85	77
5	64	68	80	76	71		69	63	65	82	73
6	82	63	66	43	91	79		78	81	60	85
7	73	47	82	80	72	92	100		92	86	77
8	87	69	81	77	79	65	88	62		81	85
9	90	45	79	75	76	96		82	74		84
Mean	70	56	75	72	72	74	73	73	76	78	80

(b) Testing the standard convolutional network (described in Table 4.1).

Figure 4.14: Using no finetuning at all (i.e., a standard generator), the proportion of filtered generated inputs for which humans judge the generated image to have the intended label, y. The filtered generated inputs, $g(z, y)$ have been filtered to remove the many examples for which the classifier prediction does not match the (incorrect) target label t. The colours represent the data visually. Can be directly compared with Figure 4.7.

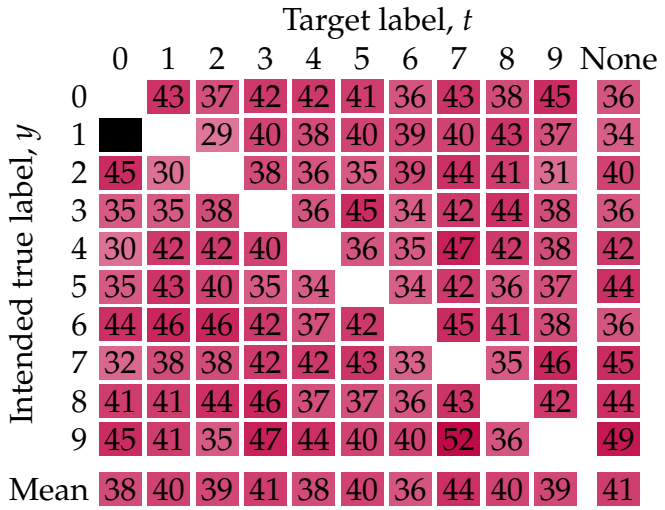
Intended true label, y \ Target label, t	0	1	2	3	4	5	6	7	8	9	None
0		48	63	53	53	62	56	46	40	37	54
1			34	52	50	37	27	52	47	46	42
2	42	39		51	39	30	36	54	49	40	43
3	41	47	55		42	55	47	53	41	47	51
4	49	52	41	46		45	50	51	45	66	57
5	38	50	43	56	47		54	44	51	48	51
6	49	52	47	50	46	55		38	62	46	48
7	39	57	59	36	49	45	32		41	57	59
8	52	50	59	52	56	57	43	39		51	59
9	51	51	66	53	74	61	53	73	48		68
Mean	45	50	52	50	51	50	44	50	47	49	53

(a) Tests targeting the Wong and Kolter robust network (described in Table 4.1).

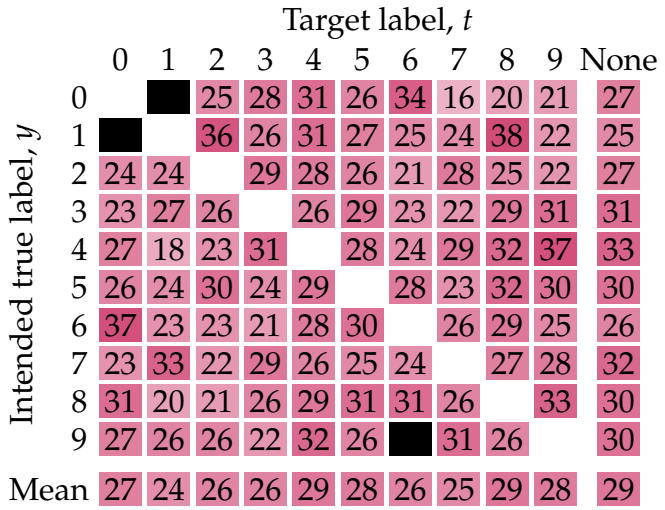
Intended true label, y \ Target label, t	0	1	2	3	4	5	6	7	8	9	None
0			59	56	61	65	63	55	51	54	63
1			56	62	76	76	70	72	80	77	74
2	53	55		66	54	48	50	62	64	52	61
3	68	54	62		48	64	52	64	71	65	71
4	57	54	55	57		63	52	60	47	66	73
5	69	64	58	62	57		73	53	60	62	63
6	63	64	58	59	67	71		62	63	63	67
7	54	63	71	61	71	62	52		67	74	77
8	65	60	60	71	57	60	81	55		69	67
9	64	51	65	64	81	71		76	68		68
Mean	62	58	60	62	64	64	62	62	63	65	68

(b) Tests targeting the standard convolutional network (described in Table 4.1).

Figure 4.15: Using no finetuning at all (i.e., a standard generator), the number of times out of 100 that the filtered successful generated 'tests' are not identified as the 'odd one out' in a set of ten. If the generated images always passed as training examples, the expected result would be 90; if they were always spotted, it would be 0. The colours represent the data visually. Can be directly compared with Figure 4.12.



(a) Tests targeting the Wong and Kolter robust network (described in Table 4.1).



(b) Tests targeting the standard convolutional network (described in Table 4.1).

Figure 4.16: Using no finetuning at all (i.e., a standard generator), the number of times out of 100 that the filtered successful generated 'tests' are not identified as the 'odd one out' in a side-by-side comparison. If the generated images always passed as training examples, the expected result would be 50; if they were always spotted, it would be 0. The colours represent the data visually. Can be directly compared with Figure 4.13.

4.6.2 Training strategies

We investigate the effect of the training strategies described in Section 4.1.3 by removing each in turn. We did not have the resources to prioritise further human experiments on these more minor ablations, but a visual comparison is sufficient to assure us of their effects. Figure 4.17 provides a point of comparison, showing the progression of the finetuning phase of our procedure, applied with no ablations.

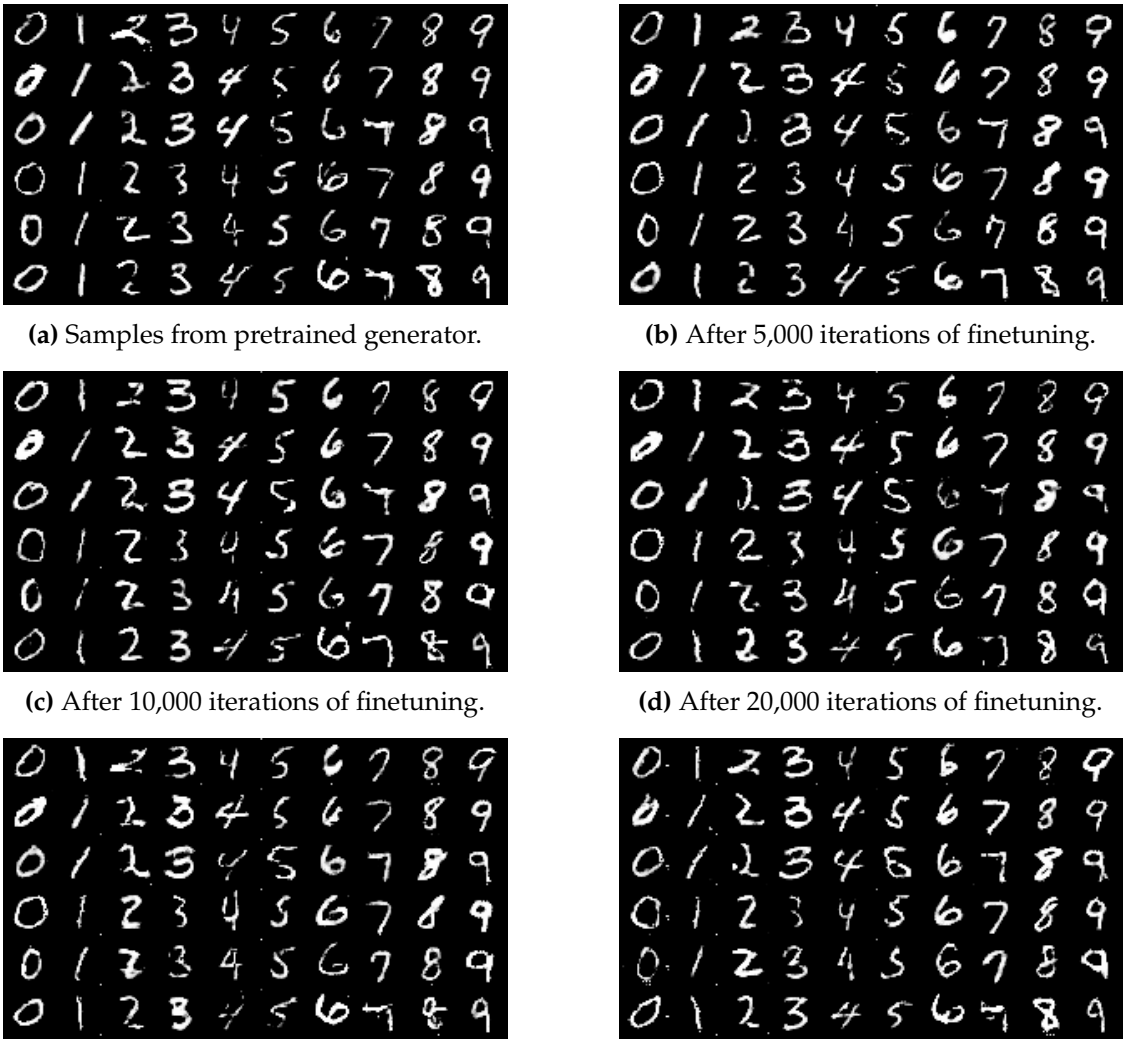
We find that without a standard pretraining phase, training with our dual-objective function does eventually converge, but it takes longer and gives generated results that appear to be less similar to the training distribution and more prone to not retaining their intended labels. See Figure 4.18 for the development of outputs over training.

We also find that the finetuning rate hyperparameter is an effective technique for trading off test generation efficacy with image realism; without this technique, the finetuning rate is in effect set to 1, which again produces results that appear to be inferior. See Figure 4.19 for a comparison of the effect of different settings of μ.

Last, we find that choosing $l_{ordinary} + l_{(un)targeted}$ as the generator's loss results in a catastrophic collapse of the training, likely because $l_{(un)targeted}$ is optimised for too heavily. It is possible that manipulating other hyperparameters could cause this naïve loss to work, but GAN training convergence is not easy to obtain, and we are confident that our loss term contributes to convergence. See Figure 4.20.

4.6.3 Perturbing standard GAN outputs

Finally, we compare to a naïve baseline where test inputs are straightforwardly generated using a standard generative network and then their pixel values perturbed using the standard projected gradient descent algorithm [88] with an ℓ_∞ norm bound of $\epsilon = 0.1$. Testing the W&K network used throughout this chapter, under 1% of correctly classified generated data result in a classification failure after perturbation; we find this approach is no better than the same pixel



(a) Samples from pretrained generator.

(b) After 5,000 iterations of finetuning.

(c) After 10,000 iterations of finetuning.

(d) After 20,000 iterations of finetuning.

(e) After 30,000 iterations of finetuning.

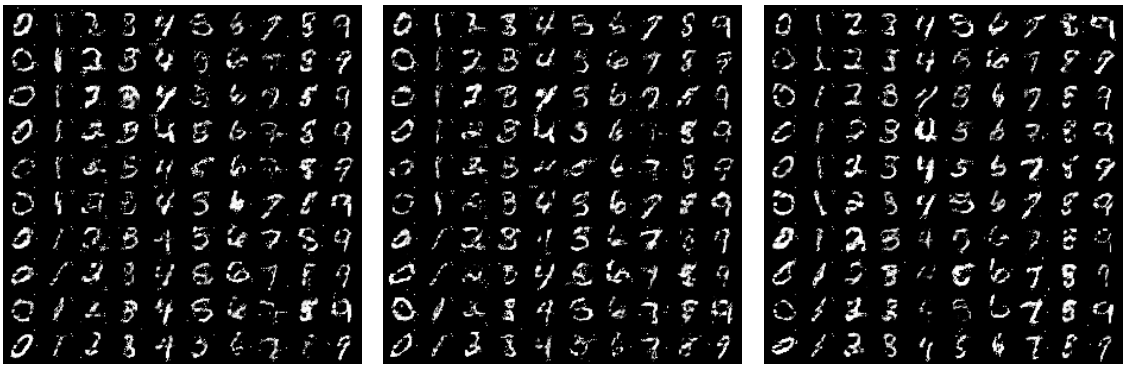
(f) The end of finetuning at 45,000 iterations.

Figure 4.17: A sequence of images tracking the output of the generator network for one fixed random sample in latent space as adversarial finetuning takes place. Five samples are given for each intended true label. The finetuning is for untargeted tests for Wong and Kolter's provably-robust network.

perturbations on the hold-out test set. This is as expected, since the generator models the training data distribution, which the test set is also drawn from.

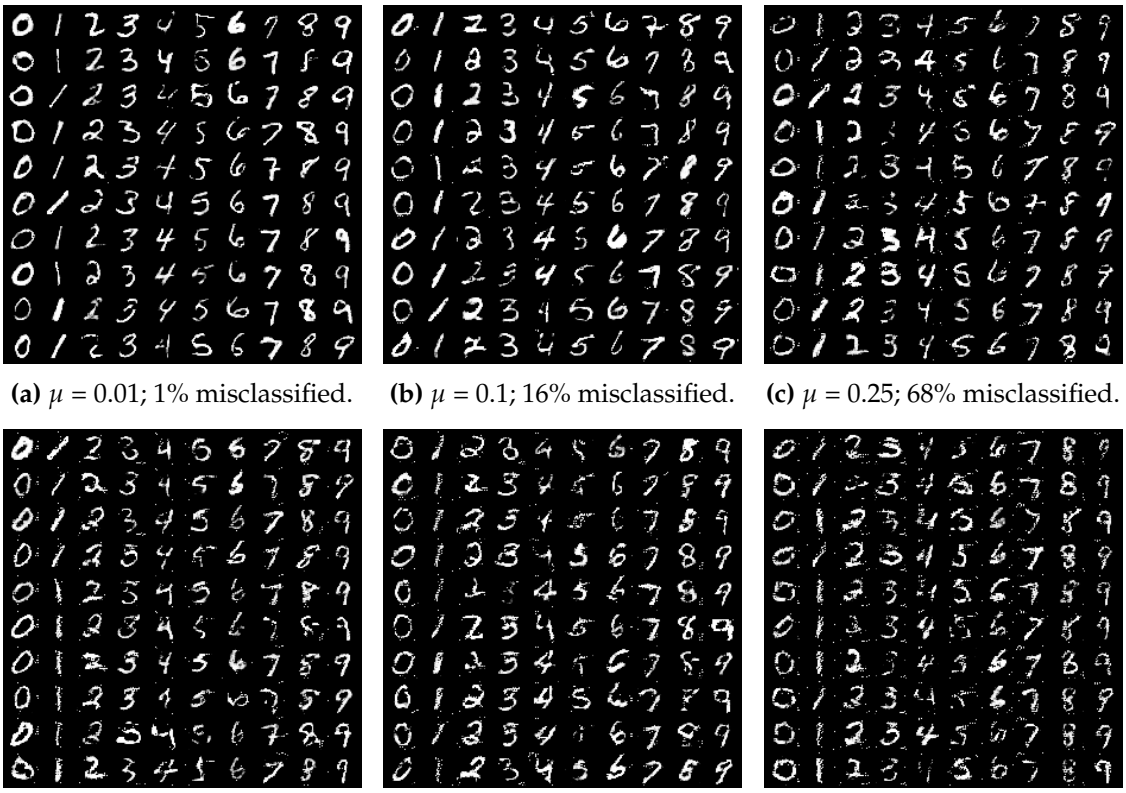
4.7 Scaling to ImageNet

To investigate the scalability of our procedure, we apply it to the much larger and more complex ImageNet-1K dataset [50], with three colour channels rather than one, 128 $\times$ 128 pixels rather than 28 $\times$ 28, and 1000 different class labels



(a) After 705,000 iterations. **(b)** After 750,000 iterations. **(c)** After 2.53 million iterations.

Figure 4.18: Demonstration of the effect of using omitting the standard pretraining phase of our procedure before optimising for both objectives simultaneously. Results are shown after the numbers of iterations equivalent to pretraining, to pretraining plus finetuning, and at convergence. The final results both took longer and are visually less convincing than otherwise comparable results with standard pretraining in Figure 4.17.



(a) $\mu = 0.01$; 1% misclassified. **(b)** $\mu = 0.1$; 16% misclassified. **(c)** $\mu = 0.25$; 68% misclassified.

(d) $\mu = 0.5$; 94% misclassified. **(e)** $\mu = 0.75$; 98% misclassified. **(f)** $\mu = 1$; >99% misclassified.

Figure 4.19: The effect of finetuning rate μ on image quality and proportion misclassified, using otherwise the same setup as in Figure 4.17. A finetuning rate of 1 is equivalent to not having the finetuning rate as an option. As expected, finetuning rates control the balance between the quality of the generated images and the propensity of the generated outputs to cause model misclassifications (ignoring whether the examples retain their intended semantics).

(a) After 500 dual-objective finetuning steps; 40% misclassified.

(b) After 1500 dual-objective finetuning steps; 91% 'misclassified'.

(c) After 10,000 dual-objective finetuning steps; all 'misclassified'.

Figure 4.20: Dual-objective finetuning with naïve generator loss $l_{ordinary} + l_{untargeted}$. As expected, our custom loss function as described in Section 4.1.3 significantly improves convergence to generator weights that continue to generate images realistic enough to maintain their intended classes.

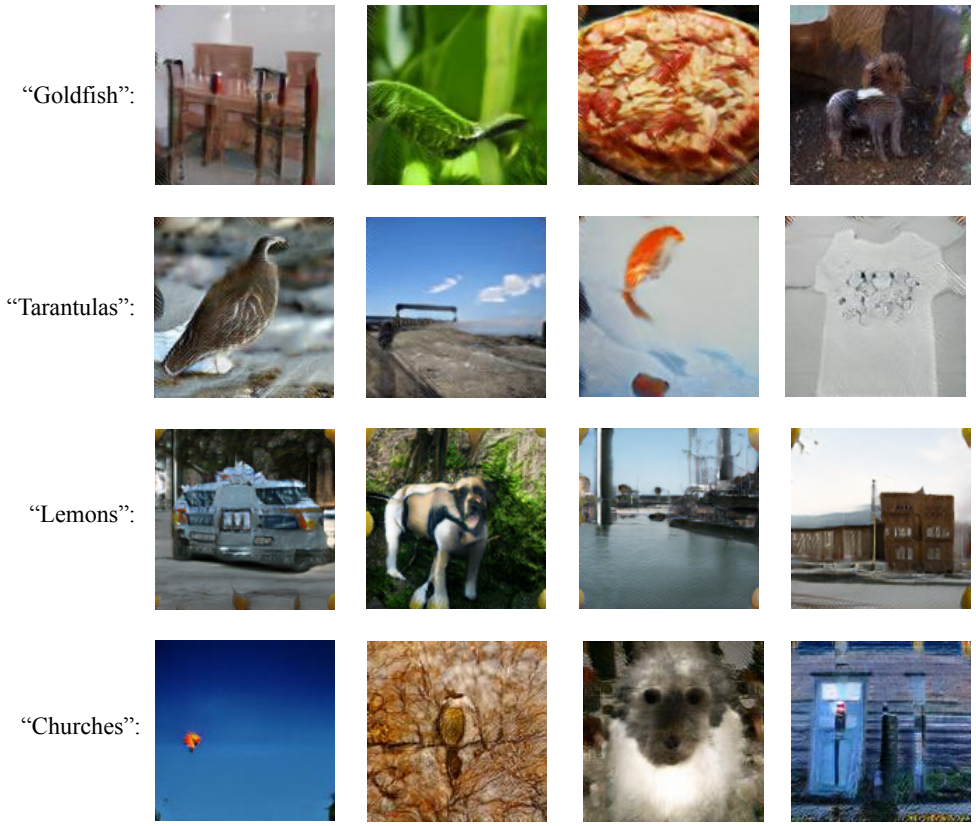


Figure 4.21: Some hand-selected examples of targeted ImageNet test cases, generated by a BigGAN finetuned using our procedure. As discussed below, most generated examples are much less legible, likely simply because of the original generator quality. The target class for each row is labelled. Observe the appearance of lemon-like objects at the edges of the lemon-targeted row – although this does not affect the meaning a human would assign, it presumably contributes toward the classification model's lemon prediction.

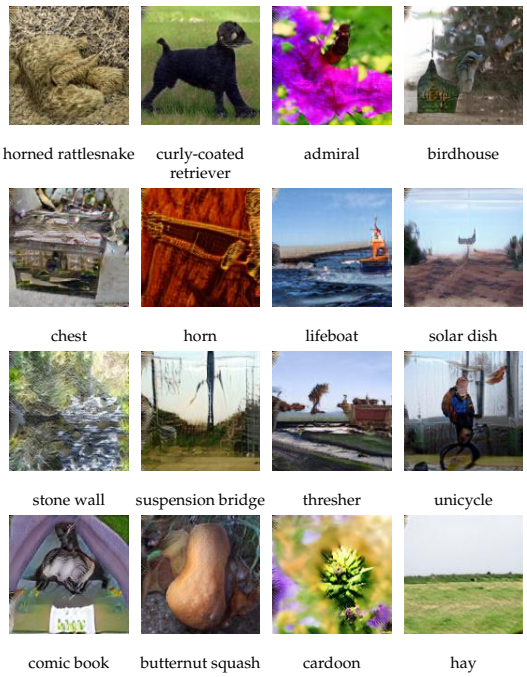


Figure 4.22: Successful targeted unrestricted adversarial examples for target class 'tabby cat'.

Figure 4.23: Successful targeted unrestricted adversarial examples for target class 'slug'.

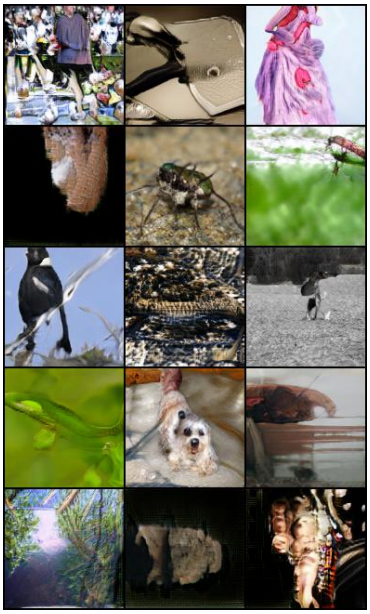
Figure 4.24: Successful targeted unrestricted adversarial examples for target class 'orange'.

Figure 4.25: Successful targeted unrestricted adversarial examples for target class 'church'.

rather than 10. The generator network used is the author's 'officially unofficial' published code and checkpoints for the current state-of-the-art, BigGAN [34], chosen because it is the best GAN checkpoint published with both generator and discriminator. Note that the official BigGAN release included much higher checkpoints (including one used in subsequent chapters), but these included the generator networks only, excluded the discriminators necessary for our procedure. The model tested is a ResNet-152 [205], the highest accuracy ImageNet classifier packaged in PyTorch at the time.

In the untargeted case, our procedure is able to finetune this BigGAN to generate examples such that the classification model's predictions are incorrect >99% of the time within 40 gradient steps (compared with the 10^5 taken to train BigGAN). This much smaller number of steps required may be because the greater resolution of the images provides more degrees of freedom that the optimisation can exploit to achieve its goals.

Our main focus, though, is on the more challenging targeted case – causing an image to be classified as the one-in-a-thousand specific target class is a much smaller target than allowing any of the 999 incorrect classes. We find that typically, on the order of 100 gradient steps were required for >10% of generated examples to be classified (top-1) as the target class. Compared to MNIST, each ImageNet gradient step takes about 100x longer to compute due to the sheer size of the models, but the 100x decrease in the number of gradient steps required compared to MNIST roughly compensates for this, resulting in a similar compute time overall to obtain a reasonable fraction of test cases that cause the misclassifications as desired. Image quality as measured by the Inception Score metric [206] typically decreased from about 75 to 70 as a result of this finetuning using our procedure. This remains slightly better than the mid-2018 state-of-the-art Inception Score of 52 [32] or the mid-2017 state-of-the-art of 12 using WGAN-GP [207]. Figure 4.21 shows some selected examples of generated targeted test cases; Figures 4.22 to 4.25 show random examples.



(a) No dual-objective finetuning: ordinary training continues.

(b) With 15 gradient steps of dual-objective finetuning.

Figure 4.26: Samples for fixed inputs to the BigGAN implementation we use, taken 15 gradient steps after the checkpoint we begin training from.

Comparison to no finetuning

The ImageNet results above are not as high-quality as those reported in the BigGAN paper [34]. There are two causes of this which need to be disentangled: the effect of finetuning, and the limitations of the particular BigGAN instance used. One limitation is that the BigGAN checkpoint we use has a worse generated image quality. The Inception Score is much lower than is reported in the paper (75 vs. 166.5), and the resolution is much less than the best images (128×128 vs. 512×512). Another is that our limited access to expensive hardware forces us to use a batch size of 15; small batch sizes like this can "lead to inaccurate estimation of the batch statistics, and reducing batch normalisation's batch size increases the model error dramatically" [208, p.1].

We therefore provide samples from the BigGAN running on our machine to identify how detrimental our procedure is to image quality relative to standard training. Figure 4.26a shows a set of samples taken after continuing ordinary training; Figure 4.26b shows the output of the generator for the same input after

dual-objective finetuning, instead.

A visual inspection shows that the samples in Figure 4.26a are similar in quality to those in Figure 4.26b (and the other ImageNet results presented). Indeed, it is often quite unclear what the images are supposed to be portraying, making it a challenge to identify compare which of the two is a more faithful depiction. This suggests that a sufficient reason for the relatively poor images generated as ImageNet tests by our procedure is the relatively poor quality of the generator network used. Unfortunately, this was the best network available at the time.

Conclusion

So in short, while our method appears able to generate test cases that induce failures at ImageNet scale without a large increase in computation required relative to the MNIST case (due to the fewer gradient steps required), the relatively poor quality of the GAN checkpoints available (and perhaps hardware constraints forcing small batch sizes) hinders our investigation of whether our finetuning procedure is able to scale its ability to generate test inputs that maintain their semantics and relevance.

4.8 Threats to validity

This section discusses possible threats to the validity of the results in this chapter. This includes threats to internal validity (whether the results as presented can be trusted) and threats to external validity (whether the results presented can be expected to generalise beyond the specific setup here.)

4.8.1 Internal validity

MTurk experiments

The evaluation of our procedure relies entirely on the quality of the data provided by the MTurk workers. As this comprises our main threat to internal validity, we therefore took a number of measures to safeguard the quality of this data,

especially to ensure that participants understood instructions and completed the tasks diligently:

- Only workers with good track records were permitted to participate.
- The instructions specified that particular answers should be given to specified questions to prove that the instructions had been read carefully. Approximately 10% of work was rejected for failing this check.
- For the image labelling tasks, some images with known labels were included to check that the right labels were being given. Reassuringly, almost no work was rejected for failing this check.
- For the identification of the generated images, a bonus nearly doubling the pay per image was given for each correctly-identified image, providing an extra incentive to try hard.
- To provide a disincentive to high-speed random clicking, a minimum time spent answering each individual question was enforced by temporarily disabling the answer input section until a short timer expired.
- If more than 1% of questions were left unanswered, we interpreted this as a sign of carelessness and did not use any of the data from that particular task.

Full screenshots of the interfaces used for these experiments, including the instructions, are available in Appendix B.2.

In all, we are confident that the data are at least reliable enough that our results have internal validity.

Training technique ablative studies

The ablative studies in Section 4.6.2 ablate different parts of the training procedure in order to identify how helpful they are in ensuring satisfactory convergence of the dual-objective finetuning. A possible minor threat to validity is that

these experiments were not particularly rigorous: although they provide clear evidence of the effect of each training technique, they do not comprehensively evaluate the extent to which each makes a contribution. For instance, single runs of each relevant ablated training procedure are used. Many runs would increase the robustness of the conclusions drawn in this section, but this was not prioritised. But this is acceptable: the key results of this chapter hinge on the ability of our procedure to produce results assuming convergence; we have clearly demonstrated that convergence is possible; convergence is likely dependent on machine learning procedural knowledge; and so these training techniques are more like beneficial extra contributions, rather than an essential part of our test generation algorithm.

ImageNet generator quality

As discussed in Section 4.7, the quality of the generator used for the ImageNet experiments hinders our ability to conclude that our test generation procedure scales without problem to this dataset, although there is evidence that indicates that this is so.

4.8.2 External validity

Our empirical evaluation in this chapter has centred primarily on the MNIST dataset, with some use of the ImageNet dataset to investigate scaling. This begs the question of the external validity of this work: whether the conclusions about our test generation procedure will generalise to other image classification tasks, and indeed to other domains.

Recall that MNIST being a simple dataset that is easy to classify is precisely a reason to expect external validity, rather than to question it. Because MNIST classifiers are among the highest performance on any dataset, we should expect that generating tests that cause failures in these classifiers is particularly difficult. So we should expect our test generation procedure to succeed quite easily on other image classification tasks, because they are likely to be more difficult than

MNIST and so the classifiers are very likely to have more failure modes that can be identified.

An open question is whether our test generation procedure would generalise beyond image classification to other domains. Common sense suggests that it would generalise, assuming that GAN training is possible for that domain, because there is nothing specific to images baked into our algorithm. Likewise, there is little specific to classification – the loss term representing classification accuracy could easily be replaced with another performance measure for a different task such as regression. But it is also reasonable to take the view that there could be an unexpected barrier in generalising to some other domain, and so we cannot assume this kind of external validity. Fortunately, we do not depend on this kind of external validity for the significance of our contribution.

4.9 Performance on requirements

This chapter has presented a new procedure for the generation of test cases for discriminative deep neural networks. By finetuning a generative network using an additional loss term, it can be trained to directly generate useful test cases. But does it meet the requirements we set out in Section 1.2.3? To conclude, let us consider each requirement in turn in light of the evidence presented in this chapter. Note that not all sections of the chapter – such as the ablative studies – are directly relevant here.

4.9.1 Production of meaningful test cases

Two types of experiments were conducted to verify that the generated test inputs were meaningful according to the oracle (human judgement).

First, human judges were asked to identify which handwritten numeral, if any, each generated test input corresponded to. For 90% of untargeted test inputs for a standard MNIST classifier, human judges assigned the label that matched the intended meaning for that test. Broadly similar results were found for other

classifiers and for targeted tests. So the overwhelming majority of generated tests are meaningful according to the task oracle.

Second, human judges were asked to identify which one example from a selection of ten was a generated test case, as opposed to an example drawn from the training set. 50% of the time, the judges were unable to correctly identify which example was the generated test in the case of untargeted tests for a standard classifier – again, similar results were found for other classifiers and for targeted tests. This implies a stronger property: not only were the generated tests clearly meaningful according to the task oracle, but they were often visually indistinguishable from the training data. This is sufficient both for the generated tests to be meaningful under the test oracle, and for failures identified by these tests to be of practical concern.

4.9.2 Production of test cases that induce poor performance

After the training procedure, it is easy for over 99% of the outputs of the generator network to be classified as intended (i.e., as something other than the intended label y) by the system being evaluated. The fewer than 1% that do not meet this criterion can easily be automatically filtered out.

The remaining check is of the proportion of generated tests for which the test input really does belong to the intended target class t, according to the oracle. As described in Section 4.9.1, this is around 80 to 90%. So the overwhelming majority of generated tests induce incorrect behaviour in the evaluated image classifier.

4.9.3 Detection of a wide range of problems

Whereas existing work takes a perturbation approach to solve the test oracle problem, only creating test inputs that are small deviations from known labelled examples, our new approach does not have any such constraints, instead solving the oracle problem by using a conditional generative network. Because of this lack of constraints on its possible outputs, our test generation procedure is able to detect many more failures than existing procedures.

There are four sources of empirical evidence that the tests generated are able in practice to detect problems not detectable by existing approaches. First, a straightforward check that generated tests are not within typical perturbation constraints of any dataset examples. Second, that the algorithm is able to detect problems with almost as much ease on classification models trained to be robust to pixel perturbations. Third, tests for one classification model tend not to also induce incorrect behaviour in other classification models, implying that in these cases, the tests are detecting failure modes that are distinct to each model being tested. Fourth, in contrast with prior perturbation-based approaches, even if the model under test is repeatedly further (adversarially) trained using the generated test inputs, our algorithm remains able to generate new test cases that reveal ways that the classifier fails to generalise.

4.9.4 Efficiency and practicality

The test generation procedure requires the training of a GAN pair. Ideally, such a GAN will be already available for the relevant task, in which case a relatively inexpensive finetuning process is all that is needed, before tests can be generated at the small cost of one forward pass through the generator per batch of tests. If such a GAN is not available, then one needs to be trained, which requires relevant data and sufficient compute (which can be expensive for larger datasets).

To verify whether the test generation procedure works at scale, it was applied at scale on the ImageNet dataset, with somewhat ambiguous results: it is unclear whether the generated test cases tended to be not particularly recognisable as their intended classes because the pretrained generator used was low quality, because hardware constraints forced small batch sizes and few gradient steps, or because the test generation algorithm does not scale well. We have seen some evidence that it is likely to be the generator checkpoint that is the main problem, but this still prevents our reaching a firm conclusion that the algorithm scales well.

5

Generating Tests by Perturbing Generative Network Activations

Contents

The main contribution of this thesis is two new test generation procedures that evaluate image classifiers' ability to generalise well, not relying on the usually false assumption that the specific distribution of inputs encountered during their training is comprehensive and representative. This chapter introduces the second of these two new algorithms, which identifies context-sensitive feature perturbations to test inputs (affecting for instance object shape, orientation, location, texture, and colour). These changes are found by performing small adjustments to the activation values of different layers of a trained generative neural network. Because this approach can perturb the activations at every layer, it is able to exploit the full range of learned representations to access a rich space of possible feature changes. Perturbing at layers earlier in the generator

causes changes to high-level, coarser-grained features such as object shape, location, colour or orientation; perturbations further on cause finer-grained, local changes such as to texture. Because this approach is able to adjust a richer space of feature changes than previous works, it can detect a wider range of generalisation failures.

The full procedure is described in Section 5.1; the rest of the chapter describes and discusses an empirical evaluation of the approach. Chapter 6 evaluates whether the procedure introduced in this chapter is able to detect faults in deep neural networks that were not detectable using existing approaches. And Chapter 7 exploits the procedure introduced in this chapter to make a surprising empirical finding: networks trained to be more robust to pixel-level perturbations are *less* robust to the kinds of high-level feature perturbations encoded in the earlier layers of generative networks.

5.1 Procedure for perturbing latent generator activations

In short, we make context-sensitive changes to image features by taking a pre-trained generative neural network and perturbing its latent activation values as it generates images. We expect this to work because the neurons at different layers of a generator encode the various useful features for generating images [60]. We can control the granularity of the downstream change by selecting the layers at which the activations are perturbed: perturbations to earlier layers result in coarser-grained changes (e.g., the shape of a building), while later perturbations result in finer-grained changes (e.g., the texture of some fur). In this way, we are able to probe generalisation to the kinds of change that occur in the data.

5.1.1 Problem setup

Recall that image classification is the approximation of an oracle $o : \mathbb{X} \rightharpoonup \mathbb{Y}$ that assigns each meaningful image x its correct (by definition) class, $o(x) \in \mathbb{Y}$. Given a set of N labelled datapoints $D = (x_i, o(x_i))_{i=1}^{N} \subset \mathbb{X} \times \mathbb{Y}$, we can train a

deep neural network image classifier $f\colon \mathbb{X} \to \mathbb{R}^{|\mathbb{Y}|}$ that attempts to approximate o. Given an input, f outputs a real-valued confidence for each possible class $y \in \mathbb{Y}$. Let $f_y(x)$ be the classifier's confidence of input x being of class y, and $f_{pred}(x) = \arg\max_y f_y(x)$, such that f_{pred} is an approximation of o. Typically, the final layer of a DNN is a softmax function, so that for all output confidences $f_y(x)$, $0 \leq f_y(x) \leq 1$, and $\Sigma_{y=1}^{k} f_y(x) = 1$.

When testing such a trained DNN, our goal is to identify test cases (x, y) that cause failures in the model, where a failure means that the output is incorrect: $f_{pred}(x) \neq y = o(x)$. These failures are indicative of a fault in the DNN.

Definition 5.1. A test case with test input $x \in \mathbb{X}$ for DNN $f\colon \mathbb{X} \to \mathbb{R}^{|\mathbb{Y}|}$ is said to *fail* if $f_{pred}(x) \neq o(x)$.

However, we quickly run into the test oracle problem: we do not have direct access to o (if we did, there would be no need to train an approximation f) and it is too costly to seek human labelling for each test input. So a practical test generation algorithm needs to provide not only test inputs x, but additionally the desired system output $o(x)$ so that failing test cases can be identified.

5.1.2 Solving the test oracle problem using perturbations

The standard approach to solving the test oracle problem is to make use of the set D of labelled data that is available. We can partition D into a large training set D_{train} and a small holdout test set D_{test}; by using only D_{train} during training, we can be confident that the DNN has not overfitted to any of the examples in D_{test}, so these examples can be used during testing. For a test case $(x_{test}, o(x_{test})) \in D_{test}$, any new input $x_{new} \in \mathbb{X}$ that we can be confident shares the same desired output as x_{test} can therefore be used as a new test case, because we simply assume that $o(x_{new}) = o(x_{test})$.

To identify such x_{new} that share a label with a known test case, most procedures begin by choosing a perturbation function $t\colon \mathbb{X} \times \mathbb{P} \to \mathbb{X}$ with parameter space $\mathbb{P}$. The intention is that, given a labelled test input x_{test}, this function is able to

generate new test inputs as its parameter varies: $x_{new} = t(x_{test}, p)$. But to be confident that x_{new} is similar enough to x_{test} to have the same true label, we must also introduce a similarity constraint over the parameter p.

Definition 5.2. A *similarity constraint* d for a perturbation function t is a function $d\colon \mathbb{P} \to \{\top, \bot\}$, such that for all $x \in \mathbb{X}$, if $d(p) = \top$ then $o(t(x, p)) = o(x)$.

If we have a suitable perturbation function and similarity constraint, then the problem of identifying test inputs reduces to a search for suitable parameter values p:

Proposition 1. *If we have a labelled test case* $(x_{test}, o(x_{test})) \in D_{test}$*, a perturbation function* $t\colon \mathbb{X} \times \mathbb{P} \to \mathbb{X}$*, a similarity constraint* $d\colon \mathbb{P} \to \{\top, \bot\}$*, and a parameter* $p \in \mathbb{P}$*, and if* $d(p) = \top$ *and* $f_{pred}(t(x_{test}, p)) \neq o(x_{test})$*, then* $t(x_{test}, p)$ *is a failing test case.*

Proof. Since d is a similarity constraint for t and $d(p) = \top$, $o(t(x, p)) = o(x)$ for any $x \in \mathbb{X}$. In particular, $o(t(x_{test}, p)) = o(x_{test})$. But $f_{pred}(t(x_{test}, p)) \neq o(x_{test})$ by assumption, so $f_{pred}(t(x_{test}, p)) \neq o(t(x_{test}, p))$, so $t(x_{test}, p)$ is a failing test case. □

Test generation using pixel-space perturbations

Most existing algorithms that generate tests for DNNs use a pixel-space perturbation approach. In our framework, we have that $\mathbb{P} = \mathbb{X}$ and $t(x, p) = x + p$. In effect, t simply changes each pixel value in the image independently. The similarity constraint used typically constrains the magnitude of p: if the pixel values do not change too much, then the label should remain the same. This magnitude is typically measured using the ℓ_∞ or ℓ_2 norm metric. That is, $d(p) = \|p\|_2 \leq \epsilon$ or $d(p) = \|p\|_\infty \leq \epsilon$, for a manually chosen constant ϵ small enough that the change is nearly imperceptible.

Given a labelled test case $(x_{test}, o(x_{test})) \in D_{test}$, then, a pixel perturbation algorithm must find a suitable p. This is almost always done using an optimisation over p, using a loss function chosen to be minimised when $f_{pred}(t(x_{test}), p) \neq$

$o(x_{test})$ and $d(p) = \top$. Since DNNs are differentiable, the derivative of the loss function with respect to p can be computed, and this gradient can be walked to minimise the loss and thereby identify a failing test case. If additional properties are desired of the test cases, this can be reflected in the choice of loss function.

5.1.3 Using GANs to perturb images

The approach we present here, as existing approaches, uses a perturbation-based approach to solve the test oracle problem. However, rather than directly perturbing the pixels of a labelled test dataset image, there are two differences:

1. Instead of beginning with a labelled test dataset image, we use a conditional generative network to *generate* a fresh test seed for which we know the correct label.

2. Instead of perturbing the individual pixel values of this seed, we make perturbations to meaningful features of the input by exploiting the generative network's learnt features. This second difference is the more important.

Generating test seeds

Suppose we have a pretrained conditional generator network $g\colon \mathbb{Z} \times \mathbb{Y} \to \mathbb{X}$, which as described in section 2.1.2, takes a normally distributed $z \in \mathbb{Z} = \mathbb{R}^m$ and a class label $y \in \mathbb{Y}$ and returns an image $g(z, y) \in \mathbb{X} = \mathbb{R}^{c \times w \times h}$ from the training image distribution such that $o(g(z, y)) = y$. If a conditional generator network is not available, we can instead make the assumption that the output of the classifier being tested is initially correct. In either case, rather than relying on the finite examples in a test dataset as our source of seeds from which to create test cases, we can generate fresh labelled test seeds on demand, by sampling new generator inputs z.

While this may be valuable in itself, our main intention in using generated images is that it allows much greater control over the test inputs we create.

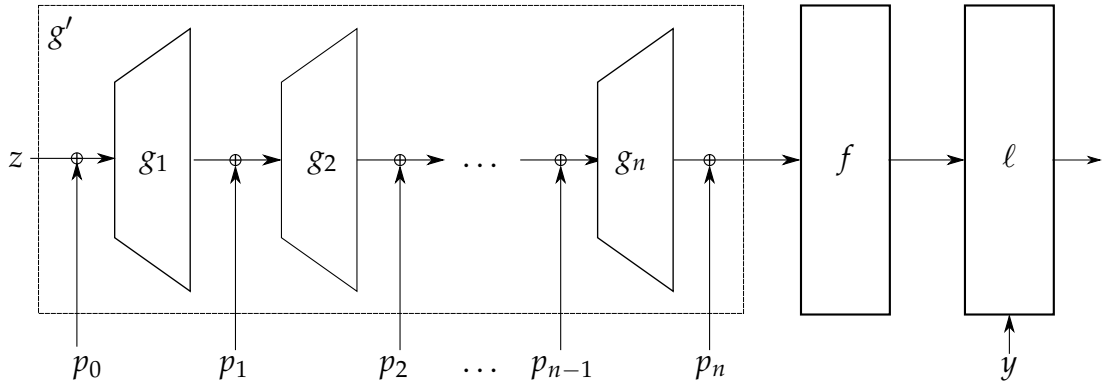


Figure 5.1: Illustration of a forward pass with perturbations to the latent activation values at n layers in the generator network.

Making perturbations

Since a feedforward neural network is a sequence of n discrete layers, each of which operates solely on the output of its predecessor, we can consider the generator g to be a composition of n functions. For instance, we can decompose BigGAN [34] into its residual blocks. We can write the ith layer as a function $g_i \colon \mathbb{A}_{i-1} \to \mathbb{A}_i$, taking activations $a_{i-1} \in \mathbb{A}_{i-1}$ from the previous layer and outputting the resulting activation tensor $g_i(a_{i-1}) \in \mathbb{A}_i$. Using $\mathbb{A}_0 = \mathbb{Z} \times \mathbb{Y}$ and $\mathbb{A}_n = \mathbb{R}^{c \times w \times h}$ for convenience, we can write $g \colon \mathbb{Z} \times \mathbb{Y} \to \mathbb{X}$ as the function composition $g = g_n \circ g_{n-1} \circ \ldots \circ g_1$.

The thrust of our procedure is to introduce a perturbation function that perturbs high-level features rather than individual pixel values. Since the neurons in a generative network encode meaningful features [60], we perturb the activations of these neurons so as to adjust the features of the generated image in a context-sensitive way. Decomposing the generator network g into a sequence of layers operating on latent activation spaces $\mathbb{A}_i$ allows us to do exactly that. For example, we can introduce a perturbation $p_i \in \mathbb{A}_i$ to layer i's activations, before continuing the forward pass through the rest of the generator. See figure 5.2 for an illustration of how a perturbation at layer i fits into the computation of a forward pass of the whole generator network.

Given a perturbation tensor $p_i \in \mathbb{A}_i$ for each layer i, we can define perturbed layer functions $g'_i(a_{i-1}) = g_i(a_{i-1}) + p_i$. By performing such perturba-

Original	**Perturbed**	**Perturbed at all layers**
$a_1 = g_1(z, y)$	$a_1 = g_1(z, y)$	$a_1 = g'_1(z + p_0, y) = g_1(z + p_0, y) + p_1$
$a_2 = g_2(a_1)$	$a_2 = g_2(a_1)$	$a_2 = g'_2(a_1) = g_2(a_1) + p_2$
...	...	...
$a_i = g_i(a_{i-1})$	$a_i = g_i(a_{i-1}) + p_i$	$a_i = g'_i(a_{i-1}) = g_i(a_{i-1}) + p_i$
$a_{i+1} = g_{i+1}(a_i)$	$a_{i+1} = g_{i+1}(a_i)$	$a_i = g'_{i+1}(a_i) = g_{i+1}(a_i) + p_{i+1}$
...	...	...
$a_n = g_n(a_{n-1})$	$a_n = g_n(a_{n-1})$	$a_n = g'_n(a_{n-1}) = g_n(a_{n-1}) + p_n$

Figure 5.2: Clarification of where the perturbation tensors p_i are applied during the forward computation of the generative model. The first column shows the standard forward pass of the generator. The middle column shows a perturbation applied at a single latent activation layer, i. The last column shows a forward computation where a perturbation is applied at all possible latent spaces.

tions at every activation space, we obtain the perturbed output of the entire generator, $g'(z, y;\ p_0, ..., p_n) = (g'_n \circ g'_{n-1} \circ ... \circ g'_1)(z + p_0, y)$. This is again illustrated in figure 5.2.

Since the neurons in a generative network encode meaningful features [60], we perturb the activations of these neurons so as to adjust the features of the generated image in a context-sensitive way. That is, rather than using a perturbation parameter space $\mathbb{P} = \mathbb{X}$, our parameter space allows adjustments at the output of multiple layers in the generator: $\mathbb{P} = \mathbb{A}_0 \times \mathbb{A}_1 \times \ldots \times \mathbb{A}_n$.

Then we define our perturbation function

$$t(g(z, y), p) = (g'_n \circ g'_{n-1} \circ ... \circ g'_1)(z, y, p), \tag{5.1}$$

where $g'_i(a_{i-1}, p_{i-1}) = g_i(a_{i-1}) + p_{i-1}$. That is, at each layer in the generator, t simply performs element-wise addition of the parameter tensor with the layer output.

Finding a suitable perturbation

Suppose that we have sampled some generator input z, and picked some label y for the unperturbed example, $g'(z, y;\ 0, ..., 0)$. We now need a procedure to

identify suitable perturbation tensors $\boldsymbol{p}^* = (p_0^*, ..., p_n^*) \in \mathbb{A}_0 \times \ldots \times \mathbb{A}_n$ such that the classifier's output on $g'(z;\ \boldsymbol{p}^*)$ is not y but some other label, while the true label of $g'(z;\ \boldsymbol{p}^*)$ remains y. Let us introduce a similarity constraint to ensure that the true label remains y. Our similarity constraint measures the total size of the changes being made to the activations: $d(p) = \|\bar{p}\|_2 < \epsilon$, where $\bar{p}$ is the one-dimensional vector formed by flattening all the elements of $p_0, p_1, \ldots, p_n$ into a list.

Now, as per Proposition 1, finding a value $p \in \mathbb{P}$ that satisfied the similarity constraint while also changing the classifier output suffices to produce a failing test case for the DNN under test, f. We use a gradient-walking optimisation to find such satisfactory values of p. In particular, we introduce a loss function $L(p) = \max_y f_{pred}(t(g(z, y), p))_y$, which penalises confidence in the DNN's top output class. Assuming that the classifier is initially correct, minimising this loss makes the classifier make a different prediction. Note that the usual back-propagation algorithm is sufficient to compute gradients of L with respect to p since f, t and each layer of g are differentiable.

To ensure that the label of the perturbed output remains y, we want to satisfy the similarity constraint. It is also possible to include a penalty on the perturbation magnitude $\|\bar{p}\|_2$, or to gradually relax a strict upper bound on it, but in practice we found this to be unnecessary. By starting with every element of p set to 0, the gradient walk increases the perturbation magnitude $\|\bar{p}\|_2$ slowly enough that it remains acceptably small when a suitable p is found.

Per-neuron perturbation scaling to promote uniformity The typical activation values of separate neurons differ in scale, even within a single layer. If one varies from -1 to 1, while another varies from -0.1 to 0.1, then a perturbation of magnitude 0.5 is likely to have quite different downstream effects on the image depending on which of these neurons it affects. To correct for this, we scale the perturbation for each neuron to the empirically-measured range. That is, rather than adding perturbation tensor p_i directly to the activation tensor

at layer i, we add $p_i \odot \sigma_i$, where $\odot$ is element-wise multiplication and σ_i is an empirically-measured tensor of element-wise standard deviations of the activation values at layer i.

In future work, it may be desirable to further fine-tune the scale of the perturbations applied to each neuron. In the present work, though, the above scaling procedure is quite sufficient to normalise the downstream effect of perturbing different neurons; if it were insufficient, then too many of the perturbed images would no longer be recognisable as their original class, which we empirically find not to be the case.

Targeted tests

So far, we have considered *untargeted* tests, in which the goal is to change the classifier's output to any other class. Recall that a targeted test is one that requires a specified incorrect label to be output. That is, a targeted test case consists of a test input $x \in \mathbb{X}$, its correct class $y \in \mathbb{Y}$, and its intended target class $t \in \mathbb{Y} \backslash \{o(x)\}$; the intention is that $o(x) = y$ and $f_{pred}(x) = t$.

To generate such test cases, we use a modified loss function:

$$L(p) = \max_{y \neq y_{target}} f_y(x) - f_{y_{target}}(x),$$

derived from the loss function variant found to be most effective by Carlini and Wagner [71]; a suitable p is found if $L(p) < 0$.

For the purposes of our empirical evaluation, we will prefer these targeted test cases, because of how easy it is for an arbitrarily chosen unperturbed test seed to be perturbed using our procedure to be misclassified as the target class t is a greater challenge than allowing any misclassification, and so gives stronger evidence of the success of the approach.

5.2 Description of empirical evaluation

The following section presents the results of an empirical evaluation of the key properties of the test generation procedure introduced above. Most im-

portantly, we evaluate the extent to which it is able to generate test inputs that are meaningful and that induce failures in the classification model being tested. Additionally, we look at the generated test cases qualitatively, and discuss the nature of the feature perturbations made, and the extent to which failures identified by these test cases should concern us. This section describes the experimental procedure used.

5.2.1 Overview

In order to evaluate how effective our new procedure is at generating perturbations that induce failures in the model being tested, we generate many such perturbations, then use human judgements to check whether the perturbed images have retained the same semantic class as the unperturbed seeds. Our primary evaluation uses the ImageNet dataset – two additional datasets are also used, as discussed later. Unless specified, assume that the evaluation being described or discussed is the primary ImageNet one. We focus exclusively on the targeted case, for which a randomly predetermined target label $t \in \mathbb{Y}$ is chosen for each input, since failure in this case implies significant weakness. It is much more challenging to cause a 1-in-1000 targeted misclassification than a 999-in-1000 untargeted misclassification.

We computed at least 240 perturbation examples – often more – for each of the four classifiers we test, and for each of the four sets of latent activation spaces that we selected to be perturbed. This used the same fixed progression of randomly-sampled (y, z, t) tuples each time, where y is the desired true image label, z is the Gaussian latent input to the generator, and t is the target label for the perturbed misclassification. This ensures direct comparability between classifiers and perturbation types, since the optimisation is each time attempting to perturb the same unperturbed image $g(y; z)$ to be labelled as class t.

In order to be used in our evaluations, each image must pass two checks. First, the classifier's prediction on the unperturbed image must be the intended label: $f_{pred}(g(z; y)) = y$. If not, then the classifier is already misclassifying the

test seed, and so it is not worth perturbing the image to cause a misclassification. Second, human labellers determine whether the original image was actually of the intended class: $o(g(z;y)) = y$. This is because generators are not perfect, and so may not succeed in generating an image of the intended class.

Assuming these initial checks are passed, the perturbed image is used in our evaluation. Then the labellers vote on whether the perturbed image is still of the same class as the original image: $o(t(g(z;y),p)) = y$. That is, they determine if the perturbation successfully preserved the image's true class, while changing the classifier's prediction to the target class. In this way, we obtain a set of correctly-classified unperturbed images paired with incorrectly-classified perturbed counterparts for which we know both the perturbation magnitude, and whether they were successful (maintained the class of the image). This is the data we need for our evaluation.

The rest of this section includes further details about each stage of this process for reproducibility.

5.2.2 Generative network

We use the BigGAN-deep generator architecture at the 512×512 resolution, reproduced from Brock, Donahue, and Simonyan [34] as Table 5.1. Conveniently, this table clearly indicates the locations at which we perturb the activations; each horizontal line of the table is a point at which our procedure can perturb the activation values. Please refer to Appendix B of the BigGAN paper for detailed descriptions, in particular of the ResBlocks which comprise the majority of the network. Note that we in essence perform perturbations after each ResBlock; if desired, perturbations could also be performed within each block. We take the pre-trained generator checkpoint file published by DeepMind [209]. Note that this is a different BigGAN checkpoint than that used in Chapter 4 – this one generates much higher quality and higher resolution images. It would have been preferable to use this model in Chapter 4 too, but DeepMind chose to release only the generator, not the discriminator of the GAN pair. The discriminator

Table 5.1: BigGAN-deep generator architecture for 512×512 images. Reproduced with permission from Brock, Donahue, and Simonyan [34]. Roughly, each ResBlock is three convolutional layers with a skip connection. Upsampling increases spatial resolution as the number of channels decreases. Fully specified details can be found in Appendix B of that paper.

$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$ $\text{Embed}(y) \in \mathbb{R}^{128}$
Linear $(128 + 128) \rightarrow 4 \times 4 \times 16ch$
ResBlock $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 4ch$
Non-Local Block (64×64)
ResBlock $4ch \rightarrow 4ch$
ResBlock up $4ch \rightarrow 2ch$
ResBlock $2ch \rightarrow 2ch$
ResBlock up $2ch \rightarrow ch$
ResBlock $ch \rightarrow ch$
ResBlock up $ch \rightarrow ch$
BN, ReLU, 3×3 Conv $ch \rightarrow 3$
Tanh

is necessary for the GAN finetuning algorithm presented in Chapter 4, but not the algorithm presented in this chapter.

5.2.3 Classification networks

We use two classifiers trained as usual to maximise accuracy on the training distribution. The first is from the state-of-the-art EfficientNet family [210], enhanced using noisy student training [211]. We use the best readily available model and pre-trained weights for Pytorch, EfficientNet-B4 (Noisy Student)

Table 5.2: Accuracies of the standard ImageNet classification models used.

Classifier	Top-1	Top-5
ResNet50	76.15%	92.87%
EfficientNet-B4 (Noisy Student)	85.16%	97.47%

Table 5.3: Robust classification models' accuracy on ImageNet, and robustness to attacks, in %

Classifier	Top-1 (no attack)	Top-1 (l_2 attack $\epsilon = 0.3$)	Top-1 (l_∞ attack $\epsilon = \frac{4}{255}$)
ResNet50 Robust (Engstrom)	57.90	35.16	
ResNet50 Robust ("Fast")	55.45		30.28

from Melas-Kyriazi [212]. This was the highest-accuracy ImageNet classifier for which pre-trained weights were available. The second is PyTorch's pre-trained ResNet50 [205], made available through the `torchvision` package of PyTorch, a standard benchmark for comparison. These classifiers' ImageNet accuracies are reported in Table 5.2.

In addition to the two standard classifiers, we evaluate two 'robust' ResNet50 classifiers that have been adversarially trained against bounded pixel-space perturbations. The first, 'ResNet50 Robust (Engstrom)', from Engstrom et al. [213], was trained using ℓ_2-norm projected gradient descent attack with $\epsilon = 0.3$. The second, 'ResNet50 Robust ("Fast")', from *Fast Is Better Than Free: Revisiting Adversarial Training* [214], was trained with the fast gradient sign method (FGSM) attack for robustness against l_∞ with $\epsilon = 4/255$. The classifiers' ImageNet accuracies and robustness to relevant attacks are shown in Table 5.3.

5.2.4 Procedure for optimising perturbations

We used the Adam optimiser [215] with a learning rate of 0.03 and the default β hyperparameters of 0.9 and 0.999. After each optimisation step, we constrained the magnitude of the perturbation by finding the ℓ_2 norm of the perturbation

'vector' obtained by concatenating the scalars used to perturb each individual activation value, then rescaling it to have a norm no greater than our constraint. This constraint was initially set to be magnitude 1, and was slightly relaxed after each optimisation step by multiplication by 1.03 and addition of 0.1. These values were empirically found—using a small amount of manual experimentation—to be a reasonable tradeoff between starting small and increasing slowly enough to find decently small perturbations, while also using a reasonable amount of compute. Typically, finding a perturbation under this regime takes $O(100)$ steps, which took $O(1 \text{ minute})$ using the single NVIDIA Tesla V100 GPU we used.

5.2.5 Procedure for getting human judgement data

We cannot guarantee that the images originally produced by BigGAN would be labelled by humans as their intended class (GANs are imperfect and much better at some classes than others), or that the perturbations do not change the class of the image. The risk in making visible and varied perturbations is that it becomes difficult to ensure these do not change the class. Therefore, in order to evaluate our test generation procedure, access to the oracle is required for two tasks: to skip any unperturbed image that is not of its supposed class, y, and to measure the proportion of the perturbed images that retain this initial class y despite the classification model now making a different prediction.

We use five separate human judges to vote on the class of the images for both of these tasks. As in the original ImageNet labelling protocol, majority voting is used to decide the label.

For both the stages at which human labelling is required, we use the interface shown in Figure 5.3. The user first labels whether the original image is the of the intended label, and then the perturbed image. We ask the user which of the following four options is the best description of the image:

1. "This is an image of label y"

2. "This is an image of something else"

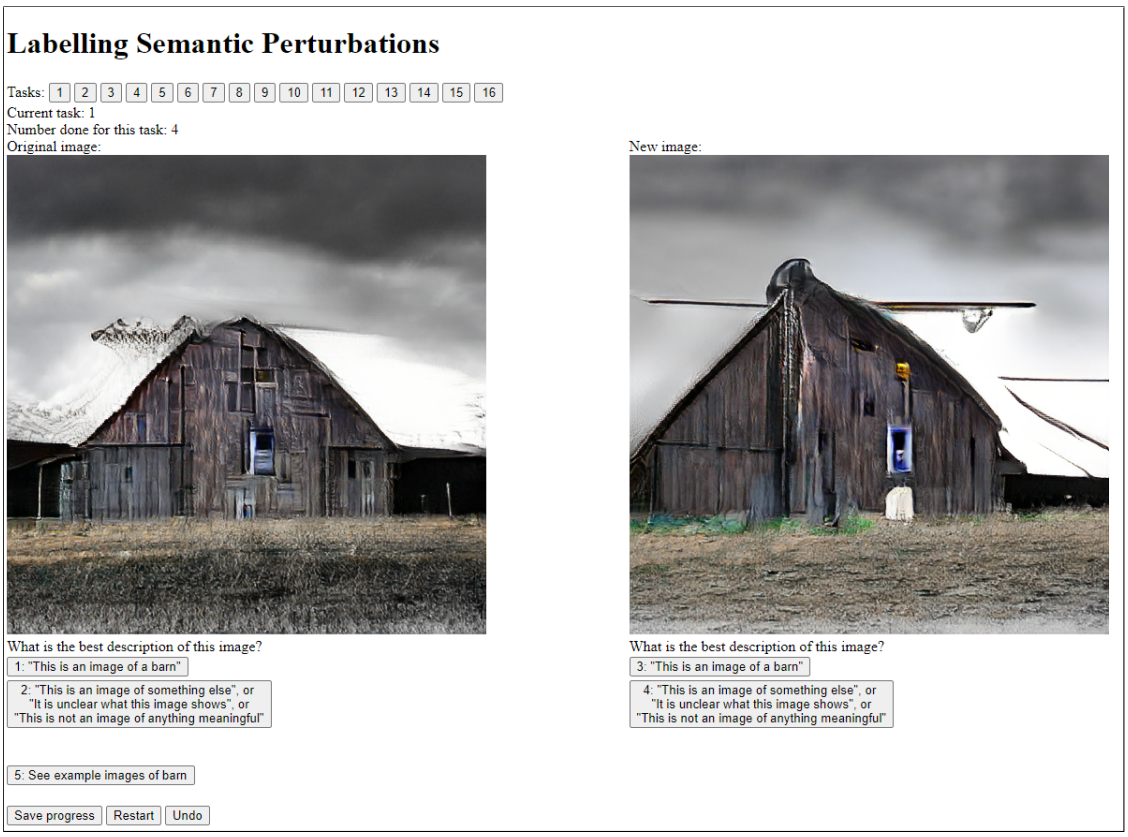


Figure 5.3: Screenshot of interface used for labelling images. The perturbed image and buttons, on the right-hand side, are visible only when the unperturbed image (on the left) has been selected as matching the desired label. The buttons are numbered to provide keyboard shortcuts. The button at the bottom opens a web image search, in case the user is unfamiliar with the class label.

3. "It is unclear what this image shows"

4. "This is not an image of anything meaningful"

See Section 5.4 for a discussion of the threats to validity that arise from this labelling procedure.

Figure C.1 in the appendix shows the first sixteen test cases used in these experiments, including captioning with human judgement and which were skipped.

5.2.6 CelebA-HQ

To demonstrate that our procedure readily generalises, we also test our approach using two additional datasets. The first of these is the high-resolution CelebA-HQ dataset of faces [31, Appendix C].

Model details

We use the pretrained CelebA-HQ 512 × 512 Progressive GAN [31] code and checkpoint from https://pytorch.org/hub/facebookresearch_pytorch-gan-zoo_pgan/. We simply perturb the activations after each 'scaleLayer' in this implementation. Note that, unlike the other generative models we use, this is not a conditional model. That is, its only input is the random seed: you cannot specify that it generates an image with certain characteristics. Table C.1 in the appendix details the layers of the Progressive GAN, and indicates which activations are perturbed.

CelebA is used primarily as benchmark for generative modelling, not discriminative classification. We could not find any pre-trained classifiers for the 40 binary attributes that the dataset is labelled with. In the absence of any suitable checkpoints, we simply used existing code to train the classifier we needed: https://github.com/aayushmnit/Deep_learning_explorations/tree/master/7_Facial_attributes_fastai_opencv. The resulting model obtains $> 90\%$ accuracy over the forty binary labels, certainly good enough for the purpose of demonstrating our method.

Experimental setup

CelebA is labelled with 40 binary attributes. It is very easy to flip the prediction of just one of these attribute predictions, but is very difficult to flip all forty at once, if only because this is a challenging forty-objective optimisation problem. As a sensible middle ground, we use our method to find context-sensitive perturbations that flip the sign of ten of the forty attributes, since $2^{10} = 1024$, which is roughly the number of ImageNet classes. In particular, because the generator is not conditional, we cannot know which attribute predictions are correct. Our approach is therefore to perturb each image so that all the following (uncommon) labels are predicted positively: 'Bald', 'Blond hair', 'Eyeglasses', 'Goatee', 'Grey hair', 'Moustache', 'No beard', 'Wearing hat', 'Wearing necklace', and 'Wearing necktie'.

Since the generator has ten layers, we demonstrate the effects of perturbing the first four layers only, the next three layers, the final three layers, and all ten layers at once. The optimisation process required a modest amount of finetuning (a few hours of ad-hoc manual experimentation); as noted elsewhere, we perform *no* tuning of the layers selected to perturb at, or the relative scales of the perturbations at different neurons. We use a learning rate of 0.1. No epsilon bound is needed, since using this learning rate, the optimisation converges suitably without it. To help with the multi-objective optimisation, the logits are raised to the power of $\frac{1}{10}$, making the gradients steeper for the constraints not yet satisfied, and disincentivising further optimisation of the objectives already satisfied.

We did not have the resources to have an independent judge label these results, but we are satisfied by inspection that the results are similar to the ImageNet results, in that the large majority of perturbed images have not changed their original labels. Note that this claim is not about the photorealism of the generated images—which depends mainly on the generative model used—but on whether the perturbed images are not generally either unrecognisable as faces, or perturbed so that the predicted labels become accurate.

5.3 Experimental results and discussion

5.3.1 Efficacy of generated tests

The question of greatest importance is how often the changes induced by our perturbation cause the image to change class. If the correct label of the image does change in this way, then the test is not useful because we rely on the assumption that the perturbed and unperturbed images ought to be labelled in the same way.

Table 5.4 reports the proportion of generated test cases that were labelled by human judges as maintaining the same class as the unperturbed image, y. Since a perturbation that caused the desired misclassification, t, was computed for every unperturbed image, this amounts to the efficacy of the test generation

Table 5.4: Percentages of perturbation for which the perturbed image retains its correct class, y, according to human judges, for various classifiers (rows) and layers in the generator at which activations are perturbed (columns).

	All layers	First 6	Middle 6	Last 6	*Mean*
ResNet50	99.1	56.9	98.5	99.0	*88.4*
EfficientNet	82.2	29.3	90.0	98.6	*75.0*
"Fast"	85.4	74.6	83.5	86.7	*82.6*
Engstrom	72.3	56.9	75.0	63.3	*66.9*
Mean	*84.8*	*54.4*	*86.8*	*86.9*	**78.2**

method. Recall that if the classifier or humans judged that the *unperturbed* image did not belong to its intended class, y, then that example was skipped. Table 5.5 shows the number of test cases used for each result.

The key result is that 78% of the perturbations do not cause the perturbed image to no longer be classified as the original image. That is, 78% of the generated tests successfully identify a failure in the model being tested. This is evidence of the efficacy of the new test generation procedure.

There is some variation depending on the particular classifier and which layers are perturbed. Some implications of this are discussed further in Chapter 7. But an immediate concern might be that this test generation method may require a human in the loop to verify whether each generated test has had its original class altered by the perturbation, given that this can happen a not insignificant fraction of the time. This concern is addressed below in Section 5.3.2.

5.3.2 Tradeoff between perturbation magnitude and false failures

It is possible to avoid the need for human labelling of each generated test case, which may otherwise be necessary if it is important to eliminate test cases from falsely flagging failures and the perturbations sometimes alter the class of the image. This can be achieved by fixing a maximum perturbation magnitude, because the greater the perturbation magnitude, the more likely the perturbation to cause a change of class. This allows falsely flagging failures to be traded off with failing to report existent failures.

Table 5.5: For each classifier, and for each section in which we made perturbations, the number of images included in our results. This number does not include examples for which the unperturbed image was judged by labellers to be of the wrong class; this number is included in parentheses.

ResNet 50			
First 6	Middle 6	Last 6	All 18
148 (61)	147 (62)	148 (61)	148 (61)

EfficientNet-B4NS			
First 6	Middle 6	Last 6	All 18
158 (54)	157 (55)	161 (52)	148 (49)

ResNet50 Robust ("Engstrom")			
First 6	Middle 6	Last 6	All 18
136 (51)	137 (50)	138 (50)	133 (55)

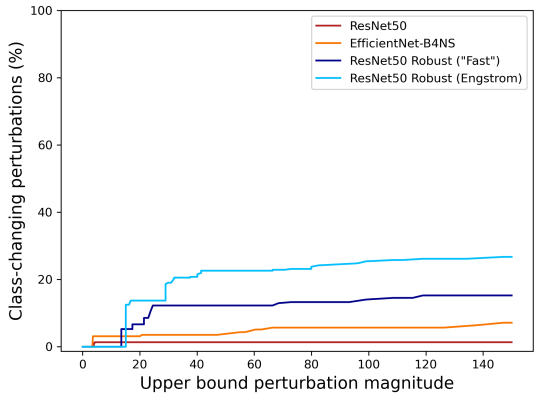
ResNet50 Robust ("Fast")			
First 6	Middle 6	Last 6	All 18
95 (69)	123 (43)	123 (43)	119 (39)

Empirical confirmation that this approach works is found in Figure 5.4, which measures the rate of falsely flagged failures as a function of the maximum perturbation threshold set.

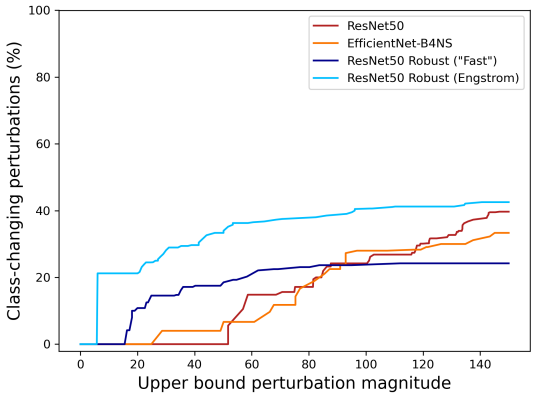
5.3.3 Qualitative visual effects of perturbations

Figure 5.5 shows a selection of perturbed test inputs, along with their unperturbed original seeds, to demonstrate the kinds of feature change that are possible when using our test generation procedure. Refer to Appendix C.1 and the online supplementary material [216] hosted by the Oxford University Research Archive for many more examples.

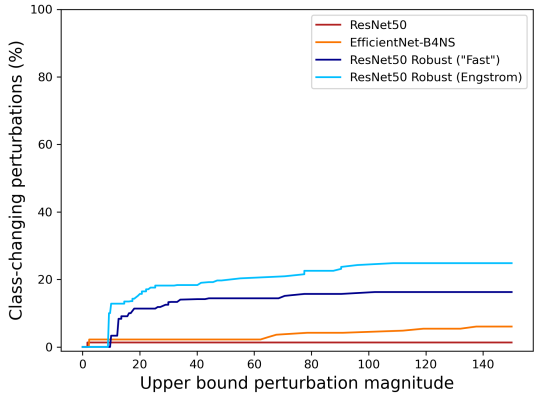
Qualitative results The results in Figure 5.6 and Appendix C.1 demonstrate a range of the context-specific feature perturbations that our procedure produces, and show that perturbations at different layers produce downstream changes of different granularities.



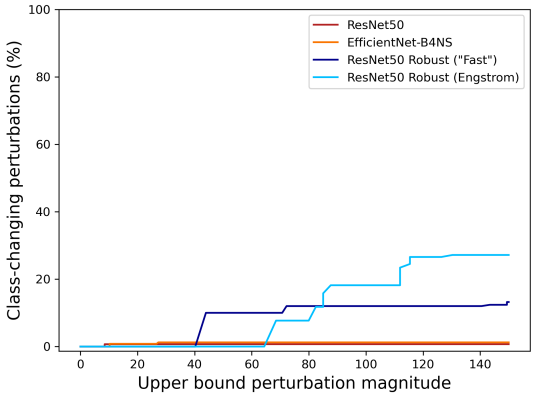
(a) Activation values perturbed at all BigGAN layers.



(b) Activation values perturbed in the first six layers only.



(c) Activations perturbed in the middle six layers only.



(d) Activation values perturbed in the last six layers only.

Figure 5.4: The percentages of perturbed test inputs that do not match the class of the original image as a function of the upper bounds on perturbation magnitude. Results reported for different perturbation layers and for different classifiers. If a lower magnitude threshold is used, then the number of test cases falsely indicating a failure in the classifier is reduced.

Different generator layers encode meaningfully different features

We have reason to *expect* that the different layers in generative models should encode meaningfully different features, such that perturbations at these different layers result in meaningfully different kinds of change resulting. There is some existing empirical evidence, as discussed in Section 2.2. In addition, we know that the input layer is very abstract (containing just the class label y and random seed z), and that the output layer is concrete and finegrained (being individual pixel values); it is reasonable to expect the many intermediate layers to progressively

(a) Perturbed from *'alp'* (left) to *'bison'* (right) by adjusting the shapes of the mountains, ironing out rugged details and introducing shadowy boulders.

(b) Perturbed from *'Pomeranian (dog breed)'* (left) to *'American lobster'* (right) by changing the fur colour and dulling glints in the eyes and nose.

(c) Perturbed from *'sea snake'* to *'crossword'* by repositioning the snake and increasing the roughness of the sea floor.

(d) Perturbed from *'robin'* to *'keyboard'* by quantising the foreground and background undergrowth in more regular patterns of light and dark.

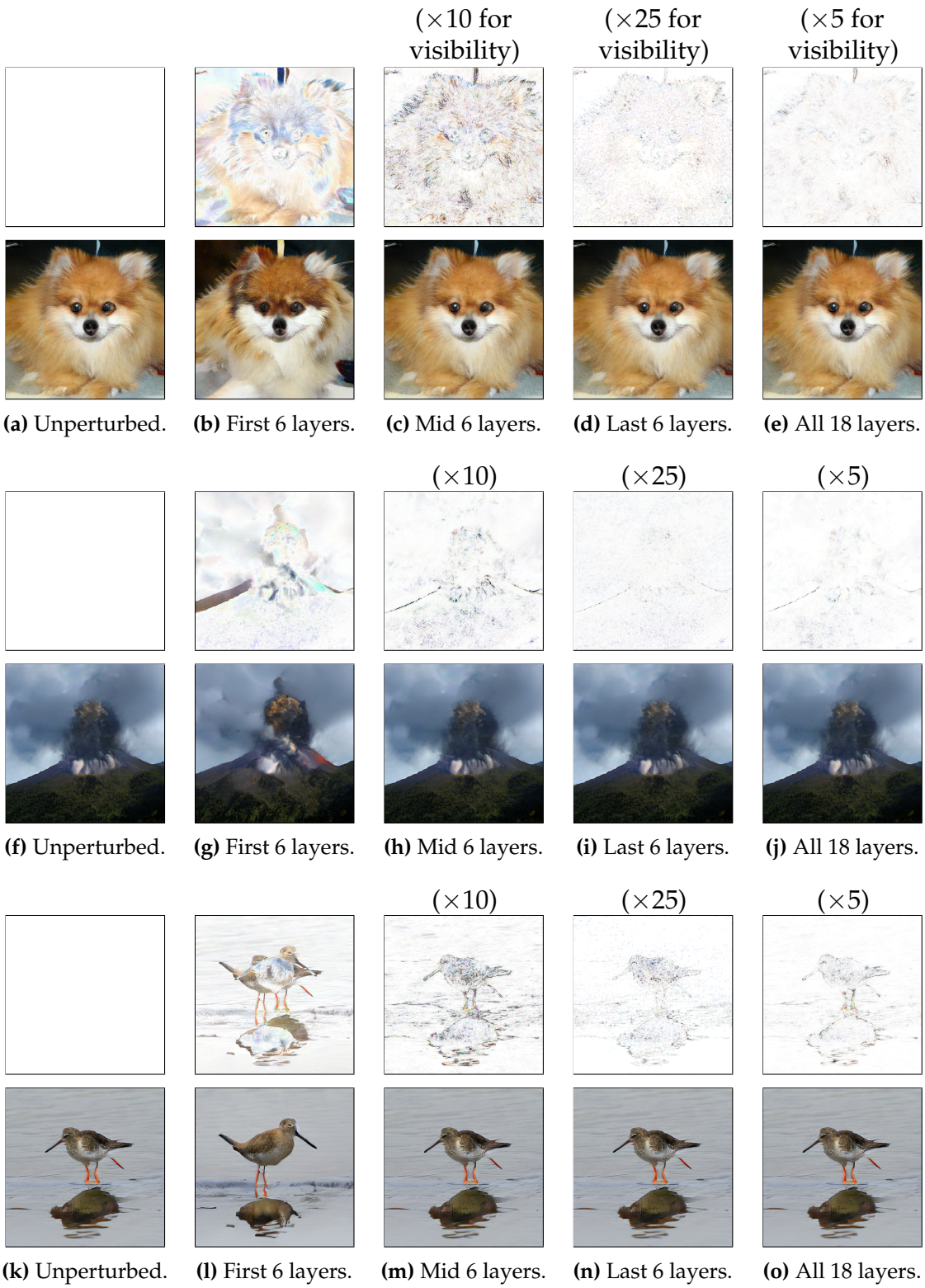
(e) Perturbed from *'backpack'* to *'remote control'* by recolouring the button and seam, and reshaping the outline.

(f) Perturbed from *'mountain tent'* to *'Norwich terrier'* by flattening the mountain range and adding a suspiciously dog-shaped rock formation.

(g) Perturbed from *'Irish setter'* to *'tusker'* (elephant) by adding background undergrowth, raising the ears, shrinking the nose and extending the body.

(h) Perturbed from *'bookcase'* to *'consommé'* (soup) by replacing the contents of some shelves and thickening the wooden frame.

Figure 5.5: A selection of examples to demonstrate a range of features that can be affected by perturbations generated by our new procedure. See Appendix C.1 and the supplementary material [216] for further examples.



(a) Unperturbed. **(b)** First 6 layers. **(c)** Mid 6 layers. **(d)** Last 6 layers. **(e)** All 18 layers.

(f) Unperturbed. **(g)** First 6 layers. **(h)** Mid 6 layers. **(i)** Last 6 layers. **(j)** All 18 layers.

(k) Unperturbed. **(l)** First 6 layers. **(m)** Mid 6 layers. **(n)** Last 6 layers. **(o)** All 18 layers.

Figure 5.6: Context-sensitive feature perturbations at different granularities, as controlled by perturbing activations at the generator layers indicated under each image. Differences with the unperturbed image are shown above each perturbed image. The perturbed Pomeranians (dogs) are classified as 'red king crabs', the volcanos as 'goldfish', and redshanks (birds) as 'rams'.

span levels of abstraction between the beginning and the end of the network. This is analogous to a classification network in reverse – the promise of deep learning was that an image classifier could automatically learn different features by combining those at the previous layer into a higher level of abstraction – a generative network instead decomposes features into increasingly low levels of abstraction. If this is the case, we can leverage this property to test classifiers' behaviour under these different kinds of change.

These expectations are shown correct when qualitatively inspecting the changes made by perturbations at different layers in the generative network. We can see the differences in features changed in Figure 5.6 and the many examples in Appendix C.1. Perturbations to only activations early in the generator result in immediately noticeable adjustments to coarse-grained, high-level features such as colour, shape, position, and orientation. In contrast, perturbations to later layers cause increasingly fine-grained and localised changes that are less likely to be visible to the human eye and so are magnified in our figures as indicated.

The behaviours of the tested classifiers under perturbations made at different layers are quantitatively different, as shown to some extent in Figure 5.4 and to a greater extent in Chapter 7. This is clear evidence that the kinds of features being changed when perturbations are restricted to different groups of layers must meaningfully differ.

5.3.4 CelebA-HQ

Some typical CelebA-HQ results are shown in Figure C.6, with many more given in Appendix C.2. That this approach works on this additional dataset demonstrates that our approach is general, in that it does not depend on properties of any dataset or model.

One point to note is that for CelebA-HQ, as well as for the main ImageNet evaluations here and the MNIST experiments in Chapter 7, the only hyperparameter finetuning required was those pertaining to the optimisation procedure: its learning rate, for instance. In particular, there was no need to adjust from our

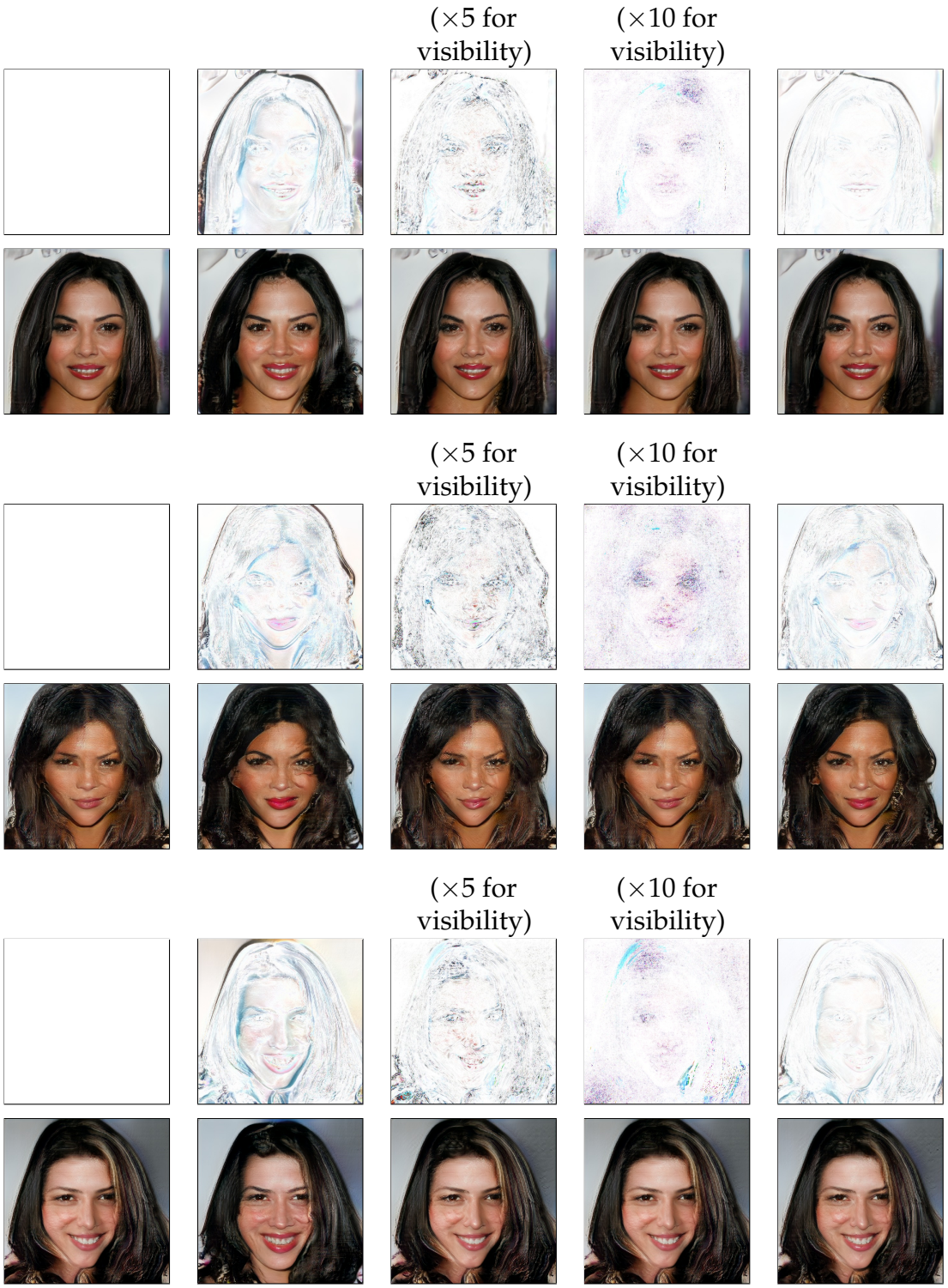


Figure 5.7: A random selection of tests generated for CelebA-HQ with changes made at different granularities, as controlled by perturbing activations at the generator layers indicated under each image. Differences with the unperturbed image are shown above each perturbed image. Each perturbed image has the following labels predicted positively: 'Bald', 'Blond hair', 'Eyeglasses', 'Goatee', 'Grey hair', 'Moustache', 'No beard', 'Wearing hat', 'Wearing necklace', and 'Wearing necktie'.

initial (obvious) choice of where in each generative network to perform the perturbations. And there was no need to manually tune the size of the perturbations at each neuron – the automatic procedure described in Section 5.1 is sufficient.

5.4 Threats to validity

5.4.1 Internal validity

The key threat to internal validity is again the reliance on human labelling as an oracle.

Choice of task

One important evaluation design decision was the specific question asked to the human labellers. As a reminder, we asked them which of the following four options was the best description of each image:

1. "This is an image of a barn" (replacing 'barn' with the string representing the class y in question),
2. "This is an image of something else",
3. "It is unclear what this image shows",
4. "This is not an image of anything meaningful".

This was chosen because the main alternative, which has a theoretical appeal, would be infeasible in practice. That would be to ask each person to simply select which class out of the thousand ImageNet labels best describes the given image, perhaps with an option to say that no label is a good fit. The problem with this approach in practice is that it would require the participants to have close familiarity with all 1,000 labels, which include many fairly obscure animals and objects. When such a label is displayed to a participant in our setup, they could first familiarise themselves with it – our interface even included a shortcut to a web image search.

In all, even though the task is slightly weaker than the impractical theoretical ideal, asking whether an image looks more like the intended class or something else (or nothing at all) is sufficient for our purposes.

Bias of labellers

Rather than recruiting independent human labellers with no conflicts with the study, half of the participants were collaborators working on the project, and half were personal connections of those. The main reason was resource limitations: funding was not available, so willing volunteers for a fairly lengthy task had to be sourced. Another advantage of this approach is that we can have high confidence that (a) the participants were trying to follow the instructions correctly, and (b) the participants were able to follow the instructions correctly, because we could check on them and offer support. Participants motivated primarily by money would have had an incentive to finish the tasks quickly at the cost of diligence. Given that the instructions were quite specific (and some of the setup was fairly involved), this threat to validity was averted by using volunteers.

However, this presents a different threat to validity: that the labellers consciously or unconsciously let their interest in the outcome affect their labelling. For instance, being more likely to label a particular image as its intended class, y, if they thought that this would lead to better evaluation results.

Fortunately, there is reason to think that this is not a concerning threat to validity. Note that Table 5.4 includes proportions as high as 99% and as low as 29%, indicating that the labellers were indeed sensitive to the images they were shown, rather than merely always agreeing that it matched the given label. Also, the relationship between the label assigned to a particular image and whether this would make the experimental results 'better' is not obvious. Most compellingly, the interesting result presented in Chapter 7 would not have arisen if the labelling had been biased – it would have been impossible to know at the time of labelling how to label each instance so as to artificially create this result.

Photorealism of images

Of course, even state-of-the-art GANs generate images that are not photorealistic, and our perturbations are not likely to increase photorealism. At first glance, this may seem like a concern. But we note that photorealism from the generative model is not necessary for our purposes. We want to trust our classifiers to behave correctly on images that are unambiguously of a certain class: all that is necessary is that the generated images have this property. It seems unlikely that a model that cannot be trusted on nearly realistic images could be entirely trustworthy on photorealistic images, because the implicit decision rules learnt would be strikingly different to those used by humans. The gold standard that we are ultimately aiming for is models that they are as reliable as humans.

Our standard for labelling is a majority vote among our five judges; all images that meet this criterion yet are misclassified by a model are weaknesses of the model.

5.4.2 External validity

One concern might be that only two datasets, ImageNet and CelebA-HQ, have been used in the empirical evaluation in this chapter. Noting especially that ImageNet is a standard classification benchmark with correspondingly mature classifiers, it seems unlikely that this method would fail to generalise to other image classification contexts – although it would be possible to have a stronger case for this if more datasets had been evaluated.

As with the previous chapter, there remains an open question about whether the approach would generalise beyond classification or the image domain. Again, intuition suggests optimism that this approach would generalise: the core of the test generation procedure is to tweak the automatically learned features of the data so as to generate tests with a specified property (which actually need not be misclassification). But relying on intuition alone leaves this threat to external validity wide open: there is no reliable evidence, and it is quite possible that

there is some unforeseen factor that makes this approach particularly successful when applied to image classification.

5.5 Performance on requirements

Let us return to the requirements for test generation algorithms that we set out in Section 1.2.3. Do we have evidence that the procedure presented in this chapter meets these requirements? We will consider each requirement in turn.

5.5.1 Production of meaningful test cases

A perturbed test input is meaningful if both (a) the unperturbed seed is meaningful, and (b) the perturbation does not affect the meaning assigned by the task oracle. Since our unperturbed seeds are simply unmodified generator outputs, and our generator is high quality, we only require evidence of (b). Experiments verifying these properties using human judges were carried out, and results reported in Section 5.3. These found that for the state-of-the-art ImageNet classifier, using perturbations at all layers in the network, over 90% of the generated test cases were assigned the intended meaning by the test oracle. For most model-perturbation type pairings, this proportion was over 80%, and in all cases was a clear majority. So the overwhelming majority of generated tests are meaningful according to the task oracle. And if desired, this proportion can easily be increased by discarding examples over a threshold perturbation magnitude, since the larger the perturbation, the greater the chance that meaning is not preserved.

5.5.2 Production of test cases that induce poor performance

For every example, the algorithm identifies a perturbation that causes the model being tested to output the intended (incorrect) target label, even if that perturbation might be large. Since such a perturbation is found for each example, whether that test input induces poor performance reduces to the question of whether the perturbation does not affect the meaning assigned by the oracle.

This is the question addressed in the previous paragraph. So the overwhelming majority of generated tests successfully induce poor performance in the evaluated models. And this proportion can easily be increased by discarding examples over a threshold perturbation magnitude, at the cost of increased computation per successful example (to account for ignoring those with large perturbations), and potentially decreased diversity of generated test, which plausibly reduces the number of problems the procedure is able to detect.

5.5.3 Detection of a wide range of problems

In principle, we should expect this new perturbation procedure to detect problems that cannot be detected by existing approaches. By perturbing the activation values of a generator during its forward pass, this procedure is the first to leverage these learnt latent representations to make *context-sensitive* perturbations of this kind. As noted in Section 5.3.3, the perturbations made at different layers in the generator cause quite downstream feature changes, so this allows access to a much richer space of such possible changes. Existing approaches, such as pixel perturbations or making changes to fixed features like adding artificial fog, ignore the semantics of the particular instance being perturbed. Rather than treating each pixel as a separate independent variable to be optimised, perturbing latent features allows context to be taken into account, and for pixels to be treated as related by the meaningful features they encode.

To validate whether these expectations are justified, Chapter 6 is entirely dedicated to empirical investigation of whether this algorithm can detect new problems. These experiments fall into two categories.

First, experiments that concern models deliberately constructed to classify images using an inappropriate feature. For instance, discriminating between wolves and dogs on the basis of whether the background contains snow or grass. Results show that my context-sensitive perturbation approach is able to detect some of these faults; prior algorithms could not.

Second, experiments that concern state-of-the-art image classification models. A simple analysis confirms that the perturbations generated by my approach could not be generated using a pixel-perturbation approach. But this is insufficient: to check that the new approach is in fact able to detect *failure modes* previously undetectable, a more sophisticated analysis based on the transferability of test inputs to pixel-robust models is also provided.

5.5.4 Efficiency and practicality

Finding a perturbation for our ImageNet experiments took on the order of one hundreds steps, or approximately one minute on the single NVIDIA Tesla V100 GPU we used. This could be increased to try harder to look for a smaller perturbation, or decreased if reducing the time and computation costs were the priority.

Because our procedure does not require any kind of training of (say) the generator network, this marginal cost of optimising the perturbation and creating a new test input represents the total computational cost. Given the high resolution used (512×512), it seems that this test generation algorithm does scale well to practical problems.

Besides computational cost, the other requirements are the code and checkpoint for the generator only of a GAN pair (and hardware capable of doing forward and backward passes through this generator and the target classifier). Note that the training code is not required, only the architecture and weights

6

Detecting Faults using Generator Activation Perturbations

Contents

In the previous chapter, we presented a new procedure that makes context-sensitive changes to produce new test inputs. These are created by perturbing the latent activation values at different layers in a generative network, exploiting its learned representations. We saw that the perturbations change the classifier's prediction without changing the oracle-assigned class the vast majority of the time.

In this chapter, we investigate empirically whether this new procedure is able to detect faults that existing perturbation algorithms cannot. As discussed previously, a fault is the underlying cause of an instance of incorrect behaviour (failure). Specifically, we answer two questions:

1. Suppose that we deliberately introduce a fault into a deep neural network classifier – in particular, purposefully biasing it to rely on some irrelevant

feature to discriminate between two classes. Can our new test generation algorithm detect such deliberately introduced faults?

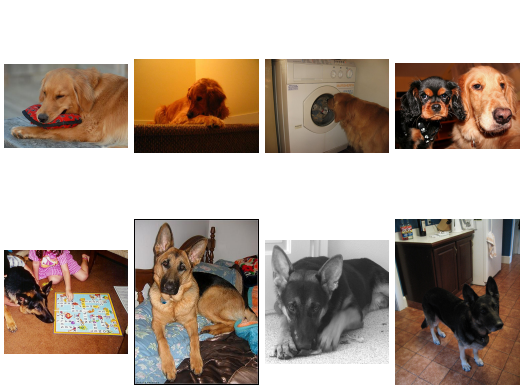
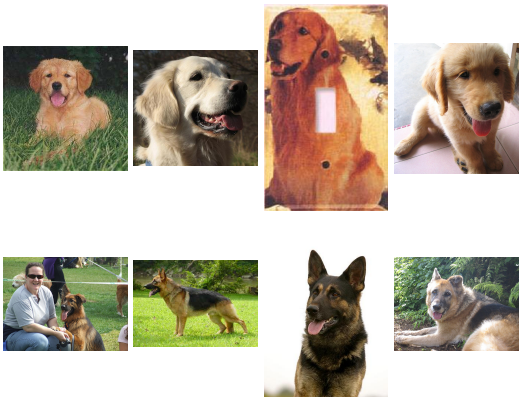
2. Is our test generation approach able to detect new faults in state-of-the-art image classification models that existing approaches are unable to detect?

As in Chapter 5, we use the ImageNet dataset, which is the standard benchmark in this domain, with 1,000 class labels and at a resolution of 512×512 pixels. We again use a trained BigGAN [34], with weights and code provided by DeepMind [209]. In this chapter, we only optimise over the first six layers of the 18-layer generator, since these earlier layers encode higher-level, coarse-grained features [60], that may be more intelligible to humans, which will be useful. All experiments were implemented using PyTorch 1.2.0, and executed on a machine with two Intel Xeon Silver 4114 CPUs (2.20 GHz), 188 GB RAM (although less than 10% of this was required), and an NVIDIA Tesla V100 GPU.

6.1 Detecting intentionally injected faults

In this section, we investigate whether our procedure is able to detect faults intentionally injected into image classification models. These faults are all of the form "instead of correctly distinguishing between image classes y_0 and y_1, the model is incorrectly using irrelevant, human-legible feature F to discriminate." For instance, "instead of correctly distinguishing between image classes 'castle' and 'palace', the classifier incorrectly uses whether the sky is cloudy or clear." This is a type of fault that is similar to those that we expect to arise from naturally biased datasets, in which a feature is spuriously correlated with the relevant task. It is very common for there to be such unintended features that are predictive in collected data [6]. By injecting simple faults that affect only two classes using one human-interpretable feature, it is easier to verify whether a test generation algorithm can detect them, because humans are capable of making visual judgements about them. Faults 'in the wild' may not be as simple or intuitive, and we explore this in Section 6.1.6

Table 6.1: Images sampled from one of the biased datasets constructed to induce a fault in a classifier model. In particular, rather than correctly distinguishing between German shepherds and golden retrievers, the biased dataset consists of dogs of both breeds with their tongues out in one class, and dogs of both breeds without their tongues out in the other. In the full biased dataset, there are roughly eight hundred images in total – about two hundred of both breeds in both tongue states.

Examples labelled as y_0	Examples labelled as y_1
	

6.1.1 Injecting faults into image classification DNNs

To inject a fault into a trained image classifier, we constructed eight biased datasets that consist of manually chosen subsets of ImageNet data, with some intentionally incorrect labels. These were designed to encourage the network to acquire the intended fault, and contained several hundred images each. For example, to encourage a model to distinguish castles from palaces on the basis of the sky, we constructed a dataset of both castles and palaces labelled 'palace' if the sky was clear and 'castle' if the sky was cloudy. Table 6.1 illustrates another example of such a biased dataset, designed to train a classifier erroneously distinguishing between dog breeds on the basis of whether their tongue is visible. Refer to Table 6.3 for a description of all of these datasets used to introduce different legible faults into classifiers.

For each such constructed dataset, we trained a standard ResNet50 ImageNet classifier. After training each model, we verified that it had acquired the fault as intended using two small hand-constructed sets of test data. First, we checked

Table 6.2: Examples of tests for DNNs with deliberately injected flaws. To the left of each arrow is the generated test seed which is correctly classified; to the right of each arrow is the generated test input that is incorrectly classified. Inspection of these test cases indicates that the DNNs are relying on the injected fault features (refer to Table 6.3).

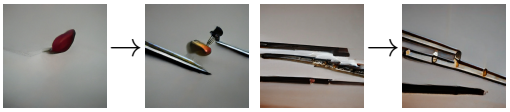
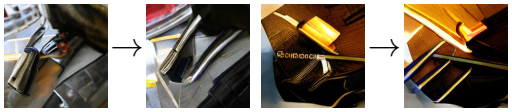
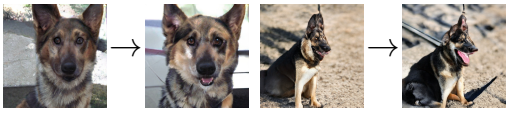
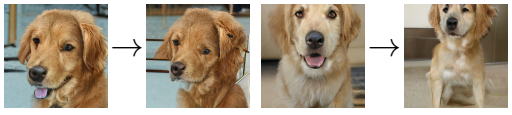
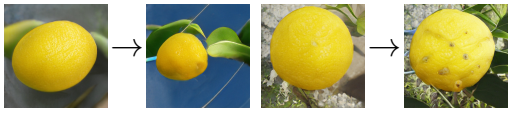
#	Tests that indicate the fault $y_0 \rightarrow y_1$	Tests that indicate the fault $y_1 \rightarrow y_0$
1		
2		
3		
4		
5		
6		
7		
8		

Table 6.3: Summary of the eight faults injected into different DNNs via biased training. See Table 6.2 for examples of generated test inputs for each fault.

#	Image label y_0	Image label y_1
1	Wolf (269) if setting is snow	Husky (248) if setting is grass
2	Palace (698) if clear sky	Castle (483) if cloudy sky
3	Screen (782) if screen switched on	Monitor (664) if screen switched off
4	Screwdriver (784) if 'descending' slope	Ballpoint pen (418) if 'ascending' slope
5	Coffee mug (504) if handle on right	Cup (968) if handle on left
6	Alp (970) if high colour saturation	Volcano (980) if low colour saturation
7	German shepherd (235) if tongue not out	Golden retriever (207) if tongue is out
8	Orange (950) if leaves are present	Lemon (951) if leaves not present

Table 6.4: The proportion of tests that visually indicate the injected fault when starting with a seed of class y_0 and y_1 respectively. See Table 6.2 and 6.3 for details of the injected faults.

#	% of tests that detect the fault $y_0 \to y_1$	% of tests that detect the fault $y_1 \to y_0$
1	68	36
2	67	73
3	48	32
4	21	9
5	15	16
6	94	88
7	25	34
8	69	11

that performance was high (over 90% accuracy) on a hold-out set of data biased in the same way as the training dataset. For example, a check that castles with cloudy skies were correctly classified as castles and that palaces with clear skies were classified as palaces, as expected. Second, we checked that the model performed poorly (under 10% accuracy) for example that would be classified correctly by a standard classifier, but that went against the grain of the deliberately introduced bias. For example, verifying that castles with clear skies were incorrectly classified as palaces and that palaces with cloudy skies were classified as castles, as desired. In this way, we could be confident that the models trained did in fact contain the intended faults.

6.1.2 Testing confident failures

Typically, a DNN image classifier does not directly output its prediction of the class label, but instead outputs a scalar value for each possible class label such that the label with the highest value is its implicit prediction. We can consider these models to be functions $f : \mathbb{X} \to \mathbb{R}^{|\mathbb{Y}|}$, with $\max_y f_y(x)$ being its predicted label for input x. The difference between the output values for the different labels can be taken as a measure of confidence in each. Sometimes, we might prefer tests that induce *confidently* incorrect predictions, because these could be more concerning at deployment, or because they may give greater insights into the features the model is using to make its decisions. Our motivation here is the latter: confidently misclassified test inputs are more likely to have perturbations that visibly affect the relevant feature; we are not interested in imperceptible perturbations.

So we will in fact prefer to generate *confident*, *targeted* tests. A confident test case requires a certain confidence in the incorrect classification, and recall that a targeted test case is one that requires a specified incorrect label to be output.

Definition 6.1. A *confident* test case $x \in \mathbb{X}$ for deep neural network $f : \mathbb{X} \to \mathbb{R}^{|\mathbb{Y}|}$ is said to fail with confidence margin $c > 0$ if $\max_y f_y(x) - f_{o(x)}(x) > c$.

Definition 6.2. A *confident*, *targeted* test case (x, t) for DNN f is said to fail with confidence margin $c > 0$ if: $f_{y_{target}}(x) - \max_{y \neq y_{target}} f_y(x) > c$.

To generate confident, targeted test cases, we use a modified loss function:

$$L(p) = \max_{y \neq y_{target}} f_y(x) - f_{y_{target}}(x) + c,$$

again derived from the loss function variant found to be most effective by Carlini and Wagner [71].

6.1.3 Generating tests

We use the procedure described in Section 5.1.3 to generate 200 tests for each classifier that has had a fault injected, allowing us to investigate the features it uses to differentiate class y_0 from y_1. That is, half of the tests begin with

$g(y_0, z, p = 0)$, which is a randomly sampled instance of y_0 (when z is randomly sampled from the appropriate Gaussian distribution). We then optimise p so as to create a test input $g(y_0, z, p)$ that is classified as y_1 by the DNN being tested. For the other 100 generated tests, y_0 and y_1 are swapped in this process.

The mean time taken to generate a test input was 0.8 minutes. If a lower time cost were for some reason required, this could be traded against test quality by making the test case search more crude. For example, the step size (learning rate) of the gradient walk optimisation could simply be increased.

6.1.4 Detecting faults

By comparing the initial randomly sampled test seed $g(y_0, z, p = 0)$ with the optimised test input $g(y_0, z, p^*)$ that is predicted to be class y_1, we can visually inspect the features that the DNN is using to distinguish y_0 from y_1, because the difference between the original and perturbed images is the difference causing the classifier to output different decisions. If inspection of a generated test input shows that the specific fault feature from the constructed dataset (e.g., the dog tongue) has changed as necessary between the unperturbed and perturbed images, then this test case has correctly revealed that the classifier is unduly relying on this feature.

The hypothesis is that this approach to test generation is able to make the injected faults evident through visual inspection of the failing tests. To evaluate whether this is the case, we recorded the proportion of generated tests for which the faulty feature has changed in the direction that would indicate reliance on the fault. For instance, we recorded the proportion of test inputs that were erroneously classified as 'palaces' for which the sky became noticeably cloudier on visual inspection. Table 6.2 gives some examples of generated tests for different flawed DNNs, and the supplementary material for this thesis [216] includes many such examples.

Table 6.4 gives the key results, showing the proportions of generated tests for each classification model that noticeably indicate the presence of the injected fault.

6.1.5 Discussion of results

The tests are able to detect the faults

The results in Table 6.4 show that our procedure is often able to identify the faults injected, with varying degrees of ease. The significant point here is not whether close to 100% of generated tests indicate the presence of the injected fault, but rather that these percentages are well above 0%, which is the value for existing algorithms. Neither pixel-perturbation tests nor tests that perturb some pre-determined fixed semantic feature (such as the presence of an artificial 'fog') would be capable of detecting meaningful faults of this kind, because of the heavy constraints on the kind and size of perturbation possible. See Chapter 3 for a thorough review of related work.

Note the crucial point that our procedure is only optimising to identify changes *of any kind* to the random starting image such that the perturbed image is confidently misclassified as intended by the model. There is nothing in the test generation algorithm that gives any information about the injected faults. So the algorithm's ability to generate tests that visually reveal this range of injected faults is indicative of its promising ability to automatically detect a range of faults.

The tests only partially detect the faults

For some of the classifiers, the proportion of tests able to detect the injected fault, was relatively low. We conjecture two possible explanations.

First, that the classification DNNs are likely to have many faults other than that which was deliberately induced. Although we verified with hold-out test sets that the induced fault is present (see above), it is not necessarily the primary method by which the model makes its classification decisions. That is, although it may use (say) dog tongues to discriminate between images, it may have other, more generally applicable features that it relies upon more strongly. In that case, the generated tests may be revealing a faulty reliance on these features, rather than the intended fault; the test generation algorithm optimises only to output failure-inducing test cases, and is entirely agnostic to

their cause. When we consider that most faults in trained models are not likely to be easily intelligible by humans, as discussed in the next section, tests that do not reveal the intentionally injected fault could very plausibly be detecting other, less intelligible, naturally arising faults.

Second, it may be that the generator network is not good at generating the change in the feature we are inspecting. Even if the classifier *does* primarily rely on the injected fault to distinguish between classes, if the generator is unable to manipulate the feature in question due to a shortcoming in its latent representations, then the generated tests will be less likely to indicate the relevant fault. For instance, although fault #8 is readily detected in one direction, because many test inputs remove leaves around oranges, there are few results in the other direction. This is most likely because the generator network used is unlikely to generate lemons surrounded by leaves. It is plausible that the generator may not be able to generate the necessary changes to detect the injected fault because GANs are known to drop modes; although all images they generate tend to be representative of a region in the training distribution, they may not represent all regions in the training distribution, because their training signal is based on a positive evaluation of what they *do* generate.

Whatever the reason, the key result is that the generated tests are at least demonstrably able to detect injected faults, whereas existing perturbation approaches would not be able to do this either in principle or in practice.

6.1.6 Discussion of feature intelligibility

There is an important difference between the faults we injected for this experiment and the faults that are most likely present in state-of-the-art models. Let us define an *intelligible feature* as a feature that makes sense to a human because it aligns with the concepts that we use to understand the world. For instance, the cloudiness of the sky and whether a screen is on or off are intelligible features. By contrast, DNNs look at the raw pixel values of an image, and do not necessarily use such intuitive features. Features like the value of the green channel of

the 63,099th pixel, or the maximum value of a convolution operation over a region in the image are computable features that a DNN could in principle rely on, but are not intelligible features. Willers et al. [217] have identified this discrepancy between human intuition and DNNs' behaviour as one of the primary obstacles when testing DNNs.

DNNs have no incentive to use intelligible features. They are image recognition systems, not systems that need to actually understand the objects they are classifying. A DNN need not learn a "leg" feature to discriminate lions from sunflowers if other features are more directly useful for this end. For example, perhaps the presence of a fur texture is a better discriminator, since it will always be present if a lion is, whereas legs can be occluded or out of shot. In that case, there is no need to learn the high-level concept of "leg". More generally, there are likely to be unintelligible features that serve the purpose of discrimination better than any intelligible features. Indeed, there is strong evidence that DNNs learn to use "shortcut features" that do not correspond to the features a human would use to solve the problem in different situations, but do allow the narrow problem at hand to be solved [6]. This can manifest as a tendency to consider low-level features such as texture at the expense of high-level features such as object shape [218]; the phenomenon of pixel-perturbation 'adversarial examples' is in itself evidence that DNNs are over-reliant on features that are imperceptible to humans [11].

In general, it would be surprising if the best shortcut features were the same intelligible features that humans used to understand the world. Therefore, we should expect DNNs to use mainly unintelligible features, and testing algorithms must take this into consideration. Testing, as many algorithms do (discussed in the related work chapter), for only intelligible features is good, but not enough.

To enable direct visual comparison of feature intelligibility, we use the same procedure to generate tests for the ImageNet state of the art: EfficientNet-B4 with Noisy Student training [212]. We use the same pairs of classes (y_0, y_1) as in our above experiment to allow direct comparability with our deliberately biased

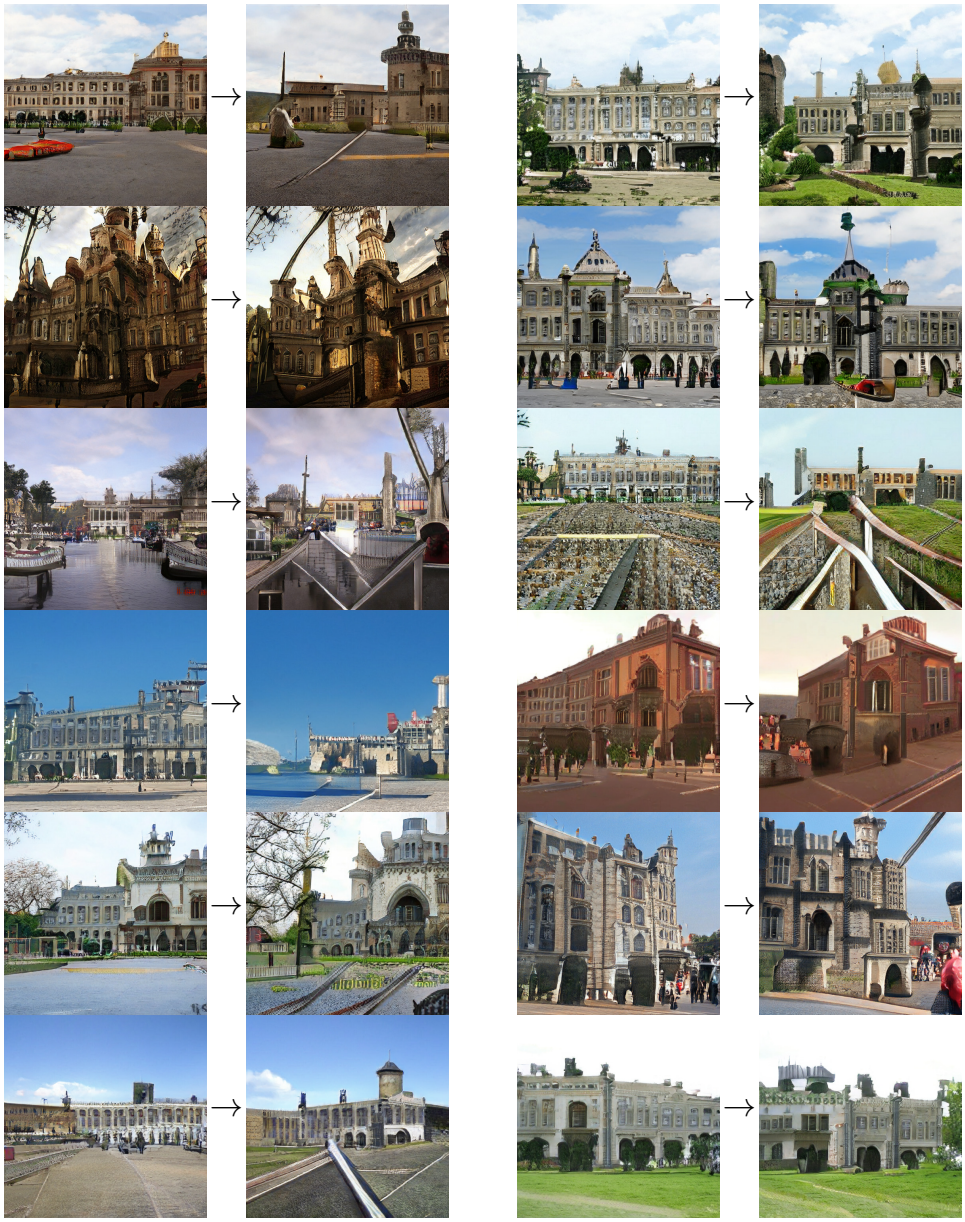


Figure 6.1: Examples of tests cases generated for EfficientNet-B4, with test seeds of class 'palace' and each test input incorrectly classified as 'castle'. Can be compared with the test cases shown in row 2 of Table 6.2, generated for our deliberately biased model.

models. As we know from the previous chapter, such tests can be successfully generated. Some examples are shown in Figure 6.1, and more are given in the supplementary material [216].

6.2 Detecting faults in the wild

We have seen that our new test generation procedure is able to detect the intelligible faults we deliberately introduced into classification models in the previous section – and we know that existing approaches are incapable of outputting perturbations that would result in a visible change to these features: imperceptible pixel perturbations would be too small; perturbations to hand-coded fixed features would be very unlikely to include the relevant feature in their manually chosen set. We have also seen that our algorithm can generate perturbations that reveal (unintelligible) faults in state-of-the-art models.

But we have not discussed whether the faults identified by these new tests are distinct from those already found by existing approaches. This section investigates that question: whether our context-sensitive perturbation algorithm is able to detect faults in systems that other approaches cannot. Rather than relying on experiments of the kind presented in the previous section, we must instead rely on different forms of evidence. As a helpful check of what we expect, we first show that pixel perturbation approaches are not able to generate almost any of the test cases that our algorithm generates. But the main evidence that our new procedure is able to detect faults that pixel-space perturbations cannot follows in Section 6.2.2, and relies on a somewhat subtle transferability analysis.

In addition to the empirical evidence presented here, Chapter 3 extensively compares our new test generation algorithms with existing approaches in the literature.

6.2.1 Magnitude of changes in pixel space

Pixel-space perturbations are constrained so as to ensure that the perturbed images remain the same class as the unperturbed image. Concretely, on ImageNet, pixel-space perturbations are typically constrained to have an ℓ_2 magnitude of at most 3 [213]. A model adversarially trained against perturbations constrained this way can be described as "highly robust" [219, p. 6]. Indeed, there exist pixel

perturbations with an ℓ_2 magnitude of 22 that can completely change the true class label of an image (refer to Figure 3 of Tramèr et al. [112], also reproduced as Figure 3.1 in this thesis). A magnitude of 22 would therefore be large for a pixel-space perturbation. For the ℓ_∞ metric, a maximum pixel-space perturbation magnitude of $\epsilon = 16/255$ is typical [213].

Our procedure performs perturbations to learnt feature representations, which then affect the downstream pixel values. Therefore, a small change to the output of early layers in the generator can result in a large change to the pixel values as measured by an ℓ_2 norm. But because these changes are context-sensitive to learned features, they preserve the meaning of the image. For example, suppose that a perturbation results in a dog moving position on a grassy background: although there is no change to the meaning of the image, the distance as measured by an ℓ_2 norm will be great, since many pixels will change value. In short, by leveraging generative models to direct changes to meaningful features, we can induce large changes in pixel space.

We investigate the empirical distribution of pixel-space distances between initial test seeds and final perturbed test inputs across 1000 initial seeds. Figure 6.2 shows that 100% of semantically perturbed test inputs are much further than the maximum pixel-perturbation constraint under either popular distance metric. Since the ℓ_∞ distance is the greatest amount any one pixel changes, there is a cluster around 1.0 because there is often at least one pixel that completely changes its value. By contrast, an ℓ_∞ pixel-space constraint of $\epsilon = 1.0$ is equivalent to no constraint: all pixels can change value arbitrarily.

6.2.2 Transferability analysis

We have established that pixel perturbation algorithms cannot generate the test cases output by our algorithm. In this section, we strengthen the case that furthermore, our algorithm is able to find *faults* that pixel perturbation approaches cannot. For faults concerning intelligible features, such as those deliberately introduced in the previous section, the case is clear: the faults involve visible

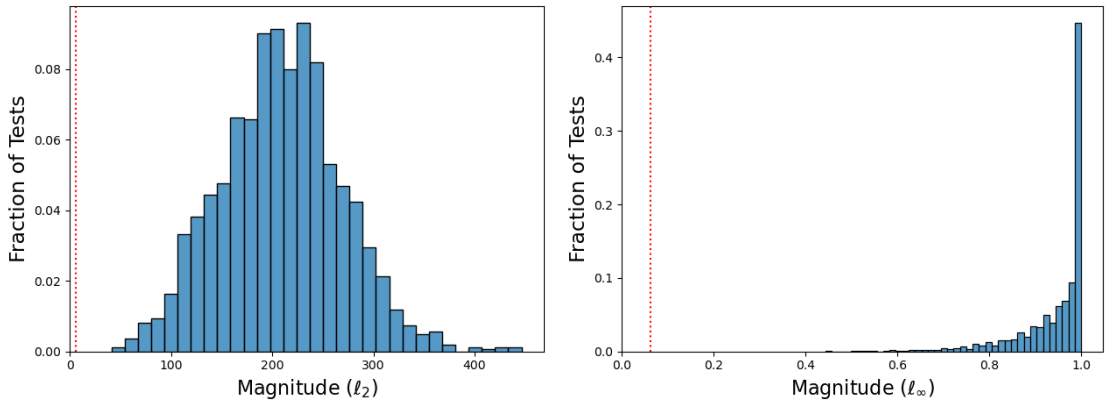


Figure 6.2: Magnitudes of perturbations produced by our procedure, as measured in pixel space using the standard ℓ_2 and ℓ_∞ metrics. In dotted red, a typical upper bound ϵ for pixel perturbations. The key point is that the changes induced by our perturbations result in a much greater pixel-space magnitude than this.

Table 6.5: The accuracies of pixel-perturbation robust classifiers on test cases originally generated for non-robust classifiers, using both pixel perturbations and our test generation procedure. The significantly higher accuracies on the pixel perturbation tests suggest that our approach detects faults of a different nature.

(a) Accuracies on tests originally for EfficientNet-B4NS.

		Pixel Perts	Our Perts
Test	Robust ResNet50 [213]	56%	27%
	Robust ResNet50 [214]	53%	24%

(b) Accuracies on tests originally for ResNet50.

		Pixel Perts	Our Perts
Test	Robust ResNet50 [213]	36%	25%
	Robust ResNet50 [214]	32%	22%

changes to meaningful features, and therefore these changes result in an ℓ_p distance greater than is allowed by pixel perturbations. In short, algorithms that generate imperceptibly different test cases are unable to detect faults concerning exclusively visibly different features.

However, most faults are plausibly not of this type; it is not immediately obvious whether pixel perturbations in fact suffice to detect almost all faults. Therefore, it could be possible that even though the particular test inputs generated by our algorithm and pixel perturbation algorithms are disjoint, they are both indicative of the same underlying faults in the DNNs, in the sense that they would

both be solved with the same fix. To demonstrate that this is not the case, we use *adversarially trained* DNNs [106]. Recall that adversarial training is a technique that performs worst-case pixel perturbations during the training of a DNN. When training converges, the result is that the DNN is more robust to these kinds of faults, and has learned to ignore the spurious features pixel perturbations affect. While this does not completely 'fix' sensitivity to pixel perturbations, it greatly improves it [213]. We check whether this 'fix' also applies to our perturbations.

We analyse whether the test cases generated *transfer* to models that have been adversarially trained to be robust to pixel-space perturbations. By "transfer", we mean that we measure whether test cases generated so as to induce a fault in (say) EfficientNet-B4 also induce faults in an adversarially trained DNN. Table 6.5 shows the proportion of test inputs for EfficientNet-B4 and a standard ResNet50 that are classified correctly by two DNNs trained to be robust against pixel-space perturbations: one by Wong et al. [214], which is robust to 31% of ℓ_∞ perturbations with $\epsilon = 4/255$, and one by Engstrom et al. [213], which is robust to 35% of ℓ_2 perturbations with $\epsilon = 3$.

We can see that pixel-perturbation tests generated for EfficientNet-B4 tend not to transfer to the pixel-robust models, likely because the faults found by the EfficientNet-B4 tests are not present due to the adversarial training. Conversely, we can see that the tests generated by our algorithm for EfficientNet-B4 *do* tend to transfer to the pixel-robust models. The results for ResNet50 are similar, although perturbations transfer slightly better, likely because the architecture is the same as the robust models. Note that all test cases confident, targeted tests: the difference in accuracy is not due to the pixel perturbations being only just misclassified.

Because the test cases generated by our algorithm continue to detect faults in adversarially trained classifiers, we have confidence that these must be detecting different kinds of faults to those detected by the pixel-perturbation algorithm. If the failing test cases were indicative of the same underlying faults, then we would see that the accuracies of the transferred test cases would be similar.

6.3 Threats to validity

The testing of DNNs is fundamentally different from the testing of conventional handwritten software, because of the training process: there is not necessarily any human-interpretable meaning to each 'line of code' (parameter value). It is therefore difficult to pin down exactly what a *fault* is in the context of DNNs, or to attribute a fault to any one cause. In this chapter, we first chose to deliberately introduce consistent biases into the DNNs' behaviours, and show that our algorithm is in turn able to consistently produce examples that highlight this bias. Doing things in this way clarifies what the fault is, and whether we have identified it. There exist numerous papers in which the model was shown to be wrong on a large number of inputs, but our experiments in Section 6.1 go beyond this by showing our ability to pick up on not just local, but global biases in the network (such as always relying on an erroneous feature). But these experiments are limited to our artificially biased networks; for normal classifiers, the seeming unintelligibility of their decision making and therefore faults represents a challenge that deserves significant future attention.

A second possible threat is the validity of the labelling done to produce Table 6.3. We hand-label whether perturbations have revealed the bias injected into the classifier, and while we take care to label perturbations without bias and according to a common-sense standard, there is some subjectivity involved. The labelling having been done blind to the direction of the perturbation (y_0 to y_1 or vice versa) will have removed desirability bias. And that the results for different models do in fact significantly vary should give confidence in the results. Many examples are given in Table 6.2 and the online supplementary material [216] hosted by the Oxford University Research Archive for the reader to review. Even under the worst case assumption that our judgements were somewhat biased, this bias is unlikely to be so pronounced that the key point no longer stands. Our algorithm is able to detect the faults at least some of the time, whereas existing algorithms cannot.

A third possible threat relates to problems with GANs. GANs are known to drop modes [220], meaning they may not generate certain parts of the input distribution. However, they need only represent enough of the distribution to identify at least some faults; our results show that they do. GANs are also not perfect generators, and so images may look unrealistic. In fact, realism is not required for our purposes. As long as a class is recognisable, we can still show that the classifier is paying attention to the wrong features. If a classifier can identify an unrealistic palace, but adding clouds in the sky changes its prediction to a castle, this betrays a problem in the classifier's internal 'logic'. In addition, if our aim is to create *human-aligned* classifiers, performance on unrealistic but recognisable images is important. Finally, our algorithm does not explicitly require a GAN, and could easily use a VAE or other generative model that does not drop modes. It is likely that recent rapid advances in generative machine learning will continue, making approaches that leverage it increasingly promising.

6.4 Conclusion

In this chapter, we have compared the ability of the context-sensitive perturbations introduced in Chapter 5 to detect faults in deep neural networks with that of existing pixel perturbation approaches. This comparison was made for models deliberately trained to include specific known faults that were then uncovered using perturbation test generation, and also for state-of-the-art ImageNet classifiers. In both cases, we have seen that the new procedure is able to detect faults that existing approaches cannot. This is possible because our procedure leverages generative machine learning, allowing it to manipulate higher-level features of generated test inputs (e.g. position, colour, texture of objects) rather than just low-level features (i.e. individual pixel values). As a result, the generated tests are much more varied and can explore weaknesses not reachable when changes are constrained to be within a small ℓ_p distance in pixel space.

Of course, we do not expect that our procedure will be able to detect *all* faults in a given system. But exploiting features learned from data during test input generation seems a promising approach worthy of future investigation. More generally, we encourage future work that seeks to meaningfully broaden the set of faults detectable by our tests. In addition to this bottom-up approach, top-down attempts to identify a superset of the requirements for a DNN might also be worth investigating.

In general, most of the procedures for producing test-cases for DNNs either only implicitly, or do not at all, address some of the main issues in testing DNNs. To test DNNs in practice, it will become increasingly important to provide specifications, consider a DNN's behaviour as part of a larger system, and pin down how to identify and correct faults [221]. Unlike conventional software, for which debugging tools allow direct inspection of the program fault, a DNN cannot be meaningfully inspected by a developer. Even if it could, there is little hope of trying to manually adjust the weights of a trained DNN. Instead, developers act on DNNs indirectly, through training code and data. Since all faults are mediated through this opaque training process, it is difficult to link a DNN failure to an action that might introduce a fix. We encourage future work that aims to make such diagnostic debugging possible, either by directly debugging training code and datasets, or by analysing the link between the trained DNN and these training artefacts.

7

Adversarial Training Can Worsen Generalisation

Contents

Recall from Section 2.3.2 that the most promising existing approach to improving models' out-of-distribution generalisation is adversarial training. By including examples during training that have had worst-case perturbations applied to their pixels, a model's robustness to ℓ_p constrained pixel perturbations can be greatly increased.

So we can use this technique to improve a deep neural network's ability to generalise to certain classes of inputs that were not part of the original training task. We might hope that improving the model's performance on this class of out-of-distribution inputs might improve its ability to generalise on other kinds of out-of-distribution input. In the conceptual framework of shortcut learning [6], we might hope that because adversarial training forces the model to perform

well on a wider class of inputs, it is less likely to choose a shortcut proxy that will not perform well on other inputs we care about.

This chapter presents a novel empirical finding: using adversarial training to increase a model's robustness to pixel-space perturbations in fact *worsens* that model's ability to generalise to at least some kinds of out-of-distribution data. In particular, it seems that adversarial training may make models unduly sensitive to and dependent on higher-level features at a coarser level of granularity, such as object position, shape, orientation or colour. This is striking because it casts doubt over the promise of adversarial training as a general solution to improve models' capabilities beyond the specific original training task.

This result was found using the context-sensitive perturbation procedure introduced in Chapter 5 – so it is arguably evidence that it can be used to identify new problems. The present chapter presents a series of experiments that compare the performance of standard and pixel-robust ImageNet classifiers under context-sensitive perturbations of different granularities. Unsurprisingly, we easily find perturbations of all granularities that cause each model evaluated to output an incorrect prediction. As we might expect, the classifiers made robust to pixel-space perturbations through adversarial training were found to generalise better to fine-grained perturbations. But those same classifiers were significantly *less* able to generalise in the presence of coarser-grained perturbations to high-level features. This may be because such classifiers must necessarily depend more on coarser-grained features of images than classifiers optimised for accuracy on i.i.d. inputs, which tend to rely on fine-grained features such as texture [218]. Our results strengthen and expand upon related findings from [183], who find that classifiers robust to pixel-level perturbations are less robust to corruptions of certain context-insensitive features such as artificial 'fog' and 2D sinusoids.

We also perform additional experiments with the MNIST dataset, described in Section 7.4. The simplicity of the MNIST classification task suggests that constructing a robust classifier for MNIST should be significantly easier than for ImageNet. We find that adversarial training against pixel perturbations does not

improve robustness to coarse-grained perturbations on MNIST, but neither does it worsen it. This is likely because the simplicity and low resolution of the dataset significantly reduces the range of possible granularities, relative to ImageNet.

7.1 Experimental setup

The experimental setup is identical to that described in Section 5.2 – the results presented in this chapter come primarily from the same run of ImageNet experiments, but now with a different focus.

A classifier is more robust to a class of perturbations if larger magnitude perturbations of that kind are required to induce the targeted misclassification. Recall that the optimisation procedure we use gradually increases the magnitude of the perturbation to activation values with each step. By measuring the smallest magnitude for which the classifier outputs the target class t, we can build a picture of how robust a given classification model is to the perturbations being used.

As a quick summary of the experimental setup, the generative model used is a pre-trained high resolution BigGAN [34]. We evaluate four classifiers. First, two standard classifiers: the state-of-the-art on ImageNet, EfficientNet-B4 with NoisyStudent training [211], along with the standard ResNet50 classifier [205]. Second, two 'robust' ResNet50 classifiers adversarially trained against pixel-space perturbations: one from Engstrom et al. [213] trained using an ℓ_2-norm PGD attack with radius $\epsilon = 0.3$, and another from Wong, Rice, and Kolter [214], trained with the FGSM attack for robustness against ℓ_∞ with $\epsilon = 4/255$. We use only those unperturbed images that are correctly classified by the model under test. In addition, we use a majority vote of five human judgements to eliminate those unperturbed images that do not actually belong to the intended class. This same procedure is then applied to determine whether the perturbed image has retained its original intended class (while being misclassified as the target class by the classifier). Reference Section 5.2 for the full details. The key difference is that we are now measuring the perturbation magnitudes necessary

Table 7.1: Mean magnitudes of the perturbations that were sufficient to induce the target misclassification for various classifiers (rows) and layers in the generator at which activations are perturbed (columns). Compare results within each column to compare robustness to each perturbation granularity.

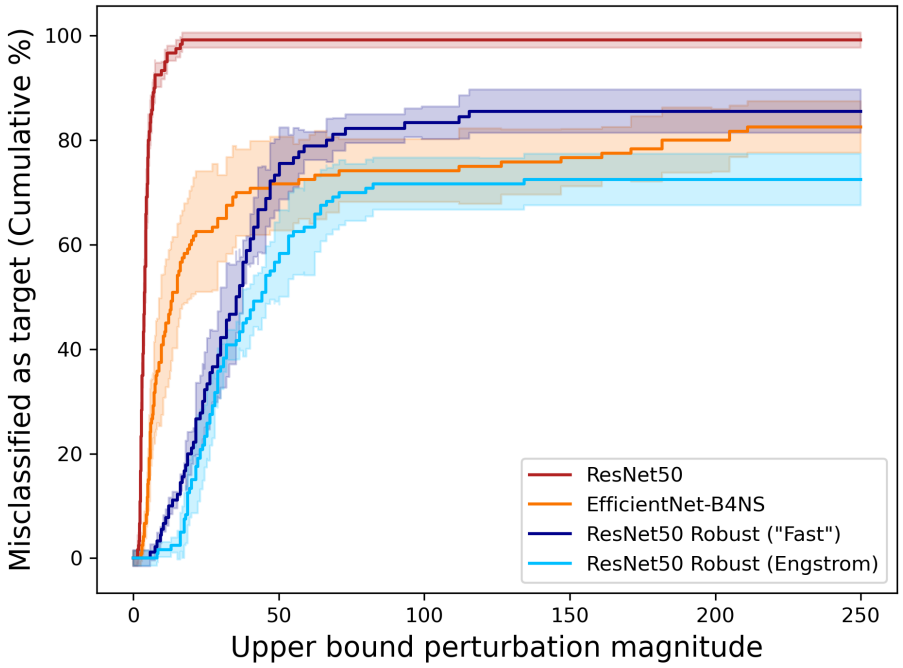
	All layers	First 6	Middle 6	Last 6
ResNet50	4.2	89	4.2	7.4
EfficientNet	36	97	22	24
ResNet50 Robust ("Fast")	35	29	22	102
ResNet50 Robust (Engstrom)	36	33	21	141

in each case as a measure of each model's ability to generalise correctly to each kind of perturbation.

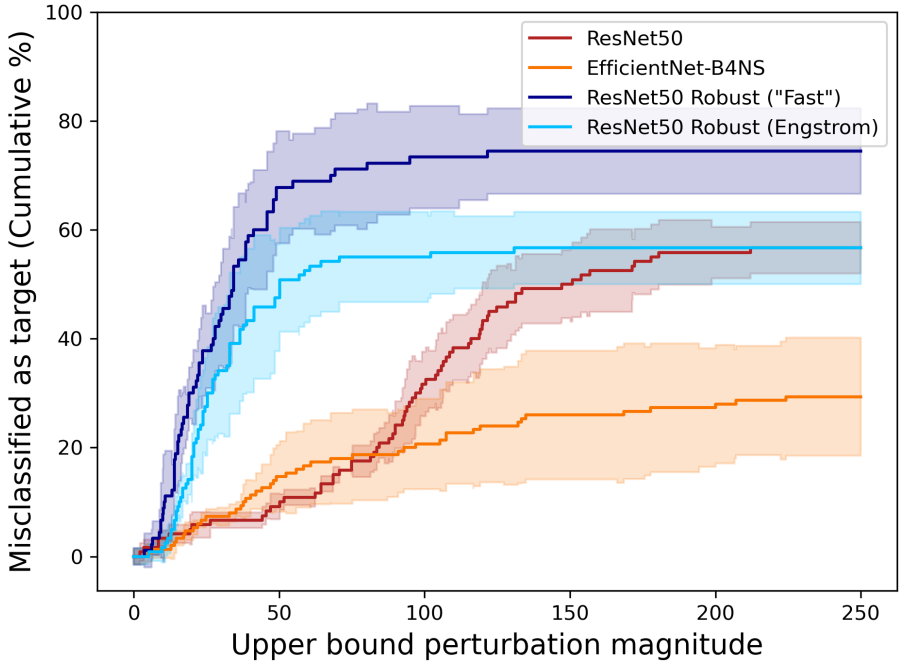
7.2 Results

Table 7.1 reports the average magnitude of the misclassification-inducing perturbations. Figure 7.1 elaborates on this, plotting the relationships between perturbation magnitude and the cumulative proportion of inputs for which this magnitude (or smaller) is sufficient to cause each classifier to predict the target class. The steeper the gradient of a line, the less the corresponding classifier is able to generalise well to that perturbation type.

We separate the images into several 'experiments' of 30 test cases so that some measure of the certainty of these results can be expressed – it is helpful to be confident that we are reporting robust results that are not down to the quirks of the specific cases being used. The lines and translucent areas shown in Figure 7.1 are the means and standard deviations between the various experiments of 30 images each. For each type of perturbation, for each classifier, 192 ± 20 (min. 158) unperturbed images were labelled by the human judges, of which 53 ± 8 (max. 69) images were rejected by the majority for not matching the intended label. Note that this latter quantity depends only on the pre-trained generator, not our procedure.

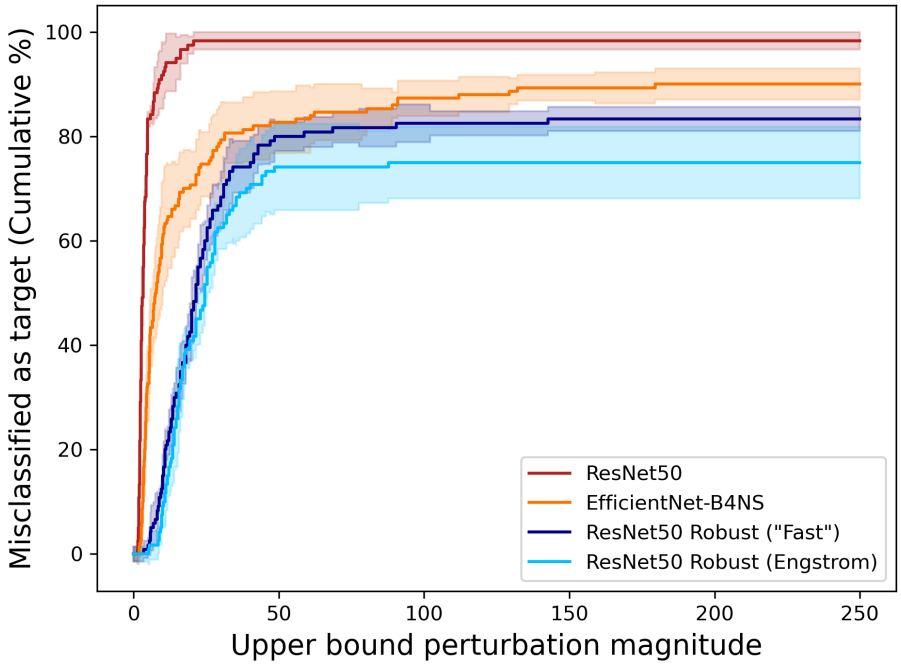


(a) Activation values perturbed at all BigGAN layers.

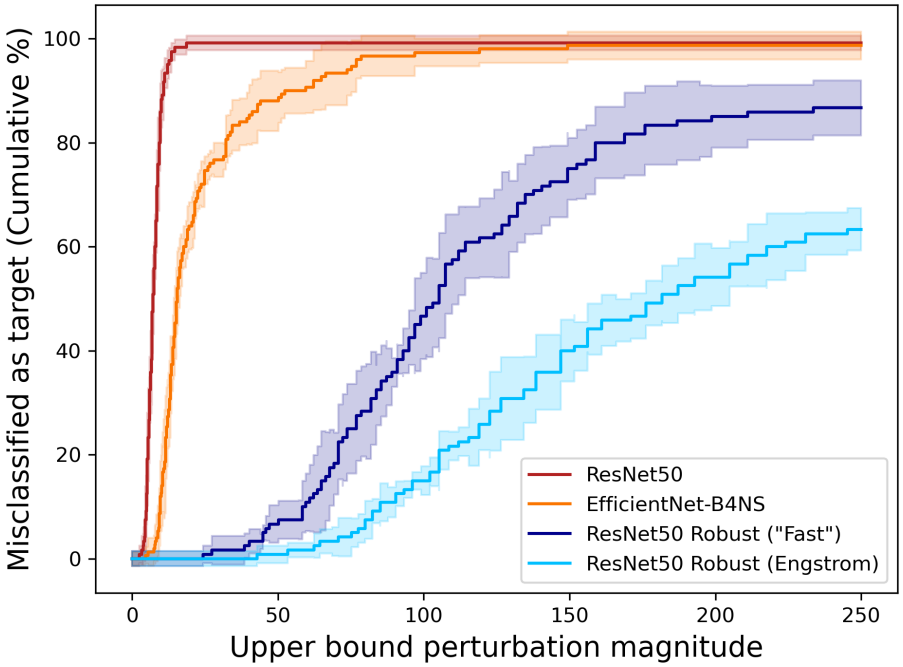


(b) Activation values perturbed in the first 6 layers only.

Figure 7.1: Cumulative proportion of perturbations inducing the targeted misclassification as a function of maximum perturbation magnitude.



(c) Activations perturbed in the middle 6 layers only.



(d) Activation values perturbed in the last 6 layers only.

Figure 7.1: Continued.

7.3 Discussion

7.3.1 Expected results

As expected, and as seen in Chapter 5, these results show that the test generation procedure finds tests that successfully induce targeted misclassifications in all four classifiers.

Also as we might expect, the results show that the pixel-space robustness conferred by adversarial training against ℓ_p constrained pixel perturbations improves a model's ability to generalise in the presence of finer-grained, localised perturbations. This can be seen in the lower average magnitudes required for the pixel-robust classifiers when perturbing the final six layers, as in the last column of Table 7.1 and the correspondingly flatter curves in Figure 7.1d. The slightly gentler gradient at the beginning of Figure 7.1c suggests that adversarial training even provides some limited robustness to small perturbations of medium granularity. In both cases, this may be because the changes fall within or nearby the pixel-space ℓ_p-norm ball that the classifier is trained to be robust within, or may be because adversarial training incentivises reliance on features that generalise better even outside this radius.

7.3.2 Key finding

The main implication of our results is that standard adversarial training, while conferring improved generalised in some ways, is a double-edged sword. Perturbations to activations in the early layers of a generator induce context-sensitive, coarse-grained changes to the features of an image. These have a large magnitude when measured in pixel space, so it is unsurprising that classifiers trained to be robust to norm-constrained pixel perturbations do not have improved robustness to such feature perturbations. More surprisingly, Figure 7.1b and Table 7.1 show that pixel-space robustness in fact considerably *worsens* robustness to the coarse-grained features encoded in the first six generator layers: the adversarially trained

classifiers required a much smaller perturbation magnitude in this case before being made to output the target misclassification.

This may be because non-robust classifiers ordinarily depend mainly on fine-grained features like texture [218]. Conversely, pixel-space robust classifiers have been trained to ignore these fine-grained features, and so depend instead on coarse-grained features. But they can still rely on fragile correlations in the coarse-grained features, so their robustness to context-sensitive coarse-grained feature perturbations is decreased.

As discussed in Section 3.6, there has been a small number of related results suggesting that adversarial training may sometimes be unhelpful, but these previous works concerned more limited possible features changes, such as low-frequency 2D sinusoidal perturbations. Our finding strengthens and generalises these results, using the context-sensitive feature changes induced by our latent perturbations that seem more similar to the kinds of feature change for which robustness may matter in practice.

7.3.3 Implications

We have seen that using adversarial training to improve a deep neural network's ability to generalise well to images perturbed using pixel-space perturbations significantly *worsens* its ability to generalise well to images perturbed at a coarser, higher level of granularity using the context-sensitive approach introduced in Chapter 5. From this, we can draw specific and broader conclusions.

Specifically, despite the large amount of attention given to the pixel perturbation paradigm in the literature, we should not expect that solving the ℓ_p constrained pixel robustness problem will solve or even necessarily help with our broader desire to develop trustworthy models. Even if a model has been trained to obtain pixel-space robustness, we should not expect its generalisation or robustness to be improved in any other domain. We especially should worry that it may be *less* well suited to deal with macro-level changes it may encounter.

But we can also draw a broader lesson from this finding. It seems that a model's ability to generalise outside its training distribution cannot be boiled down to a scalar in a single dimension. Instead, it seems that improved ability for one kind of out-of-distribution data says nothing about performance on another kind. And it is unclear what is even meant by a 'kind of data' here.

If we are to develop trustworthy models, there are two approaches that we could take. One approach is to precisely characterise exactly which regions of input space are those that we care about, then to develop models that verifiably perform well on those inputs. The problem with this approach is that it is very unclear which inputs we in fact require good performance on. From a safety perspective, we need good enough behaviour on any region of input space that could plausibly be input at deploy time. (If security is also considered, the set of necessary inputs grows significantly; depending on the exact threat model used, an adversary may be able to produce inputs that would not arise otherwise. The presence of an adversary may also affect the standard of performance required – whereas 99% correctness may suffice for safety purposes, an adversary could exploit the 1% failure rate and may be able to always find one of the inputs from this category to cause a failure.) But how can we know what inputs might be encountered at deploy time? The world is constantly changing. Humans are constantly generalising to new situations that would have been very difficult to predict in advance.

Indeed, for this reason, it is tempting to take the other approach to develop trustworthy models. This assumes a humble view regarding our ability to predict the kinds of inputs that we might eventually care about, and instead aims for human-level performance on *any* input that might be encountered. Put another way, a model is trustworthy in this view only if it does not get things wrong that a human would get right. This is appealing because such a system would clearly be trustworthy, because of the low bar for the evidence needed to show that a system is untrustworthy, and because we might use an analogy with human cognitive processes as inspiration for engineering innovations. The downside

is that this is a high bar, perhaps as high as AI-completeness for some kinds of tasks. It may also be difficult to show that a system meets this bar – failure to find a failure case does not imply that none exists – so further development of procedures for the evaluation of deep learning systems is essential.

Although both of these approaches seem difficult, the empirical result in this chapter makes it clear that we cannot hope to simply ratchet up a model's general out-of-distribution generalisation until it seems to be sufficiently good. Excellent performance, even on all kinds of data we can think to test it on, will not be sufficient for trustworthiness unless we are confident that this enumeration is in practice exhaustive.

7.4 Additional dataset: MNIST

In addition to the ImageNet experiments, we performed additional smaller scale experiments using the much simplicity MNIST dataset [198], comparing a standard classifier with an adversarially trained classifier. Its low resolution and small number of straightforward classes mean that it is particularly easy to robustly classify, and so a particularly challenging case to generate tests for.

7.4.1 Model details

For MNIST, we tried a range of generators and found that they all worked roughly as well as one another. For the experiments, we use the same simple convolutional generator, inspired by the Deep Convolutional GAN [28], as was used in Chapter 4 – architecture shown in Table B.1 in the appendix. Inputs to the generator are drawn from a 128-dimensional standard Gaussian. The sigmoid output transformation ensures that pixels are in the range $[0, 1]$, as expected by the classifiers. We perform context-sensitive perturbations before ReLU layers, rather than after, to prevent ReLU output values from being perturbed to become negative, which would not have been encountered during training and so may not result in plausible images being generated since they are out-of-distribution for the rest of the generator. Note that perturbing before and after the sigmoid

transformation has different effects because perturbations to values not close to 0 are diminished in magnitude if passed through the sigmoid function.

We use two neural networks that classify MNIST. One is a simple standard classifier with two convolutional layers and three fully-connected layers, trained to give an accuracy over 99%. The other is an LeNet5 classifier adversarially trained against l_2-norm bounded perturbations for $\epsilon = 0.3$. This was trained using the AdverTorch library [222]. It has a standard accuracy of 98%, reduced to 95% by an l_2-norm bounded attack with $\epsilon = 0.3$.

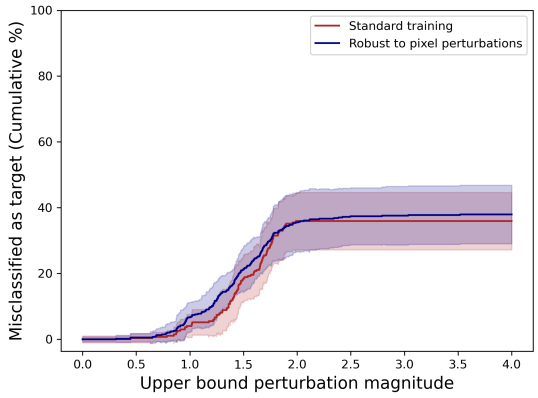
7.4.2 Experimental setup

We find context-sensitive feature perturbations as in the primary ImageNet evaluation, with a few differences. First, since the generator is much smaller, we divide it nearly in half, comparing the effect of perturbing the activation values at first four layers only with the effect of perturbing at the last four layers only. Second, because MNIST (and the generator) is much smaller, so are the perturbation magnitudes required. So the learning rate is reduced to 0.004, the we start with an initial perturbation magnitude constraint of 0.1, and this is gradually relaxed after each optimisation step by increasing this upper bound by 0.001. The procedure for collecting human judgements is repeated as described in the main ImageNet evaluation.

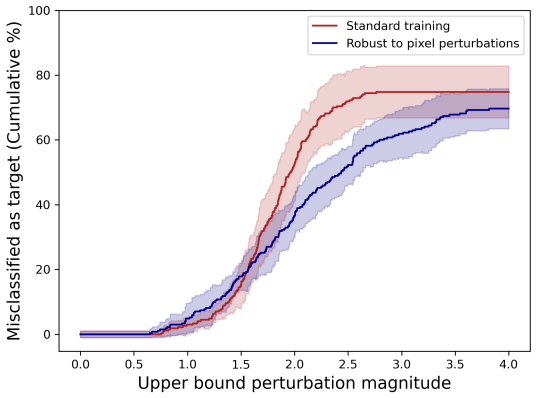
7.4.3 Results and discussion

Figure 7.2 shows the robustness of the two classifiers to the two kinds of context-sensitive perturbation. We can see from Figure 7.2b that the classifier trained to be robust to pixel-space perturbations is indeed more robust than the standard classifier, with its considerably shallower slope indicating that a bigger perturbation magnitude is required to the finer-grained features encoded in the last four layers of the generator.

Conversely, Figure 7.2a gives an almost-identical shape for both classifiers, indicating that adversarial training against pixel-space perturbations does not



(a) Generator activations perturbed at first 4 layers only.

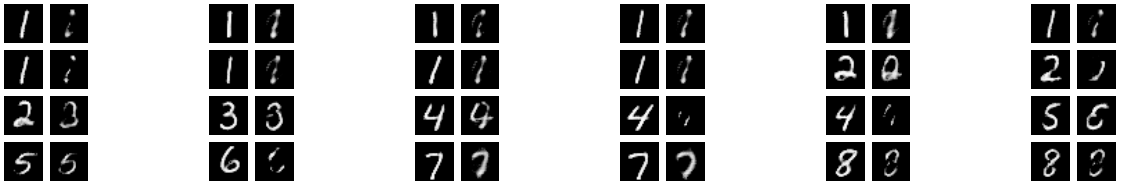


(b) Generator activations perturbed at last 4 layers only.

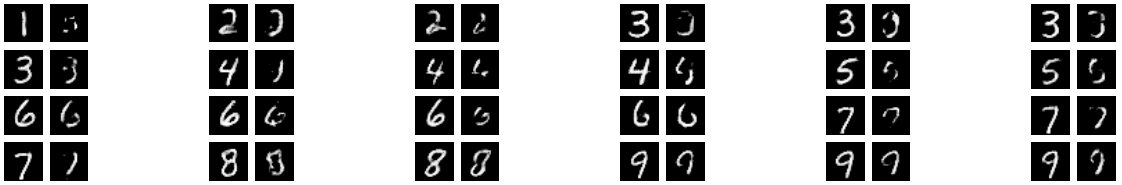
Figure 7.2: Graphs showing how the proportion of perturbations that induce the targeted misclassification increases with perturbation magnitude. These should be interpreted in the same way as Figure 7.1. The solid lines exclude the perturbed images for which a human judges that the perturbation does not change the true label of the image; the dotted lines, for reference, include these.

confer any robustness to the coarse-grained feature perturbations being evaluated here. This has an interesting relationship with our main finding, that adversarial training against pixel-space perturbations seriously harms robustness to high-level context-sensitive features perturbations on ImageNet. The difference can likely be best explained by the great difference in datasets. The dimensionality of ImageNet is over 1000$\times$ greater, and the high-level feature space of MNIST is trivially small in comparison. The result of this is that in some loose sense, there is a smaller 'gap' between the highest- and lowest-granularity features encoded in the generator; put another way, there is a much less rich space of coarse-grained context-sensitive features that a classifier must be robust to on MNIST.

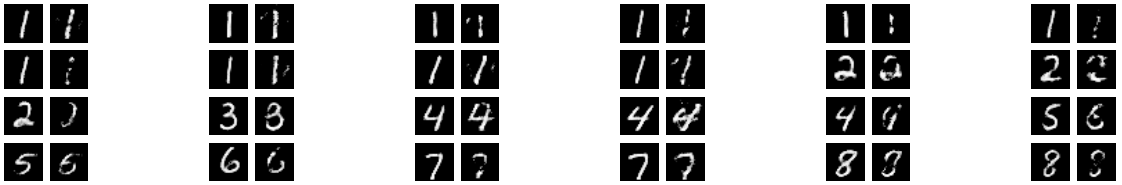
Whether this is the correct intuition or not it is clear that even on the very simplest datasets, robustness to fine-grained features completely fails to generalise to coarser-grained features.



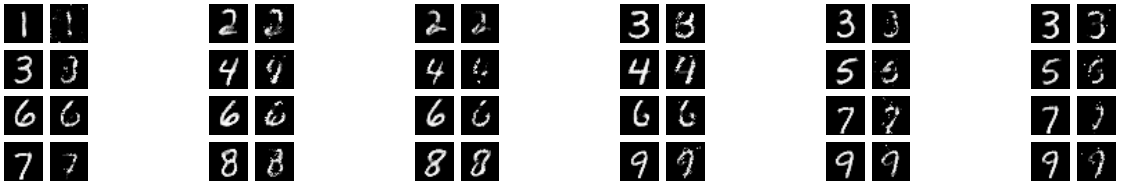
(a) Standard classifier, first four generator layers perturbed.



(b) Adversarially trained classifier, first four generator layers perturbed.



(c) Standard classifier, last four generator layers perturbed.



(d) Adversarially trained classifier, last four generator layers perturbed.

Figure 7.3: Random sample of generated test cases targeting label 0. Only the first or last four layers of generator activations are perturbed. In each image pair, the perturbed image is to the right of the unperturbed image. Note that the proportions of perturbed images for which the original class is maintained is reflected in the graphs in Figure 7.2.

7.5 Threats to validity

7.5.1 Internal validity

One possible threat to internal validity is the use of human labelling to determine whether a given perturbed test input retains the same true label as its unperturbed seed. This was discussed in Section 5.4. In short, the use of several labellers and the difficulty of knowing how to label so as to create 'better' results mitigate much of the concern. Note in particular that there was no way of knowing the magnitude of the perturbation, nor which classifier was being tested, making it very

unlikely that the results in this chapter are erroneous because of biased labelling.

Another possible threat is the metric that we used as a proxy for the classifiers' ability to generalise well in the presence of different kinds of perturbations. As a reminder, the proxy used was the ℓ_2 magnitude of the perturbations vector applied to the latent activations that caused the tested classifier to output the targeted misclassification. One might worry that this metric says more about the particular optimisation procedure than properties of the classifier. But there is no particular reason to expect this – the classifiers were not deliberately designed so as to cause gradient walking perturbation algorithms to behave poorly (unlike the classifiers mentioned in Athalye, Carlini, and Wagner [95], for instance), so the difficulty of the algorithm finding a suitable perturbation should correspond to how well the classifier performs in the region local to the test seed.

One last threat to mention is the number of ImageNet classifiers used. Although the relevant patterns in the two standard classifiers and two robust classifiers seem clear and reliable, and there are plausible mechanisms for the differences between them, it cannot be ruled out that these are, by coincidence, only properties of these particular models, rather than underlying differences resulting from standard versus adversarial training.

7.5.2 External validity

Although the results from the ImageNet experiments are unambiguous in their conclusion that the adversarially trained networks perform worse on perturbations causing changes to higher-level features, this result is much weaker on the MNIST dataset. On MNIST, the result was that adversarial training did not improve performance under these features, but neither did it worsen it. While there are reasons to expect this to be more likely on MNIST – the simplicity of the dataset makes classifiers particularly strong, and means that the highest-level feature perturbations are not so different from low-level pixel perturbations – it does raise a question about the situations in which the ImageNet result generalises. As ever, a higher powered study with more datasets, classifiers

and test cases would of course be desirable. This could perhaps be tied in with future work that investigates the general question of the situations and ways in which adversarial training is and is not helpful. At the very least, we can be quite confident that our main result is likely to point out a general concern with adversarial training – it would be very surprising if there were something specific to the ImageNet dataset that caused this result to arise only here.

8 Conclusion

Contents

8.1 Summary of research

Ideally, we would like to trust our models to perform well in new situations; unpredictable changes in the world together with inevitable limitations in the gathering of data mean that systems that are deployed will, over time, surely encounter data different in some way than the training data. But if a model is optimised only for performance on its training task, it has no reason to avoid learning proxies that are useful on the training task, but harmful in related situations. In practice, it seems that deep neural networks *do* learn such 'shortcut' proxies.

The research presented in this thesis aims to probe the ability of deep neural networks to correctly generalise beyond the distribution of images they were trained on. We have introduced two new algorithms that generate test

cases for image classification models, aiming in particular to meet the following requirements:

1. to generate test cases that:
   (a) are well-formed inputs that are assigned meaning by the oracle for the task, and
   (b) induce failures, so the system being tested gives an incorrect output,
2. to detect as wide range of such failures as possible, and
3. to be efficient and practical.

By improving our ability to identify failures, these algorithms hope to better inform model deployment decisions, to develop our understanding of the faults in deep neural networks caused by the learning of unhelpful shortcut proxies, and eventually contribute to our improved ability to develop trustworthy systems.

8.1.1 Training generative networks to generate tests

The first DNN testing algorithm, presented in Chapter 4, finetunes the generative network of a GAN pair so that it generates data that are not only meaningful, but also induce failures in the specific model being tested. After algorithmically filtering generated examples to only those that the classifier mislabels as desired, empirical evaluations show that the overwhelming majority of these examples (90% for the untargeted standard MNIST classifier, similar for targeted tests, 80% for adversarially trained classifier) are assigned the intended label by human labellers. So the algorithm meets our first requirement.

It also meets our second requirement: rather than being constrained to make only small perturbations to a finite set of existing test inputs, this algorithm is free from constraints on the test cases it can generate. Empirical experiments (including a transferability analysis and a comparison of behaviour under adversarial training) support the view that this algorithm can identify many inputs that existing approaches could not.

There are some senses in which this algorithm is efficient – after finetuning the generator, production of many test cases is very cheap. But the computation of finetuning a generator could prove to be a comparable cost to the training of the model being tested, which would not be prohibitive, but not cheap either. This computational cost and the limited quality of publicly-available generative models resulted in some ambiguity in our empirical evidence about the ability to scale up to ImageNet.

8.1.2 Perturbing activations in generative networks

The second new DNN testing algorithm, presented in Chapter 5, uses a fixed, pretrained generative network that can generate meaningful inputs to an image classifier. Perturbations dynamically applied to the activation values in that generative network optimise the generated image so that it induces a failure in the model being tested. Empirical experiments show that these perturbations (a) are small enough changes to preserve the meaning of the original image, so that our first requirement is met, and (b) manipulate the features learned by the generative network, which are high-level features such as object position or colour in earlier layers, and finer grained and more localised in later layers.

Chapter 6 evaluates not only whether this algorithm is able to detect failures that other approaches cannot, but whether it is able to detect *faults* that existing approached cannot. By 'fault', we mean the underlying cause of the image classifier giving an incorrect answer, most likely caused by the learning of a shortcut proxy that fails to generalise. Experiments in which image classifiers had incorrect shortcut proxies deliberately introduced found that this algorithm was able to detect the problems, whereas existing approaches would not be able to. And experiments involving an analysis of how test cases behave on different kinds of models suggest that the failures detected by our new algorithm must be detecting faults in state-of-the-art models that are not detectable using existing approaches. So the algorithm meets our second requirement.

It also meets our third. No additional training is required given a suitable pretrained generator. (If such a generator is missing, then its creation would be expensive, but comparable in cost to the development of the discriminative model being tested.) Instead, creation of a new test case involves optimisation of the perturbation to the latent activation values in the generator. On a standard GPU, this takes about one minute.

8.1.3 Empirical finding

From a comparison of how different models behaved under testing from our second new algorithm, we made a surprising empirical finding. Experiments presented in Chapter 7 show that image classifiers that have been adversarially trained to improve their robustness to small pixel-space perturbations are in fact significantly *less* able to generalise correctly to inputs that have had perturbations made to higher-level, coarser-grained features (for example, object shape, orientation or colour).

8.2 Significance of contributions

We have established that this thesis has introduced two new practical test generation algorithms that are both able to detect problems in deep neural networks that existing approaches cannot. Both exploit generative networks to this end: the first training such a network to generate useful tests, and the second manipulating its learnt representations. So what? That is, what is the significance of this research for the wider field?

One update for researchers might be that imaginative applications of generative machine learning can result in useful tools that are likely to form at least part of the suite of approaches used in testing and evaluation. It is unclear whether such approaches will be sufficient in themselves, but they are at least complementary to existing techniques. Generative learning allows data to be leveraged to generate test inputs that otherwise may not be possible to

automatically identify as suitable – the prominence of the pixel-space or manually-programmed perturbation is evidence of this. We recommend that further work be done in developing testing and evaluation techniques that build on our work. Section 8.3 contains several specific suggestions for future work along these lines.

The original motivation for this work was that improving our ability to detect failures and faults in deep neural networks would allow us to better identify, understand and address their limitations regarding generalisation beyond the training distribution. There was also a hope that such work might lead to improved conceptual clarity about the specific properties we actually desire or require from these systems. Any progress on these issues would be significant for the wider field, so let us evaluate each in turn.

8.2.1 Identifying problems to inform deployment decisions

If our tools to identify relevant shortcomings in machine learning systems are limited, then we necessarily must make decisions about how much trust to place in these systems with limited information. Currently, we know how to evaluate model performance under the assumption that the distribution of data during deployment will be the same as that during training. But our understanding of how models will perform when this assumption is not true is much less mature – and in almost all practical situations, this assumption cannot be relied on.

This thesis has contributed new algorithms that allow us to identify at least some problems in deep neural networks that were not previously identifiable. Therefore, one significance of this work is that it improves our ability to evaluate a machine learning system for the purpose of deciding whether to deploy it. Of course, it is unlikely that the software tools we developed during this research will be directly used for this purpose. But the more general point stands: by expanding the range of possible problems that can be identified, this research makes it more likely that future decisions to deploy or not deploy a model will be better informed thanks to analysis from a tool whose development can ultimately be traced in small part to the ideas in this thesis.

8.2.2 Better diagnosis to improve our understanding of generalisation

Another motivation for improving our ability to detect models' generalisation failures is that we might improve our understanding of the causes of these failures. That is, if several individual models each have their limitations analysed, then general patterns may be drawn out; we should expect improved diagnosis of problems to lead to improved understanding of those problems.

In this thesis, we have one instance of such an insight arising from our improved tooling to probe the limitations of deep neural networks: the result described in Chapter 7. Although we know that adversarial training improves out-of-distribution generalisation for the same kinds of data that are used during this training, we found that adversarial training against constrained pixel-space perturbations *decreases* models' ability to generalise to high-level, coarse-grained changes. The most straightforward implication of this is that adversarial training should not be treated like the holy grail. It is not the case that "out-of-distribution generalisation" is a simple metric that is increased by adversarial training – rather, models perform differently well on different kinds of data depending on what they have learned, and adversarial training simply incentivises good performance on the specific kind of data used.

If we are lucky, our empirical finding could just be a bump in the road. The best-case interpretation of the result is that standard training causes models to learn fine-grained pixel-level shortcuts; that pixel-perturbation adversarial training prevents this and causes models to learn coarse-grained shortcuts instead; and that there exists a simple form of adversarial training that counteracts both fine- and coarse-grained shortcuts, thereby causing the desired features to be learned instead of shortcuts.

But this interpretation seems overly optimistic. In the worst case, we could imagine a world in which we repeatedly identify the shortcut features learned, and include these in our adversarial, but a new kind of shortcut pops up. Such

a situation would feel like a game of "whack-a-mole", in which dealing with one problem causes another to arise, with no end in sight.

More likely, the true situation is neither the best-case nor the worst-case interpretation. In any case, an implication of the work in this thesis is that it would be valuable to conduct experiments with different kinds of adversarial training to determine whether it is possible to reduce a model's overall reliance on shortcuts, or whether adversarial training is only able to change the kinds of shortcuts that the network learns.

To the extent that the result in Chapter 7 makes us pessimistic about adversarial training as a general solution to out-of-distribution generalisation, we must either characterise the kinds of data to which we require generalisation so that they can be used in adversarial training, or turn our attention to developing new techniques that may improve out-of-distribution generalisation in general.

So the development of algorithms that improved our ability to detect problems has led to one instance of developing our understanding of out-of-distribution generalisation. This is certainly one source of wider significance for our work. But it should be noted that this is still one relatively small development in understanding – it should not be overstated how much the present work has led directly to increased knowledge.

8.2.3 Conceptual clarity

If we cannot specify what we want from our machine learning systems, then we cannot evaluate how successfully a particular model has satisfied our desired properties. There is of course consensus that a model should perform well on its training task, including on a hold-out test set drawn from the same distribution as the training data.

But the field of machine learning remains largely confused about what other properties we require. To which kinds of 'out of distribution' data do we require generalisation before trusting a system in practice? Is performance narrower than human capability acceptable, or do we require generalisation at least as

good as a person? What is a fault in a machine learning system, and which faults are tolerable? How does application domain affect our requirements? How can success be measured? Is security or safety a greater concern?

One hope was that our research would lead to improved conceptual clarity about the properties we require from deep neural networks – with improved ability to diagnose problems, perhaps empirical insights into how best to characterise the kinds of data we would like good performance on might arise.

There is perhaps one small contribution that this thesis has made in this respect: by more solidly establishing that adversarial training can decrease generalisation to some kinds of data while increasing it to others, it has become clearer that out-of-distribution generalisation is not a scalar metric that is monotonically increased by interventions that aim to do so. Instead, there may be (at least initially) trade-offs between a model's performance on different kinds of data, which suggests that it is necessary to characterise the kinds of data to which generalisation is required (currently a vague and underspecified task that is itself conceptually unclear).

But this insight is small, and was plausible even without our empirical evidence. So improved conceptual clarity is only a minor source of significance for this work.

8.2.4 Development of improved models

The last motivation to discuss is the hope that better diagnosis and understanding of faults might lead to the development of improved models. The research in this thesis has not directly led to any improvements in model architectures or training processes, so this is not a major source of significance.

That said, it is possible that our contributions indirectly lead to the development of improved models in the future. By allowing detection of a wider range of problems, developers may be made aware of shortcomings in their models that they were previously unaware of, and so attempt to resolve these;

being able to measure these shortcomings can also give feedback to developers about their progress.

Whether diagnosis tools can offer greater help to developers than this depends on the nature of the faults uncovered. Whereas a fault in conventional software is the direct result of a human error that can be understood and often straightforwardly rectified, that need not be the case for machine learning models. The final trained model is a product of the model architecture, training algorithm (and its implementation), hyperparameters, training data, and even random seeds used. Programmer mistakes in the training or model code are likely to result in faults in the trained model.

But the more subtle and perhaps more prevalent source of faults in the end product is simply that the training process was never properly aligned with what we wanted, either because we were confused about what we wanted from the model, or more likely because we never tried to specify what we wanted in the first place. These faults may or may not be intelligible to humans. If they are, and it is possible to understand what is going wrong at a higher level of abstraction than simply the various multiplications and additions constituting a forward pass through the network giving the wrong answer, then this understanding could lead to insight into what to do to rectify the problem and improve the network. If they are not, and perhaps 'fault' is not even a helpful concept when it comes to neural networks, then it will not be possible to help developers by identifying such particular problems that they can understand. This remains an open question.

8.2.5 Summary

In short, our new test generation algorithms allow the identification of failures and faults that could not be found using previous approaches. This is significant because these algorithms, and any that build on them, provide a better understanding of whether particular models should be trusted – decisions that are currently made largely in the dark. The success of these algorithms

provides a useful indication that the application of generative machine learning to test generation is fruitful, and provides a starting point for any future research that seeks to further capitalise on this. More speculatively, we might expect an improved ability to identify problems with deep neural networks to lead in time to greater understanding of the causes of generalisation failures and so eventually to improved models. We might also learn to better specify the properties we wish a model to satisfy.

By testing and comparing different models, we have found evidence suggesting that adversarial training with one kind of data can significantly decrease generalisation performance to other kinds of data, a result that has not yet been firmly established. This finding has further implications of significance: it implies that models that generalise well on all data we care about require us to either seek an alternative to adversarial training as it is currently used, or to ensure that all relevant kinds of data are included in the adversarial training.

8.3 Building on this thesis

While the previous section made some general and broad suggestions for the field, this section describes some specific suggestions for possible future work that builds on the contributions made in this thesis.

8.3.1 Next-generation generative models

In parallel with the research conducted for this thesis, the state of the art in generative models has progressed. Whereas we have used class-conditional GANs, that is generative models able to generate examples from a number of discrete classes, the next generation of state-of-the-art generative models have different capabilities. Recent transformer-based models, such as DALLE 2 [223], Imagen [224] and Parti [225], instead take arbitrary text as their input. This has allowed much larger training sets, because images and captions can be scraped from the web rather than being manually labelled. In all, the visual quality of the

results and the usefulness of the representations learned by this new generation of generative models seems a promising resource for testing.

It is plausible that leveraging this impressive recent progress could improve the capabilities of our testing algorithms in particularly – besides increased photorealism, the richer feature representations learned by these models could provide more ways that a particular image can be optimised as a test case. Although the nature of these next-generation generative models is quite different, the basic principles of our test generation algorithms should still apply. They can still be finetuned by training with an additional loss term to incentivise the generation of tests that find failures. And given a fixed particular input to a generative model, the process to identify a suitable perturbation to its latent activations should remain largely unchanged. So the main work here would be the significant engineering effort, which could the improved generated tests.

There is also potential scope for not only re-engineering our algorithms, but using them as a starting point to develop new testing algorithms that exploit the text-to-image capabilities.

8.3.2 Choosing specific features to perturb

Our second new test generation algorithm, which perturbs activation values in a generative network, makes changes to different features in an image so that the perturbed image causes the classifier being tested to fail. The procedure that optimises the activation values is free to adjust any activations (and so features) without constraint. The only control we have exerted of the kinds of features changed is by restricting the layers in the generative network at which the activations could be perturbed. For instance, perturbing earlier layers only constrains the changes made to be restricted to higher-level features at a coarser granularity.

But perhaps this algorithm could be adjusted to give a degree of control over the kinds of features that the perturbation is constrained to optimise. This would allow a more precise probing of the proxies used by the model being

tested – if vegetation were the only feature being adjusted, for instance, then the dependence of the model on that feature in different contexts could be measured. In order to facilitate this kind of control, it would first be necessary to identify which directions in the generator's latent activation spaces correspond to the features of interest – the perturbations could then be constrained to act only in these directions. This could be achieved by training the generator in the first place so that these representations are predictably 'disentangled', or by using a process such as that in Bau et al. [60] that identifies relevant representations in existing generators.

8.3.3 Learning generative representations suitable for perturbations

In this thesis, we introduced a procedure that generates tests by training a generative network so that it directly outputs useful tests, and a procedure that generates tests by perturbing the latent activations of a fixed, pretrained generator. It may that combining these two approaches leads to a procedure with advantages over either one. For instance, it may be possible to train a generative model so that its latent feature representations are especially well suited to being perturbed. That is, rather than perturbing the activations of a standard generator, perhaps it could be beneficial to design a generative model whose activations are intended to be perturbed.

8.3.4 Counterfactual explanations

In some situations, it is helpful to provide an explanation of why a system made a decision. Not only for debugging, but to provide helpful user feedback – for instance to explain to someone why their application for a loan was rejected. A counterfactual explanation achieves this by providing one or more examples of similar inputs that would have resulted in a different system output (if the income had been £8,000 greater, for example). There are existing algorithms such as those by Wachter, Mittelstadt, and Russell [226] and Sharma, Henderson,

and Ghosh [227] that provide counterfactual explanations, but they could be extended by exploiting the context-sensitive perturbation algorithm introduced in Chapter 5. In this way, the explanation algorithm could be freed from only making small changes to the raw features, and could make changes to higher-level latent underlying features. For instance, rather than keeping all else equal and raising only the income, this approach might make plausible changes to inputs besides income that could all be caused by an upstream change, such as getting a secure job. In short, exploitation of generative modelling using the techniques introduced in this thesis could improve the quality of the counterfactual explanations provided.

8.3.5 Adaptation to other domains

In this thesis, the new test generation algorithms were presented and evaluated solely in the context of image classification. But there is not anything fundamental to this choice of application domain; the algorithms should work when adapted to work with other kinds of data. For example, audio, video, natural language, or even code. The only requirement is the training of a generative model that can learn to output examples of the relevant data type. Besides natural language, these have received less attention than generative models for images. There are other difficulties associated with this adaptation, such as the greater complexity of video and the less immediate output of audio from a computer to a human, but these are all practical in nature. In practice, this project would require fresh implementations of our testing algorithms, rather than attempting to adapt the image-specific existing codebase.

Moving beyond classification to other test other discriminative models would also certainly be possible. The algorithms would simply need new loss terms to be defined that measured the performance of the model being tested on a given input – for instance, some form of error term for a regression model.

8.3.6 Testing of systems besides DNNs

The new algorithms were designed to test deep neural networks in particular – they both rely on their differentiability to allow the standard backpropagation algorithm to make updates to either the weights of the generator or the perturbation tensors. So in principle, they could be applied to any differentiable system.

If they could be adapted to not require differentiability in the system being tested, then they could be applied to test any software program or even hardware design. It is not immediately obvious whether such an adaptation is possible, but it would likely exploit techniques from reinforcement learning in order to optimise the generator weights or perturbation tensors.

References

[1] Isaac Dunn, Hadrien Pouget, Tom Melham, and Daniel Kroening. "Adaptive Generation of Unrestricted Adversarial Inputs". In: *CoRR* abs/1905.02463 (2019). arXiv: 1905.02463. URL: http://arxiv.org/abs/1905.02463.

[2] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. "Constructing Unrestricted Adversarial Examples with Generative Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by Samy Bengio, Hanna M Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 8322–8333. URL: http://papers.nips.cc/paper/8052-constructing-unrestricted-adversarial-examples-with-generative-models.

[3] Isaac Dunn, Laura Hanu, Hadrien Pouget, Tom Melham, and Daniel Kroening. "Evaluating Robustness to Context-Sensitive Feature Perturbations of Different Granularities". In: *CoRR* abs/2001.11055 (2020). arXiv: 2001.11055. URL: https://arxiv.org/abs/2001.11055.

[4] Isaac Dunn, Hadrien Pouget, Daniel Kroening, and Tom Melham. "Exposing previously undetectable faults in deep neural networks". In: *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*. Ed. by Cristian Cadar and Xiangyu Zhang. ACM, 2021, pp. 56–66. DOI: 10.1145/3460319.3464801. URL: https://doi.org/10.1145/3460319.3464801.

[5] Tim Salimans, Katherine Heller, David Blei, Max Welling, Zoubin Ghahramani, and Matt Hoffman. "Panel on the Foundations and Future of Approximate Inference". In: *Thirty-first Conference on Neural Information Processing Systems* (Dec. 2017).

[6] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard S. Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. "Shortcut learning in deep neural networks". In: *Nat. Mach. Intell.* 2.11 (2020), pp. 665–673. DOI: 10.1038/s42256-020-00257-z. URL: https://doi.org/10.1038/s42256-020-00257-z.

[7] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6572.

[8] Andrey Kuehlkamp, Benedict Becker, and Kevin W. Bowyer. "Gender-from-Iris or Gender-from-Mascara?" In: *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*. IEEE Computer Society, 2017, pp. 1151–1159. DOI: 10.1109/WACV.2017.133. URL: https://doi.org/10.1109/WACV.2017.133.

[9] Sara Beery, Grant Van Horn, and Pietro Perona. "Recognition in Terra Incognita". In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Vol. 11220. Lecture Notes in Computer Science. Springer, 2018, pp. 472–489. DOI: 10.1007/978-3-030-01270-0_28.

[10] John R. Zech, Marcus A. Badgeley, Manway Liu, Anthony B. Costa, Joseph J. Titano, and Eric K. Oermann. "Confounding variables can degrade generalization performance of radiological deep learning models". In: *CoRR* abs/1807.00431 (2018). arXiv: 1807.00431. URL: http://arxiv.org/abs/1807.00431.

[11] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. "Adversarial Examples Are Not Bugs, They Are Features". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. 2019, pp. 125–136. URL: http://papers.nips.cc/paper/8307-adversarial-examples-are-not-bugs-they-are-features.

[12] Justin Gilmer and Dan Hendrycks. "A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Adversarial Example Researchers Need to Expand What Is Meant by 'Robustness'". en. In: *Distill* 4.8 (Aug. 2019), e00019.1. DOI: 10.23915/distill.00019.1. URL: https://distill.pub/2019/advex-bugs-discussion/response-1 (visited on 08/30/2019).

[13] Pang Wei Koh et al. "WILDS: A Benchmark of in-the-Wild Distribution Shifts". In: *Proceedings of the 38th International Conference on Machine Learning*. PMLR, July 2021, pp. 5637–5664.

[14] Paul Trichelair, Ali Emami, Adam Trischler, Kaheer Suleman, and Jackie Chi Kit Cheung. "How Reasonable are Common-Sense Reasoning Tasks: A Case-Study on the Winograd Schema Challenge and SWAG". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Association for Computational Linguistics, 2019, pp. 3380–3385. DOI: 10.18653/v1/D19-1335. URL: https://doi.org/10.18653/v1/D19-1335.

[15] Timothy Niven and Hung-Yu Kao. "Probing Neural Network Comprehension of Natural Language Arguments". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Ed. by Anna Korhonen, David R. Traum, and

Lluís Màrquez. Association for Computational Linguistics, 2019, pp. 4658–4664. DOI: 10.18653/v1/p19-1459. URL: https://doi.org/10.18653/v1/p19-1459.

[16] John Miller, Karl Krauth, Benjamin Recht, and Ludwig Schmidt. "The Effect of Natural Distribution Shift on Question Answering Models". In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, Nov. 2020, pp. 6905–6916.

[17] Alexander D'Amour et al. "Underspecification Presents Challenges for Credibility in Modern Machine Learning". In: *CoRR* abs/2011.03395 (2020). arXiv: 2011.03395.

[18] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[19] Tom B Brown, Nicholas Carlini, Chiyuan Zhang, Catherine Olsson, Paul Francis Christiano, and Ian J Goodfellow. "Unrestricted Adversarial Examples". In: *CoRR* abs/1809.0 (2018). arXiv: 1809.08352. URL: http://arxiv.org/abs/1809.08352.

[20] J.N. Buxton and B. Randell. "Software Engineering Techniques". In: *NATO Science Committee* (1970). URL: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF.

[21] "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[22] Divya Kumar and K.K. Mishra. "The Impacts of Test Automation on Software's Cost, Quality and Time to Market". In: *Procedia Computer Science* 79 (2016). Proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016, pp. 8–15. DOI: https://doi.org/10.1016/j.procs.2016.03.003. URL: https://www.sciencedirect.com/science/article/pii/S1877050916001277.

[23] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D Lawrence, and Kilian Q Weinberger. 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.

[24] Ian J Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *CoRR* abs/1701.0 (2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[25] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. "Are GANs Created Equal? A Large-Scale Study". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 698–707. URL: http://papers.nips.cc/paper/7350-are-gans-created-equal-a-large-scale-study (visited on 10/16/2019).

[26] Zhengwei Wang, Qi She, and Tomas E. Ward. "Generative Adversarial Networks: A Survey and Taxonomy". In: *CoRR* abs/1906.01529 (2019). URL: http://arxiv.org/abs/1906.01529 (visited on 10/16/2019).

[27] Yongjun Hong, Uiwon Hwang, Jaeyoon Yoo, and Sungroh Yoon. "How Generative Adversarial Networks and Their Variants Work: An Overview". In: *ACM Comput. Surv.* 52.1 (2019), 10:1–10:43. DOI: 10.1145/3301282.

[28] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1511.06434 (visited on 10/16/2019).

[29] Martín Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 214–223. URL: http://proceedings.mlr.press/v70/arjovsky17a.html.

[30] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C Courville. "Improved Training of Wasserstein GANs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M Wallach, Rob Fergus, S V N Vishwanathan, and Roman Garnett. 2017, pp. 5769–5779. URL: http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.

[31] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=Hk99zCeAb (visited on 10/16/2019).

[32] Han Zhang, Ian J. Goodfellow, Dimitris N. Metaxas, and Augustus Odena. "Self-Attention Generative Adversarial Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7354–7363. URL: http://proceedings.mlr.press/v97/zhang19d.html (visited on 10/16/2019).

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need (visited on 10/16/2019).

[34] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=B1xsqj09Fm.

[35] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *CoRR* abs/1411.1 (2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784.

[36] Augustus Odena, Christopher Olah, and Jonathon Shlens. "Conditional Image Synthesis with Auxiliary Classifier GANs". In: *International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 2642–2651. URL: http://proceedings.mlr.press/v70/odena17a.html.

[37] Brendan J Frey, J Frey Brendan, and Brendan J Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT press, 1998.

[38] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio". In: *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*. ISCA, 2016, p. 125. URL: http://www.isca-speech.org/archive/SSW_2016/abstracts/ssw9_DS-4_van_den_Oord.html (visited on 10/16/2019).

[39] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=BJrFC6ceg (visited on 10/16/2019).

[40] Aapo Hyvärinen and Petteri Pajunen. "Nonlinear Independent Component Analysis: Existence and Uniqueness Results". In: *Neural Networks* 12.3 (1999), pp. 429–439.

[41] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-Linear Independent Components Estimation". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1410.8516 (visited on 10/16/2019).

[42] Diederik P. Kingma and Prafulla Dhariwal. "Glow: Generative Flow with Invertible 1x1 Convolutions". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 10236–10245. URL: http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions (visited on 10/16/2019).

[43] Geoffrey E Hinton, Terrence J Sejnowski, et al. "Learning and Relearning in Boltzmann Machines". In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1.282-317 (1986), p. 2.

[44] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: http://arxiv.org/abs/1312.6114 (visited on 10/16/2019).

[45] Gaurav Kumar and Pradeep Kumar Bhatia. "A Detailed Review of Feature Extraction in Image Processing Systems". In: *2014 Fourth International Conference on Advanced Computing Communication Technologies*. 2014, pp. 5–12. DOI: 10.1109/ACCT.2014.74.

[46] David G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[47] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. "Zoom In: An Introduction to Circuits". In: *Distill* (2020). https://distill.pub/2020/circuits/zoom-in. DOI: 10.23915/distill.00024.001.

[48] Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. "Thread: Circuits". In: *Distill* (2020). https://distill.pub/2020/circuits. DOI: 10.23915/distill.00024.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594. URL: https://doi.org/10.1109/CVPR.2015.7298594.

[50] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[51] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization". In: *Distill* (2017). https://distill.pub/2017/feature-visualization. DOI: 10.23915/distill.00007.

[52] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. "An Overview of Early Vision in InceptionV1". In: *Distill* (2020). https://distill.pub/2020/circuits/early-vision. DOI: 10.23915/distill.00024.002.

[53] Nick Cammarata, Gabriel Goh, Shan Carter, Ludwig Schubert, Michael Petrov, and Chris Olah. "Curve Detectors". In: *Distill* (2020). https://distill.pub/2020/circuits/curve-detectors. DOI: 10.23915/distill.00024.003.

[54] Nick Cammarata, Gabriel Goh, Shan Carter, Chelsea Voss, Ludwig Schubert, and Chris Olah. "Curve Circuits". In: *Distill* (2021). https://distill.pub/2020/circuits/curve-circuits. DOI: 10.23915/distill.00024.006.

[55] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 8748–8763. URL: http://proceedings.mlr.press/v139/radford21a.html.

[56] Gabriel Goh, Nick Cammarata †, Chelsea Voss †, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. “Multimodal Neurons in Artificial Neural Networks”. In: *Distill* (2021). https://distill.pub/2021/multimodal-neurons. DOI: 10.23915/distill.00030.

[57] Jesse Mu and Jacob Andreas. “Compositional Explanations of Neurons”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/c74956ffb38ba48ed6ce977af6727275-Abstract.html.

[58] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. “Network Dissection: Quantifying Interpretability of Deep Visual Representations”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 3319–3327. DOI: 10.1109/CVPR.2017.354. URL: https://doi.org/10.1109/CVPR.2017.354.

[59] Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, Anthony Bau, and James R. Glass. “What Is One Grain of Sand in the Desert? Analyzing Individual Neurons in Deep NLP Models”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 6309–6317. DOI: 10.1609/aaai.v33i01.33016309. URL: https://doi.org/10.1609/aaai.v33i01.33016309.

[60] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. “GAN Dissection: Visualizing and Understanding Generative Adversarial Networks”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=Hyg_X2C5FX (visited on 01/29/2020).

[61] Ali Jahanian, Lucy Chai, and Phillip Isola. “On the "steerability" of generative adversarial networks”. In: *CoRR* abs/1907.07171 (2019). arXiv: 1907.07171. URL: http://arxiv.org/abs/1907.07171.

[62] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996. DOI: 10.1017/CBO9780511812651.

[63] Battista Biggio and Fabio Roli. “Wild Patterns: Ten Years after the Rise of Adversarial Machine Learning”. In: *Pattern Recognition* 84 (2018), pp. 317–331. DOI: 10.1016/j.patcog.2018.07.023.

[64] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: http://arxiv.org/abs/1312.6199.

[65] Dong C. Liu and Jorge Nocedal. "On the Limited Memory BFGS Method for Large Scale Optimization". en. In: *Mathematical Programming* 45.1 (Aug. 1989), pp. 503–528. DOI: 10.1007/BF01589116. URL: https://doi.org/10.1007/BF01589116 (visited on 10/23/2019).

[66] Mehdi Hussain, Ainuddin Wahid Abdul Wahab, Yamani Idna Bin Idris, Anthony T. S. Ho, and Ki-Hyun Jung. "Image Steganography in Spatial Domain: A Survey". en. In: *Signal Processing: Image Communication* 65 (July 2018), pp. 46–66. DOI: 10.1016/j.image.2018.03.012. URL: http://www.sciencedirect.com/science/article/pii/S092359651830256X (visited on 10/23/2019).

[67] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. Ed. by Johannes Fürnkranz and Thorsten Joachims. Omnipress, 2010, pp. 807–814. URL: https://icml.cc/Conferences/2010/papers/432.pdf (visited on 10/23/2019).

[68] Kurt Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". en. In: *Neural Networks* 4.2 (Jan. 1991), pp. 251–257. DOI: 10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T (visited on 10/24/2019).

[69] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil K. Jain. "Adversarial Attacks and Defenses in Images, Graphs and Text: A Review". In: *CoRR* abs/1909.08072 (2019). URL: http://arxiv.org/abs/1909.08072 (visited on 10/25/2019).

[70] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Examples in the Physical World". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=HJGU3Rodl (visited on 10/16/2019).

[71] Nicholas Carlini and David A. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 39–57. DOI: 10.1109/SP.2017.49. URL: https://doi.org/10.1109/SP.2017.49.

[72] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 2574–2582. DOI: 10.1109/CVPR.2016.282.

[73] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. "The Limitations of Deep Learning in Adversarial Settings". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. IEEE, 2016, pp. 372–387. DOI: 10.1109/EuroSP.2016.36. URL: https://doi.org/10.1109/EuroSP.2016.36.

[74] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. "Transferability in Machine Learning: From Phenomena to Black-Box Attacks Using Adversarial Samples". In: *CoRR* abs/1605.07277 (2016). URL: http://arxiv.org/abs/1605.07277 (visited on 10/16/2019).

[75] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. "Universal Adversarial Perturbations". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 86–94. DOI: 10.1109/CVPR.2017.17.

[76] Nicholas Carlini and David A. Wagner. "Audio Adversarial Examples: Targeted Attacks on Speech-to-Text". In: *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*. IEEE Computer Society, 2018, pp. 1–7. DOI: 10.1109/SPW.2018.00009.

[77] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani B Srivastava, and Kai-Wei Chang. "Generating Natural Language Adversarial Examples". In: *Proc. 2018 Conf. Empir. Methods Nat. Lang. Process. Brussels, Belgium, Oct. 31 - Novemb. 4, 2018*. Ed. by Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii. Association for Computational Linguistics, 2018, pp. 2890–2896. URL: https://aclanthology.info/papers/D18-1316/d18-1316.

[78] Wei Emma Zhang, Quan Z. Sheng, Ahoud Abdulrahmn F. Alhazmi, and Chenliang Li. "Adversarial Attacks on Deep Learning Models in Natural Language Processing: A Survey". In: *CoRR* abs/1901.06796 (2019). URL: http://arxiv.org/abs/1901.06796 (visited on 10/17/2019).

[79] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. "Adversarial Attacks on Neural Network Policies". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=ryvlRyBKl (visited on 10/16/2019).

[80] Adam Gleave, Michael Dennis, Neel Kant, Cody Wild, Sergey Levine, and Stuart Russell. "Adversarial Policies: Attacking Deep Reinforcement Learning". In: *CoRR* abs/1905.10615 (2019). URL: http://arxiv.org/abs/1905.10615 (visited on 10/16/2019).

[81] Tong Chen, Jiqiang Liu, Yingxiao Xiang, Wenjia Niu, Endong Tong, and Zhen Han. "Adversarial Attack and Defense in Reinforcement Learning-from AI Security View". en. In: *Cybersecurity* 2.1 (Dec. 2019), p. 11. DOI: 10.1186/s42400-019-0027-x. URL: https://cybersecurity.springeropen.com/articles/10.1186/s42400-019-0027-x (visited on 10/16/2019).

[82] Adi Shamir, Itay Safran, Eyal Ronen, and Orr Dunkelman. "A Simple Explanation for the Existence of Adversarial Examples with Small Hamming Distance". In: *arXiv:1901.10861 [cs, stat]* (Jan. 2019). arXiv: 1901.10861 [cs, stat]. URL: http://arxiv.org/abs/1901.10861 (visited on 07/16/2019).

[83] Ali Shafahi, W. Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. "Are Adversarial Examples Inevitable?" In: *CoRR* abs/1809.02104 (2018). arXiv: 1809.02104. URL: http://arxiv.org/abs/1809.02104.

[84] Saumya Jetley, Nicholas A. Lord, and Philip H. S. Torr. "With Friends Like These, Who Needs Adversaries?" In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 10772–10782. URL: http://papers.nips.cc/paper/8273-with-friends-like-these-who-needs-adversaries (visited on 10/24/2019).

[85] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. "Adversarially Robust Generalization Requires More Data". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 5019–5031. URL: http://papers.nips.cc/paper/7749-adversarially-robust-generalization-requires-more-data (visited on 10/16/2019).

[86] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. "Learning with a Strong Adversary". In: *CoRR* abs/1511.03034 (2015). URL: http://arxiv.org/abs/1511.03034 (visited on 10/16/2019).

[87] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Machine Learning at Scale". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=BJm4T4Kgx (visited on 10/16/2019).

[88] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=rJzIBfZAb.

[89] J. M. Danskin. *The Theory of Max-Min and Its Application to Weapons Allocation Problems*. en. Ökonometrie Und Unternehmensforschung Econometrics and Operations Research. Berlin Heidelberg: Springer-Verlag, 1967. DOI: 10.1007/978-3-642-46092-0. URL: https://www.springer.com/gp/book/9783642460944 (visited on 10/23/2019).

[90] Florian Tramèr and Dan Boneh. "Adversarial Training and Robustness for Multiple Perturbations". In: *CoRR* abs/1904.13000 (2019). URL: http://arxiv.org/abs/1904.13000 (visited on 10/23/2019).

[91] Pratyush Maini, Eric Wong, and J. Zico Kolter. "Adversarial Robustness Against the Union of Multiple Perturbation Models". In: *arXiv:1909.04068 [cs, stat]* (Sept. 2019). arXiv: 1909.04068 [cs, stat]. URL: http://arxiv.org/abs/1909.04068 (visited on 10/03/2019).

[92] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 97–117. DOI: 10.1007/978-3-319-63387-9_5.

[93] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks". In: *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 582–597. DOI: 10.1109/SP.2016.41.

[94] Nicholas Carlini and David A. Wagner. "Defensive Distillation Is Not Robust to Adversarial Examples". In: *CoRR* abs/1607.04311 (2016). URL: http://arxiv.org/abs/1607.04311 (visited on 10/25/2019).

[95] 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)*. Ed. by Jennifer G Dy and Andreas Krause. Vol. 80. 2018, pp. 274–283. URL: http://proceedings.mlr.press/v80/athalye18a.html.

[96] Nicholas Carlini and David A. Wagner. "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods". In: *Workshop on Artificial Intelligence and Security at the ACM Conference on Computer and Communications Security (AISec@CCS*. ACM, 2017, pp. 3–14.

[97] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. "On Evaluating Adversarial Robustness". In: *arXiv:1902.06705 [cs, stat]* (Feb. 2019). arXiv: 1902.06705 [cs, stat]. URL: http://arxiv.org/abs/1902.06705 (visited on 06/18/2019).

[98] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J. Kochenderfer. "Algorithms for Verifying Deep Neural Networks". In: *CoRR* abs/1903.06758 (2019). arXiv: 1903.06758. URL: http://arxiv.org/abs/1903.06758.

[99] Matthew Mirman, Timon Gehr, and Martin T. Vechev. "Differentiable Abstract Interpretation for Provably Robust Neural Networks". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 3575–3583. URL: http://proceedings.mlr.press/v80/mirman18b.html (visited on 10/17/2019).

[100] 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 (ICML)*. Ed. by Jennifer G Dy and Andreas Krause. Vol. 80. JMLR Workshop and Conference Proceedings. JMLR.org, 2018, pp. 5283–5292. URL: http://proceedings.mlr.press/v80/wong18a.html.

[101] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. "Provable Robustness of ReLU Networks via Maximization of Linear Regions". In: *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2057–2066. URL: http://proceedings.mlr.press/v89/croce19a.html (visited on 10/17/2019).

[102] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. "MixTrain: Scalable Training of Formally Robust Neural Networks". In: *CoRR* abs/1811.02625 (2018).

[103] Francesco Croce and Matthias Hein. "Provable Robustness against All Adversarial Lp-Perturbations for P>=1". In: *CoRR* abs/1905.11213 (2019). URL: http://arxiv.org/abs/1905.11213 (visited on 10/23/2019).

[104] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. "DLFuzz: differential fuzzing testing of deep learning systems". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 739–743. DOI: 10.1145/3236024.3264835. URL: https://doi.org/10.1145/3236024.3264835 (visited on 10/13/2020).

[105] Seokhyun Lee, Sooyoung Cha, Dain Lee, and Hakjoo Oh. "Effective white-box testing of deep neural networks with adaptive neuron-selection strategy". In: *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2020. New York, NY, USA: Association for Computing Machinery, July 2020, pp. 165–176. DOI: 10.1145/3395363.3397346. URL: https://doi.org/10.1145/3395363.3397346 (visited on 10/13/2020).

[106] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. "Adversarial attacks and defenses in deep learning". In: *Engineering* 6.3 (2020), pp. 346–360.

[107] Nicholas Carlini. *A Complete List of All Adversarial Example Papers*. https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html. June 2019.

[108] Justin Gilmer, Ryan P Adams, Ian J Goodfellow, David Andersen, and George E Dahl. "Motivating the Rules of the Game for Adversarial Example Research". In: *CoRR* abs/1807.0 (2018). arXiv: 1807.06732. URL: http://arxiv.org/abs/1807.06732.

[109] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. "Adversarial Robustness as a Prior for Learned Representations". In: *arXiv:1906.00945 [cs, stat]* (Sept. 2019). arXiv: 1906.00945 [cs, stat]. URL: http://arxiv.org/abs/1906.00945 (visited on 10/24/2019).

[110] Shibani Santurkar, Dimitris Tsipras, Brandon Tran, Andrew Ilyas, Logan Engstrom, and Aleksander Madry. "Image Synthesis with a Single (Robust) Classifier". In: *arXiv:1906.09453 [cs, stat]* (Aug. 2019). arXiv: 1906.09453 [cs, stat]. URL: http://arxiv.org/abs/1906.09453 (visited on 10/24/2019).

[111] Zhou Wang and Alan C. Bovik. "Mean Squared Error: Love It or Leave It? A New Look at Signal Fidelity Measures". In: *IEEE Signal Processing Magazine* 26.1 (Jan. 2009), pp. 98–117. DOI: 10.1109/MSP.2008.930649.

[112] Florian Tramèr, Jens Behrmann, Nicholas Carlini, Nicolas Papernot, and Jörn-Henrik Jacobsen. "Fundamental Tradeoffs between Invariance and Sensitivity to Adversarial Perturbations". In: *CoRR* abs/2002.04599 (2020). arXiv: 2002.04599. URL: https://arxiv.org/abs/2002.04599.

[113] Jamie Hayes and George Danezis. "Learning Universal Adversarial Perturbations with Generative Models". In: *2018 IEEE Security and Privacy Workshops*. IEEE Computer Society, 2018, pp. 43–49. DOI: 10.1109/SPW.2018.00015. URL: https://doi.org/10.1109/SPW.2018.00015.

[114] Shumeet Baluja and Ian Fischer. "Learning to Attack: Adversarial Transformation Networks". In: *AAAI Conference on Artificial Intelligence*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 2687–2695. DOI: 10.1609/aaai.v32i1.11672.

[115] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. "Generating Adversarial Examples with Adversarial Networks". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by Jérôme Lang. ijcai.org, 2018, pp. 3905–3911. DOI: 10.24963/ijcai.2018/543. URL: https://doi.org/10.24963/ijcai.2018/543.

[116] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge J. Belongie. "Generative Adversarial Perturbations". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 4422–4431. DOI: 10.1109/CVPR.2018.00465. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Poursaeed_Generative_Adversarial_Perturbations_CVPR_2018_paper.html.

[117] Houssem Ben Braiek and Foutse Khomh. "On Testing Machine Learning Programs". In: *arXiv:1812.02257 [cs]* (Dec. 2018). arXiv: 1812.02257 [cs]. URL: http://arxiv.org/abs/1812.02257 (visited on 10/24/2019).

[118] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. "Machine Learning Testing: Survey, Landscapes and Horizons". In: *CoRR* abs/1906.10742 (2019). URL: http://arxiv.org/abs/1906.10742 (visited on 10/17/2019).

[119] Hong Zhu, Patrick A. V. Hall, and John H. R. May. "Software Unit Test Coverage and Adequacy". In: *ACM Comput. Surv.* 29.4 (1997), pp. 366–427. DOI: 10.1145/267580.267590.

[120] John Joseph Chilenski and Steven P. Miller. "Applicability of Modified Condition/Decision Coverage to Software Testing". In: *Software Engineering Journal* 9.5 (Sept. 1994), pp. 193–200. DOI: 10.1049/sej.1994.0025.

[121] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. "DeepXplore: Automated Whitebox Testing of Deep Learning Systems". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1–18. DOI: 10.1145/3132747.3132785. URL: https://doi.org/10.1145/3132747.3132785 (visited on 10/13/2020).

[122] Lei Ma et al. "DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems". In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. Ed. by Marianne Huchard, Christian Kästner, and Gordon Fraser. ACM, 2018, pp. 120–131. DOI: 10.1145/3238147.3238202.

[123] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. "Testing Deep Neural Networks". In: *CoRR* abs/1803.04792 (2018). URL: http://arxiv.org/abs/1803.04792 (visited on 10/17/2019).

[124] Robert S. Boyer, Bernard Elspas, and Karl N. Levitt. "SELECT - a Formal System for Testing and Debugging Programs by Symbolic Execution". In: *Proceedings of the International Conference on Reliable Software 1975, Los Angeles, California, USA, April 21-23, 1975*. Ed. by Martin L. Shooman and Raymond T. Yeh. ACM, 1975, pp. 234–245. DOI: 10.1145/800027.808445.

[125] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs". In: *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*. Ed. by Richard Draves and Robbert van Renesse. USENIX Association, 2008, pp. 209–224.

[126] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. "The S2E Platform: Design, Implementation, and Applications". In: *ACM Trans. Comput. Syst.* 30.1 (2012), 2:1–2:49. DOI: 10.1145/2110356.2110358.

[127] Fish Wang and Yan Shoshitaishvili. "Angr - The Next Generation of Binary Analysis". In: *IEEE Cybersecurity Development, SecDev 2017, Cambridge, MA, USA, September 24-26, 2017*. IEEE Computer Society, 2017, pp. 8–9. DOI: 10.1109/SecDev.2017.14.

[128] Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S. Pasareanu, and Sarfraz Khurshid. "Symbolic Execution for Deep Neural Networks". In: *CoRR* abs/1807.10439 (2018). arXiv: 1807.10439.

[129] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. "Concolic Testing for Deep Neural Networks". In: *ACM/IEEE International Conference on Automated Software Engineering (ASE)*. Ed. by Marianne Huchard, Christian Kästner, and Gordon Fraser. ACM, 2018, pp. 109–119.

[130] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. "Structural Coverage Criteria for Neural Networks Could Be Misleading". In: *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019, Montreal, QC, Canada, May 29-31, 2019*. Ed. by Anita Sarma and Leonardo Murta. IEEE / ACM, 2019, pp. 89–92. DOI: 10.1109/ICSE-NIER.2019.00031.

[131] Barton P. Miller, Lars Fredriksen, and Bryan So. "An Empirical Study of the Reliability of UNIX Utilities". In: *Commun. ACM* 33.12 (1990), pp. 32–44. DOI: 10.1145/96267.96279.

[132] Augustus Odena, Catherine Olsson, David Andersen, and Ian J. Goodfellow. "TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 4901–4911. URL: http://proceedings.mlr.press/v97/odena19a.html (visited on 10/17/2019).

[133] Sven Gowal, Chongli Qin, Po-Sen Huang, Taylan Cemgil, Krishnamurthy Dvijotham, Timothy A. Mann, and Pushmeet Kohli. "Achieving Robustness in the Wild via Adversarial Mixing with Disentangled Representations". In: *CoRR* abs/1912.03192 (2019). arXiv: 1912.03192. URL: http://arxiv.org/abs/1912.03192.

[134] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems". In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ASE 2018. New York, NY, USA: Association for Computing Machinery, Sept. 2018, pp. 132–142. DOI: 10.1145/3238147.3238187. URL: https://doi.org/10.1145/3238147.3238187 (visited on 10/28/2020).

[135] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. "Unsupervised Image-to-Image Translation Networks". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, pp. 700–708. URL: http://papers.nips.cc/paper/6672-unsupervised-image-to-image-translation-networks.

[136] Ameya Joshi, Amitangshu Mukherjee, Soumik Sarkar, and Chinmay Hegde. "Semantic Adversarial Attacks: Parametric Transformations That Fool Deep Classifiers". In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 4772–4782. DOI: 10.1109/ICCV.2019.00487. URL: https://doi.org/10.1109/ICCV.2019.00487.

[137] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. "A General Framework for Adversarial Examples with Objectives". In: *ACM Trans. Priv. Secur.* 22.3 (2019), 16:1–16:30. DOI: 10.1145/3317611. URL: https://doi.org/10.1145/3317611.

[138] Anand Bhattad, Min Jin Chong, Kaizhao Liang, Bo Li, and David A. Forsyth. "Big but Imperceptible Adversarial Perturbations via Semantic Manipulation". In: *arXiv:1904.06347 [cs]* (Apr. 2019). arXiv: 1904.06347 [cs]. URL: http://arxiv.org/abs/1904.06347 (visited on 09/25/2019).

[139] Haonan Qiu, Chaowei Xiao, Lei Yang, Xinchen Yan, Honglak Lee, and Bo Li. "SemanticAdv: Generating Adversarial Examples via Attribute-Conditional Image Editing". In: *arXiv:1906.07927 [cs, eess]* (June 2019). arXiv: 1906.07927 [cs, eess]. URL: http://arxiv.org/abs/1906.07927 (visited on 09/25/2019).

[140] Zhengli Zhao, Dheeru Dua, and Sameer Singh. "Generating Natural Adversarial Examples". In: *arXiv:1710.11342 [cs]* (Oct. 2017). arXiv: 1710.11342 [cs]. URL: http://arxiv.org/abs/1710.11342 (visited on 09/25/2019).

[141] Taejoon Byun, Abhishek Vijayakumar, Sanjai Rayadurgam, and Darren Cofer. "Manifold-Based Test Generation for Image Classifiers". In: *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. Aug. 2020, pp. 15–22. DOI: 10.1109/AITEST49225.2020.00010.

[142] Shuo Wang, Shangyu Chen, Tianle Chen, Surya Nepal, Carsten Rudolph, and Marthie Grobler. "Generating Semantic Adversarial Examples via Feature Manipulation". In: *arXiv:2001.02297 [cs, stat]* (Jan. 2020). arXiv: 2001.02297 [cs, stat]. URL: http://arxiv.org/abs/2001.02297 (visited on 02/11/2020).

[143] Fan Yang, Ninghao Liu, Mengnan Du, and Xia Hu. "Generative Counterfactuals for Neural Networks via Attribute-Informed Perturbation". In: *ACM SIGKDD Explorations Newsletter* 23.1 (May 2021), pp. 59–68. DOI: 10.1145/3468507.3468517.

[144] Felipe Toledo, David Shriver, Sebastian Elbaum, and Matthew B. Dwyer. "Distribution Models for Falsification and Verification of DNNs". In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Nov. 2021, pp. 317–329. DOI: 10.1109/ASE51524.2021.9678590.

[145] Eric Wong and J. Zico Kolter. "Learning perturbation sets for robust machine learning". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=MIDckA56aD.

[146] Swaroopa Dola, Matthew B. Dwyer, and Mary Lou Soffa. "Distribution-Aware Testing of Neural Networks Using Generative Models". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. May 2021, pp. 226–237. DOI: 10.1109/ICSE43902.2021.00032.

[147] Matthias Minderer, Olivier Bachem, Neil Houlsby, and Michael Tschannen. "Automatic Shortcut Removal for Self-Supervised Representation Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, Nov. 2020, pp. 6927–6937.

[148] Lu Yang, Qing Song, and Yingqi Wu. "Attacks on State-of-the-Art Face Recognition Using Attentional Adversarial Attack Generative Network". In: *Multimedia Tools and Applications* 80.1 (2021), pp. 855–875. DOI: 10.1007/s11042-020-09604-z.

[149] Xiaosen Wang, Kun He, and John E. Hopcroft. "AT-GAN: A Generative Attack Model for Adversarial Transferring on Generative Adversarial Nets". In: *CoRR* abs/1904.07793 (2019). arXiv: 1904.07793. URL: http://arxiv.org/abs/1904.07793.

[150] Francis Baek, Daeho Kim, Somin Park, Hyoungkwan Kim, and SangHyun Lee. "Conditional Generative Adversarial Networks with Adversarial Attack and Defense for Generative Data Augmentation". In: *Journal of Computing in Civil Engineering* 36.3 (May 2022), p. 04022001. DOI: 10.1061/(ASCE)CP.1943-5487.0001015.

[151] Hsueh-Ti Derek Liu, Michael Tao, Chun-Liang Li, Derek Nowrouzezahrai, and Alec Jacobson. "Beyond Pixel Norm-Balls: Parametric Adversaries Using an Analytically Differentiable Renderer". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=SJl2niR9KQ (visited on 10/17/2019).

[152] Lakshya Jain, Wilson Wu, Steven Chen, Uyeong Jang, Varun Chandrasekaran, Sanjit A. Seshia, and Somesh Jha. "Generating Semantic Adversarial Examples with Differentiable Rendering". In: *CoRR* abs/1910.00727 (2019). URL: http://arxiv.org/abs/1910.00727 (visited on 10/17/2019).

[153] Vincenzo Riccio and Paolo Tonella. "Model-Based Exploration of the Frontier of Behaviours for Deep Learning System Testing". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event USA: ACM, Nov. 2020, pp. 876–888. DOI: 10.1145/3368089.3409730.

[154] Alessio Gambi, Marc Mueller, and Gordon Fraser. "Automatically Testing Self-Driving Cars with Search-Based Procedural Content Generation". In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. Beijing China: ACM, July 2019, pp. 318–328. DOI: 10.1145/3293882.3330566.

[155] Hossein Hosseini and Radha Poovendran. "Semantic Adversarial Examples". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 1614–1619. DOI: 10.1109/CVPRW.2018.00212. URL: http://openaccess.thecvf.com/content_cvpr_2018_workshops/w32/html/Hosseini_Semantic_Adversarial_Examples_CVPR_2018_paper.html (visited on 10/17/2019).

[156] Zhengyu Zhao, Zhuoran Liu, and Martha A. Larson. "Towards Large Yet Imperceptible Adversarial Image Perturbations With Perceptual Color Distance". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 1036–1045. DOI: 10.1109/CVPR42600.2020.00112. URL: https://doi.org/10.1109/CVPR42600.2020.00112.

[157] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. "Exploring the Landscape of Spatial Robustness". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1802–1811. URL: http://proceedings.mlr.press/v97/engstrom19a.html (visited on 10/16/2019).

[158] Yongqiang Tian, Shiqing Ma, Ming Wen, Yepang Liu, Shing-Chi Cheung, and Xiangyu Zhang. "Testing Deep Learning Models for Image Analysis Using Object-Relevant Metamorphic Relations". In: *CoRR* abs/1909.03824 (2019). arXiv: 1909.03824. URL: http://arxiv.org/abs/1909.03824.

[159] Jasper Snoek, Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D. Sculley, Joshua V. Dillon, Jie Ren, and Zachary Nado. "Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty under Dataset Shift". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. 2019, pp. 13969–13980. URL: http://papers.nips.cc/paper/9547-can-you-trust-your-models-uncertainty-evaluating-predictive-uncertainty-under-dataset-shift (visited on 06/03/2020).

[160] Yaroslav Bulatov. *Machine Learning, Etc: notMNIST Dataset*. Sept. 2011. URL: http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html (visited on 06/03/2020).

[161] Xiaofei Xie et al. "DeepHunter: a coverage-guided fuzz testing framework for deep neural networks". In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2019. New York, NY, USA: Association for Computing Machinery, July 2019, pp. 146–157. DOI: 10.1145/3293882.3330579. URL: https://doi.org/10.1145/3293882.3330579 (visited on 10/27/2020).

[162] Xiang Gao, Ripon K. Saha, Mukul R. Prasad, and Abhik Roychoudhury. "Fuzz testing based data augmentation to improve robustness of deep neural networks". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE '20. New York, NY, USA: Association for Computing Machinery, June 2020, pp. 1147–1158. DOI: 10.1145/3377811.3380415. URL: https://doi.org/10.1145/3377811.3380415 (visited on 11/10/2020).

[163] Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. "AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[164] Jeet Mohapatra, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. "Towards Verifying Robustness of Neural Networks Against A Family of Semantic Perturbations". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 241–249. DOI: 10.1109/CVPR42600.2020.00032. URL: https://doi.org/10.1109/CVPR42600.2020.00032.

[165] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. "DeepTest: automated testing of deep-neural-network-driven autonomous cars". In: *Proceedings of the 40th International Conference on Software Engineering*. ICSE '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 303–314. DOI:

10.1145/3180155.3180220. URL: https://doi.org/10.1145/3180155.3180220 (visited on 10/16/2020).

[166] Dan Hendrycks and Thomas G. Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HJz6tiCqYm.

[167] Robert Geirhos, Carlos R. Medina Temme, Jonas Rauber, Heiko H. Schütt, Matthias Bethge, and Felix A. Wichmann. "Generalisation in Humans and Deep Neural Networks". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 2018, pp. 7549–7561.

[168] Norman Mu and Justin Gilmer. "MNIST-C: A Robustness Benchmark for Computer Vision". In: *CoRR* abs/1906.02337 (2019). arXiv: 1906.02337. URL: http://arxiv.org/abs/1906.02337.

[169] Luca Scimeca, Seong Joon Oh, Sanghyuk Chun, Michael Poli, and Sangdoo Yun. "Which Shortcut Cues Will DNNs Choose? A Study from the Parameter-Space Perspective". In: *International Conference on Learning Representations*. 2022.

[170] Meike Nauta, Ricky Walsh, Adam Dubowski, and Christin Seifert. "Uncovering and Correcting Shortcut Learning in Machine Learning Models for Skin Cancer Diagnosis". In: *Diagnostics* 12.1 (Jan. 2022), p. 40. DOI: 10.3390/diagnostics12010040.

[171] Axel Sauer and Andreas Geiger. "Counterfactual Generative Networks". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[172] Kaiyang Zhou, Yongxin Yang, Timothy M. Hospedales, and Tao Xiang. "Learning to Generate Novel Domains for Domain Generalization". In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XVI*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Vol. 12361. Lecture Notes in Computer Science. Springer, 2020, pp. 561–578. DOI: 10.1007/978-3-030-58517-4_33.

[173] Xiaodan Liang, Hao Zhang, Liang Lin, and Eric P. Xing. "Generative Semantic Manipulation with Mask-Contrasting GAN". In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Vol. 11217. Lecture Notes in Computer Science. Springer, 2018, pp. 574–590. DOI: 10.1007/978-3-030-01261-8_34. URL: https://doi.org/10.1007/978-3-030-01261-8_34.

[174] Cihang Xie and Alan L. Yuille. "Intriguing Properties of Adversarial Training at Scale". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[175] Ishaan Gulrajani and David Lopez-Paz. "In Search of Lost Domain Generalization". In: *International Conference on Learning Representations*. Sept. 2020.

[176] Olivia Wiles, Sven Gowal, Florian Stimberg, Sylvestre-Alvise Rebuffi, Ira Ktena, Krishnamurthy Dj Dvijotham, and Ali Taylan Cemgil. "A Fine-Grained Analysis on Distribution Shift". In: *International Conference on Learning Representations*. 2022.

[177] Daniel Kang, Yi Sun, Dan Hendrycks, Tom Brown, and Jacob Steinhardt. "Testing Robustness Against Unforeseen Adversaries". In: *CoRR* abs/1908.08016 (2019). URL: http://arxiv.org/abs/1908.08016 (visited on 10/16/2019).

[178] Yash Sharma and Pin-Yu Chen. *Attacking the Madry Defense Model with L_1-Based Adversarial Examples*. July 2018. DOI: 10.48550/arXiv.1710.10733. arXiv: 1710.10733 [cs, stat].

[179] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. "Measuring Robustness to Natural Distribution Shifts in Image Classification". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020.

[180] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. "Robustness May Be at Odds with Accuracy". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[181] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. "Theoretically Principled Trade-off between Robustness and Accuracy". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7472–7482.

[182] Justin Gilmer, Nicolas Ford, Nicholas Carlini, and Ekin D. Cubuk. "Adversarial Examples Are a Natural Consequence of Test Error in Noise". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2280–2289.

[183] Dong Yin, Raphael Gontijo Lopes, Jon Shlens, Ekin Dogus Cubuk, and Justin Gilmer. "A Fourier Perspective on Model Robustness in Computer Vision". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. 2019, pp. 13255–13265. URL: http://papers.nips.cc/paper/9483-a-fourier-perspective-on-model-robustness-in-computer-vision.

[184] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. "Do ImageNet Classifiers Generalize to ImageNet?" In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5389–5400. URL: http://proceedings.mlr.press/v97/recht19a.html.

[185] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. "Natural Adversarial Examples". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, Virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 15262–15271.

[186] Josip Djolonga et al. "On Robustness and Transferability of Convolutional Neural Networks". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Nashville, TN, USA: IEEE, June 2021, pp. 16453–16463. DOI: 10.1109/CVPR46437.2021.01619.

[187] Hanbin Hu, Mit Shah, Jianhua Z. Huang, and Peng Li. "Global Adversarial Attacks for Assessing Deep Learning Robustness". In: *CoRR* abs/1906.07920 (2019). arXiv: 1906.07920. URL: http://arxiv.org/abs/1906.07920.

[188] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. "Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=BkJ3ibb0-.

[189] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. "Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models". In: *CoRR* abs/1705.1 (2017). arXiv: 1705.10843. URL: http://arxiv.org/abs/1705.10843.

[190] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient". In: *AAAI Conference on Artificial Intelligence*. Ed. by Satinder P Singh and Shaul Markovitch. AAAI Press, 2017, pp. 2852–2858. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14344.

[191] Nicola De Cao and Thomas Kipf. "MolGAN: An Implicit Generative Model for Small Molecular Graphs". In: *CoRR* abs/1805.1 (2018). arXiv: 1805.11973. URL: http://arxiv.org/abs/1805.11973.

[192] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. *Domain Generalization: A Survey*. May 2022. arXiv: 2103.02503 [cs].

[193] Garrett Wilson and Diane J. Cook. "A Survey of Unsupervised Deep Domain Adaptation". In: *ACM Transactions on Intelligent Systems and Technology* 11.5 (July 2020), 51:1–51:46. DOI: 10.1145/3400066.

[194] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M. Hospedales. "Deeper, Broader and Artier Domain Generalization". In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 5543–5551. DOI: 10.1109/ICCV.2017.591.

[195] Xingxuan Zhang, Yue He, Renzhe Xu, Han Yu, Zheyan Shen, and Peng Cui. *NICO++: Towards Better Benchmarking for Domain Generalization*. Apr. 2022. arXiv: 2204.08040 [cs].

[196] Martín Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. "Invariant Risk Minimization". In: *CoRR* abs/1907.02893 (2019). arXiv: 1907.02893.

[197] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive Computation and Machine Learning. MIT Press, 2016. URL: http://www.deeplearningbook.org/.

[198] Yann LeCun, Corinna Cortes, and Chris Burges. *MNIST Handwritten Digit Database*. 1998. URL: http://yann.lecun.com/exdb/mnist/ (visited on 03/28/2019).

[199] Kamran Kowsari, Mojtaba Heidarysafa, Donald E. Brown, Kiana Jafari Meimandi, and Laura E. Barnes. "RMDL: Random Multimodel Deep Learning for Classification". In: *CoRR* abs/1805.01890 (2018).

[200] Li Wan, Matthew D. Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. "Regularization of Neural Networks using DropConnect". In: *International Conference on Machine Learning (ICML)*. Vol. 28. Proceedings of Machine Learning Research. PMLR, 2013, pp. 1058–1066. URL: http://proceedings.mlr.press/v28/wan13.html.

[201] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. "Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for L0 Norm". In: *CoRR* abs/1804.05805 (2018). arXiv: 1804.05805. URL: http://arxiv.org/abs/1804.05805.

[202] Pei-Hsuan Lu, Pin-Yu Chen, Kang-Cheng Chen, and Chia-Mu Yu. "On the Limitation of MagNet Defense Against L_1-Based Adversarial Examples". In: *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN)*. IEEE Computer Society, 2018, pp. 200–214. DOI: 10.1109/DSN-W.2018.00065. URL: http://doi.ieeecomputersociety.org/10.1109/DSN-W.2018.00065.

[203] Lukas Schott, Jonas Rauber, Wieland Brendel, and Matthias Bethge. "Towards the first adversarially robust neural network model on MNIST". In: *CoRR* abs/1805.09190 (2018). arXiv: 1805.09190. URL: http://arxiv.org/abs/1805.09190.

[204] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. "Delving into Transferable Adversarial Examples and Black-box Attacks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[205] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: https://doi.org/10.1109/CVPR.2016.90.

[206] Tim Salimans, Ian J Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved Techniques for Training GANs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by Daniel D Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett. 2016, pp. 2226–2234. URL: http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.

[207] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. "How Good Is My GAN?" In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part II*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Vol. 11206. Lecture Notes in Computer Science. Springer, 2018, pp. 218–234. DOI: 10.1007/978-3-030-01216-8_14. URL: https://doi.org/10.1007/978-3-030-01216-8_14.

[208] Yuxin Wu and Kaiming He. "Group Normalization". In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Vol. 11217. Lecture Notes in Computer Science. Springer, 2018, pp. 3–19. DOI: 10.1007/978-3-030-01261-8_1. URL: https://doi.org/10.1007/978-3-030-01261-8_1.

[209] DeepMind. *Biggan Deep 512 | .* 2019. URL: https://tfhub.dev/deepmind/biggan-deep-512/1 (visited on 06/11/2020).

[210] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6105–6114. URL: http://proceedings.mlr.press/v97/tan19a.html.

[211] Qizhe Xie, Eduard H. Hovy, Minh-Thang Luong, and Quoc V. Le. "Self-training with Noisy Student improves ImageNet classification". In: *CoRR* abs/1911.04252 (2019). arXiv: 1911.04252. URL: http://arxiv.org/abs/1911.04252.

[212] Luke Melas-Kyriazi. *lukemelas/EfficientNet-PyTorch*. original-date: 2019-05-30T05:24:11Z. June 2020. URL: https://github.com/lukemelas/EfficientNet-PyTorch (visited on 06/03/2020).

[213] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, and Dimitris Tsipras. *Robustness (Python Library)*. 2019. URL: https://github.com/MadryLab/robustness.

[214] Eric Wong, Leslie Rice, and J. Zico Kolter. "Fast is better than free: Revisiting adversarial training". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=BJx040EFvH.

[215] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980 (visited on 06/11/2020).

[216] I. Dunn. "Supplementary Material for Testing Deep Image Classifiers Using Generative Machine Learning (DPhil Thesis)". In: (2022). DOI: 10.5287/bodleian:5zJwdjna0.

[217] Oliver Willers, Sebastian Sudholt, Shervin Raafatnia, and Stephanie Abrecht. "Safety Concerns and Mitigation Approaches Regarding the Use of Deep Learning in Safety-Critical Perception Tasks". In: *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops - DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings*. Ed. by António Casimiro, Frank Ortmeier, Erwin Schoitsch, Friedemann Bitsch, and Pedro M. Ferreira. Vol. 12235. Lecture Notes in Computer Science. Springer, 2020, pp. 336–350. DOI: 10.1007/978-3-030-55583-2_25. URL: https://doi.org/10.1007/978-3-030-55583-2_25.

[218] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. "ImageNet-Trained CNNs Are Biased towards Texture; Increasing Shape Bias Improves Accuracy and Robustness". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=Bygh9j09KX.

[219] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. "Do Adversarially Robust ImageNet Models Transfer Better?" In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/24357dd085d2c4b1a88a7e0692e60294-Abstract.html.

[220] Tengyu Ma. "Generalization and equilibrium in generative adversarial nets (GANs) (invited talk)". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. Ed. by Ilias Diakonikolas, David Kempe, and Monika Henzinger. ACM, 2018, p. 2. DOI: 10.1145/3188745.3232194. URL: https://doi.org/10.1145/3188745.3232194.

[221] Sanjit A. Seshia, Somesh Jha, and Tommaso Dreossi. "Semantic Adversarial Deep Learning". In: *IEEE Des. Test* 37.2 (2020), pp. 8–18. DOI: 10.1109/MDAT.2020.2968274. URL: https://doi.org/10.1109/MDAT.2020.2968274.

[222] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. "advertorch v0.1: An Adversarial Robustness Toolbox based on PyTorch". In: *CoRR* abs/1902.07623 (2019). arXiv: 1902.07623. URL: http://arxiv.org/abs/1902.07623.

[223] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. "Hierarchical Text-Conditional Image Generation with CLIP Latents". In: *CoRR* abs/2204.06125 (2022). DOI: 10.48550/arXiv.2204.06125. arXiv: 2204.06125.

[224] Chitwan Saharia et al. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding". In: *CoRR* abs/2205.11487 (2022). DOI: 10.48550/arXiv.2205.11487. arXiv: 2205.11487.

[225] Jiahui Yu et al. "Scaling Autoregressive Models for Content-Rich Text-to-Image Generation". In: *CoRR* abs/2206.10789 (2022). DOI: 10.48550/arXiv.2206.10789. arXiv: 2206.10789.

[226] Sandra Wachter, Brent Mittelstadt, and Chris Russell. *Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR*. en. SSRN Scholarly Paper ID 3063289. Rochester, NY: Social Science Research Network, Oct. 2017. URL: https://papers.ssrn.com/abstract=3063289 (visited on 10/28/2019).

[227] Shubham Sharma, Jette Henderson, and Joydeep Ghosh. "CERTIFAI: Counterfactual Explanations for Robustness, Transparency, Interpretability, and Fairness of Artificial Intelligence Models". In: *arXiv:1905.07857 [cs, stat]* (May 2019). arXiv: 1905.07857 [cs, stat]. URL: http://arxiv.org/abs/1905.07857 (visited on 12/16/2019).

[228] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[229] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123. URL: https://doi.org/10.1109/ICCV.2015.123.

[230] *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/group?id=ICLR.cc/2019/Conference.

Appendices

A

Introduction to Deep Neural Networks

Deep neural networks are functions $f_\theta : \mathbb{X} \to \mathbb{Y}$ that map inputs $x \in \mathbb{X}$ to outputs $f_\theta(x) \in \mathbb{Y}$, parametrised by parameters $\theta \in \Theta$. By searching for appropriate values of θ, they can be 'trained' using data to approximate some target function over the distribution that the training data are drawn from. In fact, various universal approximation theorems assert that suitably rich neural networks can approximate various classes of target functions, including continuous functions and Lebesque integrable functions. But these theoretical results are not the main reason for their popularity. They are popular because of their (somewhat unexplained) success in practice.

A.1 Single-layer perceptron

A.1.1 Scalar output

Single-layer perceptrons with single scalar outputs are the nodes that are the building blocks of deep neural networks. Consider a function with input space $\mathbb{X} = \mathbb{R}^m$ for some positive integer m, output space $\mathbb{Y} = \mathbb{R}$, and parameters $\Theta = \mathbb{R}^{m+1}$. The parameters θ can be split into the weights $w \in \mathbb{R}^m$ and the bias $b \in \mathbb{R}$. A single-layer perceptron first performs a linear transformation, then

applies a non-linear *activation function* ϕ to the result:

$$f_\theta(x) = \phi(w \cdot x + b) = \phi(\theta \cdot (x \oplus 1)),$$

where $\theta = (w, b)$ and $x \oplus 1$ denotes the $(m+1)$-dimensional vector consisting of the elements of x followed by additional element of value 1. Possible activation functions ϕ include:

- the Heaviside step function, $\phi(z) = 1$ if $z > 0$ else 0,
- the hyperbolic tangent, $\phi(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$,
- the logistic function, $\phi(z) = \frac{1}{1+e^{-z}}$,
- the rectified linear unit (ReLU), $\phi(z) = z$ if $z > 0$ else 0,
- the leaky ReLU, $\phi(z) = z$ if $z > 0$ else $0.01z$.

In practice, since the derivatives of these functions provide the update values during training, the Heaviside step function is rarely used, and the sigmoid functions have fallen out of favour due to the low magnitude of their derivatives for much of their range. The ReLU and related activation functions are commonly used.

A.1.2 Vector output

Consider duplicating the above scalar-output node n times in parallel, producing n parallel outputs separately using n different sets of weights and biases. The result is a single-layer perceptron $f_\theta : \mathbb{X} \to \mathbb{Y}$, now with output space $\mathbb{Y} = \mathbb{R}^n$ and parameter space $\Theta = \mathbb{R}^{n \times m+1}$. Defining $x \oplus 1$ as above,

$$f_\theta(x) = \phi(\theta \cdot (x \oplus 1)),$$

where the $n \times m+1$ weight matrix θ is multiplied by the $m+1 \times 1$ matrix $(x \oplus 1)$, and where the activation function ϕ operates elementwise.

A.2 Multi-layer network

A single-layer perceptron is a function that maps from m- to n-dimensional vectors. So we can stack these layers, applying one perceptron to the output of another if the input dimension of the former matches the output dimension of the latter. In this way, we obtain a multi-layer network, which is a function $f_\theta : \mathbb{X} \to \mathbb{Y}$, consisting of a sequence of l layers, each of which is a single-layer perceptron. The parameters θ consist of a matrix of parameters for each layer: $\theta = (\theta_1, \theta_2, \ldots, \theta_l)$. So the ith layer, is the single-layer perceptron:

$$f_\theta^{(i)}(a_{i-1}) = \phi(\theta_i \cdot (a_{i-1} \oplus 1)),$$

where a_{i-1} will be the output from the previous layer (its *activation value*, since the activation function ϕ has been applied to it). Besides needing to map from inputs in $\mathbb{X}$ to eventual outputs in $\mathbb{Y}$, the number of layers in the network and the number of units in the various hidden (internal) layers is flexible, and is a matter of design.

A *deep* neural network (DNN) is simply a multi-layer network with a large number of layers. In practice, these have proven successful.

A.3 Training

Given a particular neural network $f_\theta : \mathbb{X} \to \mathbb{Y}$, we need a procedure for picking values of θ that make f a useful model for some purpose. Since this is a machine learning model, we learn θ using data. In general, we design a differentiable loss function $l : \mathbb{X} \times \mathbb{Y} \to \mathbb{R}$ that measures the error (or other undesirable property) in the network output $f_\theta(x)$ for a particular input x. A network whose outputs induce low loss values should be desirable. In particular, we must have some source of data that we can use to train our network. For example, one application that will be revisited in this thesis is supervised classification. In this domain, there will be a dataset of correctly labelled examples, and the loss function will penalise a model to the extent that it does not predict the correct label.

Modelling the data source as a probability distribution, p_{train}, we want to search for parameters θ that minimise the expected training loss over this distribution:

$$\arg\min_{\theta} \mathop{\mathbb{E}}_{x \sim p_{train}} l(x, f_\theta(x)).$$

This can be done using gradient descent. Assuming that we can compute the derivatives $\frac{\partial}{\partial \theta} \mathbb{E}\, l(x, f_\theta(x))$, with respect to each element of θ, then gradient descent makes the following update to these parameters:

$$\theta := \theta - \gamma \frac{\partial}{\partial \theta} \mathop{\mathbb{E}}_{x \sim p_{train}} l(x, f_\theta(x)),$$

where γ is a scalar known as the learning rate, which controls the size of each update. Given a finite training dataset D of inputs x sampled from p_{train}, the most straightforward approach to computing the expectation $\mathbb{E}_{x \sim p_{train}}\, l(x, f_\theta(x))$ is to compute the empirical average $\frac{1}{|D|} \sum_{x \in D} l(x, f_\theta(x))$.

In practice, though, a variant known as stochastic gradient descent (SGD) is used. Rather than computing the gradient over the entire dataset for each update, the dataset is partitioned into "mini-batches" D_i of m inputs each. Then the ith gradient update to the parameters becomes

$$\theta := \theta - \gamma \frac{\partial}{\partial \theta} \frac{1}{m} \sum_{x \in D_i} l(x, f_\theta(x)).$$

In practice, graphical processing units (GPUs) are exploited for their ability to perform the same operation in parallel on multiple data. So a mini-batch of examples may be processed simultaneously, resulting in rank-3 tensor, rather than merely matrix (rank 2), computation; the additional rank comes from processing more than one example at once.

The last remaining question is how to compute the derivative of the loss function with respect to the parameters θ. Since the loss function is differentiable, this reduces to computing the derivative of the output of the network. Noting that the only operations in the network are the activation function ϕ, which must be differentiable, and matrix multiplication, it is clearly possible. The backpropagation algorithm [228] exploits that the network is a directed acyclic graph of

operations by traversing the network in reverse, starting with its outputs, and recursively computing the derivative of each node using the already-computed derivatives of its successors; the derivatives of the weights are calculated at the same time as those of the activation values they are multiplied with. This is often referred to as a backwards pass through the network, as opposed to a forwards pass, which computes the output of the network given an input.

This relatively concise section should have provided sufficient background on feedforward (acyclic) neural networks for the purposes of this dissertation. For a more comprehensive introduction, refer to Goodfellow, Bengio, and Courville [197].

B

Training Generative Networks Experimental Particulars

Contents

B.1 Details of experimental setup

The WGAN-GP [30] and ACGAN [36] architectures were the starting points for the design of these neural networks. Only a small amount of manual hyperparameter tuning was performed.

The generator is a convolutional neural network, conditioned on class label.

The discriminator network is a combination of a conditional WGAN-GP critic, which learns an approximation of the Wasserstein distance between the generated and training-set conditional distributions, and an auxiliary classifier, which predicts the likelihood of the possible values of $h(x)$. We combined these two architectures in an attempt to strengthen the gradient provided to the generator, helping to generate data which are both realistic and for which

the true (i.e., human-judged) labels match the intended true labels. The critic is given the true label of the data $h(x)$ to improve its training, but the auxiliary classifier must not have access to this information since its purpose is to predict it. We therefore split the discriminator d into three sub-networks. Network $d_0 \colon X \to \mathbb{R}^i$ effectively preprocesses the input, passing an intermediate representation to the critic network $d_1 \colon \mathbb{R}^i \times Y \to \mathbb{R}$ and the auxiliary classifier network $d_2 \colon \mathbb{R}^i \to \mathbb{R}^{|Y|}$. In our experiments, both d_1 and d_2 were single fully-connected layers of the appropriate dimension. The loss terms from the WGAN-GP and ACGAN algorithms are simply summed. The auxiliary classifier helps the training converge, but is not necessary.

Table B.1: Architecture for generator network, g.

Layer Type	Kernel	Strides	Feature Maps	Batch Norm.	Dropout	Activation
Fully-Connected	N/A	N/A	64	No	0	ReLU
Transposed Conv.	5×5	2×2	32	Yes	0.35	LeakyReLU
Transposed Conv.	5×5	2×2	8	Yes	0.35	LeakyReLU
Transposed Conv.	5×5	2×2	4	Yes	0.35	LeakyReLU
Fully-Connected	N/A	N/A	784	No	0	Tanh

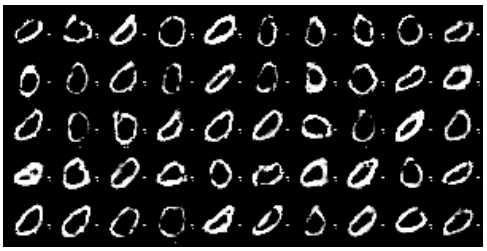
Table B.2: Architecture for discriminator subnetwork, d_0. No batch normalisation used.

Layer Type	Kernel	Strides	Feature Maps	Dropout	Activation
Convolution	3×3	2×2	8	0.2	LeakyReLU
Convolution	3×3	1×1	16	0.2	LeakyReLU
Convolution	3×3	2×2	32	0.2	LeakyReLU
Convolution	3×3	1×1	64	0.2	LeakyReLU
Convolution	3×3	2×2	128	0.2	LeakyReLU
Convolution	3×3	1×1	256	0.2	LeakyReLU

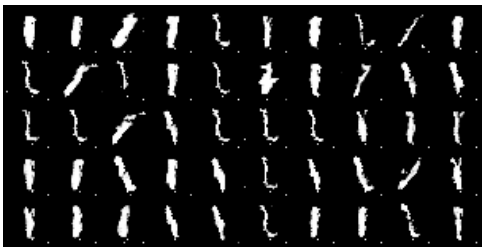
Table B.3: Hyperparameters for all networks.

Hyperparameter	Value
Attack rate	$\mu = 0.1$
Learning rate	$\alpha = 0.000005$
Adam betas	$\beta_1 = 0.6, \beta_2 = 0.999$
Leaky ReLU slope	0.2
Minibatch size	100
Dimensionality of latent space	128
Weight initialisation	Normally distributed [229]
Coefficient of gradient penalty loss term	$\lambda = 10$

B.1.1 Samples of MNIST Unrestricted Adversarial Examples



(a) Intended true label '0'.

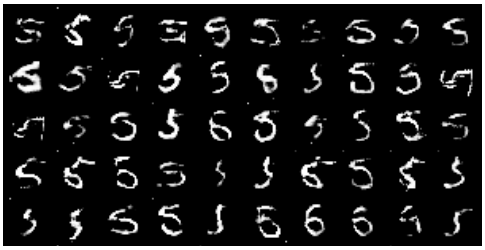


(b) Intended true label '1'.

(c) Intended true label '2'.

(d) Intended true label '3'.

(e) Intended true label '4'.

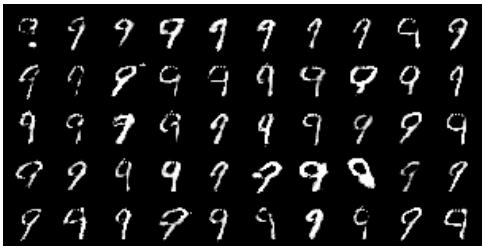


(f) Intended true label '5'.

(g) Intended true label '6'.

(h) Intended true label '7'.

(i) Intended true label '8'.



(j) Intended true label '9'.

Figure B.1: Examples generated by one adversarially-finetuned GAN to perform an untargeted attack on Wong and Kolter's (2018) classifier, which is provably robust to perturbation attacks.

B.2 Interfaces used by human judges

Figures B.2, B.3 and B.4 show full screenshots of the interfaces used for the experiments presented in Sections 4.3 and 4.5, including the instructions for participants.

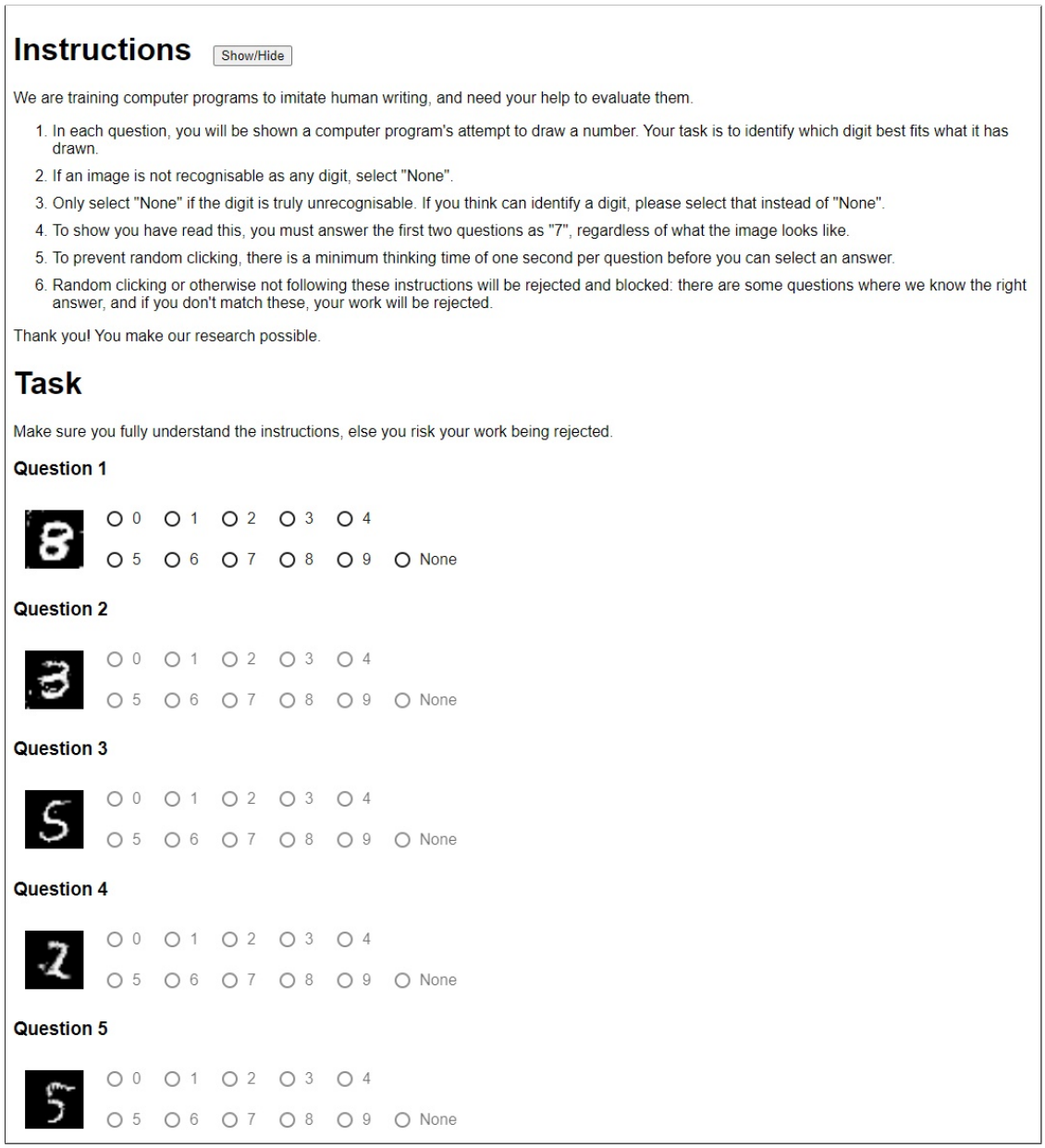


Figure B.2: Screenshot of the interface used by participants to label generated test inputs for the experiment described in Section 4.3, including the instructions.

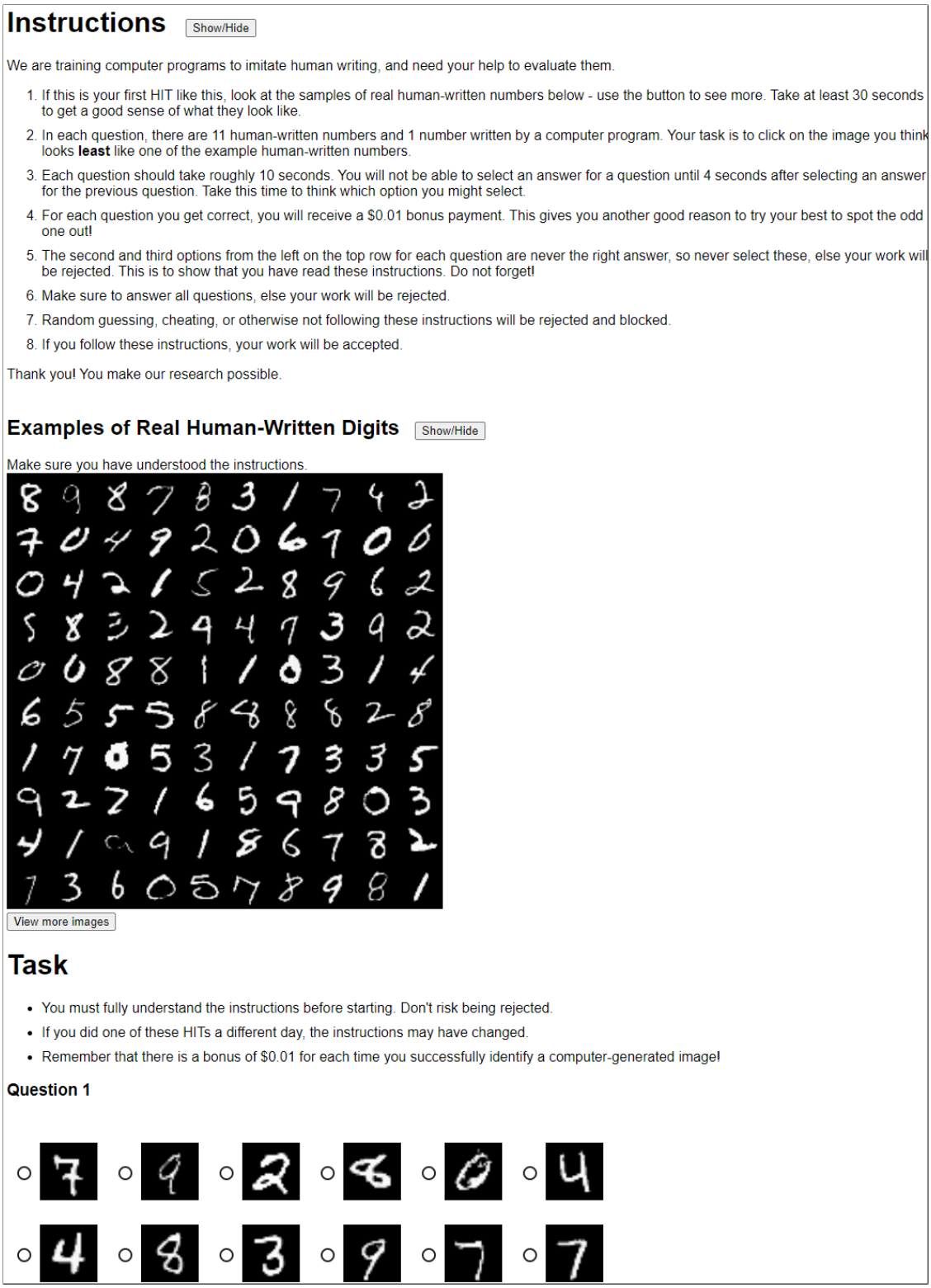


Figure B.3: Screenshot of the interface used by participants when trying to pick out which one image of ten is not drawn by a human for the experiments described in Section 4.5.

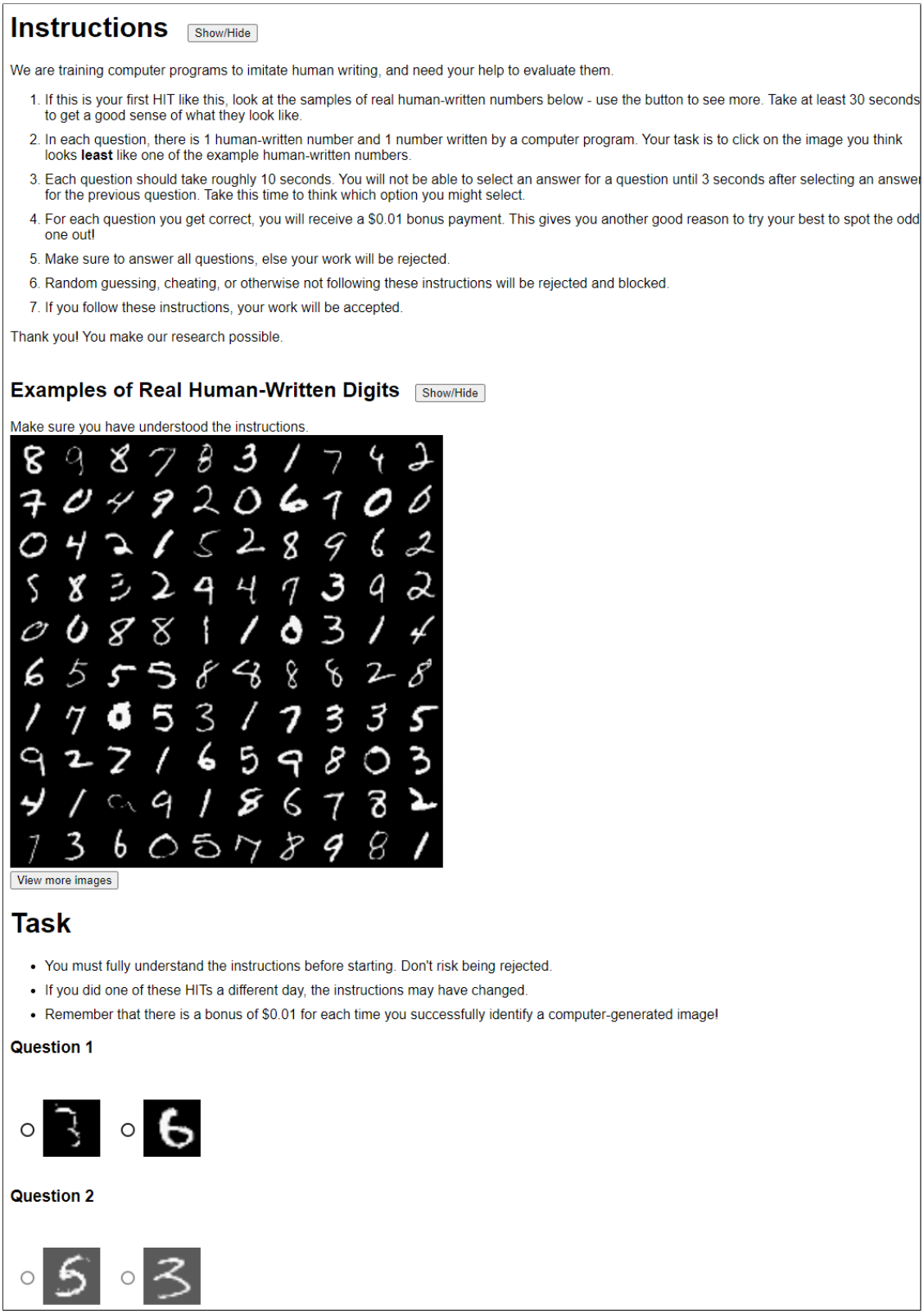


Figure B.4: Screenshot of the interface used by participants when trying to pick out which one image of two is not drawn by a human for the experiments described in Section 4.5.

C

Latent Generator Perturbations: Supplementary Materials

Contents

C.1 ImageNet: further examples

Figure C.1 shows the results for the first sixteen randomly-selected (y, z, t) tuples that were used throughout our experiments. The captions indicates when an example was skipped because the classifier misclassified the unperturbed seed image, and when an example was skipped because the true class of the unperturbed seed image did not match the intended class, y. The captions also indicate whether one of the human participants judged the perturbed image to match the class of the unperturbed image. The Engstrom robust ResNet50 is the classifier used for this figure.

Figures C.2 to C.5 give more examples of context-sensitive feature perturbations for each classifier and each different set of generator activations being perturbed.

Please refer to the additional online supplementary materials to this thesis [216], hosted by the Oxford University Research Archive. These include many more ImageNet examples, as well as many animations showing the effect of gradually introducing the perturbations to the latent activations of the generator. These give a much clearer intuition for the nature of the changes being made to the images; comparing static images alone can be difficult to interpret.

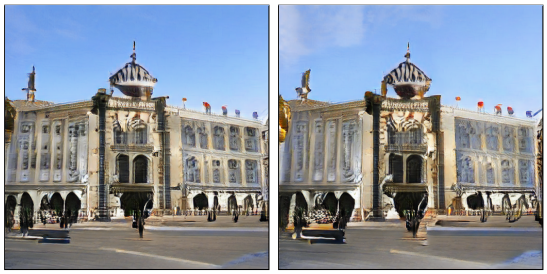
(a) Skipped because the classifier did not predict the desired label, 'Labrador retriever'.

(b) Skipped because the human did not agree that 'velvet' was the best description of the unperturbed image.

(c) Perturbed from 'Komodo dragon' (left) to 'overskirt' (right), but the human labeller did not agree that the label 'Komodo dragon' remained the best description of the perturbed image.

(d) Perturbed from 'file cabinet' (left) to 'doormat' (right); the human judged the true label of the perturbed image to remain unchanged.

(e) Perturbed from 'barn' to 'Afghan hound'; the human judged the true label of the perturbed image to remain unchanged.



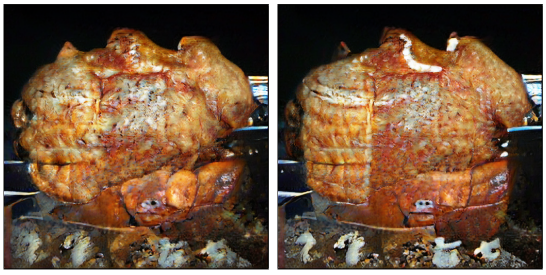
(f) Perturbed from 'palace' to 'throne'; the human judged the true label of the perturbed image to remain unchanged.

(g) Perturbed from 'reflex camera' to 'sunglass'; the human judged the true label of the perturbed image to remain unchanged.

(h) Perturbed from 'Blenheim spaniel' to 'black-and-tan coonhound'; the human judged the true label of the perturbed image to remain unchanged.

Figure C.1: The first examples used in our experiments. Perturbed images for Engstrom et al.'s adversarially-trained classifier [213] are to the right of each unperturbed image.

(i) Skipped because the classifier did not predict the desired label, 'bath towel'.



(j) Perturbed from 'rotisserie' to 'rain barrel'; the human judged the true label of the perturbed image to remain unchanged.

(k) Perturbed from 'breastplate' to 'mud turtle'; the human judged the true label of the perturbed image to remain unchanged.

(l) Skipped because the human did not agree that 'spotted salamander' was the best description of the unperturbed image.

(m) Perturbed from 'yurt' to 'library'; the human judged the true label of the perturbed image to remain unchanged.

(n) Perturbed from 'cougar' to 'shower cap'; the human judged the true label of the perturbed image to remain unchanged.

(o) Perturbed from 'chickadee' to 'soap dispenser'; the human judged the true label of the perturbed image to remain unchanged.

(p) Skipped because the classifier did not predict the desired label, 'trombone'.

Figure C.1: Continued.

Figure C.2: Examples of feature perturbations for the two standard classifiers. For each, the bottom row shows the perturbed images for perturbations at different parts of the generator. The top row shows the pixel-wise difference between the original image and the perturbed image. Some of these have been scaled to be made more visible. The name of the classifier is shown in the top left; the top right shows the original and target label.

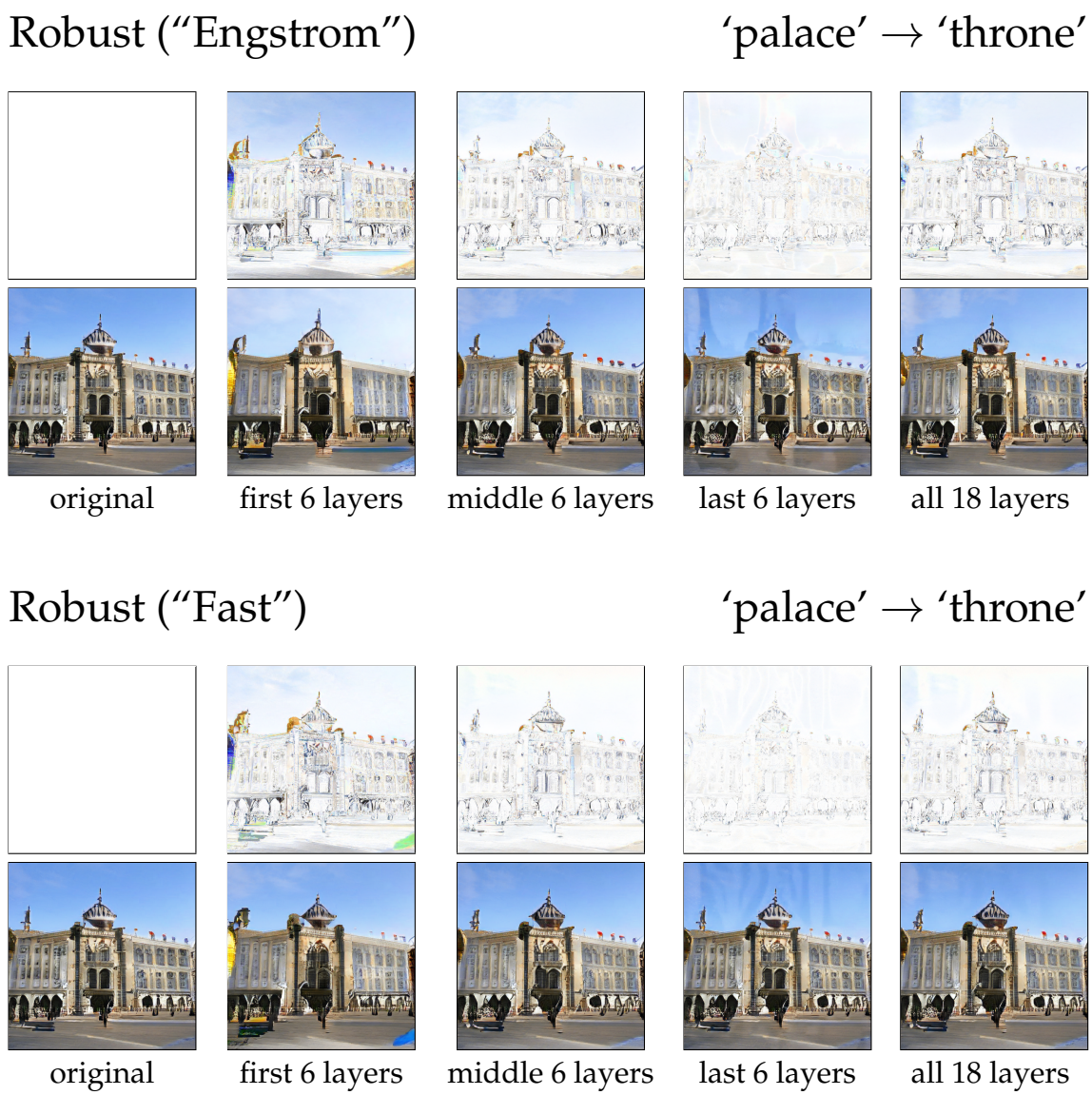


Figure C.3: Examples of feature perturbations for the two pixel-robust classifiers. For each, the bottom row shows the perturbed images for perturbations at different parts of the generator. The top row shows the pixel-wise difference between the original image and the perturbed image. The name of the classifier is shown in the top left; the top right shows the original and target label.

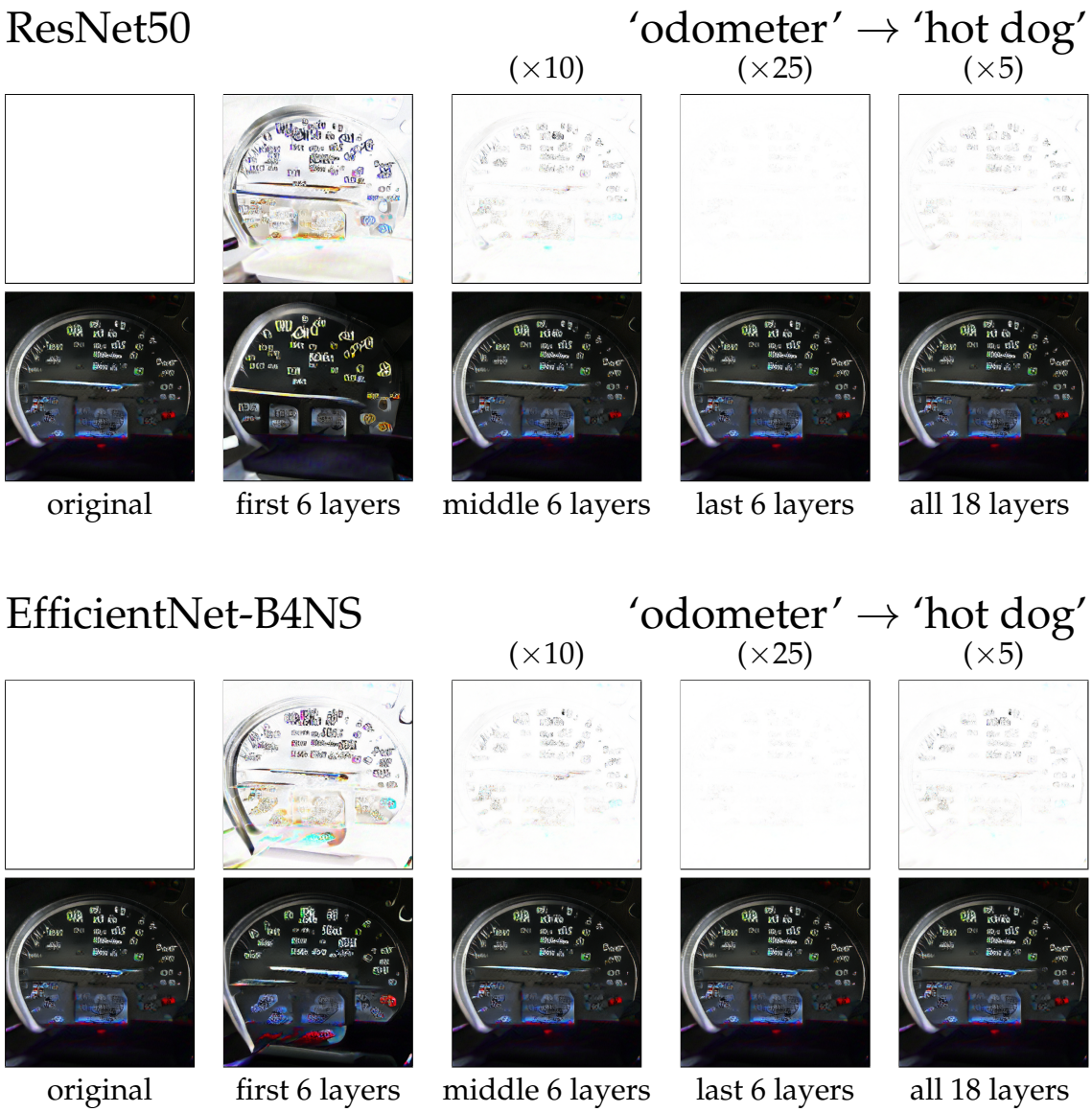


Figure C.4: Examples of feature perturbations for the two standard classifiers. For each, the bottom row shows the perturbed images for perturbations at different parts of the generator. The top row shows the pixel-wise difference between the original image and the perturbed image. Some of these have been scaled to be made more visible. The name of the classifier is shown in the top left; the top right shows the original and target label.

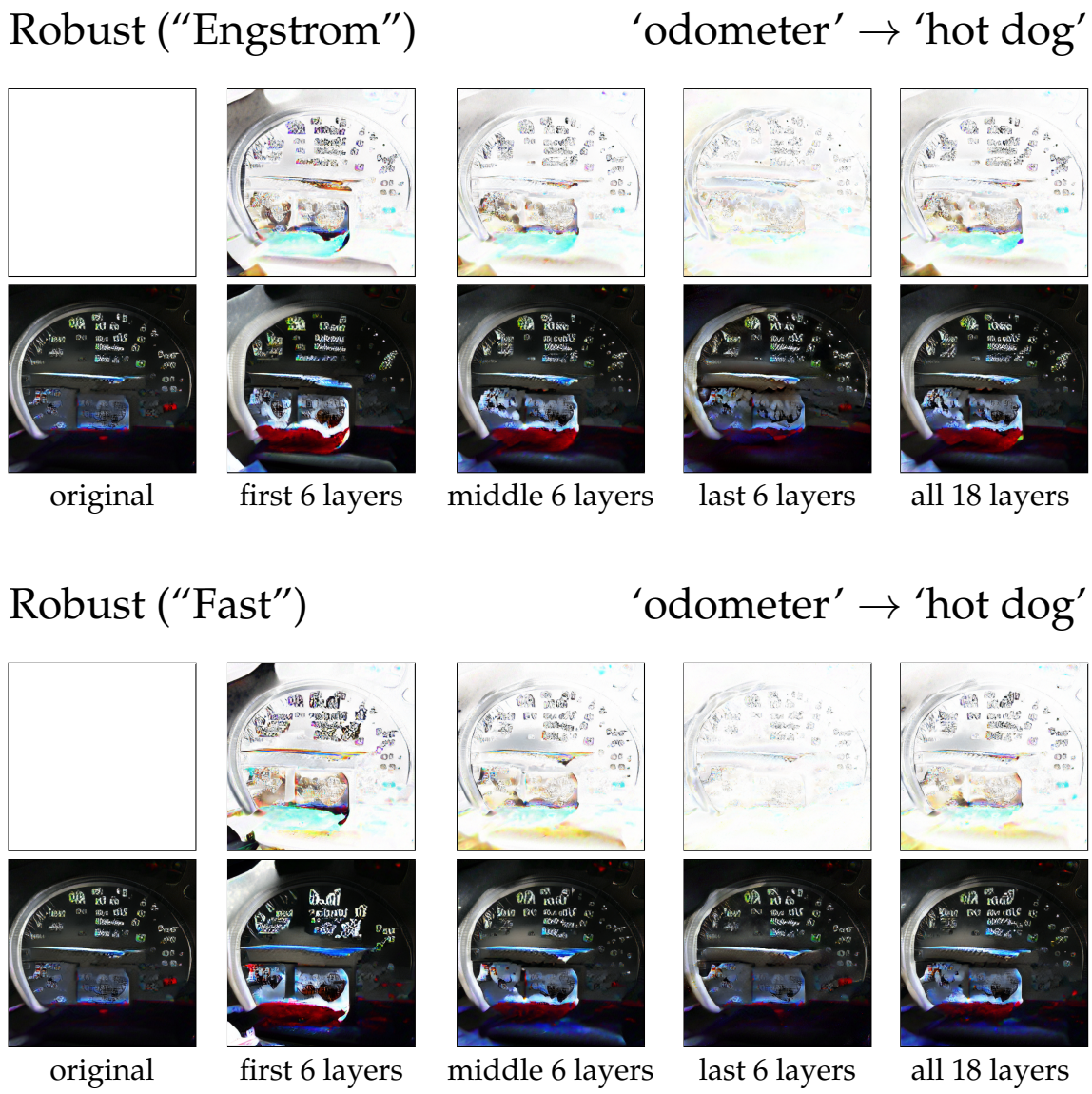


Figure C.5: Examples of feature perturbations for the two pixel-robust classifiers. For each, the bottom row shows the perturbed images for perturbations at different parts of the generator. The top row shows the pixel-wise difference between the original image and the perturbed image. The name of the classifier is shown in the top left; the top right shows the original and target label.

C.2 CelebA-HQ

Additional experiments were done with the CelebA-HQ dataset. This section contains a table specifying the architecture of the generator architecture used, and gives several more examples of perturbed test cases.

Table C.1: CelebA-HQ convolutional generator architecture. Each row represents a layer. Each horizontal rule marks an activation tensor at which perturbations are performed.

Fully-Connected	(8192 units)
LeakyReLU	(Slope -0.2)
Reshape	(To batch of $512 \times 4 \times 4$ tensors)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 512 feature maps)
LeakyReLU	(Slope -0.2)
Upscale	(To 8×8)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 512 feature maps)
LeakyReLU	(Slope -0.2)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 512 feature maps)
LeakyReLU	(Slope -0.2)
Upscale	(To 16×16)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 512 feature maps)
LeakyReLU	(Slope -0.2)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 512 feature maps)
LeakyReLU	(Slope -0.2)
Upscale	(To 32×32)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 512 feature maps)
LeakyReLU	(Slope -0.2)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 512 feature maps)
LeakyReLU	(Slope -0.2)
Upscale	(To 64×64)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 256 feature maps)
LeakyReLU	(Slope -0.2)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 256 feature maps)
LeakyReLU	(Slope -0.2)
Upscale	(To 128×128)
2D Convolution	(64×64 kernel, stride 1, padding size 1, 128 feature maps)
LeakyReLU	(Slope -0.2)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 128 feature maps)
LeakyReLU	(Slope -0.2)
Upscale	(To 256×256)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 64 feature maps)
LeakyReLU	(Slope -0.2)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 64 feature maps)
LeakyReLU	(Slope -0.2)
Upscale	(To 512×512)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 32 feature maps)
LeakyReLU	(Slope -0.2)
2D Convolution	(3×3 kernel, stride 1, padding size 1, 32 feature maps)
LeakyReLU	(Slope -0.2)
2D Convolution	(1×1 kernel, stride 1, 3 feature maps)

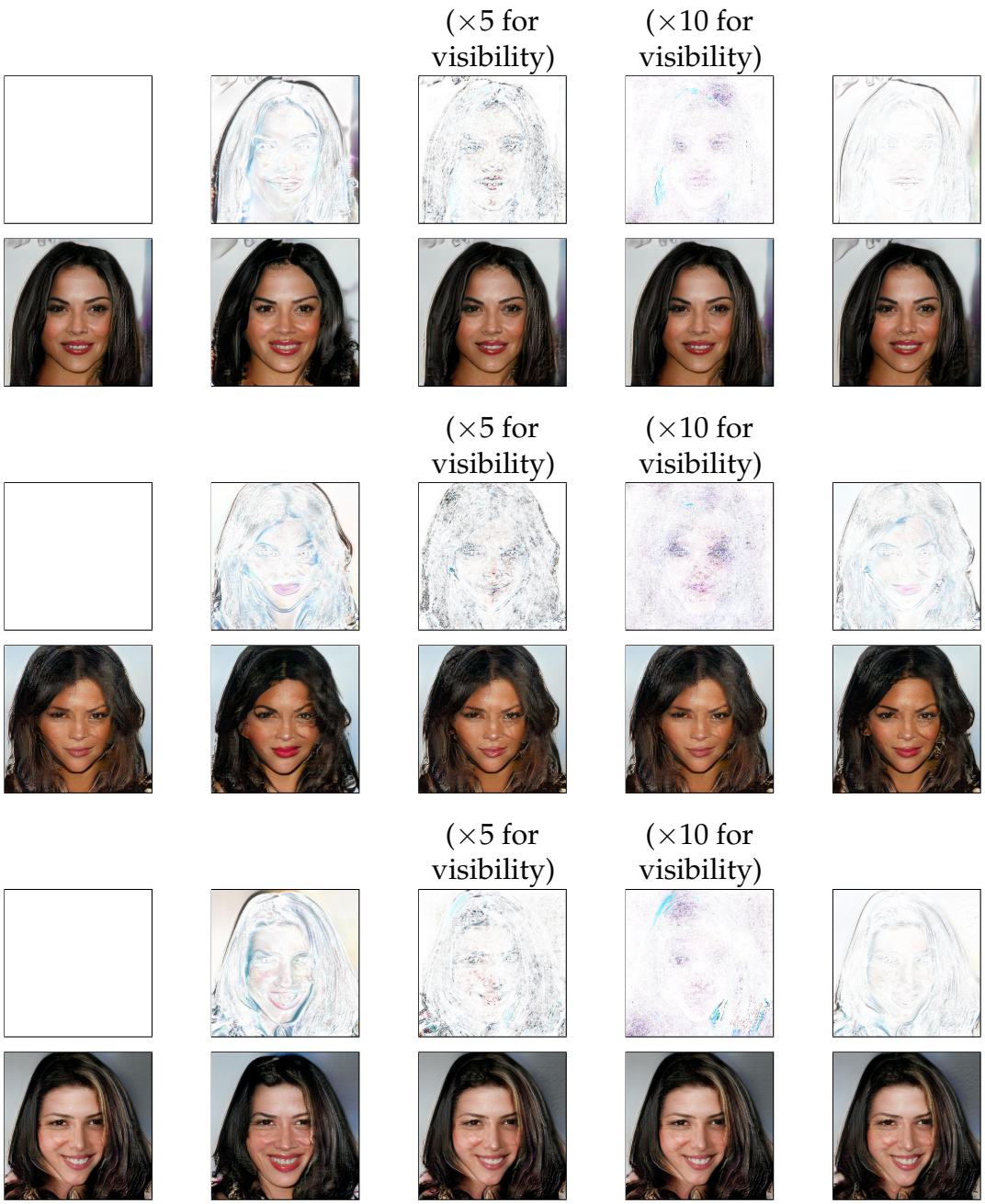


Figure C.6: A random selection of context-sensitive feature perturbations at different granularities, as controlled by perturbing activations at the generator layers indicated under each image. Differences with the unperturbed image are shown above each perturbed image. Each perturbed image has the following labels predicted positively: 'Bald', 'Blond hair', 'Eyeglasses', 'Goatee', 'Grey hair', 'Moustache', 'No beard', 'Wearing hat', 'Wearing necklace', and 'Wearing necktie'.

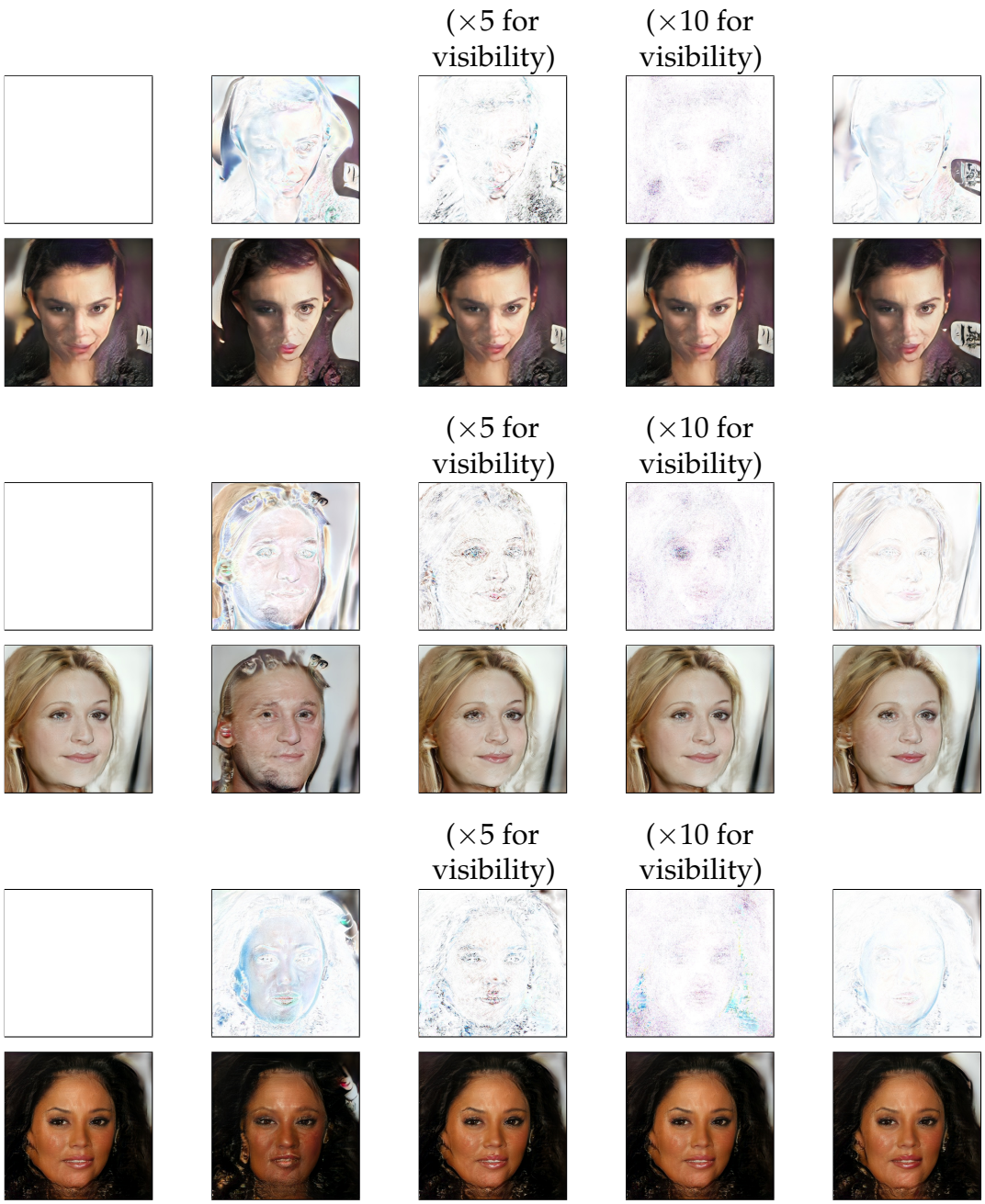


Figure C.7: A random selection of context-sensitive feature perturbations at different granularities, as controlled by perturbing activations at the generator layers indicated under each image. Differences with the unperturbed image are shown above each perturbed image. Each perturbed image has the following labels predicted positively: 'Bald', 'Blond hair', 'Eyeglasses', 'Goatee', 'Grey hair', 'Moustache', 'No beard', 'Wearing hat', 'Wearing necklace', and 'Wearing necktie'.

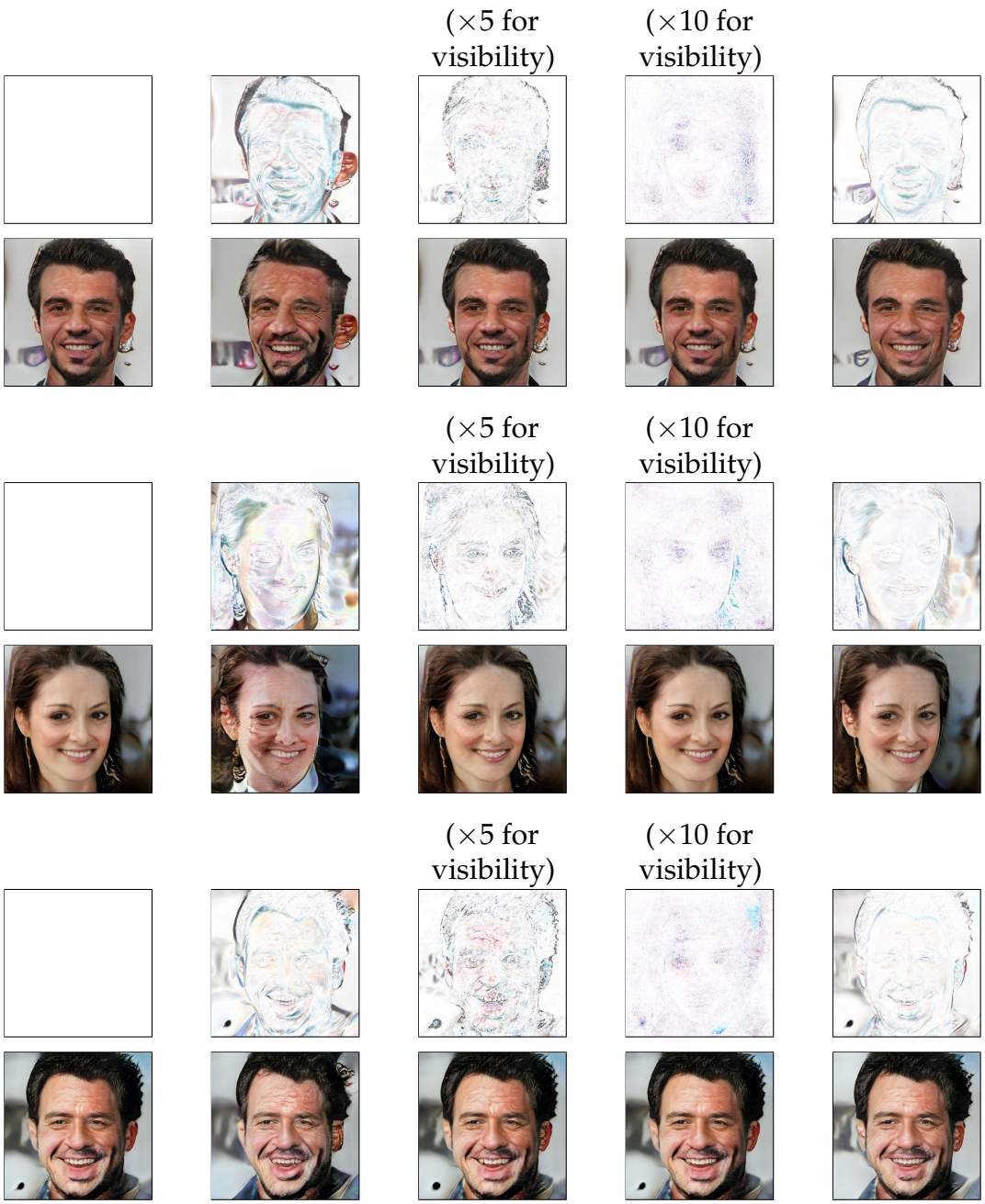


Figure C.8: A random selection of context-sensitive feature perturbations at different granularities, as controlled by perturbing activations at the generator layers indicated under each image. Differences with the unperturbed image are shown above each perturbed image. Each perturbed image has the following labels predicted positively: 'Bald', 'Blond hair', 'Eyeglasses', 'Goatee', 'Grey hair', 'Moustache', 'No beard', 'Wearing hat', 'Wearing necklace', and 'Wearing necktie'.