Dissection of Deep Neural Networks

by

David Bau

A.B., Harvard University (1991)
M.S., Cornell University (1994)

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

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author .....
Department of Electrical Engineering and Computer Science
August 27, 2021

Certified by .....
Antonio Torralba
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

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

Dissection of Deep Neural Networks

by

David Bau

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

We investigate the role of neurons within the internal computations of deep neural networks for computer vision.

We introduce network dissection, a method for quantifying the alignment between human-interpretable visual concepts and individual neurons in a deep network. We apply network dissection to examine and compare the internal computations of several networks trained to classify and represent images, and we ask how well human-understandable concepts align with neurons at different layers, in different architectures, with various training objectives; we also compare neurons to random linear combinations of neurons, and examine emergence of concepts as training proceeds.

Then, we adapt network dissection to analyze generative adversarial networks. In GAN dissection, human-understandable neurons are identified by applying a semantic segmentation model to generated output. We find that small sets of neurons control the presence of specific objects within synthesized scenes. We also find that activating neurons reveals modeled rules and interactions between objects and their context.

We then ask how to dissect and understand the omissions of a generative network. Omissions of human-understandable objects can be quantified by comparing semantic segmentation statistics between the training distribution and the generated distribution. Then we develop a method that can invert and reconstruct generated images in a progressive GAN, and show that this reconstruction can visualize specific cases in which the GAN omits identified object classes.

Finally, we ask how rules within a generative model are represented. We hypothesize that the layers of a generative model serve as a memory that stores associations from representations of concepts at the input of a layer to patterns of concepts at the output to the layer, and we develop a method for rewriting the weights of a model by directly rewriting one memorized association. We show that our method can be used to rewrite several individual associative memories in a Progressive GAN or StyleGAN, altering learned rules that govern the appearance of specific object parts in the model.

Thesis Supervisor: Antonio Torralba
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

My PhD studies have occurred during the phase of my life when instead of going to school and puzzling over the role of neurons in deep networks, I should be working to support my children's college education. Yet in my choices I have always had the bedrock support and indulgence of my wife and partner in life, Heidi Yeh, and our three beloved children Cody, Piper, and Anthony. I am fortunate to have their love.

My research at MIT has been guided by the leadership, creativity, and encouragement of my advisor Antonio Torralba. I have also learned many lessons from his postdoc and my close collaborator and mentor, Jun-Yan Zhu. I will be forever grateful to have benefited from their wealth of experience and knowledge throughout my journey here. I have been blessed to have been given the chance to learn from them.

For inspiration, I thank the many senior academics who have been generous in sharing their time and imparting their wisdom as well as their obvious love of the academy and intellectual life. Their endless curiosity has nourished my own. I have been especially inspired by Hal Abelson, Gerald Sussman, Bill Freeman, Nancy Kanwisher, Aude Oliva, Jacob Andreas, Josh Tenenbaum, my advisors Daniela Rus, Rob Miller and Nick Trefethen, as well as the readers of this dissertation Phillip Isola, Alexei Efros, and Aleksander Madry.

The ideas in these pages derive from years of fertile conversation with my friends and collaborators Bolei Zhou, Jonas Wulff, Hendrik Strobelt, Ali Jahanian, Aditya Khosla, Alex Andonian, Lucy Chai, Tongzhou Wang, Javier Marín, Dídac Surís, Yonatan Belinkov, Julius Adebayo, Karren Yang, Prafull Sharma, Jan Stühmer, Adrià Recasens, Carl Vondrik, Jonathan Frankle, Leilani Gilpin, Josh Sheldon, Sheng-Yu Wang, Ser-Nam Lim, Daksha Yadav, Bill Ferguson, Jaden Fiotto-Kaufman, Josh Fasching, Pratushya Sharma, Sarah Schwettmann, Evan Hernandez, Shibani Santurkar, Dimitris Tsipras, Xavier Puig Fernandez, Nadiia Chepurko, Dim Papadopoulos, Ethan Weber, David Harwath, Wei-Chiu Ma, Manel Baradad, Shuang Li, Joanna Materzynska, Chuang Gan, Yunzhu Li, Jingwei Ma, Brian Cheung, Ankur Bhargava, Thomas Colthurst, James Synge, Rio LaVigne, Carrie Cai, Nick Hynes, Elena Glass-

man, Amy Zhang, William Peebles, Steven Liu, Mahi Elango, Christine You, Evan Shelhamer, James Gilles, Wendy Wei, Brian Park, Tony Peng, Brian Shimanuki, Kaveri Nadhamuni, Sam Boshar, Ben Gardner, Kevin Meng, Audrey Cui, and Anthony Bau. This dissertation would not have been possible without all of you.

During my PhD I have had the privilege to work on many fascinating research problems together with talented researchers at several industry and government partners, and I thank these organizations for both their collaboration and their financial support. These include the DARPA Explainable Artificial Intelligence (XAI) program, the MIT-IBM Watson AI Lab, DARPA SAIL-ON, Adobe Research, Facebook Research, and Philips Signify Research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis proposes a way to understand how deep networks work. We ask the fundamental question: *Do deep neural networks contain concepts?*

Inspired by decades of neuroscience research on biological brains,* I introduce methods for directly probing the internal structure of an artificial neural network by testing the activity of individual neurons and their interactions. Because in our computerized setting we have the luxury of examining every neuron of a trained network, we assume a detailed systems view: we investigate the role of every individual unit, and test every unit on a broad range of stimuli.

By beginning with the simplistic proposal that an individual neuron might represent one internal concept,† we pursue our fundamental question in a concrete, quantitative way: Which neurons? Which concepts? What are concept neurons used for? Then: Can we see which concepts are missing? And: Can we see rules governing concept relationships? Taken together, the research challenges the notion that a neural network is hopelessly opaque. Instead, we tear back the curtain and chart a path through the

Defense talk video, demos, code and data at `https://dissection.csail.mit.edu/`.

*In experiments ranging from the Lettvin et al. [1959] study of the frog optic nerves to the Quiroga [2012] observation of a single neuron selective for one particular celebrity, neuroscience has a long and continuing tradition of measuring the response of single neurons to particular classes of nontrivial stimuli. Several inspiring experiments are described in Chapter 2.

†This is a simplification of the neuron doctrine [Barlow, 1972], which is not the modern consensus in neuroscience; mainstream views are closer to the distributed coding model [Haxby et al., 2001]. Nevertheless, for understanding artificial neural networks we take the old neuron doctrine seriously with full awareness of distributed codes. We discuss this debate further in Chapter 2.

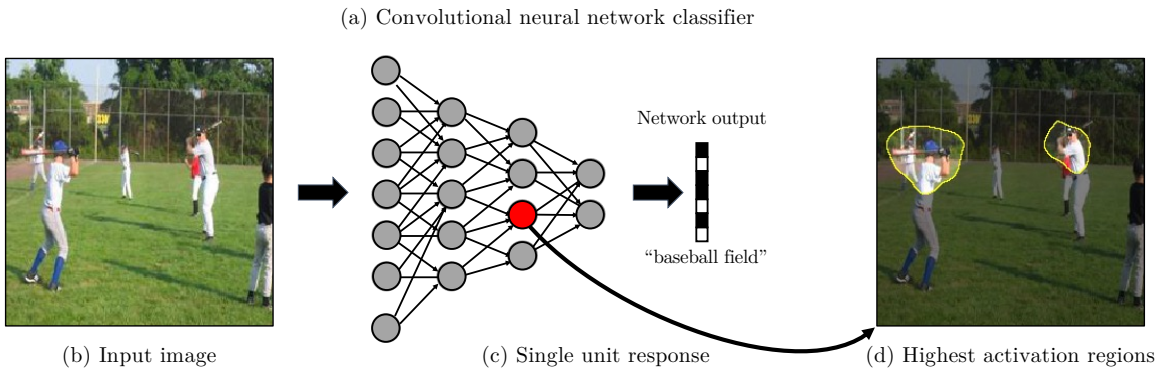


Figure 1-1: The response of one unit to one image. In a convolutional network, individual neurons are part of a *unit* of identically-parameterized neurons that perform the same calculation at each location in the visual field. To visualize the response of a neuron, we consider it together with the full grid of neurons within its unit, and highlight the portions of the image in which those neurons are firing strongest. Here unit 208 of layer `conv5_3` of the VGG-16 network activates on top parts of bodies of two baseball players in an image.

detailed structure of a network by which we can begin to understand its logic.

In Chapter 2 we review the literature to survey other approaches for understanding neural networks, and we also review the history of classical neuroscience experiments that motivated the neuron doctrine. Then in subsequent chapters we present the methods, experiments, and results of our current work. Each of Chapters 3-6 describes a set of experimental results that I have previously presented at computer vision conferences together with collaborators who are noted in each chapter. The remainder of this chapter gives an extended overview.

1.1 Dissecting a classifier

What is a neuron's purpose? We know that individual units have been seen to respond to object classes, parts, textures, tense, gender, context, and sentiment (see Figures 1-1, 1-2 and [Zeiler and Fergus, 2014, Zhou et al., 2015, Mahendran and Vedaldi, 2015, Karpathy et al., 2016, Radford et al., 2017]). And when images are synthesized to maximize the response of single neurons, the results can resemble real-world objects such as faces or animal parts or vehicles [Erhan et al., 2009, Szegedy et al., 2014, Mordvintsev et al., 2015, Nguyen et al., 2015, Olah et al., 2017, Mahendran and Vedaldi, 2015, Nguyen et al., 2016, 2017].

Figure 1-2: The specificity and invariance of a unit is revealed by testing its response on many images. Unit 208 is specific to people wearing hats. It does not activate strongly on non-hat-wearing-people, but it activates on hat-wearers in a variety of contexts, poses, and types of hats. In network dissection, we map out such selectivity by comparing the responses of every unit in a network to human-labeled segmentations of images in which more than 1000 visual concepts such as object, parts, materials, and textures are labeled. (This unit emerges in a VGG-16 network trained to classify images of places.)

But are such single-neuron phenomena systematic? To understand the selectivity of individual neurons quantitatively, we ought to investigate every neuron and test selectivity for every concept we can imagine. We should test each concept using many images, and compare concept neurons to baselines using other possible encodings.

In Chapter 3, we pursue that goal by introducing the method of Network Dissection. Rather than testing each neuron on a single image or a single concept, we test all neurons on a data set we construct for the purpose (Section 3.2.1). In all, we use more than 60,000 labeled images with more than 1,000 visual concepts to conduct tests on every neuron in every layer of several image classification networks.

Our approach allows us to investigate the specifics of the sparse neural code employed by a network. For example, beyond identifying a neuron that is selective for trees, we can find and count every neuron in the network that is selective for trees. By standardizing a metric and applying the same measurement across networks, layers, and concepts, we can also make quantitative comparisons about the emergence of concepts globally (Figure 1-3). Our experiments confirm that neurons in a convolutional network detect a hierarchy of visual concepts of increasing complexity by layer, with simple textures dominating early layers and neurons for abstract object classes emerging in later layers. We also find the presence of neurons that detect parts and object classes

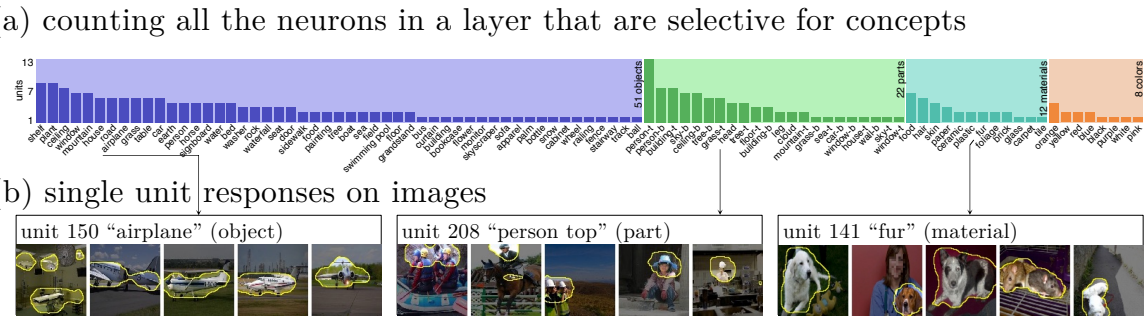


Figure 1-3: Charting the concepts of a layer. By testing every individual neuron (b) against all the visual concepts in a large set we can build a map of every neuron for each concept (a). The height of each bar shows the number of neurons that match the given concept. Individual concept names can be read by magnifying the figure. This plot uses the method of Bau et al. [2020], which builds upon the method described in Chapter 3.

that are not explicit in the training task, such as neurons specific to human faces in networks that were trained to distinguish scene classes. And we identify conditions under which more or fewer object detection neurons emerge (Sections 3.3.3-3.3.7).

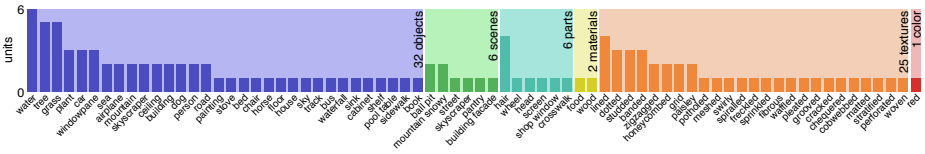
Distributed coding advocates [Plaut and McClelland, 2010, Averbeck et al., 2006, Haxby et al., 2001] might object to the special attention we pay to single neurons, because one might argue that concepts live within a population of neurons, not within individual neurons. Neurons could obtain their selectivity simply by being undistinguished members of a powerful dense distributed code in which any arbitrary feature combination would be selective for a meaningful concept. Network dissection allows us to test that hypothesis directly: we form randomized feature combinations, and compare their selectivity for concepts against that of individual neurons. When we perform that test (Section 3.3.2), we find that the dense distributed code model does not explain selectivity. Although single neurons are not perfect matches for human-meaningful concepts, they do match concepts much better than arbitrary feature combinations within the population (Figure 1-4).

Therefore we conclude that neurons are selective for meaningful concepts, and networks do contain many such neurons. These findings lead to two further puzzles:

1. What causal role does a concept neuron have within the network?[‡] Does activat-

[‡]When we ask about the causal role of a neuron, we are not asking about causality in the real world, but the simpler question of causality within the network computation: how a neuron’s output

(a) The learned neurons of Alexnet Layer 5: 72 concepts with IoU > 0.04



(b) Units of a random basis for the same layer: 18 concepts with IoU > 0.04

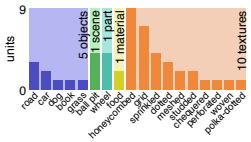


Figure 1-4: Testing the dense distributed code hypothesis. The units of a layer align with more human-meaningful visual concepts (a) than would be expected for an arbitrary linear coding of the same representation space (b). The two image representations have perfectly equivalent power and classification accuracy because each is a feature-space rotation of the other. Experiment from Section 3.3.2.

ing a concept neuron cause the network to perceive that concept?

2. Why do neurons tend toward human-meaningful concepts? Do concept detectors arise due to the supervision of human-created labels in training?

Neuron causality can be investigated in classifiers by measuring the impact of removal of neurons on accuracy. Although removing one concept neuron tends to have negligible impact on overall accuracy [Morcos et al., 2018], neuron removal does have a strong impact on the accuracy of individual classes. In Bau et al. [2020] and Zhou et al. [2018], our measurements of causal links between neuron and classes hint at a neuron's purpose. For example, removing the hat neuron damages classification accuracy for the baseball field and construction site classes, which might suggest that the network was blinded to baseball hats and construction helmets. However, evidence from such experiments remains circumstantial, since we cannot directly ask the classifier what it perceives when a neuron is removed.

The challenge of clarifying the purpose and causal role of concept-correlated neurons motivates us to move beyond classifiers to study an unsupervised generative setting, which we introduce next.

causes the network behavior to change. This can be tested by overriding the neuron's output and substituting a given value (e.g. set to zero to remove a neuron) when running the network.

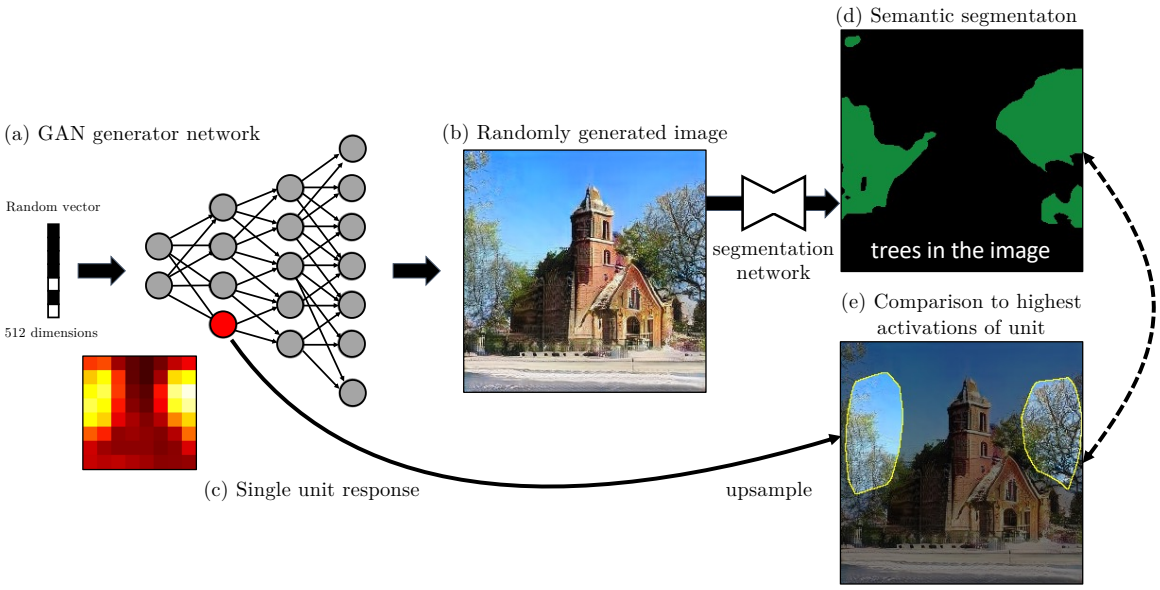


Figure 1-5: When a GAN network (a) synthesizes a random realistic image (b), the semantics for a single unit can be measured by comparing the unit response (c) to the predictions of a segmentation network trained to locate human-meaningful concepts in an image (d). Here a single unit has some agreement with the locations of trees in the generated image (e). In GAN dissection, we measure this agreement across many images.

1.2 Dissecting a generator

An unconditional Generative Adversarial Network (GAN) is trained on the task of transforming a noise vector to a random realistic image that imitates a sample from an unlabeled training set [Goodfellow et al., 2014]. A GAN trained using current methods [Karras et al., 2018, 2019, 2020] can produce high-quality complex output such as realistic scenes that contain buildings and trees and other objects. Such realistic output suggests an ability to model meaningful structure in the world, even though the GAN is trained without the detailed supervision of human-annotated labels.

When a GAN draws an image that contains a tree, we wish to understand: does a GAN know about the tree? We do not intend to ask about the full real-world idea of a tree, but rather whether the GAN models the visual concept of a tree as its own class of object distinct from buildings or roads, and if it knows that trees have their own particular appearance and propensity to appear in particular contexts in a scene. To investigate this question, in Chapter 4 we extend network dissection to GANs, testing the neurons of several Progressive GAN models [Karras et al., 2018]

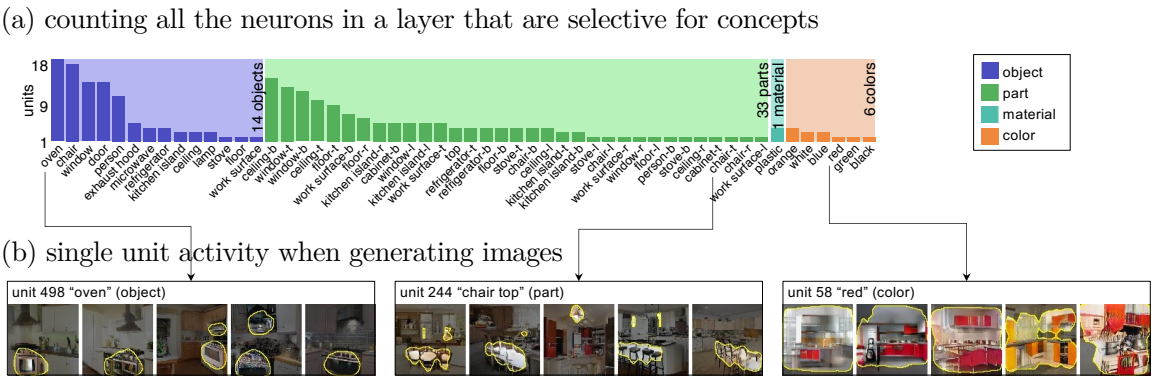


Figure 1-6: The emergence of object and part neurons in an unsupervised GAN trained to mimic kitchen scenes. Although the GAN has never been exposed to the purpose of an appliance or the use of a chair, single neurons emerge that are selective for those objects (b) as well as a variety of other objects and parts that appear in kitchens (a).

for agreement with a range of meaningful visual concepts. Because a GAN synthesizes its own images, we introduce use of a pretrained semantic segmentation network for identifying neurons with human-understandable semantics (Figure 1-5).

Our measurements of neuron selectivity within several GAN models reveal object-specific units in a GAN even in the absence of label supervision: a network trained on outdoor church scenes has neurons for object classes such as trees, doors, and domes. And a GAN trained on kitchen scenes has neurons for objects such as cabinets, ovens, and chairs (Figure 1-6).

These networks have never had the experience of seeing the branches of a tree sway in the wind, and they have no reason to know that an oven has a real-world purpose as a distinct appliance. Yet the task of learning how to draw scenes containing those objects has allowed the network to learn how to segregate the representation of each object into a set of neurons. Learning to draw seems to induce an awareness of objects.

However, correlation is not causation, and without further evidence, we risk letting our imagination go too far. The role of the neuron would be clearer if we could ask the network to describe the effects of neurons on its own thoughts. Famously, Jerome Lettvin imagined finding a set of 'mother neurons'[§] whose purpose was clear because,

[§]Mother neurons have been promoted to 'grandmother neurons' in the popular imagination. In the original telling of Lettvin's humorous allegory, the discoverer of mother neurons moved on to future work on grandmother neurons due to better availability of research funding [Barwich, 2019].

(a) A scene synthesized by Progressive GAN.

(b) Removing 20 units selective for trees.

(c) Location to activate neurons

(d) Activating 20 neurons selective for doors

Figure 1-7: Conducting Lettvin's 'mother neuron' experiment on a Progressive GAN trained on outdoor church scenes. Removing 20 tree-specific units causes a scene with trees (a) to have far fewer trees while not reducing buildings (b). Activating 20 door-specific neurons cases a scene without a door (c) to have a door (d).

when a fabled human subject was asked to discuss his family after the neurons' removal from his brain, he was utterly unable to describe his own mother, even while still talking about the red dress she wore [Barlow, 2009].

In an ideal world, we could test the causal effect of any neuron by asking the network what it sees when we stimulate or remove the neuron (this has occasionally been done with humans [Parvizi et al., 2012, Schalk et al., 2017]). While neither a classifier nor a generator can talk about its own perception, a generator does present a wonderful opportunity to conduct the experiment, because it is trained to effectively *draw what it thinks*. To see what a GAN is thinking, we simply let it generate its output image while we stimulate any set of neurons we wish.

Thus we can conduct Lettvin's 'mother neuron' test almost exactly, by generating images while removing one or more neurons that are selective for a single concept, or conversely activating them. Figure 1-7 shows the result: the more tree-neurons we remove, the fewer trees it draws. And activating a small set neurons for a concept such as doors causes network to depict a new door where it previously did not exist.

By examining the details within images produced in neuron intervention experiments, we can see two more ways in which the effects of neurons are fascinating. First, we can see that the neurons' causal impacts are highly specific. For example, in Figure 1-7(b), observe that while trees are removed, other details in the scene such as buildings are not reduced. In particular, parts of the building that used to be obscured

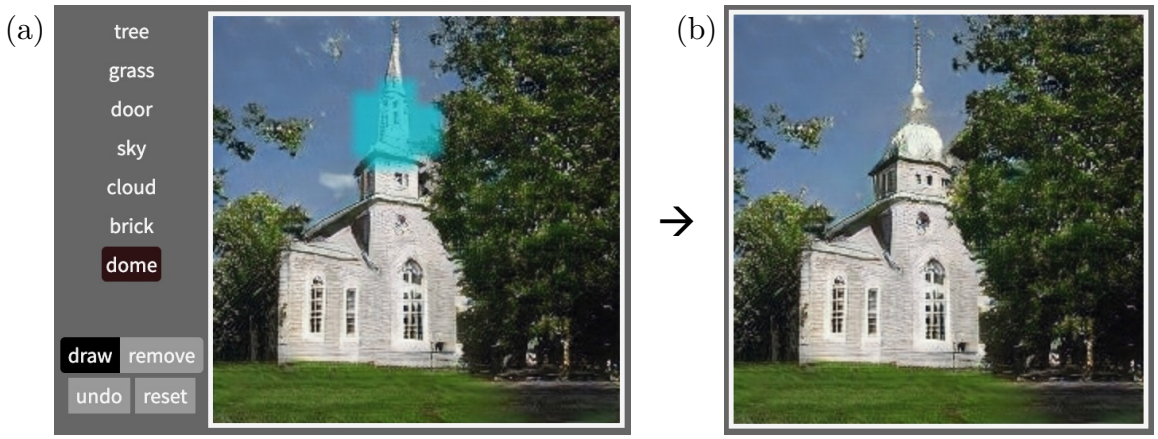


Figure 1-8: GANPaint: a user interface for painting with neurons. (a) The user can select a concept paintbrush, such as 'dome' then scribble on a region of the image. (b) 20 concept-specific neurons are activated in the scribbled area, and the GAN will add the requested object in a way that fits into the existing scene. For example, a dome shape smoothly added into the existing building. Interactive demo at `gandissect.csail.mit.edu/ganpaint`.

by the trees are now visible![¶] That new complex building shapes become visible when trees are suppressed strongly suggests that trees and buildings are processed as separate objects by the model. The second interesting phenomenon is discussed, measured and illustrated in Section 4.4.4: if door neurons are activated in the sky or the grass or a location that would not make sense for a door, the model will not add a door. The model contains a computational rule that prevents doors from being drawn in places where a door would not make sense. In both these cases, it is apparent that the GAN encodes rules governing *relationships* between concepts.

These effects lead to visually satisfying results, and the method enables us to create an application, GANPaint, that presents the user with a 'semantic paintbrush' that activates or deactivates sets of neurons specific to a visual concept that they can select. The user can paint trees or doors or other objects in a synthesized scene. With each brushstroke, small sets of concept-specific neurons are activated, and as a result the GAN alters the scene in a realistic way (Figure 1-8).

Our findings from interventions of the neurons of a GAN lead us to two further

[¶]When trees vanish while buildings remain, it is reminiscent of Lettvin's idea of the mother's red dress remaining. It is especially surprising here because our GAN has just been trained on the task of matching visible pixels; yet it has apparently learned to model parts of objects that are *not* visible, such as the parts of the buildings that had been obscured by trees.

questions concerning the limitations and rules encoded within a network:

1. How can we quantify and visualize concepts that a GAN does *not* draw?

2. Is it possible to understand how *rules* governing relationships between concepts are represented within a network?

Investigating these two questions are the subjects of Chapter 5 and Chapter 6.

1.3 Seeing what a GAN cannot generate

If the neurons of a network tell us what visual concepts a network contains, then how can we discover what visual concepts a network *fails* to contain? Understanding the *omissions* of a network requires a new kind of test, because it is difficult to explain an omission by giving just one example of an image.

In Chapter 5, we devise two types of pairwise comparisons to understand the omission of a GAN: first, we compare *distributions* of images, contrasting the distribution of outputs of a GAN with the GAN's training distribution (Figure 1-9(a)). Second, we compare *instances* of images, pairing each real training images with a GAN-generated image that is optimized to be as similar as possible to the real image (Figure 1-9(b,c)).

Surprisingly, we find that the distributions concentrate many of their differences on a few object classes. For example, the training data for a GAN model of church buildings contains many people in the images, but the output of the GAN model contains very few objects that resemble people. The same systematic omissions can be observed on other classes of complex objects, such as vehicles and text, and quantified as shown in Figure 1-9(a).

To visualize instances of the omissions, we develop a network inversion procedure that can accurately invert layers of a generator network G by calculating the noise that would produce a generated image x, that is, identifying the z for which $G(z) = x$, if there is one (see Figure 5-6). By applying this inversion on training images that cannot be synthesized by the GAN, we can create pairs of images that visualize specific cases in which the GAN omits identified object classes, as shown in Figure 1-9(c).

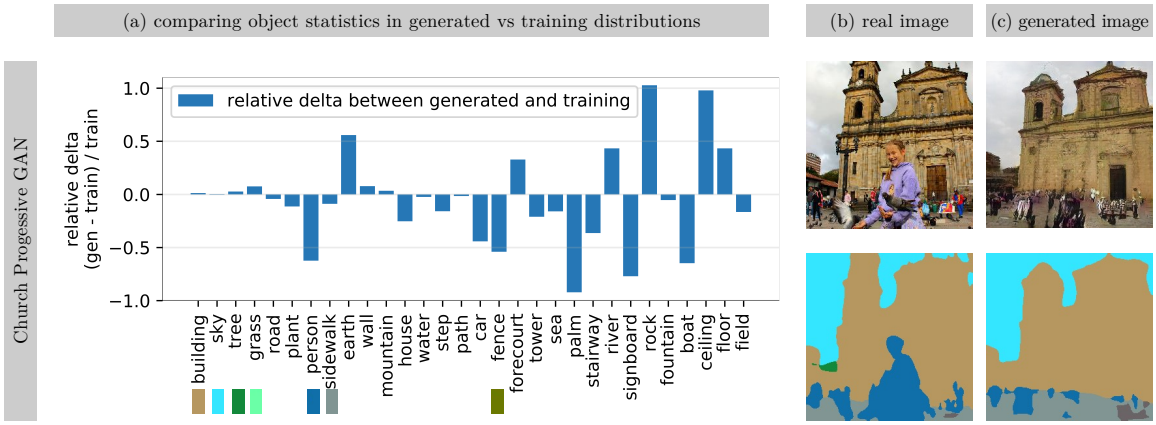


Figure 1-9: Two ways to understand GAN omissions. (a) By comparing statistics of human-understandable visual concepts in generated images to those of training images, we can see that the GAN concentrates omissions in a few visually complicated classes such as people, vehicles, and signs with text. (b) An example of a typical training set image, containing a person. (c) By inverting the GAN and drawing the image reconstruction, we can directly see an example of a GAN omitting people from a generated image; this matches the sharp dropout of the person object class seen in the statistics.

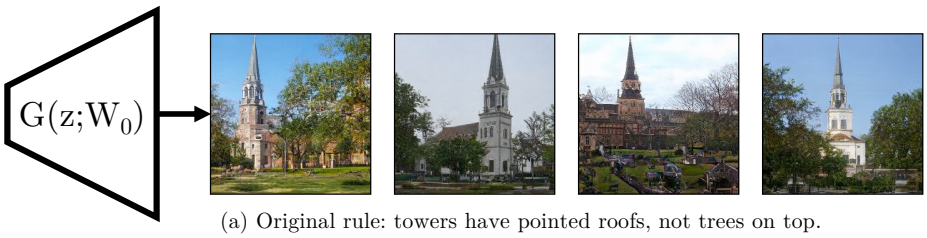
1.4 Rewriting rules in a generative model

While the *data* processsed by a network are represented by the *activations* of its neurons, the *rules* governing the data processing must be represented by the *weights* of the connections between neurons. Thus understanding the rules of a network requires us to understand its weights.

In Chapter 6, we propose a new problem setting that is equivalent to understanding how the rules of a model are encoded in the weights. We ask, for any generalized rule within a model, is there a minimal change in the weights that will cause the model to change that one rule without changing other unrelated behavior in the network?

For example, a network could contain a rule that specifies "Towers have pointy roofs, not leafy branches." (Figure 1-10) We ask, which weights need to change in order to change that rule? For example, can we change the rule so that tops of buildings are instead required to grow trees and not peaked roofs? We are not interested in just changing how one image is computed: we wish to change the rule in general, so that *all* similar buildings sprout trees instead of rooftops.

To solve this problem, we hypothesize that the model memorizes rules in its layers



(a) Original rule: towers have pointed roofs, not trees on top.

(b) Modified rule: towers have trees on top, not pointed roofs.

Figure 1-10: The goal of rewriting a generative model. (a) A pretrained model generates images that follow systematic rules learned from the data, for example, towers are topped by pointy rooftops. (b) The goal is to make a minimal change in the weights of a model to change one systematic rule without interfering with other aspects of the model.

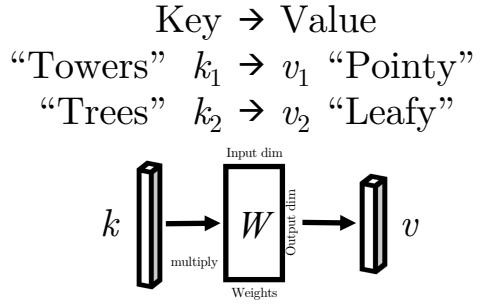


Figure 1-11: A single-layer associative memory model for the purpose of a layer in a deep network. If a network memorizes rules by storing key-value associations in a layer, then we should be able to change a rule by rewriting one such memorized association.

by treating each layer as a lookup table that maps neurons for a meaningful input context, such as 'top of a tower,' to neurons for a meaningful output command such as 'draw a pointy rooftoop' (Figure 1-11). To understand how such a table would be organized, we adopt a simple mathematical model of single-layer neural networks.[‖] An analysis of this model reveals that a single rule should correspond to a *rank-one subspace* of the weights of a layer.

To test this hypothesis, we build a user interface that would enables a person to edit a rank-one subspace of weights a GAN by highlighting examples of a rule to

[‖]The optimal linear associative model was first proposed by Kohonen [1972] and Anderson [1972], where they used it to reason about the capabilities and organization of a single-layer neural network.

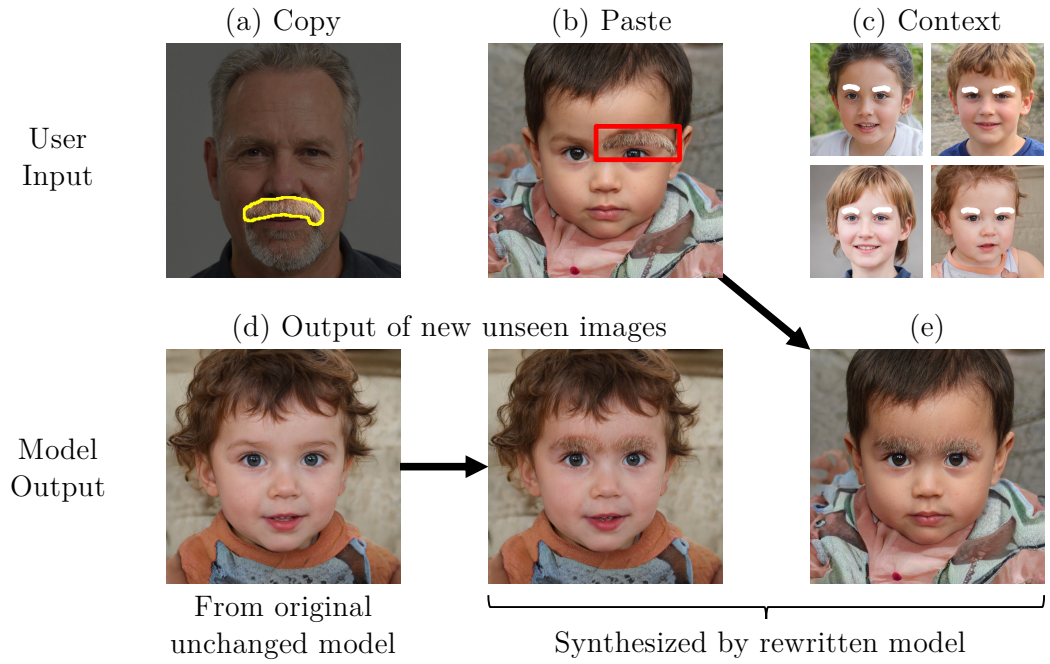


Figure 1-12: Human interaction for rewriting a generative model. A person selects a behavior such as (a) a furry mustache, and then selects an example of a rule that should be changed to use the new behavior, such as (b) the eyebrow of a child. Then to demonstrate the desired generalization the user selects (c) a few additional context examples. After rewriting the rule, (d) new unseen examples also obey the new rule, and the rule has generalized according to the examples, for example (e) altering both eyebrows instead of just one. Yet the rule change is also specific, minimizing changes outside the eyebrow change that was desired. Interactive demos at `rewriting.csail.mit.edu`.

change. And we demonstrate and measure efficacy at changing some individual rules, for example altering the appearance of eyebrows on children, or causing trees to sprout out of towers (Figure 1-12).

Although our editing method can only change some specific kinds of rules encoded by a large model, the results are interesting because they demonstrate that, by cracking open a model and understanding and manipulating its internal structure, it is feasible to directly create a model that exhibits behavior that a person designed, that does not need to be trained to mimic any new data set.

In other words, by understanding the internal language of a deep network, we have found that we can enable a person to teach a network to do something new that merges capabilities that the network has learned through machine learning, with new rules that come not from data, but from the imagination of the human user.

1.5 Summary

We start this journey by asking: Do deep neural networks contain concepts?

Following this inquiry within state-of-the-art models in computer vision leads us to insights about the computational structure of those deep networks that enable several new applications, including GANPaint semantic manipulation of objects in an image; visualization of objects that are missing from a generative model; and quick, selective editing of generalizable rules within a fully trained StyleGAN network.

The results from our investigations demonstrate that, although we train deep neural networks as black boxes, we are not compelled to use them that way. They contain computational structure that can be decomposed, understood, and manipulated.

The simple methods we develop in this dissertation presage a scientific approach to machine learning in which we will not be content to allow a model to learn freely from data, but where machine learning becomes just one step in a larger model-building process. They anticipate a future in which it will become routine for model internals to be analyzed, understood, manipulated, decomposed, and recombined to create new systems that solve human needs that cannot be expressed by imitation of data alone.

Bibliography

James A Anderson. A simple neural network generating an interactive memory. *Mathematical biosciences*, 14(3-4):197–220, 1972.

Bruno B Averbeck, Peter E Latham, and Alexandre Pouget. Neural correlations, population coding and computation. *Nature reviews neuroscience*, 7(5):358–366, 2006.

H Barlow. Grandmother cells, symmetry, and invariance: how the term arose and what the facts suggest. *The cognitive neurosciences*, pages 309–320, 2009.

Horace B Barlow. Single units and sensation: a neuron doctrine for perceptual psychology? *Perception*, 1(4):371–394, 1972.

Ann-Sophie Barwich. The value of failure in science: the story of grandmother cells in neuroscience. *Frontiers in neuroscience*, 13:1121, 2019.

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *PNAS*, 117(48):30071–30078, 2020.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

James V Haxby, M Ida Gobbini, Maura L Furey, Alumit Ishai, Jennifer L Schouten, and Pietro Pietrini. Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science*, 293(5539):2425–2430, 2001.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. In *ICLR*, 2016.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020.

Teuvo Kohonen. Correlation matrix memories. *IEEE transactions on computers*, 100 (4):353–359, 1972.

Jerome Y Lettvin, Humberto R Maturana, Warren S McCulloch, and Walter H Pitts. What the frog's eye tells the frog's brain. *Proceedings of the IRE*, 47(11):1940–1951, 1959.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *ICLR*, 2018.

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. 2015.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436, 2015.

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *NeurIPS*, 2016.

Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *CVPR*, pages 4467–4477, 2017.

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

Josef Parvizi, Corentin Jacques, Brett L Foster, Nathan Withoft, Vinitha Rangarajan, Kevin S Weiner, and Kalanit Grill-Spector. Electrical stimulation of human fusiform face-selective regions distorts face perception. *Journal of Neuroscience*, 32(43): 14915–14920, 2012.

David C Plaut and James L McClelland. Locating object knowledge in the brain: Comment on bowers's (2009) attempt to revive the grandmother cell hypothesis. *Psychological Review*, 117(1):284–288, 2010.

Rodrigo Quian Quiroga. Concept cells: the building blocks of declarative memory functions. *Nature Reviews Neuroscience*, 13(8):587, 2012.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

Gerwin Schalk, Christoph Kapeller, Christoph Guger, Hiroshi Ogawa, Satoru Hiroshima, Rosa Lafer-Sousa, Zeynep M Saygin, Kyousuke Kamada, and Nancy Kanwisher. Facephenes and rainbows: Causal evidence for functional and anatomical specificity of face and color processing in the human brain. *Proceedings of the National Academy of Sciences*, 114(46):12285–12290, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. In *ICLR*, 2015.

Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Revisiting the importance of individual units in cnns via ablation. *arXiv preprint arXiv:1806.02891*, 2018.

Chapter 2

Literature Review

At the heart of both the success and the enigma of modern computer vision is our field's reliance on machine learning to automatically learn programs from data. As long as we can calculate a robust numerical objective that measures how well a program solves a task on a set of examples, computer vision practitioners have found that we can perform a task without fully understanding how it is done. We train deep convolutional neural networks [LeCun et al., 1995] by optimizing our objectives on large data sets, and the learned weights take care of the details. The approach has proven so effective that machine learning with convolutional networks has become the central tool for solving almost every problem in computer vision including image classification [Krizhevsky et al., 2012, Szegedy et al., 2015, He et al., 2016, Tan and Le, 2019], object detection [Girshick, 2015, Ren et al., 2015, Redmon et al., 2016], scene segmentation [Long et al., 2015, Badrinarayanan et al., 2017, Zhou et al., 2017a, Chen et al., 2018b, Kirillov et al., 2019], captioning [Vinyals et al., 2015, Xu et al., 2015, Rennie et al., 2017, Yao et al., 2018, Lu et al., 2019], 3D perception [Eigen et al., 2014, Godard et al., 2017, Zhou et al., 2017b, Mahjourian et al., 2018] and image synthesis [Goodfellow et al., 2014, Radford et al., 2016, Arjovsky et al., 2017, Isola et al., 2017, Zhu et al., 2017, van den Oord et al., 2017, Brock et al., 2019, Karras et al., 2018, 2019, 2020].

The success of machine learning across computer vision poses a new problem for computer vision scientists, because now, constructing a working program no longer

suffices as proof that we understand the solution. After creating a network that performs well on a prescribed task, we can still find ourselves utterly unable to explain its limitations, such as why the system will be fooled by tiny changes in the input [Szegedy et al., 2014, Papernot et al., 2016, Madry et al., 2018]. Even more astoundingly, we can be just as unable to anticipate the capabilities of our network, such as how it can represent solutions to problems that seem more specific and complex than the original training task [Yosinski et al., 2014, Chen et al., 2020, He et al., 2020, Grill et al., 2020].

Thus if we wish to fully understand a computer vision task, we are now faced with a second puzzle after making a network work: how to explain, understand, and control the computations that our trained network has learned.

While the network dissection approach described in this thesis is one way to understand a neural network, it is one among many of approaches that have been developed, which we survey here. We will examine both recent methods for understanding artificial neural networks in computer vision, and classical experiments investigating the structure of biological vision networks.

2.1 Other ways to understand a deep network

As computational forms, neural networks are powerful because they are a family of *universal function approximators* [Hornik et al., 1989], but this flexibility also makes deep networks difficult to understand. For example, a VGG16 classifier [Simonyan and Zisserman, 2015] is a nonlinear function computed by performing 19.6 billion floating point operations using 138 million learned parameters.

2.1.1 Surrogate models and explanation models

One response to the complexity is to model the network with a simpler calculation. For example, in the neighborhood of a single image, the local behavior of a large network like VGG16 can be approximated by a simple linear model [Ribeiro et al., 2016] which is easier to analyze. Nonlinear surrogate models can also provide insight: a network can

be approximated as sets of examples in a nearest-neighbor model [Caruana et al., 1999, Kim et al., 2014]. Or its behavior can be modeled by a decision tree [Féraud and Clérot, 2002, Frosst and Hinton, 2017, Zhang et al., 2019] or a finite state machine [Koul et al., 2019]. Once a surrogate model is created, instead of explaining the original complex neural network, we can explain behavior by examining the activity of the surrogate.

Another modeling approach seeks out explanations by recruiting the help of a powerful explanation network. Here a second model is again used that is designed to be more understandable to humans than the original, but rather than choosing a simpler model, one uses a larger powerful model that is trained to generate human-readable explanatory text [Hendricks et al., 2016], or images that show where the relevant evidence is [Park et al., 2018]. Such explanation networks can be trained on a data set of human-created explanations so that the justifications are similar to the explanations that a human would give.

A supplementary model can also be used to understand information within the network by decoding its representations. A common approach is to decode specific variables from a layer by training a linear model [Alain and Bengio, 2017, Kim et al., 2017, Belinkov et al., 2017]. A decoder can also be trained to reconstruct an input image that yields the same representation [Dosovitskiy and Brox, 2016, Mahendran and Vedaldi, 2015, Vondrick et al., 2013, Weinzaepfel et al., 2011], to visualize model perception. A decoding model can also be trained to identify invariances and equivariances in a layer [Lenc and Vedaldi, 2015].

Training a second model to provide insights about the behavior of an opaque neural network can uncover insights such as which input variables seem to be the most important for a particular decision [Ribeiro et al., 2016], or whether the model contains sensitive latent information such as legally protected class membership [Kim et al., 2017]. However, one disadvantage of introducing a second model is that the limitations, biases, and structure of the second model may not be identical to those of the original network. For example, a variable that plays a causal role in a surrogate model may not play a causal role in the original network's computations [Goyal et al., 2019]. Explanations of a second model may also miss structural insights about the

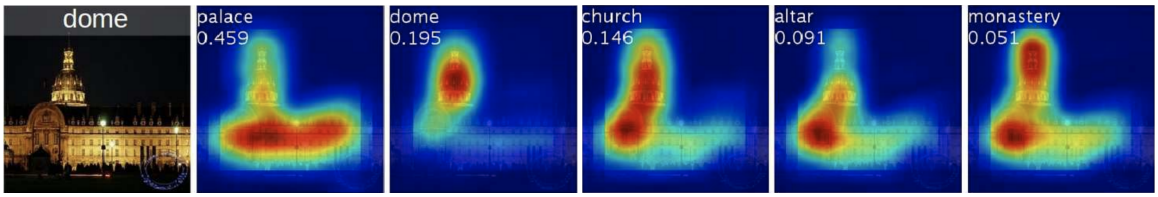


Figure 2-1: Example of a CAM heatmap, from Zhou et al. [2015], reproduced with permission of the author. By visualizing gradients of predictions of different classes weighted by feature activations, the CAM heatmap visualization shows the model attending to different parts of the image for different classes. Such single-image visualizations hint at the underlying neuron selectivity, which our methods aim to quantify directly.

original computations. If the original network has some simple underlying organization, the chance to understand that structure may be lost when translating to another kind of model that operates very differently.

In this thesis, instead of creating surrogate models or explanation models, we will focus on understanding the computations of the primary network directly.

2.1.2 Salience methods

One fundamental way to understand a network directly is to ask how its outputs vary as its inputs are changed. For example, Zeiler and Fergus [2014] proposed testing the output changes of an image classifier in response to erasing a small patch of the input image. By scanning the erased patch across the image and identifying locations which cause the network to change its prediction most, it is possible to identify individual parts of the input that are most salient to the computed output, even if we do not understand the detailed computations that lead to that specific sensitivity.

Zeiler's simple patch deletion procedure does not test every possible perturbation of the input, and a number of other saliency mapping methods have been developed for identifying other potentially important input sensitivities. One approach is to estimate input pixel sensitivity by directly computing gradients through the network [Baehrens et al., 2010, Erhan et al., 2009, Simonyan et al., 2014]. The simple gradient approach can produce noisy visualizations, but it has been found that more understandable visualizations can be derived from other gradient-based quantities such as the gradient of positive terms [Bach et al., 2015], the gradient integrated over a path [Sundararajan

et al., 2017], or the gradient averaged over perturbations of the input [Smilkov et al., 2017]. It can be particularly effective to create class activation maps which visualize feature gradients weighted according to feature activation strengths [Zhou et al., 2014, Selvaraju et al., 2017] (Figure 2-1). A different approach is to improve Zeiler's masking method by probing deletions on many parts of the input simultaneously using random masks [Petsiuk et al., 2018], or by devising a loss for explanations and optimizing masks according to that loss [Fong and Vedaldi, 2017]. Or one can appeal to game theory and score the Shapley value of each pixel [Roth, 1988, Lundberg and Lee, 2017, Sundararajan and Najmi, 2020, Chen et al., 2018a], which is its marginal contribution to the output when considered over all combinations of other pixels.

The multiplicity of reasonable salience methods leads to the question, how can we know when a salience method reveals something meaningful? One can benchmark salience methods by asking if humans are able to distinguish stronger or weaker models by looking at visualizations [Selvaraju et al., 2017], or by conducting lower-level evaluations, for example, by scoring a visualization against a "pointing game," that measures how well salient regions match human-labeled segmentations of relevant objects in a scene [Zhang et al., 2018]. Or one can score a visualization method against a "deletion game," measuring the impact on deleting masked regions on the output of the network [Samek et al., 2016, Fong and Vedaldi, 2017] or on the the ability of the network to learn from images with deletions [Hooker et al., 2018]. It has been observed that a good visualization should change when network parameters are scrambled to destroy performance [Adebayo et al., 2018], since a useful visualization should help distinguish strong models from weak ones.

Salience methods have revealed that convolutional neural networks attend to interesting parts of an image that are often suggestive of either insights or mistakes. However, salience methods give us a picture of *where* a network is looking at without directly answering the question *why* the network is looking there.

To gain insight into "why," we ask: how do the internal computations of the network work? Since the computations are too complex to consider as a whole, is it natural to decompose a network into neurons, and study properties of specific units.

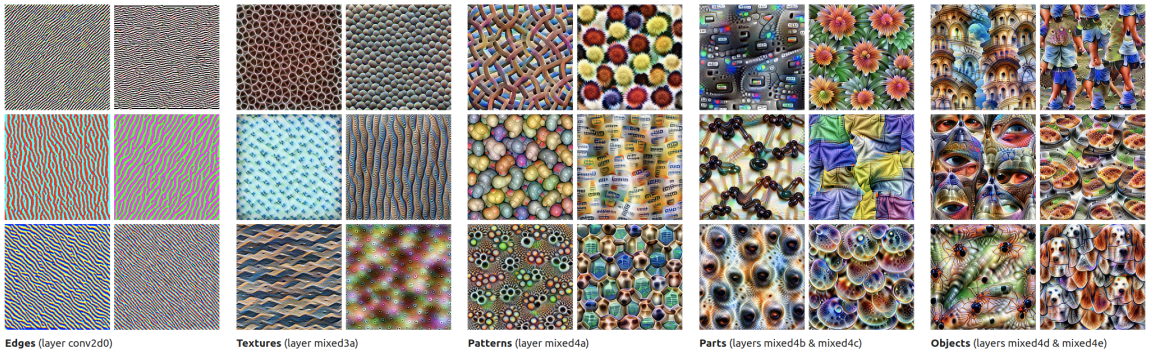


Figure 2-2: Synthesized images for feature visualization, from Olah et al. [2017], licensed under CC-BY 4.0. These visualizations are obtained by optimizing the pixels to maximize individual unit activations, while applying several regularization techniques such as jitter and whitening, to ensure that the images reveal patterns clearly. Such single-image visualizations are superb tools for understanding units that are selective for a single visual appearance, but do not provide full insight on the range of dissimilar images which a unit may detect.

2.1.3 Unit and latent vector methods

In computer vision, it has long been known that specialized neurons acting as oriented-edge Gabor filters can be learned at the early layers of a neural network [Daugman, 1988, Zeiler et al., 2010], resembling the response of edge detection neurons in biological vision [Hubel and Wiesel, 1962]. After the the successful use of deep convolutional networks on very large datasets, it was also noticed that individual units of large networks seem to show sensitivity to higher-level patterns and shapes that are much more complicated than oriented edges [Zeiler and Fergus, 2014]. This has inspired the development of several methods for visualizing and understanding individual units with a deep network. The work in this thesis is part of that tradition.

One natural approach for understanding a unit is to visualize images that cause it to activate. This can be done by simply identifying a few activation-maximizing samples from a data set of real images [Zeiler and Fergus, 2014]. In our work, when we visualize units we adopt this simple approach, though we identify regions that stimulate a neuron, rather than only whole images. Another approach is to synthesize an image that maximizes the response of the unit. This can be done by direct backpropagation [Erhan et al., 2009, Simonyan et al., 2014, Zeiler and Fergus, 2014], optimization using a generative prior [Nguyen et al., 2016, 2017], or other regularization such as jitter [Olah

et al., 2017, Mordvintsev et al., 2015] (Figure 2-2). Synthesized images can be used to visualize entire feature maps and combinations of units [Olah et al., 2018, 2020].

One limitation of unit-visualization synthesis methods is that a unit's response may not be well-represented by a single visual template: a unit may respond to sets of visually dissimilar images. Nguyen et al. [2017] has advocated synthesizing unit images with a method that generates many images, and recently, Goh et al. [2021] demonstrated a method for synthesizing multiple dissimilar images that is able to reveal neurons respond to both printed text as well as visual views of the same abstract concept within the CLIP [Radford et al., 2021] network. One of the goals of our network dissection approach is to address and measure such visually diverse detection. By characterizing units according to their behavior on a distribution of many images, our method allows semantically coherent responses to visually dissimilar appearances to be identified and quantified.

When visualizing and understanding generative model feature vectors, Radford et al. [2016] observed that a GAN latent space provides vector arithmetic for visual semantics, and this has sparked development of methods for understanding, exploring, and finding useful vector latent directions in GAN feature space. Latent feature vectors can be found using classifiers and other learning methods [Shen et al., 2020, Goetschalckx et al., 2019, Jahanian et al., 2020, Härkönen et al., 2020]. Our work differs from those methods because we focus on GAN units in interior layers, which allows us to analyze and manipulate how a GAN decomposes a scene into smaller objects and parts. Recently, exciting work analyzing individual interior-layer units in StyleGAN [Wu et al., 2021] has extended our methods and discovered remarkably disentangled individual units in the style modulation units of that network.

Our review of model explanation, unit visualization and latent vector methods is not exhaustive, and in subsequent chapters, further work is reviewed that is relevant to the experiments in each chapter. We also note that the study of individual units predates computer science; our work is motivated by the history of single-unit studies in neuroscience, which we briefly discuss now.

2.2 Inspiration from neuroscience

When studying neurons, we draw inspiration from a rich history of single-neuron experiments in neuroscience. But we must also point out that the question about the fundamental role of a neuron in biological brains is far from settled.

2.2.1 The distributed code model

On the role of neurons, neuroscience is divided into two schools of thought. The modern prevailing view is that large populations of neurons must work together to form a *distributed code*, and that elements of perception are represented in complex ways that cannot be distilled down to any single neuron. Advocates of this view point out that even when neurons or regions or neurons respond most strongly to one type of stimulus, they also contain information about many other types of stimuli [Haxby et al., 2001]. In this view, it can be misleading to say, for example, that a neuron detects a face on the basis of its strong selective activity for faces, because diffuse signals that appear only as barely detectable noise across individual neurons can also make a decisive contribution to perception when combinations of neurons are considered in the aggregate [Averbeck et al., 2006].* It is believed that generalization of perception to complex concepts such as specific instances of objects with many attributes, demands a flexible coding scheme [Plaut and McClelland, 2010].

If perception is mediated by such a distributed code, the question 'where in the brain is something perceived?' cannot be answered by any small group of neuron cells. Perception must be understood as an abstract state of the population of neurons as a whole, with no need for a special neuron to perceive any one concept.

One of the greatest successes of the distributed coding model is the design of artificial deep neural networks: when we train a network, we allow every neuron to read signals from every neuron at the previous layer, allowing for the representation

*The typical argument in favor of the distributed code view is a decoding argument, that many aspects of perception can be decoded from what might appear as mere noise in individual neural signals. Yet the presence such information does not directly prove that sensation of perception is caused by those signals. Such correlations are suggestive, but whether they cause the sensation of perception is not fully settled.

of a dense distributed code where a whole layer of neurons works together.

2.2.2 The neuron doctrine

The second school of thought is the *neuron doctrine* of perception. This view argues that no invisible large-scale population property is necessary to explain the sensation of perception: every element of human perception corresponds directly to the activity of a few neurons [Barlow, 1972].† This point of view was motivated by results from single-neuron experiments that revealed specific sensitivities, physical organization, and functional hierarchy of individual neurons in the nervous system and parts of the brain such as the visual cortex [Adrian and Matthews, 1927, Lettvin et al., 1959, Hubel and Wiesel, 1962, Gross et al., 1972]. These experiments revealed neurons with highly specific responses to inputs that are increasingly complex, from oriented edges to shapes of objects such as hands. The neuron doctrine extrapolated from such shape sensitivity to the hypothesis that there must be neurons elsewhere in the brain for all elements of perception.

Few neuroscientists would defend the simplistic notion that one neuron purely corresponds to one idea. However Jerome Lettvin's parable of an imaginary set of "Grandmother cells"‡ has captured the public imagination and sparked much research and debate about the nature of high-level perception [Barlow, 2009]. Despite the fact that "No one wants to be accused of believing in grandmother cells" [Connor, 2005], modern experimental results have continued to reveal an unexpected degree of both selectivity and invariance in small sets of neurons for classes of increasingly complex inputs, coming closer to Lettvin's allegorical grandmother cells [Bowers, 2009]. Causality has been demonstrated, for example Newsome et al. [1989] showed that small sets of perceptual cells in monkeys could be stimulated to cause the monkeys to react as if specific directional motions had been seen. Higher-level concepts have

†One way to understand the argument between the neuron-doctrine and population-code perspectives is as a difference opinion on how best to apply Occam's razor. Which is the more unnecessary concept, a physical decoding neuron, or an emergent sensation of perception?

‡Actually "mother cells" in the original telling, whose removal would cause you to be unable to perceive your own mother.

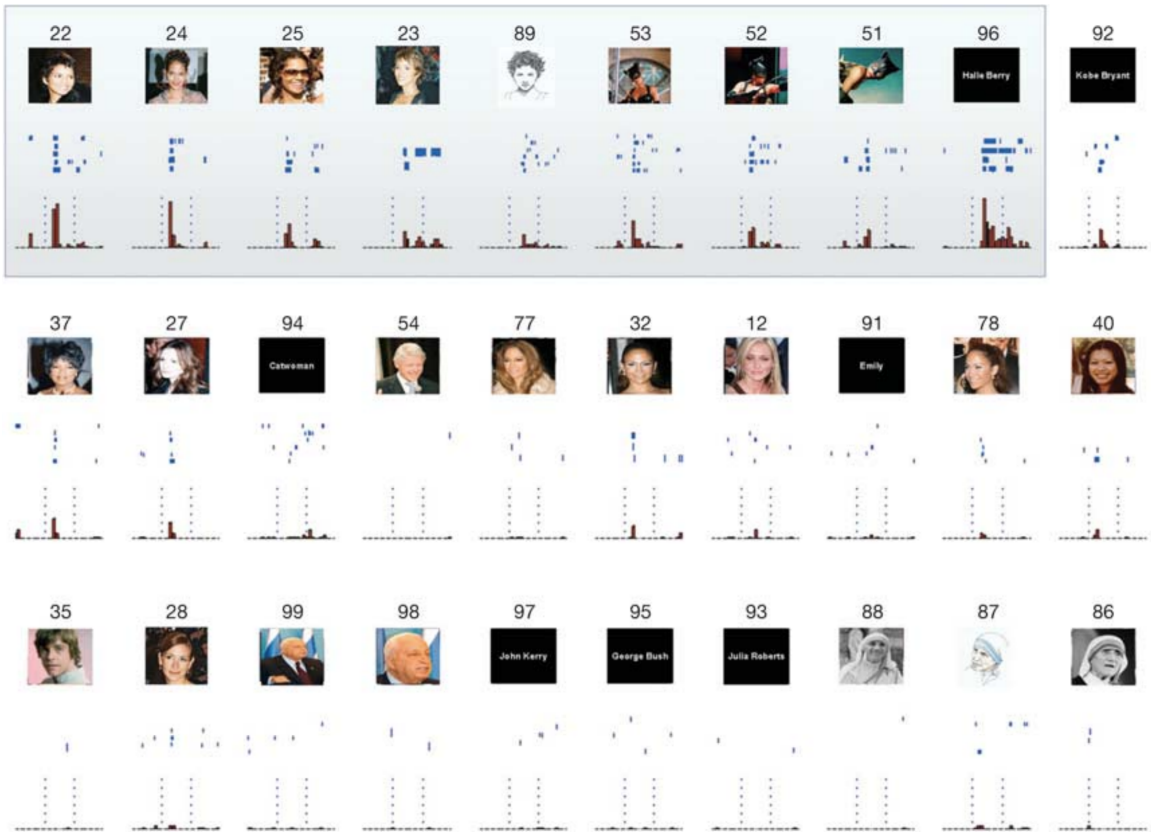


Figure 2-3: Selectivity and invariance of one neuron for Halle Berry, from Quiroga et al. [2005], reprinted with the permission of Macmillan Publishers. These single-neuron spike measurements from a human subject are shown together with representative visual stimuli. Notice that the single neuron selectively images of Halle Berry and not other people or scenes, but it also responds to the printed text 'Halle Berry', a drawing of the actress, as well as pictures of Catwoman, a role that had recently been played by the actress.

been found, for example, Kanwisher et al. [1997] localized a region of face-specific neurons in the FFA of the cortex; when this region is stimulated, a subject will report hallucinating faces on plain objects, and changing identities of seen faces [Schalk et al., 2017]. And Quiroga et al. [2005] has found individual neurons in the medial temporal lobe in which single-neuron activity was specific to an individual person, including one neuron was only sensitive to the actress Halle Berry, responding to a variety of photos of that actress as well as a drawing, the printed text 'Halle Berry', as well as depictions of the fictional character Catwoman that she had recently played (Figure 2-3).

2.2.3 Sparse coding and artificial neural networks

It is reasonable to ask whether the neuron doctrine should be fully discarded in the face of such strikingly specific neurons. There is some middle ground which may come closest to the full story. The distributed and neuron-doctrine views can be reconciled by the proposal that a neural network uses a *sparse code* in which a minimal number of neurons fire to represent any element of perception: sparse coding has been observed in some studies that have exhaustively examined small systems of neurons [Olshausen and Field, 1996, Hung et al., 2005, Honegger et al., 2011]. Even if individual neurons are not singularly responsible for perception, in a sparse coding scheme, the response of individual neurons will provide insight about the specificity, invariances, and symmetries of the code [Barlow, 2009].

With artificial neural networks we are now in the odd position of being able to perfectly calculate the operation of a network while still lacking an understanding of how the network works. Therefore, to start the process of understanding these systems, we will begin by retracing the steps of the first neuroscientists: we examine the response of individual neurons.

Bibliography

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. *NeurIPS*, 31, 2018.

Edgar Douglas Adrian and Rachel Matthews. The action of light on the eye: Part i. the discharge of impulses in the optic nerve and its relation to the electric changes in the retina. *The Journal of Physiology*, 63(4):378–414, 1927.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *ICLR Workshop*, 2017.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

Bruno B Averbeck, Peter E Latham, and Alexandre Pouget. Neural correlations, population coding and computation. *Nature reviews neuroscience*, 7(5):358–366, 2006.

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

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. 39(12):2481–2495, 2017.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *JMLR*, 11:1803–1831, 2010.

H Barlow. Grandmother cells, symmetry, and invariance: how the term arose and what the facts suggest. *The cognitive neurosciences*, pages 309–320, 2009.

Horace B Barlow. Single units and sensation: a neuron doctrine for perceptual psychology? *Perception*, 1(4):371–394, 1972.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. What do neural machine translation models learn about morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 861–872, 2017.

Jeffrey S Bowers. On the biological plausibility of grandmother cells: implications for neural network theories in psychology and neuroscience. *Psychological review*, 116 (1):220, 2009.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *ICLR*, 2019.

Rich Caruana, Hooshang Kangarloo, John David Dionisio, Usha Sinha, and David Johnson. Case-based explanation of non-case-based learning methods. In *Proceedings of the AMIA Symposium*, page 212. American Medical Informatics Association, 1999.

Jianbo Chen, Le Song, Martin J Wainwright, and Michael I Jordan. L-shapley and c-shapley: Efficient model interpretation for structured data. *arXiv preprint arXiv:1808.02610*, 2018a.

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, pages 801–818, 2018b.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*. PMLR, 2020.

Charles E Connor. Friends and grandmothers. *Nature*, 435(7045):1036–1037, 2005.

John G Daugman. Complete discrete 2-d gabor transforms by neural networks for image analysis and compression. *IEEE Transactions on acoustics, speech, and signal processing*, 36(7):1169–1179, 1988.

Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *CVPR*, 2016.

David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *Advances in Neural Information Processing Systems*, 27:2366–2374, 2014.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

Raphael Féraud and Fabrice Clérot. A methodology to explain neural network classification. *Neural networks*, 15(2):237–246, 2002.

Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *ICCV*, pages 3429–3437, 2017.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. 2071, 2017. URL http://ceur-ws.org/Vol-2071/CExAIIA_2017_paper_3.pdf.

Ross Girshick. Fast r-cnn. In *CVPR*, pages 1440–1448, 2015.

Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, pages 270–279, 2017.

Lore Goetschalckx, Alex Andonian, Aude Oliva, and Phillip Isola. Ganalyze: Toward visual definitions of cognitive image properties. In *ICCV*, 2019.

Gabriel Goh, Nick Cammarata, Chelsea Voss, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Yash Goyal, Amir Feder, Uri Shalit, and Been Kim. Explaining classifiers with causal concept effect (CACE). *arXiv preprint arXiv:1907.07165*, 2019.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Pires, Zhaohan Guo, Mohammad Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS*, 2020.

Charles G Gross, CE de Rocha-Miranda, and DB Bender. Visual properties of neurons in inferotemporal cortex of the macaque. *Journal of Neurophysiology*, 35(1):96–111, 1972.

Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. *arXiv preprint arXiv:2004.02546*, 2020.

James V Haxby, M Ida Gobbini, Maura L Furey, Alumit Ishai, Jennifer L Schouten, and Pietro Pietrini. Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science*, 293(5539):2425–2430, 2001.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations. In *ECCV*, 2016.

Kyle S Honegger, Robert AA Campbell, and Glenn C Turner. Cellular-resolution population imaging reveals robust sparse coding in the drosophila mushroom body. *Journal of neuroscience*, 31(33):11772–11785, 2011.

Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. *arXiv preprint arXiv:1806.10758*, 2018.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1): 106–154, 1962.

Chou P Hung, Gabriel Kreiman, Tomaso Poggio, and James J DiCarlo. Fast readout of object identity from macaque inferior temporal cortex. *Science*, 310(5749):863–866, 2005.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

Ali Jahanian, Lucy Chai, and Phillip Isola. On the "steerability" of generative adversarial networks. In *ICLR*, 2020.

Nancy Kanwisher, Josh McDermott, and Marvin M Chun. The fusiform face area: a module in human extrastriate cortex specialized for face perception. *Journal of neuroscience*, 17(11):4302–4311, 1997.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020.

Been Kim, Cynthia Rudin, and Julie A Shah. The bayesian case model: A generative approach for case-based reasoning and prototype classification. In *NeurIPS*, 2014.

Been Kim, Justin Gilmer, Fernanda Viegas, Ulfar Erlingsson, and Martin Wattenberg. Tcav: Relative concept importance testing with linear concept activation vectors. *arXiv preprint arXiv:1711.11279*, 2017.

Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *CVPR*, 2019.

Anurag Koul, Alan Fern, and Sam Greydanus. Learning finite state representations of recurrent policy networks. In *ICLR*, 2019. URL `https://openreview.net/forum?id=S1gOpsCctm`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *CVPR*, 2015.

Jerome Y Lettvin, Humberto R Maturana, Warren S McCulloch, and Walter H Pitts. What the frog's eye tells the frog's brain. *Proceedings of the IRE*, 47(11):1940–1951, 1959.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *NeurIPS*, pages 13–23, 2019.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NeurIPS*, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints. In *CVPR*, 2018.

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. 2015.

William T Newsome, Kenneth H Britten, and J Anthony Movshon. Neuronal correlates of a perceptual decision. *Nature*, 341(6237):52–54, 1989.

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *NeurIPS*, 2016.

Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *CVPR*, pages 4467–4477, 2017.

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

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

Dong Huk Park, Lisa Anne Hendricks, Zeynep Akata, Anna Rohrbach, Bernt Schiele, Trevor Darrell, and Marcus Rohrbach. Multimodal explanations: Justifying decisions and pointing to the evidence. In *CVPR*, pages 8779–8788, 2018.

Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. In *BMVC*, 2018.

David C Plaut and James L McClelland. Locating object knowledge in the brain: Comment on bowers's (2009) attempt to revive the grandmother cell hypothesis. *Psychological Review*, 117(1):284–288, 2010.

R Quian Quiroga, Leila Reddy, Gabriel Kreiman, Christof Koch, and Itzhak Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435 (7045):1102–1107, 2005.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*, pages 91–99, 2015.

Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *CVPR*, pages 7008–7024, 2017.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144. ACM, 2016.

Alvin E Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE transactions on neural networks and learning systems*, 28(11): 2660–2673, 2016.

Gerwin Schalk, Christoph Kapeller, Christoph Guger, Hiroshi Ogawa, Satoru Hiroshima, Rosa Lafer-Sousa, Zeynep M Saygin, Kyousuke Kamada, and Nancy Kanwisher. Facephenes and rainbows: Causal evidence for functional and anatomical specificity of face and color processing in the human brain. *Proceedings of the National Academy of Sciences*, 114(46):12285–12290, 2017.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.

Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *CVPR*, 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*, 2014.

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

Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation. In *ICML*, pages 9269–9278. PMLR, 2020.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *PMLR*, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, 2017.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *CVPR*, pages 3156–3164, 2015.

Carl Vondrick, Aditya Khosla, Tomasz Malisiewicz, and Antonio Torralba. Hoggles: Visualizing object detection features. In *CVPR*, 2013.

Philippe Weinzaepfel, Hervé Jégou, and Patrick Pérez. Reconstructing an image from its local descriptors. In *CVPR*, pages 337–344, 2011.

Zongze Wu, Dani Lischinski, and Eli Shechtman. Stylespace analysis: Disentangled controls for stylegan image generation. In *CVPR*, 2021.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pages 2048–2057. PMLR, 2015.

Ting Yao, Yingwei Pan, Yehao Li, and Tao Mei. Exploring visual relationship for image captioning. In *ECCV*, 2018.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *NeurIPS*, 2014.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pages 2528–2535. IEEE, 2010.

Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *IJCV*, 126(10): 1084–1102, 2018.

Quanshi Zhang, Yu Yang, Haotian Ma, and Ying Nian Wu. Interpreting cnns via decision trees. In *CVPR*, 2019.

Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NeurIPS*, 2014.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. In *ICLR*, 2015.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017a.

Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017b.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

Chapter 3

Network Dissection

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva,
Antonio Torralba. CVPR 2017.

3.1 Introduction

Observations of hidden units in large deep neural networks have revealed that human-interpretable concepts sometimes emerge as individual latent variables within those networks: for example, object detector units emerge within networks trained to recognize places [Zhou et al., 2015]; part detectors emerge in object classifiers [Gonzalez-Garcia et al., 2016]; and object detectors emerge in generative video networks [Vondrick et al., 2016] (Fig. 3-1). This internal structure has appeared in situations where the networks are not constrained to decompose problems in any interpretable way.

The emergence of interpretable structure suggests that deep networks may be

Figure 3-1: Examples of image regions that maximize units in several networks. Unit 13 in Zhou et al. [2015] (classifying places) detects table lamps. Unit 246 in Gonzalez-Garcia et al. [2016] (classifying objects) detects bicycle wheels. A unit in Vondrick et al. [2016] (self-supervised for generating videos) detects people.

learning disentangled representations spontaneously. While it is commonly understood that a network can learn an efficient encoding that makes economical use of hidden variables to distinguish its states, the appearance of a disentangled representation is not well-understood. A disentangled representation aligns its variables with a meaningful factorization of the underlying problem structure, and encouraging disentangled representations is a significant area of research [Bengio et al., 2013]. If the internal representation of a deep network is partly disentangled, one possible path for understanding its mechanisms is to detect disentangled structure, and simply read out the separated factors.

However, this proposal raises questions which we address in this chapter:

- What is a disentangled representation, and how can its factors be quantified and detected?
- Do interpretable hidden units reflect a special alignment of feature space, or are interpretations a chimera?
- What conditions in state-of-the-art training lead to representations with greater or lesser entanglement?

To examine these issues, we propose a general analytic framework, *network dissection*, for interpreting deep visual representations and quantifying their interpretability. Using Broden, a broadly and densely labeled data set, our framework identifies hidden units' semantics for any given CNN, then aligns them with human-interpretable concepts. We evaluate our method on various CNNs (AlexNet, VGG, GoogLeNet, ResNet) trained on object and scene recognition, and show that emergent interpretability is an axis-aligned property of a representation that can be destroyed by rotation without affecting discriminative power. We further examine how interpretability is affected by training data sets, training techniques like dropout [Srivastava et al., 2014] and batch normalization [Ioffe and Szegedy, 2015], and supervision by different primary tasks*.

*Source code and data available at `http://netdissect.csail.mit.edu`

3.1.1 Related work

A growing number of techniques have been developed to understand the internal representations of convolutional neural networks through visualization. The behavior of a CNN can be visualized by sampling image patches that maximize activation of hidden units [Zeiler and Fergus, 2014, Zhou et al., 2015], or by using variants of backpropagation to identify or generate salient image features [Mahendran and Vedaldi, 2015, Simonyan et al., 2014, Zeiler and Fergus, 2014]. The discriminative power of hidden layers of CNN features can also be understood by isolating portions of networks, transferring them or limiting them, and testing their capabilities on specialized problems [Yosinski et al., 2014, Razavian et al., 2014, Agrawal et al., 2014]. Visualizations digest the mechanisms of a network down to images which themselves must be interpreted; this motivates our work which aims to match representations of CNNs with labeled interpretations directly and automatically.

Most relevant to our current work are explorations of the roles of individual units inside neural networks. In Zhou et al. [2015] human evaluation was used to determine that individual units behave as object detectors in a network that was trained to classify scenes. Nguyen et al. [2016] automatically generated prototypical images for individual units by learning a feature inversion mapping; this contrasts with our approach of automatically assigning concept labels. Recently Alain and Bengio [2017] suggested an approach to testing the intermediate layers by training simple linear probes, which analyzes the information dynamics among layers and its effect on the final prediction.

3.2 Network Dissection

How can we quantify the clarity of an idea? The notion of a disentangled representation rests on the human perception of what it means for a concept to be mixed up. Therefore when we quantify interpretability, we define it in terms of alignment with a set of human-interpretable concepts. Our measurement of interpretability for deep visual representations proceeds in three steps:

1. Identify a broad set of human-labeled visual concepts.

2. Gather hidden variables' response to known concepts.

3. Quantify alignment of hidden variable−concept pairs.

This three-step process of *network dissection* is reminiscent of the procedures used by neuroscientists to understand similar representation questions in biological neurons Quiroga et al. [2005]. Since our purpose is to measure the level to which a representation is disentangled, we focus on quantifying the correspondence between a single latent variable and a visual concept.

In a fully interpretable local coding such as a one-hot-encoding, each variable will match exactly with one human-interpretable concept. Although we expect a network to learn partially nonlocal representations in interior layers [Bengio et al., 2013], and past experience shows that an emergent concept will often align with a combination of a several hidden units [Gonzalez-Garcia et al., 2016, Agrawal et al., 2014], our present aim is to assess how well a representation is disentangled. Therefore we measure the alignment between single units and single interpretable concepts. This does not gauge the discriminative power of the representation; rather it quantifies its disentangled interpretability. As we will show in Sec. 3.3.2, it is possible for two representations of perfectly equivalent discriminative power to have very different levels of interpretability.

To assess the interpretability of any given CNN, we draw concepts from a new broadly and densely labeled image data set that unifies labeled visual concepts from a heterogeneous collection of labeled data sources, described in Sec. 3.2.1. We then measure the alignment of each hidden unit of the CNN with each concept by evaluating the feature activation of each individual unit as a segmentation model for each concept. To quantify the interpretability of a layer as a whole, we count the number of distinct visual concepts that are aligned with a unit in the layer, as detailed in Sec. 3.2.2.

Figure 3-2: Samples from the **Broden** Dataset. The ground truth for each concept is a pixel-wise dense annotation.

Table 3.1: Statistics of each label type included in the data set.

Category	Classes	Sources	Avg sample
scene	468	ADE Zhou et al. [2017]	38
object	584	ADE Zhou et al. [2017], Pascal-Context Mottaghi et al. [2014]	491
part	234	ADE Zhou et al. [2017], Pascal-Part Chen et al. [2014]	854
material	32	OpenSurfaces Bell et al. [2014]	1,703
texture	47	DTD Cimpoi et al. [2014]	140
color	11	Generated	59,250

3.2.1 Broden: Broadly and Densely Labeled Dataset

To be able to ascertain alignment with both low-level concepts such as colors and higher-level concepts such as objects, we have assembled a new heterogeneous data set.

The Broadly and Densely Labeled Dataset (**Broden**) unifies several densely labeled image data sets: ADE [Zhou et al., 2017], OpenSurfaces [Bell et al., 2014], Pascal-Context [Mottaghi et al., 2014], Pascal-Part [Chen et al., 2014], and the Describable Textures Dataset [Cimpoi et al., 2014]. These data sets contain examples of a broad range of objects, scenes, object parts, textures, and materials in a variety of contexts. Most examples are segmented down to the pixel level except textures and scenes which are given for full-images. In addition, every image pixel in the data set is annotated with one of the eleven common color names according to the human perceptions classified by Van De Weijer et al. [2009]. A sample of the types of labels in the Broden dataset are shown in Fig. 3-2.

The purpose of Broden is to provide a ground truth set of exemplars for a broad

set of visual concepts. The concept labels in Broden are normalized and merged from their original data sets so that every class corresponds to an English word. Labels are merged based on shared synonyms, disregarding positional distinctions such as 'left' and 'top' and avoiding a blacklist of 29 overly general synonyms (such as 'machine' for 'car'). Multiple Broden labels can apply to the same pixel: for example, a black pixel that has the Pascal-Part label 'left front cat leg' has three labels in Broden: a unified 'cat' label representing cats across data sets; a similar unified 'leg' label; and the color label 'black'. Only labels with at least 10 image samples are included. Table 3.1 shows the average number of image samples per label class.

3.2.2 Scoring unit interpretability

The proposed network dissection method evaluates every individual convolutional unit in a CNN as a solution to a binary segmentation task to every visual concept in Broden (Fig. 3-3). Our method can be applied to any CNN using a forward pass without the need for training or backpropagation.

For every input image $\mathbf{x}$ in the Broden dataset, the activation map $A_k(\mathbf{x})$ of every internal convolutional unit k is collected. Then the distribution of individual unit activations a_k is computed. For each unit k, the top quantile level T_k is determined such that $P(a_k > T_k) = 0.005$ over every spatial location of the activation map in the data set.

To compare a low-resolution unit's activation map to the input-resolution annotation mask L_c for some concept c, the activation map is scaled up to the mask resolution $S_k(\mathbf{x})$ from $A_k(\mathbf{x})$ using bilinear interpolation, anchoring interpolants at the center of each unit's receptive field.

$S_k(\mathbf{x})$ is then thresholded into a binary segmentation: $M_k(\mathbf{x}) \equiv S_k(\mathbf{x}) \geq T_k$, selecting all regions for which the activation exceeds the threshold T_k. These segmentations are evaluated against every concept c in the data set by computing intersections $M_k(\mathbf{x}) \cap L_c(\mathbf{x})$, for every (k, c) pair.

The score of each unit k as segmentation for concept c is reported as a data-set-wide

intersection over union score

$$IoU_{k,c} = \frac{\sum |M_k(\mathbf{x}) \cap L_c(\mathbf{x})|}{\sum |M_k(\mathbf{x}) \cup L_c(\mathbf{x})|}, \tag{3.1}$$

where $|\cdot|$ is the cardinality of a set. Because the data set contains some types of labels which are not present on some subsets of inputs, the sums are computed only on the subset of images that have at least one labeled concept of the same category as c. The value of $IoU_{k,c}$ is the accuracy of unit k in detecting concept c; we consider one unit k as a detector for concept c if $IoU_{k,c}$ exceeds a threshold. Our qualitative results are insensitive to the IoU threshold: different thresholds denote different numbers of units as concept detectors across all the networks but relative orderings remain stable. For our comparisons we report a detector if $IoU_{k,c} > 0.04$. Note that one unit might be the detector for multiple concepts; for the purpose of our analysis, we choose the top ranked label. To quantify the interpretability of a layer, we count the number unique concepts aligned with units. We call this the number of *unique detectors*.

The IoU evaluating the quality of the segmentation of a unit is an objective confidence score for interpretability that is *comparable across networks*. Thus this score enables us to compare interpretability of different representations and lays the basis for the experiments below. Note that network dissection works only as well as the underlying data set: if a unit matches a human-understandable concept that is absent in Broden, then it will not score well for interpretability. Future versions of Broden will be expanded to include more kinds of visual concepts.

3.3 Experiments

For testing we prepare a collection of CNN models with different network architectures and supervision of primary tasks, as listed in Table 3.2. The network architectures include AlexNet [Krizhevsky et al., 2012], GoogLeNet [Szegedy et al., 2015], VGG [Simonyan and Zisserman, 2015], and ResNet [He et al., 2016]. For supervised training, the models are trained from scratch (i.e., not pretrained) on ImageNet Russakovsky

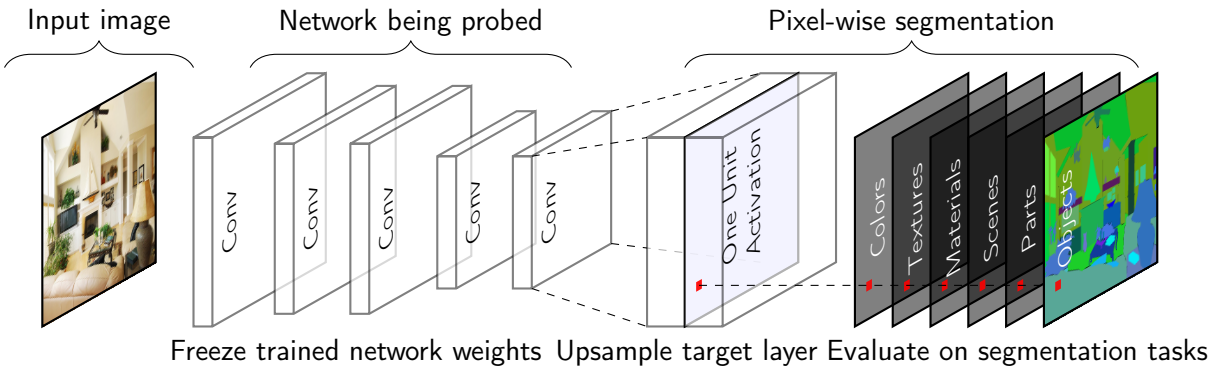


Figure 3-3: Illustration of network dissection for measuring semantic alignment of units in a given CNN. Here one unit of the last convolutional layer of a given CNN is probed by evaluating its performance on 1197 segmentation tasks. Our method can probe any convolutional layer.

Table 3.2: Tested CNN Models.

Training	**Network**	**Data set or task**
none	AlexNet	random
Supervised	AlexNet	ImageNet, Places205, Places365, Hybrid.
	GoogLeNet	ImageNet, Places205, Places365.
	VGG-16	ImageNet, Places205, Places365, Hybrid.
	ResNet-152	ImageNet, Places365.
Self	AlexNet	`context`, `puzzle`, `egomotion`, `tracking`, `moving`, `videoorder`, `audio`, `crosschannel`,`colorization`. `objectcentric`.

et al. [2015], Places205 [Zhou et al., 2014], and Places365 [Zhou et al., 2016]. ImageNet is an object-centric data set, which contains 1.2 million images from 1000 classes. Places205 and Places365 are two subsets of the Places Database, which is a scene-centric data set with categories such as kitchen, living room, and coast. Places205 contains 2.4 million images from 205 scene categories, while Places365 contains 1.6 million images from 365 scene categories. "Hybrid" refers to a combination of ImageNet and Places365. For self-supervised training tasks, we select several recent models trained on predicting context (`context`) Doersch et al. [2015], solving puzzles (`puzzle`) Noroozi and Favaro [2016], predicting ego-motion (`egomotion`) Jayaraman and Grauman [2015], learning by moving (`moving`) Agrawal et al. [2015], predicting video frame order (`videoorder`) Mikjjsra et al. [2016] or tracking (`tracking`) Wang and Gupta [2015], detecting object-centric alignment (`objectcentric`) Gao et al. [2016], colorizing images (`colorization`) Zhang et al. [2016], predicting cross-channel (`crosschannel`) Zhang et al. [2017], and predicting ambient sound from frames (`audio`) Owens et al. [2016]. The self-supervised models we analyze are comparable to each other in that they all use AlexNet or an AlexNet-derived architecture.

In the following experiments, we begin by validating our method using human evaluation. Then, we use random unitary rotations of a learned representation to test whether interpretability of CNNs is an axis-independent property; we find that it is not, and we conclude that interpretability is not an inevitable result of the discriminative power of a representation. Next, we analyze all the convolutional layers of AlexNet as trained on ImageNet [Krizhevsky et al., 2012] and as trained on Places [Zhou et al., 2014], and confirm that our method reveals detectors for higher-level concepts at higher layers and lower-level concepts at lower layers; and that more detectors for higher-level concepts emerge under scene training. Then, we show that different network architectures such as AlexNet, VGG, and ResNet yield different interpretability, while differently supervised training tasks and self-supervised training tasks also yield a variety of levels of interpretability. Finally we show the impact of different training conditions, examine the relationship between discriminative power and interpretability, and investigate a possible way to improve the interpretability of

Table 3.3: Human evaluation of our Network Dissection approach. Interpretable units are those where raters agreed with ground-truth interpretations. Within this set we report the portion of interpretations assigned by our method that were rated as descriptive. Human consistency is based on a second evaluation of ground-truth labels.

	conv1	conv2	conv3	conv4	conv5
Interpretable units	57/96	126/256	247/384	258/384	194/256
Human consistency	82%	76%	83%	82%	91%
Network Dissection	37%	56%	54%	59%	71%

CNNs by increasing their width.

3.3.1 Human evaluation of interpretations

We evaluate the quality of the unit interpretations found by our method using Amazon Mechanical Turk (AMT). Raters were shown 15 images with highlighted patches showing the most highly-activating regions for each unit in AlexNet trained on Places205, and asked to decide (yes/no) whether a given phrase describes most of the image patches.

Table 3.3 summarizes the results. First, we determined the set of interpretable units as those units for which raters agreed with ground-truth interpretations from Zhou et al. [2015]. Over this set of units, we report the portion of interpretations generated by our method that were rated as descriptive. Within this set we also compare to the portion of ground-truth labels that were found to be descriptive by a second group of raters. The proposed method can find semantic labels for units that are comparable to descriptions written by human annotators at the highest layer. At the lowest layer, the low-level color and texture concepts available in Broden are only sufficient to match good interpretations for a minority of units. Human consistency is also highest at `conv5`, which suggests that humans are better at recognizing and agreeing upon high-level visual concepts such as objects and parts, rather than the shapes and textures that emerge at lower layers.

3.3.2 Measurement of axis-aligned interpretability

We conduct an experiment to determine whether it is meaningful to assign an interpretable concept to an individual unit. Two possible hypotheses can explain the emergence of interpretability in individual hidden layer units:

Hypothesis 1. Interpretable units emerge because interpretable concepts appear in most directions in representation space. If the representation localizes related concepts in an axis-independent way, projecting to *any* direction could reveal an interpretable concept, and interpretations of single units in the natural basis may not be a meaningful way to understand a representation.

Hypothesis 2. Interpretable alignments are unusual, and interpretable units emerge because learning converges to a special basis that aligns explanatory factors with individual units. In this model, the natural basis represents a meaningful decomposition learned by the network.

Hypothesis 1 is the default assumption: in the past it has been found [Szegedy et al., 2014] that with respect to interpretability "there is no distinction between individual high level units and random linear combinations of high level units."

Network dissection allows us to re-evaluate this hypothesis. We apply random changes in basis to a representation learned by AlexNet. Under hypothesis 1, the overall level of interpretability should not be affected by a change in basis, even as rotations cause the specific set of represented concepts to change. Under hypothesis 2, the overall level of interpretability is expected to drop under a change in basis.

We begin with the representation of the 256 convolutional units of AlexNet `conv5` trained on Places205 and examine the effect of a change in basis. To avoid any issues of conditioning or degeneracy, we change basis using a random orthogonal transformation Q. The rotation Q is drawn uniformly from $SO(256)$ by applying Gram-Schmidt on a normally-distributed $QR = A \in \mathbf{R}^{256^2}$ with positive-diagonal right-triangular R, as described by Diaconis [2005]. Interpretability is summarized as the number of unique visual concepts aligned with units, as defined in Sec. 3.2.2.

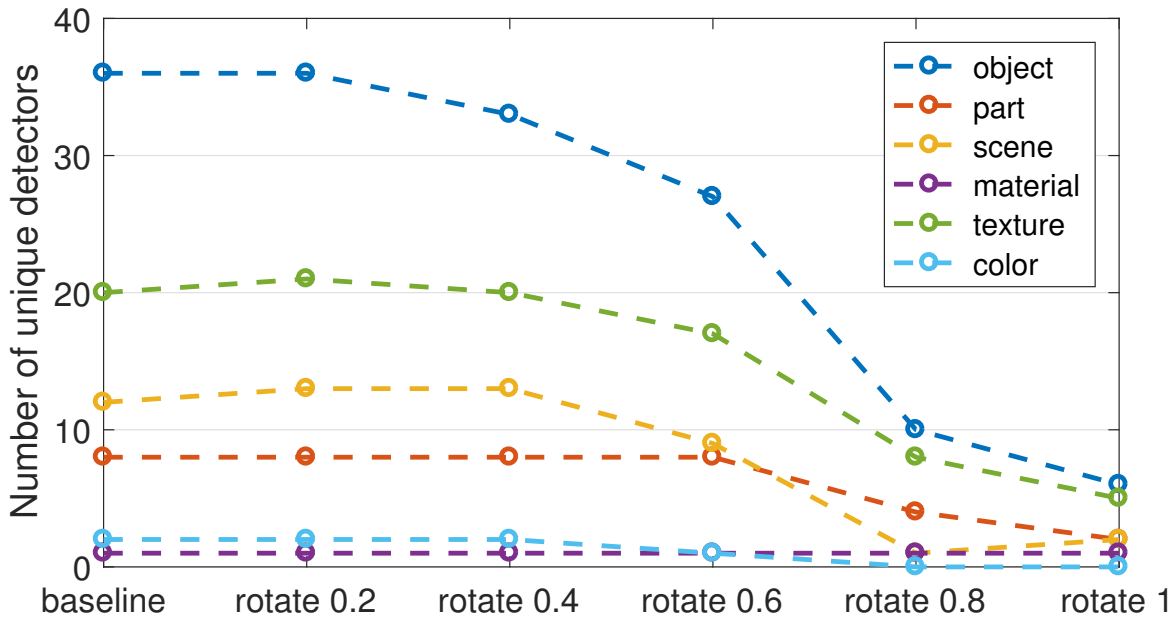


Figure 3-4: Interpretability over changes in basis of the representation of AlexNet `conv5` trained on Places. The vertical axis shows the number of unique interpretable concepts that match a unit in the representation. The horizontal axis shows α, which quantifies the degree of rotation.

Denoting AlexNet `conv5` as $f(x)$, we find that the number of unique detectors in $Qf(x)$ is 80% fewer than the number of unique detectors in $f(x)$. Our finding is inconsistent with hypothesis 1 and consistent with hypothesis 2.

We also test smaller perturbations of basis using Q^α for $0 \leq \alpha \leq 1$, where the fractional powers $Q^\alpha \in SO(256)$ are chosen to form a minimal geodesic gradually rotating from I to Q; these intermediate rotations are computed using a Schur decomposition. Fig. 3-4 shows that interpretability of $Q^\alpha f(x)$ decreases as larger rotations are applied.

Each rotated representation has exactly the same discriminative power as the original layer. Writing the original network as $g(f(x))$, note that $g'(r) \equiv g(Q^T r)$ defines a neural network that processes the rotated representation $r = Qf(x)$ exactly as the original g operates on $f(x)$. We conclude that interpretability is neither an inevitable result of discriminative power, nor is it a prerequisite to discriminative power. Instead, we find that interpretability is a different quality that must be measured separately to be understood.

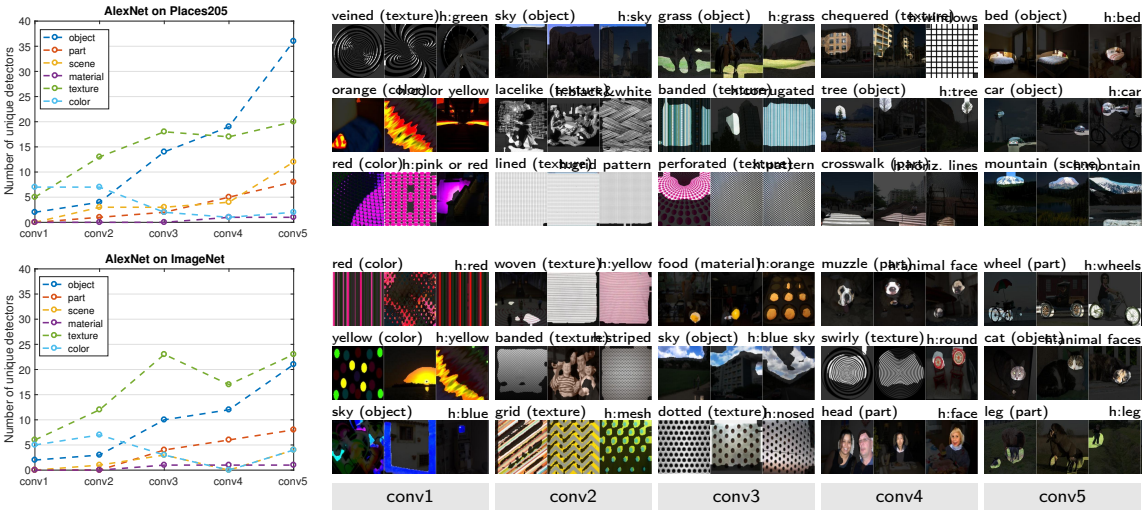


Figure 3-5: A comparison of the interpretability of all five convolutional layers of AlexNet, as trained on classification tasks for Places (top) and ImageNet (bottom). At right, three examples of units in each layer are shown with identified semantics. The segmentation generated by each unit is shown on the three Broden images with highest activation. Top-scoring labels are shown above to the left, and human-annotated labels are shown above to the right. Some disagreement can be seen for the dominant judgment of meaning. For example, human annotators mark the first `conv4` unit on Places as a 'windows' detector, while the algorithm matches the 'chequered' texture.

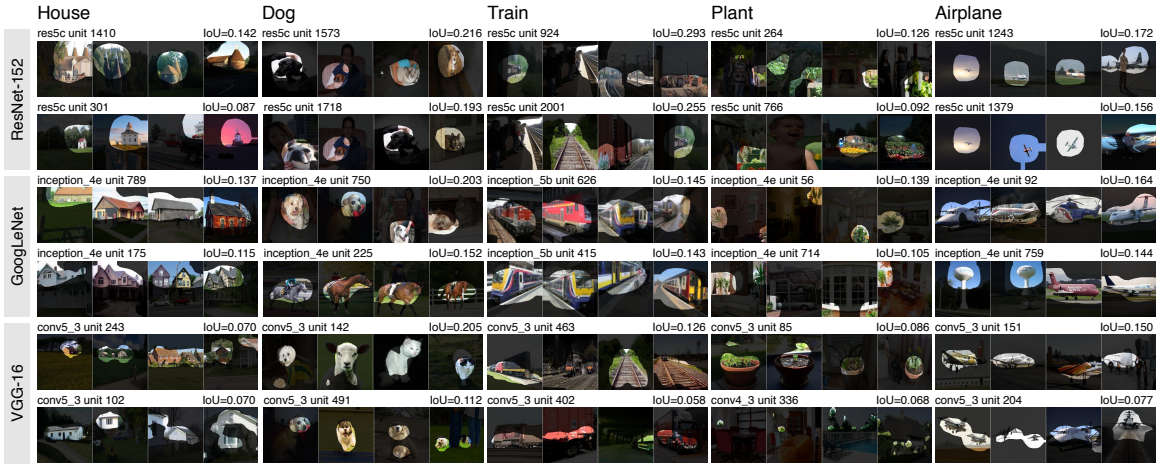


Figure 3-6: A comparison of several visual concept detectors identified by network dissection in ResNet, GoogLeNet, and VGG. Each network is trained on Places365. The two highest-IoU matches among convolutional units of each network is shown. The segmentation generated by each unit is shown on the four maximally activating Broden images. Some units activate on concept generalizations, e.g., GoogLeNet 4e's unit 225 on horses and dogs, and 759 on white ellipsoids and jets.

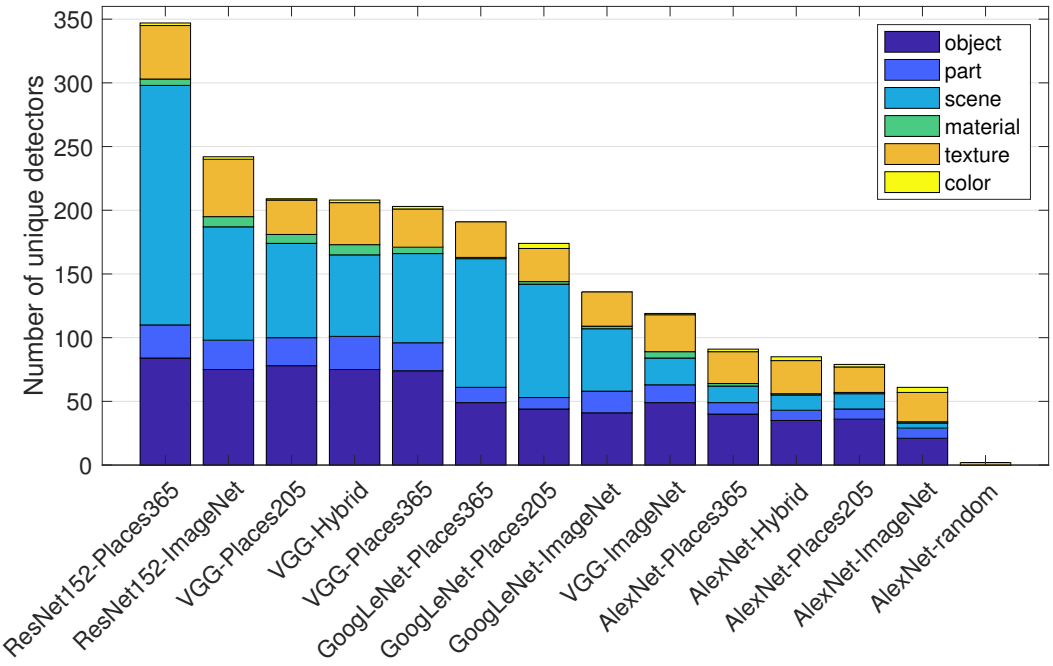


Figure 3-7: Interpretability across different architectures and training.

3.3.3 Disentangled concepts by layer

Using network dissection, we analyze and compare the interpretability of units within all the convolutional layers of Places-AlexNet and ImageNet-AlexNet. Places-AlexNet is trained for scene classification on Places205 [Zhou et al., 2014], while ImageNet-AlexNet is the identical architecture trained for object classification on ImageNet [Krizhevsky et al., 2012].

The results are summarized in Fig. 3-5. A sample of units are shown together with both automatically inferred interpretations and manually assigned interpretations taken from Zhou et al. [2015]. We can see that the predicted labels match the human annotation well, though sometimes they capture a different description of a visual concept, such as the 'crosswalk' predicted by the algorithm compared to 'horizontal lines' given by the human for the third unit in `conv4` of Places-AlexNet in Fig. 3-5. Confirming intuition, color and texture concepts dominate at lower layers `conv1` and `conv2` while more object and part detectors emerge in `conv5`.

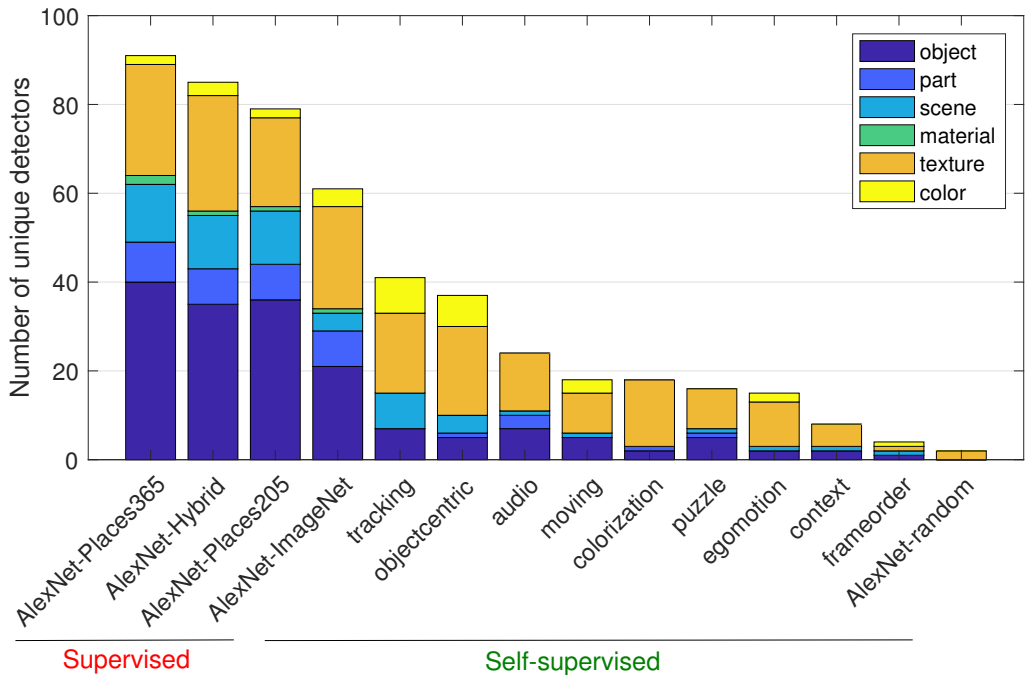


Figure 3-8: Semantic detectors emerge across different supervision of the primary training task. All these models use the AlexNet architecture and are tested at `conv5`.

3.3.4 Network architectures and supervisions

How do different network architectures and training supervisions affect disentangled interpretability of the learned representations? We apply network dissection to evaluate a range of network architectures and supervisions. For simplicity, the following experiments focus on the last convolutional layer of each CNN, where semantic detectors emerge most.

Results showing the number of unique detectors that emerge from various network architectures trained on ImageNet and Places are plotted in Fig. 3-7, with examples shown in Fig. 3-6. In terms of network architecture, we find that interpretability of ResNet > VGG > GoogLeNet > AlexNet. Deeper architectures appear to allow greater interpretability. Comparing training data sets, we find Places > ImageNet. As discussed in Zhou et al. [2015], one scene is composed of multiple objects, so it may be beneficial for more object detectors to emerge in CNNs trained to recognize scenes.

Results from networks trained on various supervised and self-supervised tasks are shown in Fig. 3-8. Here the network architecture is AlexNet for each model, We

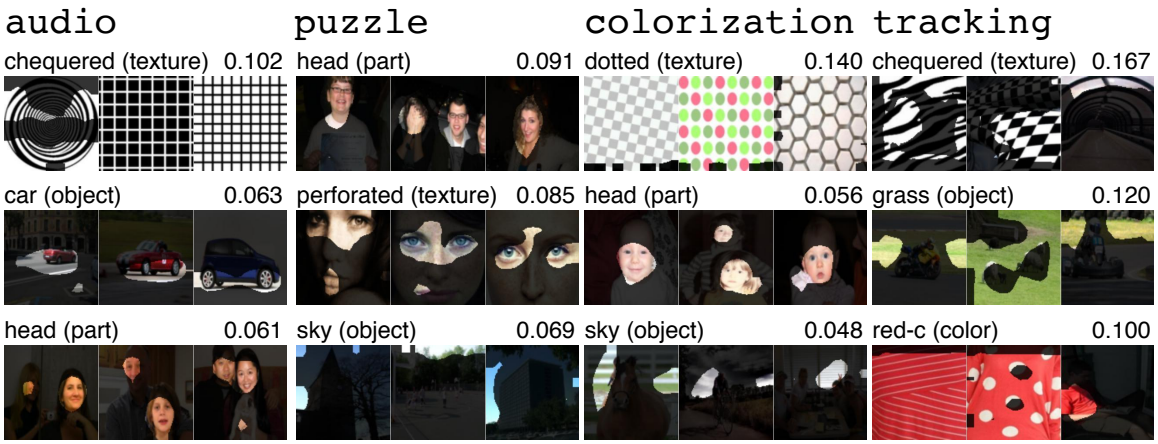


Figure 3-9: The top ranked concepts in the three top categories in four self-supervised networks. Some object and part detectors emerge in `audio`. Detectors for person heads also appear in `puzzle` and `colorization`. A variety of texture concepts dominate models with self-supervised training.

observe that training on Places365 creates the largest number of unique detectors. Self-supervised models create many texture detectors but relatively few object detectors; apparently, supervision from a self-taught primary task is much weaker at inferring interpretable concepts than supervised training on a large annotated data set. The form of self-supervision makes a difference: for example, the colorization model is trained on colorless images, and almost no color detection units emerge. We hypothesize that emergent units represent concepts required to solve the primary task.

Fig. 3-9 shows some typical visual detectors identified in the self-supervised CNN models. For the models `audio` and `puzzle`, some object and part detectors emerge. Those detectors may be useful for CNNs to solve the primary tasks: the `audio` model is trained to associate objects with a sound source, so it may be useful to recognize people and cars; while the `puzzle` model is trained to align the different parts of objects and scenes in an image. For `colorization` and `tracking`, recognizing textures might be good enough for the CNN to solve primary tasks such as colorizing a desaturated natural image; thus it is unsurprising that the texture detectors dominate.

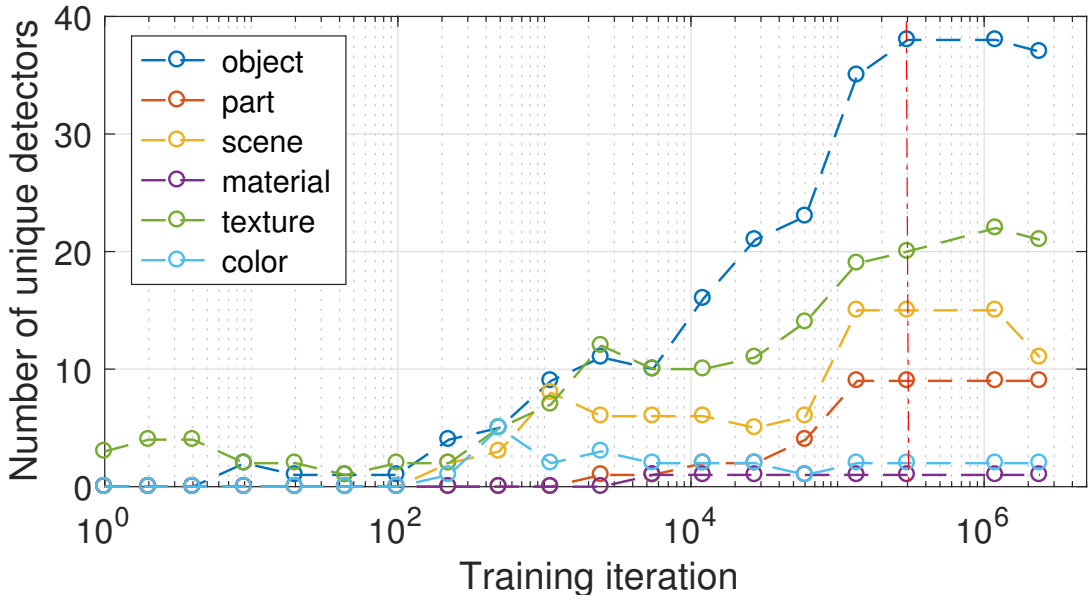


Figure 3-10: The evolution of the interpretability of `conv5` of Places205-AlexNet over 2,400,000 training iterations. The baseline model is trained to 300,000 iterations (marked at the red line).

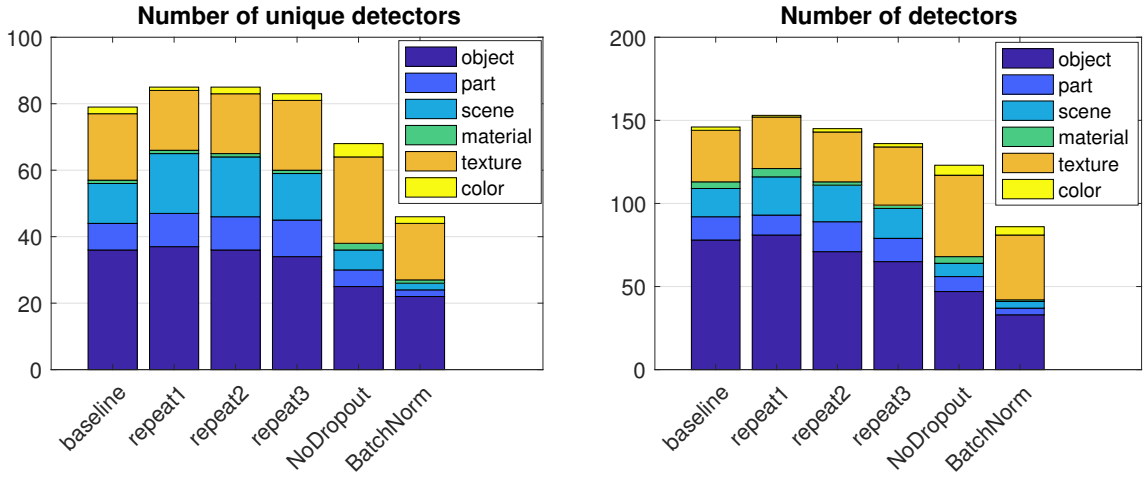


Figure 3-11: Effect of regularizations on the interpretability of CNNs.

3.3.5 Training conditions vs. interpretability

Training conditions such as the number of training iterations, dropout [Srivastava et al., 2014], batch normalization Ioffe and Szegedy [2015], and random initialization [Li et al., 2015], are known to affect the representation learning of neural networks. To analyze the effect of training conditions on interpretability, we take the Places205-AlexNet as the baseline model and prepare several variants of it, all using the same AlexNet architecture. For the variants *Repeat1*, *Repeat2* and *Repeat3*, we randomly initialize the weights and train them with the same number of iterations. For the variant *NoDropout*, we remove the dropout in the FC layers of the baseline model. For the variant *BatchNorm*, we apply batch normalization at each convolutional layers of the baseline model. Repeat1, Repeat2, Repeat3 all have nearly the same top-1 accuracy 50.0% on the validation set. The variant without dropout has top-1 accuracy 49.2%. The variant with batch norm has top-1 accuracy 50.5%.

In Fig. 3-10 we plot the interpretability of snapshots of the baseline model at different training iterations. We can see that object detectors and part detectors begin emerging at about 10,000 iterations (each iteration processes a batch of 256 images). We do not find evidence of transitions across different concept categories during training. For example, units in `conv5` do not turn into texture or material detectors before becoming object or part detectors.

Fig. 3-11 shows the interpretability of units in the CNNs over different training conditions. We find several effects: 1) Comparing different random initializations, the models converge to similar levels of interpretability, both in terms of the unique detector number and the total detector number; this matches observations of convergent learning discussed in Li et al. [2015]. 2) For the network without dropout, more texture detectors emerge but fewer object detectors. 3) Batch normalization seems to decrease interpretability significantly.

The batch normalization result serves as a caution that discriminative power is not the only property of a representation that should be measured. Our intuition for the loss of interpretability under batch normalization is that the batch normalization

'whitens' the activation at each layer, which smooths out scaling issues and allows a network to easily rotate axes of intermediate representations during training. While whitening apparently speeds training, it may also have an effect similar to random rotations analyzed in Sec. 3.3.2 which destroy interpretability. As discussed in Sec. 3.3.2, however, interpretability is neither a prerequisite nor an obstacle to discriminative power. Finding ways to capture the benefits of batch normalization without destroying interpretability is an important area for future work.

3.3.6 Discrimination vs. interpretability

Activations from the higher layers of CNNs are often used as generic visual features, showing great discrimination and generalization ability [Zhou et al., 2014, Razavian et al., 2014]. Here we benchmark deep features from several networks trained on several standard image classification data sets for their discrimination ability on a new task. For each trained model, we extract the representation at the highest convolutional layer, and train a linear SVM with $C = 0.001$ on the training data for action40 action recognition task [Yao et al., 2011]. We compute the classification accuracy averaged across classes on the test split.

Fig. 3-12 plots the number of the unique object detectors for each representation, compared to that representation's classification accuracy on the action40 test set. We can see there is positive correlation between them. Thus the supervision tasks that encourage the emergence of more concept detectors may also improve the discrimination ability of deep features. Interestingly, the best discriminative representation for action40 is the representation from ResNet152-ImageNet, which has fewer unique object detectors compared to ResNet152-Places365. We hypothesize that the accuracy on a representation when applied to a task is dependent not only on the number of concept detectors in the representation, but on the suitability of the set of represented concepts to the transfer task.

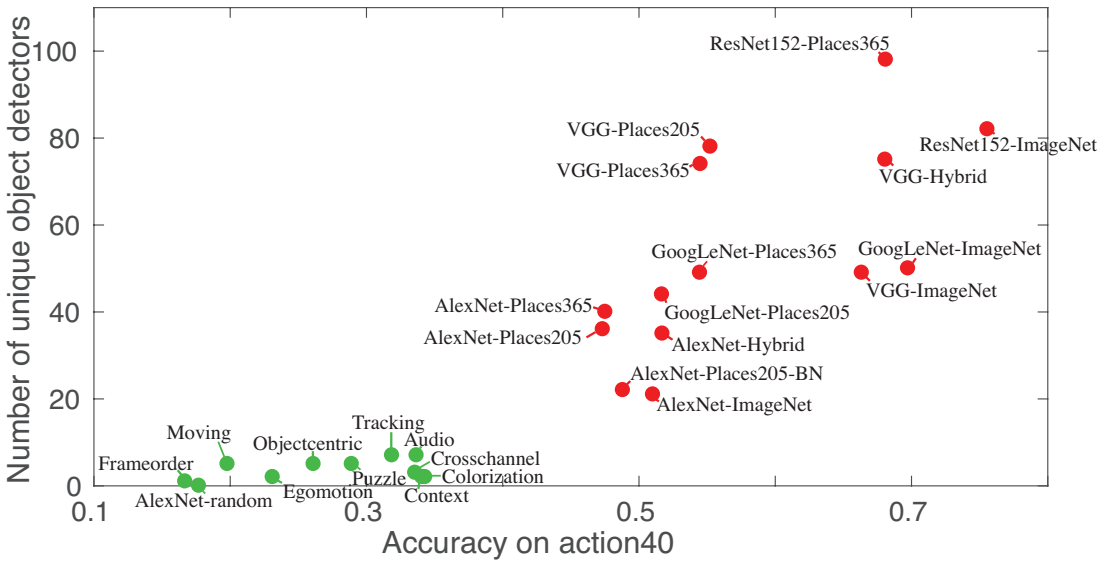


Figure 3-12: The number of unique object detectors in the last convolutional layer compared to each representation's classification accuracy on the action40 data set. Supervised and unsupervised representations clearly form two clusters.

3.3.7 Layer width vs. interpretability

From AlexNet to ResNet, CNNs for visual recognition have grown deeper in the quest for higher classification accuracy. Depth has been shown to be important to high discrimination ability, and we have seen in Sec. 3.3.4 that interpretability can increase with depth as well. However, the width of layers (the number of units per layer) has been less explored. One reason is that increasing the number of convolutional units at a layer significantly increases computational cost while yielding only marginal improvements in classification accuracy. Nevertheless, some recent work [Zagoruyko and Komodakis, 2016] shows that a carefully designed wide residual network can achieve classification accuracy superior to the commonly used thin and deep counterparts.

To explore how the width of layers affects interpretability of CNNs, we do a preliminary experiment to test how width affects emergence of interpretable detectors: we remove the FC layers of the AlexNet, then triple the number of units at the `conv5`, *i.e.*, from 256 units to 768 units. Finally we put a global average pooling layer after `conv5` and fully connect the pooled 768-feature activations to the final class prediction. We call this model *AlexNet-GAP-Wide*.

After training on Places365, the AlexNet-GAP-Wide obtains similar classification

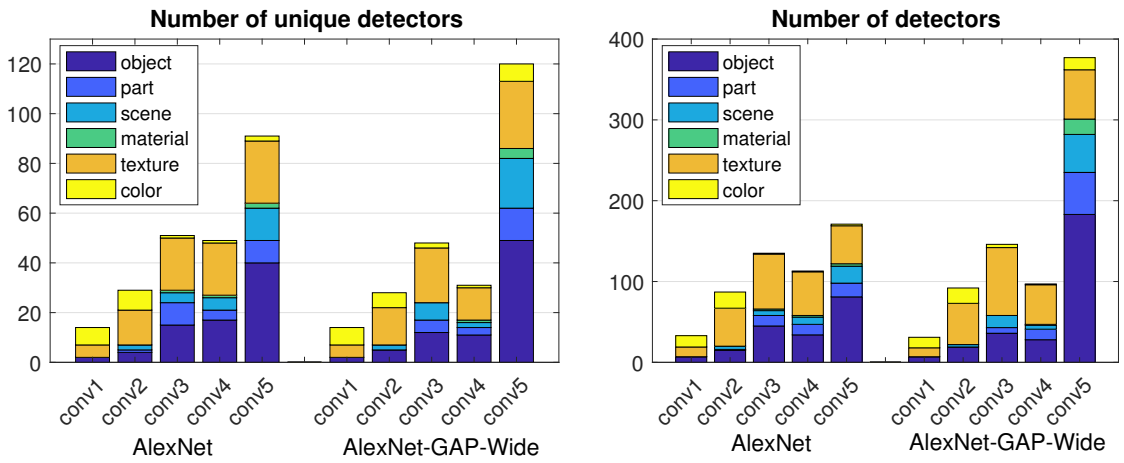


Figure 3-13: Comparison between standard AlexNet and AlexNet-GAP-Wide (AlexNet with wider `conv5` layer and GAP layer) through the number of unique detectors (the left plot) and the number of detectors (the right plot). Widening the layer brings the emergence of more detectors. Networks are trained on Places365.

accuracy on the validation set as the standard AlexNet (0.5% top1 accuracy lower), but it has many more emergent concept detectors, both in terms of the number of unique detectors and the number of detector units at `conv5`, as shown in Fig. 3-13. We have also increased the number of units to 1024 and 2048 at `conv5`, but the number of unique concepts does not significantly increase further. This may indicate a limit on the capacity of AlexNet to separate explanatory factors; or it may indicate that a limit on the number of disentangled concepts that are helpful to solve the primary task of scene classification.

3.4 Discussion

This paper proposed a general framework, network dissection, for quantifying interpretability of CNNs. We applied network dissection to measure whether interpretability is an axis-independent phenomenon, and we found that it is not. This is consistent with the hypothesis that interpretable units indicate a partially disentangled representation. We applied network dissection to investigate the effects on interpretability of state-of-the art CNN training techniques. We have confirmed that representations at different layers disentangle different categories of meaning; and that different training techniques can have a significant effect on the interpretability of the representation

learned by hidden units.

Bibliography

Pulkit Agrawal, Ross Girshick, and Jitendra Malik. Analyzing the performance of multilayer neural networks for object recognition. *ECCV*, 2014.

Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *ICCV*, 2015.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *ICLR Workshop*, 2017.

Sean Bell, Kavita Bala, and Noah Snavely. Intrinsic images in the wild. *ACM Trans. on Graphics (SIGGRAPH)*, 2014.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. Detect what you can: Detecting and representing objects using holistic models and body parts. In *CVPR*, 2014.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *CVPR*, 2014.

Persi Diaconis. What is a random matrix? *Notices of the AMS*, 52(11):1348–1349, 2005.

Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *CVPR*, 2015.

Ruohan Gao, Dinesh Jayaraman, and Kristen Grauman. Object-centric representation learning from unlabeled videos. *arXiv:1612.00500*, 2016.

Abel Gonzalez-Garcia, Davide Modolo, and Vittorio Ferrari. Do semantic parts emerge in convolutional neural networks? *arXiv:1607.03738*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Dinesh Jayaraman and Kristen Grauman. Learning image representations tied to ego-motion. In *ICCV*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations? *arXiv:1511.07543*, 2015.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

Ishan Mikjjsra, C Lawrence Zitnick, and Martial Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *ECCV*, 2016.

Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *CVPR*, 2014.

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *NeurIPS*, 2016.

Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*, 2016.

Andrew Owens, Jiajun Wu, Josh H McDermott, William T Freeman, and Antonio Torralba. Ambient sound provides supervision for visual learning. In *ECCV*, 2016.

R Quian Quiroga, Leila Reddy, Gabriel Kreiman, Christof Koch, and Itzhak Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435 (7045):1102–1107, 2005.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. *arXiv:1403.6382*, 2014.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*, 2014.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

Joost Van De Weijer, Cordelia Schmid, Jakob Verbeek, and Diane Larlus. Learning color names for real-world applications. *IEEE Transactions on Image Processing*, 18(7):1512–1523, 2009.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *NeurIPS*, 2016.

Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *CVPR*, 2015.

Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas Guibas, and Li Fei-Fei. Human action recognition by learning bases of action attributes and parts. In *ICCV*, 2011.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *NeurIPS*, 2014.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv:1605.07146*, 2016.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.

Richard Zhang, Phillip Isola, and Alexei A Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *CVPR*, 2017.

Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NeurIPS*, 2014.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. In *ICLR*, 2015.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Antonio Torralba, and Aude Oliva. Places: An image database for deep scene understanding. *arXiv:1610.02055*, 2016.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017.

Chapter 4

GAN Dissection

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou,
Joshua B. Tenenbaum, William T. Freeman,
Antonio Torralba. ICLR 2019.

4.1 Introduction

Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] have been able to produce photorealistic images, often indistinguishable from real images. This remarkable ability has powered many real-world applications ranging from visual recognition [Wang et al., 2017], to image manipulation [Isola et al., 2017, Zhu et al., 2017], to video prediction [Mathieu et al., 2016]. Since its invention in 2014, many GAN variants have been proposed [Radford et al., 2016, Zhang et al., 2018], often producing more realistic and diverse samples with better training stability.

Despite this tremendous success, many questions remain to be answered. For example, to produce a church image (Figure 4-1a), what knowledge does a GAN need to learn? Alternatively, when a GAN sometimes produces terribly unrealistic images (Figure 4-1f), what causes the mistakes? Why does one GAN variant work better than another? What fundamental differences are encoded in their weights?

In this work, we study the internal representations of GANs. To a human observer, a well-trained GAN appears to have learned facts about the objects in the image: for

Figure 4-1: Overview: (a) A number of realistic outdoor church images generated by Progressive GANs [Karras et al., 2018]. (b) Given a pre-trained GAN model (e.g., Progressive GANs), we first identify a set of interpretable units, whose featuremap is highly correlated to the region of an object class across different images. For example, one unit in `layer4` can localize tree regions with diverse visual appearance. (c) We ablate the units by forcing the activation to be zero and quantify the average casual effect of the ablation. Here we successfully remove these trees from church images. (d) We can insert these tree causal units to other locations. The same set of units can synthesize different trees visually compatible with their surrounding context. In addition, our method can diagnose and improve GANs by identifying artifact-causing units (e). We can remove the artifacts that appear in (f) and significantly improve the results by ablating the "artifact" units (g). Please see our demo video.

example, a door can appear on a building but not on a tree. We wish to understand how a GAN represents such a structure. Do the objects emerge as pure pixel patterns without any explicit representation of objects such as doors and trees, or does the GAN contain internal variables that correspond to the objects that humans perceive? If the GAN does contain variables for doors and trees, do those variables cause the generation of those objects, or do they merely correlate? How are relationships between objects represented?

We present a general method for visualizing and understanding GANs at different levels of abstraction, from each neuron, to each object, to the contextual relationship

between different objects. We first identify a group of interpretable units that are related to object concepts (Figure 4-1b). These units' featuremaps closely match the semantic segmentation of a particular object class (e.g., trees). Second, we directly intervene within the network to identify sets of units that cause a type of objects to disappear (Figure 4-1c) or appear (Figure 4-1d). We quantify the causal effect of these units using a standard causality metric. Finally, we examine the contextual relationship between these causal object units and the background. We study where we can insert the object concepts in new images and how this intervention interacts with other objects in the image (Figure 4-1d). To our knowledge, our work provides the first systematic analysis for understanding the internal representations of GANs.

Finally, we show several practical applications enabled by this analytic framework, from comparing internal representations across different layers, GAN variants and datasets; to debugging and improving GANs by locating and ablating "artifact" units (Figure 4-1e); to understanding contextual relationships between objects in scenes; to manipulating images with interactive object-level control.

4.2 Related work

Generative Adversarial Networks. The quality and diversity of results from GANs [Goodfellow et al., 2014] has continued to improve, from generating simple digits and faces [Goodfellow et al., 2014], to synthesizing natural scene images [Radford et al., 2016, Denton et al., 2015], to generating 1k photorealistic portraits [Karras et al., 2018], to producing one thousand object classes [Miyato et al., 2018, Zhang et al., 2018]. In addition to image generation, GANs have also enabled many applications such as visual recognition [Wang et al., 2017, Hoffman et al., 2018], image manipulation [Isola et al., 2017, Zhu et al., 2017], and video generation [Mathieu et al., 2016, Wang et al., 2018]. Despite the huge success, little work has been done to visualize what GANs have learned. Prior work [Radford et al., 2016, Zhu et al., 2016] manipulates latent vectors and observes how the results change accordingly.

Visualizing deep neural networks. Various methods have been developed

to understand the internal representations of networks, such as visualizations for CNNs [Zeiler and Fergus, 2014] and RNNs [Karpathy et al., 2016, Strobelt et al., 2018]. We can visualize a CNN by locating and reconstructing salient image features [Simonyan et al., 2014, Mahendran and Vedaldi, 2015] or by mining patches that maximize hidden layers' activations [Zeiler and Fergus, 2014], or we can synthesize input images to invert a feature layer [Dosovitskiy and Brox, 2016]. Alternately, we can identify the semantics of each unit [Zhou et al., 2015, Bau et al., 2017, Zhou et al., 2018a] by measuring agreement between unit activations and object segmentation masks. Visualization of an RNN has also revealed interpretable units that track long-range dependencies [Karpathy et al., 2016]. Most previous work on network visualization has focused on networks trained for classification; our work explores deep generative models trained for image generation.

Explaining the decisions of deep neural networks. We can explain individual network decisions using informative heatmaps [Zhou et al., 2018b, 2016, Selvaraju et al., 2017] or modified back-propagation [Simonyan et al., 2014, Bach et al., 2015, Sundararajan et al., 2017]. The heatmaps highlight which regions contribute most to the categorical prediction given by the networks. Recent work has also studied the contribution of feature vectors [Kim et al., 2017, Zhou et al., 2018b] or individual channels [Olah et al., 2018] to the final prediction. Morcos et al. [2018] has examined the effect of individual units by ablating them. Those methods explain discriminative classifiers. Our method aims to explain how an image can be generated by a network, which is much less explored.

4.3 Method

Our goal is to analyze how objects such as trees are encoded by the internal representations of a GAN generator $G\colon \mathbf{z} \to \mathbf{x}$. Here $\mathbf{z} \in \mathbb{R}^{|z|}$ denotes a latent vector sampled from a low-dimensional distribution, and $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ denotes an $H \times W$ generated image. We use *representation* to describe the tensor $\mathbf{r}$ output from a particular layer of the generator G, where the generator creates an image $\mathbf{x}$ from random $\mathbf{z}$ through a

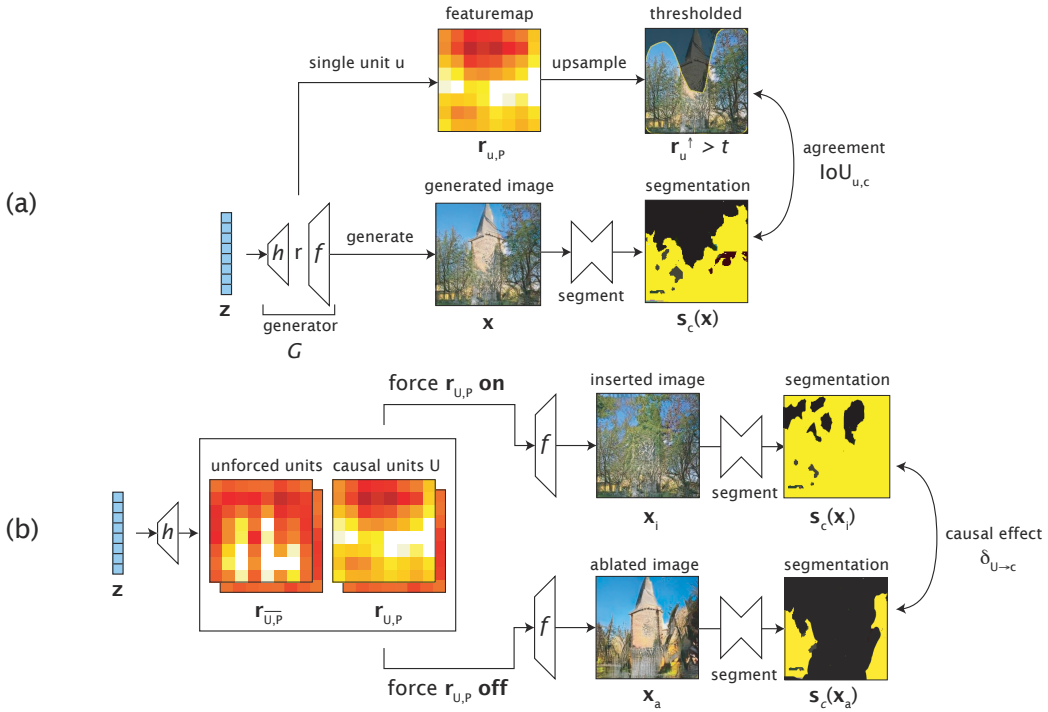


Figure 4-2: Measuring the relationship between representation units and trees in the output using (a) dissection and (b) intervention. Dissection measures agreement between a unit u and a concept c by comparing its thresholded upsampled heatmap with a semantic segmentation of the generated image $s_c(x)$. Intervention measures the causal effect of a set of units U on a concept c by comparing the effect of forcing these units on (unit insertion) and off (unit ablation). The segmentation s_c reveals that trees increase after insertion and decrease after ablation. The average difference in the tree pixels measures the average causal effect. In this figure, interventions are applied to the entire featuremap $\mathbb{P}$, but insertions and ablations can also apply to any subset of pixels $\mathrm{P} \subset \mathbb{P}$.

composition of layers: $\mathbf{r} = h(\mathbf{z})$ and $\mathbf{x} = f(\mathbf{r}) = f(h(\mathbf{z})) = G(\mathbf{z})$.

Since $\mathbf{r}$ has all the data necessary to produce the image $\mathbf{x} = f(\mathbf{r})$, $\mathbf{r}$ certainly contains the information to deduce the presence of any visible class c in the image. Therefore the question we ask is not whether information about c is present in $\mathbf{r}$ — it is — but *how* such information is encoded in $\mathbf{r}$. In particular, for any class from a universe of concepts $c \in \mathbb{C}$, we seek to understand whether $\mathbf{r}$ explicitly represents c in some way where it is possible to factor $\mathbf{r}$ at locations P into two components

$$\mathbf{r}_{\mathbb{U},\mathrm{P}} = (\mathbf{r}_{\mathrm{U},\mathrm{P}}, \mathbf{r}_{\overline{\mathrm{U}},\mathrm{P}}), \tag{4.1}$$

where the generation of the object c at locations P depends mainly on the units $\mathbf{r}_{\mathrm{U},\mathrm{P}}$, and is insensitive to the other units $\mathbf{r}_{\overline{\mathrm{U}},\mathrm{P}}$. Here we refer to each channel of the featuremap as a unit: U denotes the set of unit indices of interest and $\overline{\mathrm{U}}$ is its

Thresholding unit #65 layer 3 of a dining room generator matches 'table' segmentations with IoU=0.34.

Thresholding unit #37 layer 4 of a living room generator matches 'sofa' segmentations with IoU=0.29.

Figure 4-3: Visualizing the activations of individual units in two GANs. Top ten activating images are shown, and IoU is measured over a sample of 1000 images. In each image, the unit feature is upsampled and thresholded as described in Eqn. 4.2.

complement; we will write $\mathbb{U}$ and $\mathbb{P}$ to refer to the entire set of units and featuremap pixels in r. We study the structure of $\mathbf{r}$ in two phases:

- Dissection: starting with a large dictionary of object classes, we identify the classes that have an explicit representation in $\mathbf{r}$ by measuring the agreement between individual units of $\mathbf{r}$ and every class c (Figure 4-1b).
- Intervention: for the represented classes identified through dissection, we identify causal sets of units and measure causal effects between units and object classes by forcing sets of units on and off (Figure 4-1c,d).

4.3.1 Characterizing units by dissection

We first focus on individual units of the representation. Recall that $\mathbf{r}_{u,\mathbb{P}}$ is the one-channel $h \times w$ featuremap of unit u in a convolutional generator, where $h \times w$ is typically smaller than the image size. We want to know if a specific unit $\mathbf{r}_{u,\mathbb{P}}$ encodes a semantic class such as a "tree". For image classification networks, Bau et al. [2017] has observed that many units can approximately locate emergent object classes when the units are upsampled and thresholded. In that spirit, we select a universe of concepts $c \in \mathbb{C}$ for which we have a semantic segmentation $\mathbf{s}_c(\mathbf{x})$ for each class. Then we quantify the spatial agreement between the unit u's thresholded featuremap and a

concept c's segmentation with the following intersection-over-union (IoU) measure:

$$\text{IoU}_{u,c} \equiv \frac{\mathbb{E}_{\mathbf{z}}\left|(\mathbf{r}^{\uparrow}_{u,\mathbb{P}} > t_{u,c}) \wedge \mathbf{s}_c(\mathbf{x})\right|}{\mathbb{E}_{\mathbf{z}}\left|(\mathbf{r}^{\uparrow}_{u,\mathbb{P}} > t_{u,c}) \vee \mathbf{s}_c(\mathbf{x})\right|}, \text{where } t_{u,c} = \arg\max_t \frac{\text{I}(\mathbf{r}^{\uparrow}_{u,\mathbb{P}} > t; \mathbf{s}_c(\mathbf{x}))}{\text{H}(\mathbf{r}^{\uparrow}_{u,\mathbb{P}} > t, \mathbf{s}_c(\mathbf{x}))}, \quad (4.2)$$

where $\wedge$ and $\vee$ denote intersection and union operations, and $\mathbf{x} = G(\mathbf{z})$ denotes the image generated from $\mathbf{z}$. The one-channel feature map $\mathbf{r}_{u,\mathbb{P}}$ slices the entire featuremap $\mathbf{r} = h(\mathbf{z})$ at unit u. As shown in Figure 4-2a, we upsample $\mathbf{r}_{u,\mathbb{P}}$ to the output image resolution as $\mathbf{r}^{\uparrow}_{u,\mathbb{P}}$. $(\mathbf{r}^{\uparrow}_{u,\mathbb{P}} > t_{u,c})$ produces a binary mask by thresholding the $\mathbf{r}^{\uparrow}_{u,\mathbb{P}}$ at a fixed level $t_{u,c}$. $\mathbf{s}_c(\mathbf{x})$ is a binary mask where each pixel indicates the presence of class c in the generated image $\mathbf{x}$. The threshold $t_{u,c}$ is chosen to be informative as possible by maximizing the information quality ratio I/H (using a separate validation set), that is, it maximizes the portion of the joint entropy H which is mutual information I [Wijaya et al., 2017].

We can use $\text{IoU}_{u,c}$ to rank the concepts related to each unit and label each unit with the concept that matches it best. Figure 4-3 shows examples of interpretable units with high $\text{IoU}_{u,c}$. They are not the only units to match tables and sofas: `layer3` of the dining room generator has 31 units (of 512) that match tables and table parts, and `layer4` of the living room generator has 65 (of 512) sofa units.

Once we have identified an object class that a set of units match closely, we next ask: which units are responsible for triggering the rendering of that object? A unit that correlates highly with an output object might not actually cause that output. Furthermore, any output will jointly depend on several parts of the representation. We need a way to identify combinations of units that cause an object.

4.3.2 Measuring causal relationships using intervention

To answer the above question about causality, we probe the network using interventions: we test whether a set of units U in $\mathbf{r}$ cause the generation of c by forcing the units of U on and off.

Recall that $\mathbf{r}_{\text{U,P}}$ denotes the featuremap $\mathbf{r}$ at units U and locations P. We *ablate*

those units by forcing $\mathbf{r}_{\mathrm{U,P}} = \mathbf{0}$. Similarly, we *insert* those units by forcing $\mathbf{r}_{\mathrm{U,P}} = \mathbf{k}$, where $\mathbf{k}$ is a per-class constant, as described in Section A.4. We decompose the featuremap $\mathbf{r}$ into two parts $(\mathbf{r}_{\mathrm{U,P}}, \mathbf{r}_{\overline{\mathrm{U,P}}})$, where $\mathbf{r}_{\overline{\mathrm{U,P}}}$ are unforced components of $\mathbf{r}$:

$$\begin{aligned} &\text{Original image :} && \mathbf{x} = G(\mathbf{z}) \equiv f(\mathbf{r}) \equiv f(\mathbf{r}_{\mathrm{U,P}}, \mathbf{r}_{\overline{\mathrm{U,P}}}) \\ &\text{Image with U ablated at pixels P :} && \mathbf{x}_a = f(\mathbf{0}, \mathbf{r}_{\overline{\mathrm{U,P}}}) \\ &\text{Image with U inserted at pixels P :} && \mathbf{x}_i = f(\mathbf{k}, \mathbf{r}_{\overline{\mathrm{U,P}}}) \end{aligned} \tag{4.3}$$

An object is caused by U if the object appears in $\mathbf{x}_i$ and disappears from $\mathbf{x}_a$. Figure 4-1c demonstrates the ablation of units that remove trees, and Figure 4-1d demonstrates insertion of units at specific locations to make trees appear. This causality can be quantified by comparing the presence of trees in $\mathbf{x}_i$ and $\mathbf{x}_a$ and averaging effects over all locations and images. Following prior work [Holland, 1988, Pearl, 2009], we define the average causal effect (ACE) of units U on the generation of on class c as:

$$\delta_{\mathrm{U}\to c} \equiv \mathbb{E}_{\mathbf{z},\mathrm{P}}[\mathbf{s}_c(\mathbf{x}_i)] - \mathbb{E}_{\mathbf{z},\mathrm{P}}[\mathbf{s}_c(\mathbf{x}_a)], \tag{4.4}$$

where $\mathbf{s}_c(\mathbf{x})$ denotes a segmentation indicating the presence of class c in the image $\mathbf{x}$ at P. To permit comparisons of $\delta_{\mathrm{U}\to c}$ between classes c which are rare, we normalize our segmentation $\mathbf{s}_c$ by $\mathbb{E}_{\mathbf{z},\mathrm{P}}[\mathbf{s}_c(x)]$. While these measures can be applied to a single unit, we have found that objects tend to depend on more than one unit. Thus we need to identify a set of units U that maximize the average causal effect $\delta_{\mathrm{U}\to c}$ for an object class c.

Finding sets of units with high ACE. Given a representation $\mathbf{r}$ with d units, exhaustively searching for a fixed-size set U with high $\delta_{\mathrm{U}\to c}$ is prohibitive as it has $\binom{d}{|\mathrm{U}|}$ subsets. Instead, we optimize a continuous intervention $\boldsymbol{\alpha} \in [0,1]^d$, where each dimension $\boldsymbol{\alpha}_u$ indicates the degree of intervention for a unit u. We maximize the

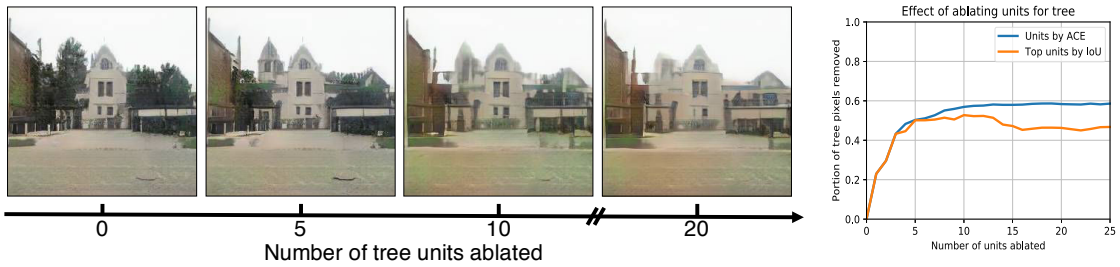


Figure 4-4: Ablating successively larger sets of tree-causal units from a GAN trained on LSUN outdoor church images, showing that the more units are removed, the more trees are reduced, while buildings remain. The choice of units to ablate is specific to the tree class and does not depend on the image. At right, the causal effect of removing successively more tree units is plotted, comparing units chosen to optimize the average causal effect (ACE) and units chosen with the highest IoU for trees.

following average causal effect formulation $\delta_{\boldsymbol{\alpha} \to c}$:

$$\begin{aligned}
&\text{Image with partial ablation at pixels P :} && \mathbf{x}'_a = f((\mathbf{1} - \boldsymbol{\alpha}) \odot \mathbf{r}_{\mathbb{U},\mathrm{P}},\ \mathbf{r}_{\mathbb{U},\overline{\mathrm{P}}}) \qquad (4.5)\\
&\text{Image with partial insertion at pixels P :} && \mathbf{x}'_i = f(\boldsymbol{\alpha} \odot \mathbf{k} + (\mathbf{1} - \boldsymbol{\alpha}) \odot \mathbf{r}_{\mathbb{U},\mathrm{P}},\ \mathbf{r}_{\mathbb{U},\overline{\mathrm{P}}})\\
&\text{Objective :} && \delta_{\boldsymbol{\alpha} \to c} = \mathbb{E}_{\mathbf{z},\mathrm{P}}\left[\mathbf{s}_c(\mathbf{x}'_i)\right] - \mathbb{E}_{\mathbf{z},\mathrm{P}}\left[\mathbf{s}_c(\mathbf{x}'_a)\right],
\end{aligned}$$

where $\mathbf{r}_{\mathbb{U},\mathrm{P}}$ denotes the all-channel featuremap at locations P, $\mathbf{r}_{\mathbb{U},\overline{\mathrm{P}}}$ denotes the all-channel featuremap at other locations $\overline{\mathrm{P}}$, and $\odot$ applies a per-channel scaling vector $\boldsymbol{\alpha}$ to the featuremap $\mathbf{r}_{\mathbb{U},\mathrm{P}}$. We optimize $\boldsymbol{\alpha}$ over the following loss with an L2 regularization:

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} (-\delta_{\boldsymbol{\alpha} \to c} + \lambda ||\boldsymbol{\alpha}||_2), \qquad (4.6)$$

where λ controls the relative importance of each term. We add the L2 loss as we seek a minimal set of casual units. We optimize using stochastic gradient descent, sampling over both $\mathbf{z}$ and featuremap locations P and clamping the coefficient $\boldsymbol{\alpha}$ within the range $[0, 1]^d$ at each step (d is the total number of units). More details of this optimization are discussed in Section A.4. Finally, we can rank units by $\boldsymbol{\alpha}^*_u$ and achieve a stronger causal effect (i.e., removing trees) when ablating successively larger sets of tree-causing units as shown in Figure 4-4.

4.4 Results

We study three variants of Progressive GANs [Karras et al., 2018] trained on LSUN scene datasets [Yu et al., 2015]. To segment the generated images, we use a recent model [Xiao et al., 2018] trained on the ADE20K scene dataset [Zhou et al., 2017]. The model can segment the input image into 336 object classes, 29 parts of large objects, and 25 materials. To further identify units that specialize in object parts, we expand each object class c into additional object part classes c-t, c-b, c-l, and c-r, which denote the top, bottom, left, or right half of the bounding box of a connected component.

Below, we use dissection for analyzing and comparing units across datasets, layers, and models (Section 4.4.1), and locating artifact units (Section 4.4.2). Then, we start with a set of dominant object classes and use intervention to locate causal units that can remove and insert objects in different images (Section 4.4.3 and 4.4.4). In addition, our video demonstrates our interactive tool.

4.4.1 Comparing units across datasets, layers, and models

Emergence of individual unit object detectors We are particularly interested in any units that are correlated with instances of an object class with diverse visual appearances; these would suggest that GANs generate those objects using similar abstractions as humans. Figure 4-3 illustrates two such units. In the dining room dataset, a unit emerges to match dining table regions. More interestingly, the matched tables have different colors, materials, geometry, viewpoints, and levels of clutter: the only obvious commonality among these regions is the concept of a table. This unit's featuremap correlates to the fully supervised segmentation model [Xiao et al., 2018] with a high IoU of 0.34.

Interpretable units for different scene categories The set of all object classes matched by the units of a GAN provides a map of what a GAN has learned about the data. Figure 4-5 examines units from GANs trained on four LSUN scene categories [Yu

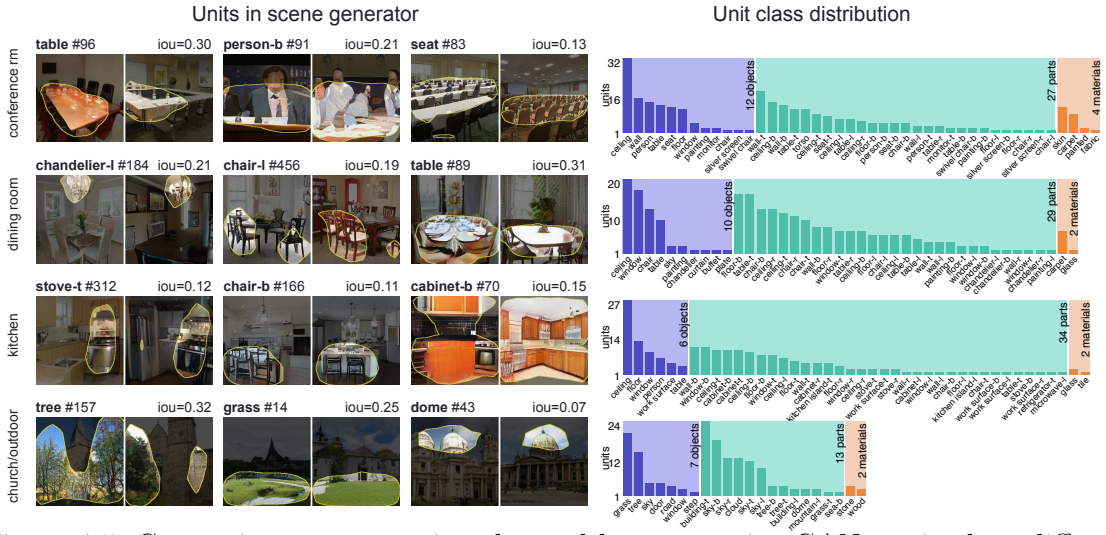


Figure 4-5: Comparing representations learned by progressive GANs trained on different scene types. The units that emerge match objects that commonly appear in the scene type: seats in conference rooms and stoves in kitchens. Units from `layer4` are shown. A unit is counted as a class predictor if it matches a supervised segmentation class with pixel accuracy > 0.75 and IoU > 0.05 when upsampled and thresholded. The distribution of units over classes is shown in the right column.

et al., 2015]. The units that emerge are object classes appropriate to the scene type: for example, when we examine a GAN trained on kitchen scenes, we find units that match stoves, cabinets, and the legs of tall kitchen stools. Another striking phenomenon is that many units represent parts of objects: for example, the conference room GAN contains separate units for the body and head of a person.

Interpretable units for different network layers. In classifier networks, the type of information explicitly represented changes from layer to layer [Zeiler and Fergus, 2014]. We find a similar phenomenon in a GAN. Figure 4-6 compares early, middle, and late layers of a progressive GAN with 14 internal convolutional layers. The output of the first convolutional layer, one step away from the input z, remains entangled: individual units do not correlate well with any object classes except for two units that are biased towards the ceiling of the room. Mid-level layers 4 to 7 have many units that match semantic objects and object parts. Units in layers 10 and beyond match local pixel patterns such as materials, edges and colors. All layers are shown in Section A.7.

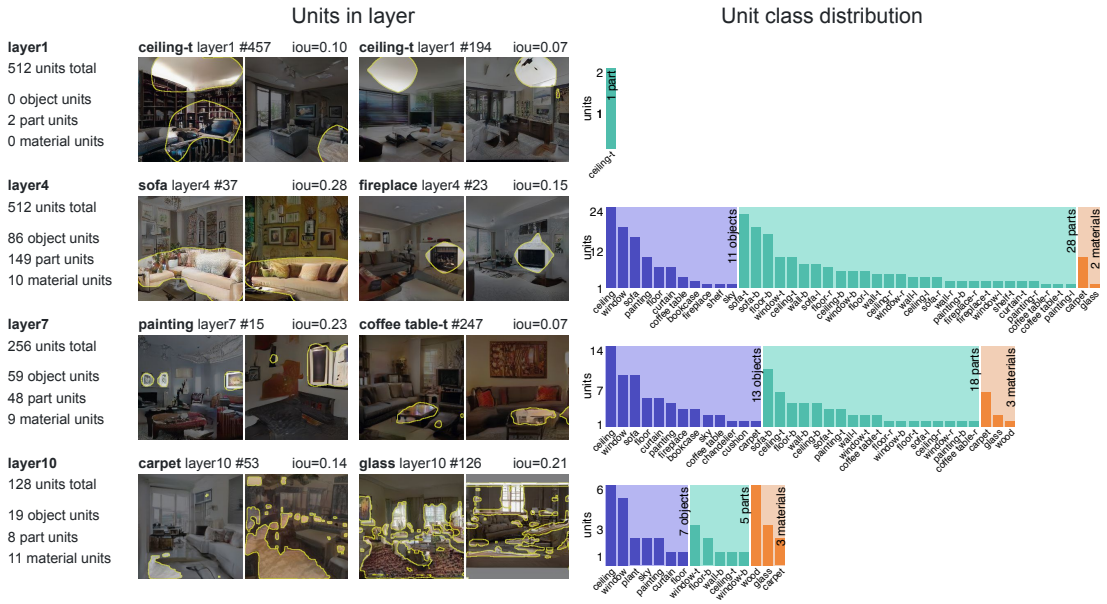


Figure 4-6: Comparing layers of a progressive GAN trained to generate LSUN living room images. The output of the first convolutional layer has almost no units that match semantic objects, but many objects emerge at layers 4-7. Later layers are dominated by low-level materials, edges and colors.

Interpretable units for different GAN models. Interpretable units can provide insights about how GAN architecture choices affect the structures learned inside a GAN. Figure 4-7 compares three models from Karras et al. [2018]: a baseline Progressive GANs, a modification that introduces minibatch stddev statistics, and a further modification that adds pixelwise normalization. By examining unit semantics, we confirm that providing minibatch stddev statistics to the discriminator increases not only the realism of results, but also the diversity of concepts represented by units: the number of types of objects, parts, and materials matching units increases by more than 40%. The pixelwise normalization increases the number of units that match semantic classes by 19%.

4.4.2 Diagnosing and improving GANs

While our framework can reveal how GANs succeed in producing realistic images, it can also analyze the causes of failures in their results. Figure 4-8a shows several annotated units that are responsible for typical artifacts consistently appearing across different images. We can identify these units efficiently by human annotation: out of a

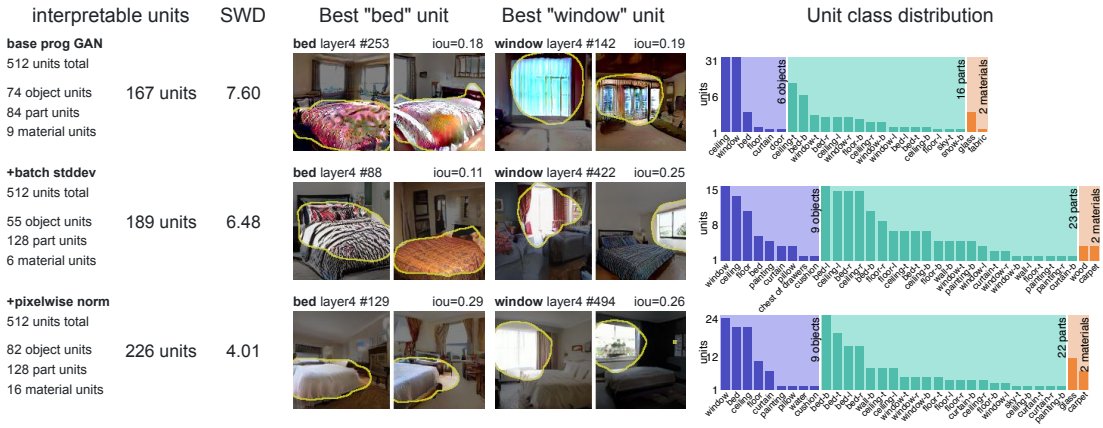


Figure 4-7: Comparing `layer4` representations learned by different training variations. Sliced Wasserstein Distance (SWD) is a GAN quality metric suggested by Karras et al. [2018]: lower SWD indicates more realistic image statistics. Note that as the quality of the model improves, the number of interpretable units also rises. Progressive GANs apply several innovations including making the discriminator aware of minibatch statistics, and pixelwise normalization at each layer. We can see batch awareness increases the number of object classes matched by units, and pixel norm (applied in addition to batch stddev) increases the number of units matching objects.

Table 4.1: We compare generated images before and after ablating 20 "artifact" units. We also report a simple baseline that ablates 20 randomly chosen units.

Fréchet Inception Distance (FID)	
original images	43.16
"artifacts" units ablated (ours)	**27.14**
random units ablated	43.17

Human preference score vs original images	
"artifacts" units ablated (ours)	**72.4**%
random units ablated	49.9%

sample of 1000 images, we visualize the top ten highest activating images for each unit, and we manually identify units with noticeable artifacts in this set. It typically takes 10 minutes to locate 20 artifact-causing units out of 512 units in `layer4`.

More importantly, we can fix these errors by ablating the above 20 artifact-causing units. Figure 4-8b shows that artifacts are successfully removed, and the artifact-free pixels stay the same, improving the generated results. In Table 4.1 we report two standard metrics, comparing our improved images to both the original artifact images and a simple baseline that ablates 20 randomly chosen units. First, we compute the widely used Fréchet Inception Distance [Heusel et al., 2017] between the generated images and real images. We use $50,000$ real images and generate $10,000$ images with high activations on these units. Second, we score $1,000$ images per method on Amazon MTurk, collecting $20,000$ human annotations regarding whether the modified

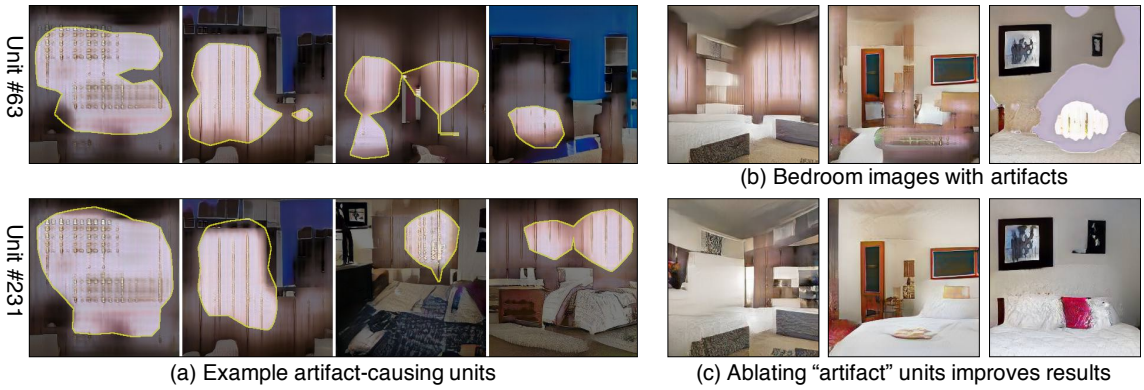


Figure 4-8: (a) We show two example units that are responsible for visual artifacts in GAN results. There are 20 units in total. By ablating these units, we can fix the artifacts in (b) and significantly improve the visual quality as shown in (c).

image looks more realistic compared to the original. Both metrics show significant improvements. Strikingly, this simple manual change to a network beats state-of-the-art GANs models. The manual identification of "artifact" units can be approximated by an automatic scoring of the realism of each unit, as detailed in Section A.1.

4.4.3 Locating causal units with ablation

Errors are not the only type of output that can be affected by directly intervening in a GAN. A variety of specific object types can also be removed from GAN output by ablating a set of units in a GAN. In Figure 4-9 we apply the method in Section 4.3.2 to identify sets of 20 units that have causal effects on common object classes in conference rooms scenes. We find that, by turning off these small sets of units, most of the output of people, curtains, and windows can be removed from the generated scenes. However, not every object can be erased: tables and chairs cannot be removed. Ablating those units will reduce the size and density of these objects, but will rarely eliminate them.

The ease of object removal depends on the scene type. Figure 4-10 shows that, while windows can be removed well from conference rooms, they are more difficult to remove from other scenes. In particular, windows are just as difficult to remove from a bedroom as tables and chairs from a conference room. We hypothesize that the difficulty of removal reflects the level of choice that a GAN has learned for a concept: a conference room is defined by the presence of chairs, so they cannot be altered. And

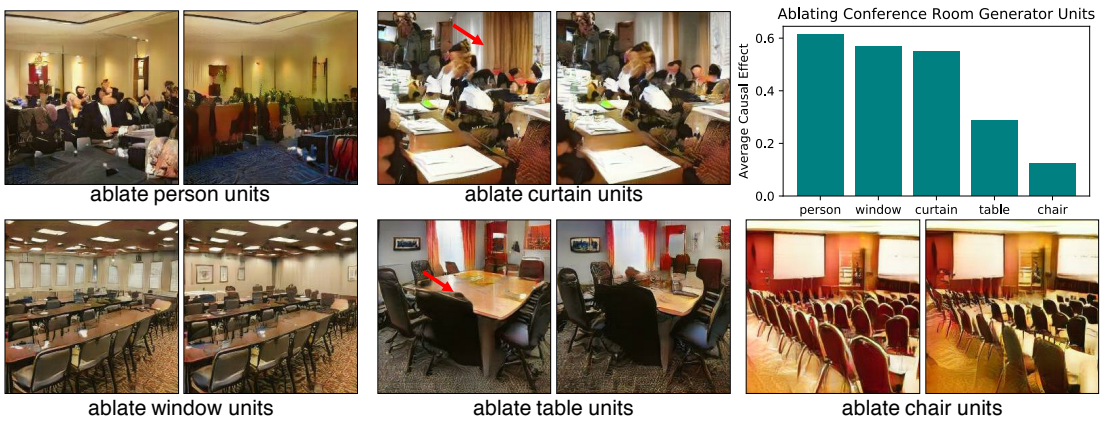


Figure 4-9: Measuring the effect of ablating units in a GAN trained on conference room images. Five different sets of units have been ablated related to a specific object class. In each case, 20 (out of 512) units are ablated from the same GAN model. The 20 units are specific to the object class and independent of the image. The average causal effect is reported as the portion of pixels that are removed in 1 000 randomly generated images. We observe that some object classes are easier to remove cleanly than others: a small ablation can erase most pixels for people, curtains, and windows, whereas a similar ablation for tables and chairs only reduces object sizes without deleting them.

modern building codes mandate that all bedrooms must have windows; the GAN seems to have caught on to that pattern.

4.4.4 Characterizing contextual relationships via insertion

We can also learn about the operation of a GAN by forcing units on and inserting these features into specific locations in scenes. Figure 4-11 shows the effect of inserting 20 `layer4` causal door units in church scenes. In this experiment, we insert these units by setting their activation to the fixed mean value for doors (further details in Section A.4). Although this intervention is the same in each case, the effects vary widely depending on the objects' surrounding context. For example, the doors added to the five buildings in Figure 4-11 appear with a diversity of visual attributes, each with an orientation, size, material, and style that matches the building.

We also observe that doors cannot be added in most locations. The locations where a door can be added are highlighted by a yellow box. The bar chart in Figure 4-11 shows average causal effects of insertions of door units, conditioned on the background object class at the location of the intervention. We find that the GAN allows doors to be added in buildings, particularly in plausible locations such as where a window is

Figure 4-10: Comparing the effect of ablating 20 window-causal units in GANs trained on five scene categories. In each case, the 20 ablated units are specific to the class and the generator and independent of the image. In some scenes, windows are reduced in size or number rather than eliminated, or replaced by visually similar objects such as paintings.

present, or where bricks are present. Conversely, it is not possible to trigger a door in the sky or on trees. Interventions provide insight on how a GAN enforces relationships between objects. Even if we try to add a door in `layer4`, that choice can be vetoed later if the object is not appropriate for the context. These downstream effects are further explored in Section A.5.

4.5 Discussion

By carefully examining representation units, we have found that many parts of GAN representations can be interpreted, not only as signals that correlate with object concepts but as variables that have a causal effect on the synthesis of objects in the output. These interpretable effects can be used to compare, debug, modify, and reason about a GAN model. Our method can be potentially applied to other generative models such as VAEs [Kingma and Welling, 2014] and RealNVP [Dinh et al., 2017].

We have focused on the generator rather than the discriminator (as did in Radford et al. [2016]) because the generator must represent all the information necessary to approximate the target distribution, while the discriminator only learns to capture the difference between real and fake images. Alternatively, we can train an encoder to invert the generator [Donahue et al., 2017, Dumoulin et al., 2017]. However, this incurs additional complexity and errors. Many GANs also do not have an encoder.

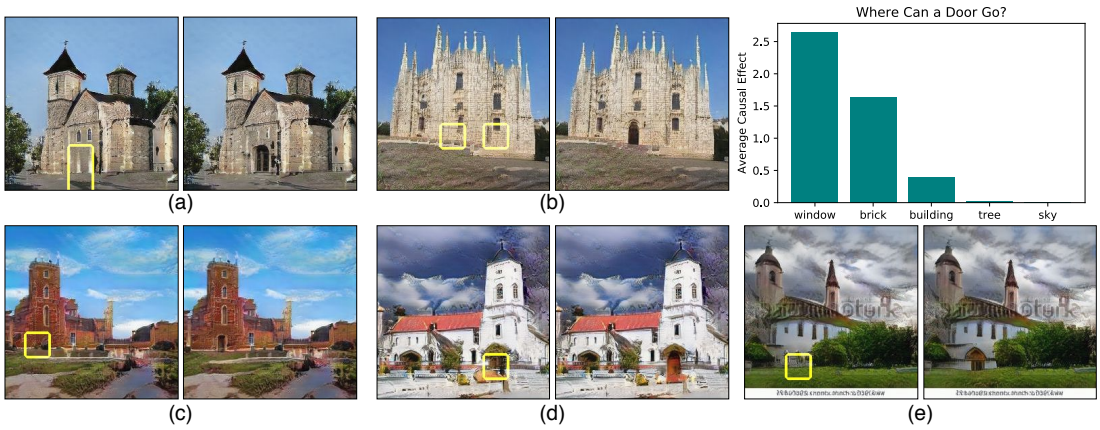


Figure 4-11: Inserting door units by setting 20 causal units to a fixed high value at one pixel in the representation. Whether the door units can cause the generation of doors is dependent on its local context: we highlight every location that is responsive to insertions of door units on top of the original image, including two separate locations in (b) (we intervene at left). The same units are inserted in every case, but the door that appears has a size, alignment, and color appropriate to the location. One way to add door pixels is to emphasize a door that is already present, resulting in a larger door (d). The chart summarizes the causal effect of inserting door units at one pixel with different contexts.

Our method is not designed to compare the quality of GANs to one another, and it is not intended as a replacement for well-studied GAN metrics such as FID, which estimate realism by measuring the distance between the generated distribution of images and the true distribution (Borji [2018] surveys these methods). Instead, our goal has been to identify the interpretable structure and provide a window into the internal mechanisms of a GAN.

Prior visualization methods [Zeiler and Fergus, 2014, Bau et al., 2017, Karpathy et al., 2016] have brought new insights into CNN and RNNs research. Motivated by that, in this work we have taken a small step towards understanding the internal representations of a GAN, and we have uncovered many questions that we cannot yet answer with the current method. For example: why can a door not be inserted in the sky? How does the GAN suppress the signal in the later layers? Further work will be needed to understand the relationships between layers of a GAN. Nevertheless, we hope that our work can help researchers and practitioners better analyze and develop their own GANs.

Bibliography

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

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *CVPR*, 2017.

Ali Borji. Pros and cons of gan evaluation measures. *arXiv preprint arXiv:1802.03446*, 2018.

Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *NeurIPS*, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *ICLR*, 2017.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *ICLR*, 2017.

Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *NeurIPS*, 2016.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. In *ICLR*, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *ICML*, 2018.

Paul W Holland. Causal inference, path analysis and recursive structural equations models. *ETS Research Report Series*, 1988(1):i–50, 1988.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. In *ICLR*, 2016.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

Been Kim, Justin Gilmer, Fernanda Viegas, Ulfar Erlingsson, and Martin Wattenberg. Tcav: Relative concept importance testing with linear concept activation vectors. *arXiv preprint arXiv:1711.11279*, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *ICLR*, 2016.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *ICLR*, 2018.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

Judea Pearl. *Causality*. Cambridge university press, 2009.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*, 2014.

Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M. Rush. LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE TVCG*, 24(1):667–676, Jan 2018.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *PMLR*, 2017.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *NeurIPS*, 2018.

Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *CVPR*, 2017.

Dedy Rahman Wijaya, Riyanarto Sarno, and Enny Zulaika. Information quality ratio as a novel metric for mother wavelet selection. *Chemometrics and Intelligent Laboratory Systems*, 160:59–71, 2017.

Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, 2018.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. In *ICLR*, 2015.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017.

Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *PAMI*, 2018a.

Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable basis decomposition for visual explanation. In *ECCV*, pages 119–134, 2018b.

Tinghui Zhou, Philipp Krahenbuhl, Mathieu Aubry, Qixing Huang, and Alexei A Efros. Learning dense correspondence via 3d-guided cycle consistency. In *CVPR*, 2016.

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, 2016.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

Chapter 5

Seeing what a GAN Cannot Generate

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou,
Joshua B. Tenenbaum, William T. Freeman,
Antonio Torralba. ICCV 2019.

5.1 Introduction

The remarkable ability of a Generative Adversarial Network (GAN) to synthesize realistic images leads us to ask: How can we know what a GAN is *unable* to generate? Mode-dropping or mode collapse, where a GAN omits portions of the target distribution, is seen as one of the biggest challenges for GANs [Goodfellow, 2016, Li and Malik, 2018], yet current analysis tools provide little insight into this phenomenon in state-of-the-art GANs.

Our paper aims to provide detailed insights about dropped modes. Our goal is not to measure GAN quality using a single number: existing metrics such as Inception scores [Salimans et al., 2016] and Fréchet Inception Distance [Heusel et al., 2017] focus on that problem. While those numbers measure *how far* the generated and target distributions are from each other, we instead seek to understand *what* is different between real and fake images. Existing literature typically answers the latter question by sampling generated outputs, but such samples only visualize what a GAN is *capable* of doing. We address the complementary problem: we want to see what a GAN *cannot*

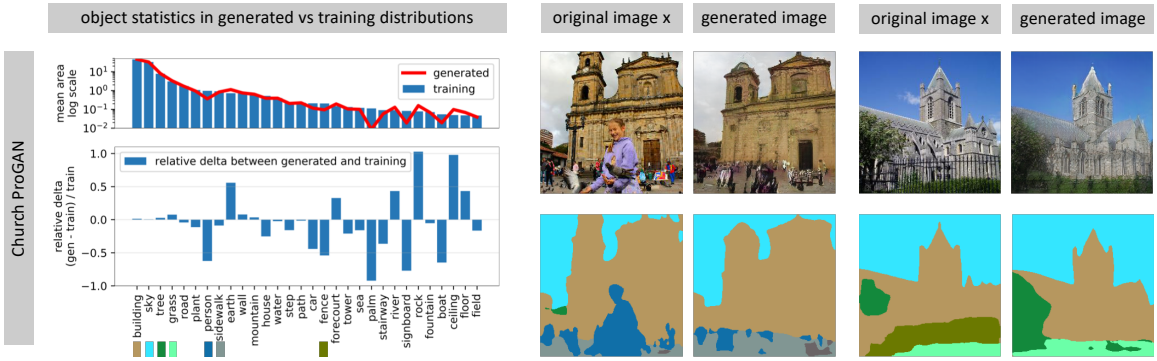


Figure 5-1: Seeing what a GAN cannot generate: (a) We compare the distribution of object segmentations in the training set of LSUN churches Yu et al. [2015] to the distribution in the generated results: objects such as people, cars, and fences are dropped by the generator. (b) We compare pairs of a real image and its reconstruction in which individual instances of a person and a fence cannot be generated. In each block, we show a real photograph (top-left), a generated reconstruction (top-right), and segmentation maps for both (bottom).

generate.

In particular, we wish to know: Does a GAN deviate from the target distribution by ignoring difficult images altogether? Or are there specific, semantically meaningful parts and objects that a GAN decides not to learn about? And if so, how can we detect and visualize these missing concepts that a GAN does not generate?

Image generation methods are typically tested on images of faces, objects, or scenes. Among these, scenes are an especially fertile test domain as each image can be parsed into clear semantic components by segmenting the scene into objects. Therefore, we propose to directly understand mode dropping by analyzing a scene generator at two levels: the distribution level and instance level.

First, we characterize omissions in the distribution as a whole, using *Generated Image Segmentation Statistics*: we segment both generated and ground truth images and compare the distributions of segmented object classes. For example, Figure 5-1a shows that in a church GAN model, object classes such as people, cars, and fences appear on fewer pixels of the generated distribution as compared to the training distribution.

Second, once omitted object classes are identified, we want to visualize specific examples of failure cases. To do so, we must find image instances where the GAN *should* generate an object class but does not. We find such cases using a new reconstruction method called *Layer Inversion* which relaxes reconstruction to a tractable problem.

Instead of inverting the entire GAN, our method inverts a layer of the generator. Unlike existing methods to invert a small generator [Zhu et al., 2016, Brock et al., 2017], our method allows us to create reconstructions for complex, state-of-the-art GANs. Deviations between the original image and its reconstruction reveal image features and objects that the generator cannot draw faithfully.

We apply our framework to analyze several recent GANs trained on different scene datasets. Surprisingly, we find that dropped object classes are not distorted or rendered in a low quality or as noise. Instead, they are simply not rendered at all, *as if the object was not part of the scene*. For example, in Figure 5-1b, we observe that large human figures are skipped entirely, and the parallel lines in a fence are also omitted. Thus a GAN can ignore classes that are too hard, while at the same time producing outputs of high average visual quality. Code, data, and additional information are available at `https://ganseeing.csail.mit.edu`.

5.2 Related work

Generative Adversarial Networks [Goodfellow et al., 2014] have enabled many computer vision and graphics applications such as generation Brock et al. [2019], Karras et al. [2018, 2019], image and video manipulation Huang et al. [2018], Isola et al. [2017], Park et al. [2019], Sangkloy et al. [2017], Taigman et al. [2017], Wang et al. [2018], Zhu et al. [2017], object recognition Bousmalis et al. [2017], Wang et al. [2017], and text-to-image translation Reed et al. [2016], Xu et al. [2018], Zhang et al. [2017]. One important issue in this emerging topic is how to evaluate and compare different methods Theis et al. [2016], Wu et al. [2017a]. For example, many evaluation metrics have been proposed to evaluate unconditional GANs such as Inception score Salimans et al. [2016], Fréchet Inception Distance Heusel et al. [2017], and Wasserstein Sliced Distance Karras et al. [2018]. Though the above metrics can quantify different aspects of model performance, they cannot explain what visual content the models fail to synthesize. Our goal here is *not* to introduce a metric. Instead, we aim to provide explanations of a common failure case of GANs: mode collapse. Our error diagnosis

tools complement existing single-number metrics and can provide additional insights into the model's limitations.

Network inversion. Prior work has found that inversions of GAN generators are useful for photo manipulation [Bau et al., 2019a, Brock et al., 2017, Peleg and Wolf, 2018, Zhu et al., 2016] and unsupervised feature learning Donahue et al. [2017], Dumoulin et al. [2017]. Later work found that DCGAN left-inverses can be computed to high precision Lipton and Tripathi [2017], Yeh et al. [2017], and that inversions of a GAN for glyphs can reveal specific strokes that the generator is unable to generate Creswell and Bharath [2018]. While previous work Zhu et al. [2016] has investigated inversion of 5-layer DCGAN generators, we find that when moving to a 15-layer Progressive GAN, high-quality inversions are much more difficult to obtain. Omissions of a generator can also be estimated using Monte Carlo methods to sample the modeled posterior near a target image Wu et al. [2017b]. In our work, we develop a layer-wise inversion method that is effective for large-scale GANs. We apply a classic layer-wise training approach Bengio et al. [2007], Hinton and Salakhutdinov [2006] to the problem of training an encoder and further introduce layer-wise image-specific optimization. Our work is also loosely related to inversion methods for understanding CNN features and classifiers Dosovitskiy and Brox [2016], Mahendran and Vedaldi [2015], Olah et al. [2017, 2018]. However, we focus on understanding generative models rather than classifiers.

Understanding and visualizing networks. Most prior work on network visualization concerns discriminative classifiers Bach et al. [2015], Bau et al. [2017], Kindermans et al. [2017], Lundberg and Lee [2017], Smilkov et al. [2017], Springenberg et al. [2014], Zeiler and Fergus [2014], Zhou et al. [2014]. GANs have been visualized by examining the discriminator Radford et al. [2016] and the semantics of internal features Bau et al. [2019b]. Different from recent work Bau et al. [2019b] that aims to understand what a GAN has learned, our work provides a complementary perspective and focuses on what semantic concepts a GAN fails to capture.

5.3 Method

Our goal is to visualize and understand the semantic concepts that a GAN generator cannot generate, in both the entire distribution and in each image instance. We will proceed in two steps. First, we measure *Generated Image Segmentation Statistics* by segmenting both generated and target images and identifying types of objects that a generator omits when compared to the distribution of real images. Second, we visualize how the dropped object classes are omitted for individual images by finding real images that contain the omitted classes and projecting them to their best reconstruction given an intermediate layer of the generator. We call the second step *Layer Inversion*.

5.3.1 Quantifying distribution-level mode collapse

The systematic errors of a GAN can be analyzed by exploiting the hierarchical structure of a scene image. Each scene has a natural decomposition into objects, so we can estimate deviations from the true distribution of scenes by estimating deviations of constituent object statistics. For example, a GAN that renders bedrooms should also render some amount of curtains. If the curtain statistics depart from what we see in true images, we will know we can look at curtains to see a specific flaw in the GAN.

To implement this idea, we segment all the images using the Unified Perceptual Parsing network [Xiao et al., 2018], which labels each pixel of an image with one of 336 object classes. Over a sample of images, we measure the total area in pixels for each object class and collect mean and covariance statistics for all segmented object classes. We sample these statistics over a large set of generated images as well as training set images. We call the statistics over all object segmentations *Generated Image Segmentation Statistics*.

Figure 5-2 visualizes mean statistics for two networks. In each graph, the mean segmentation frequency for each generated object class is compared to that seen in the true distribution. Since most classes do not appear on most images, we focus on the most common classes by sorting classes by descending frequency. The comparisons can

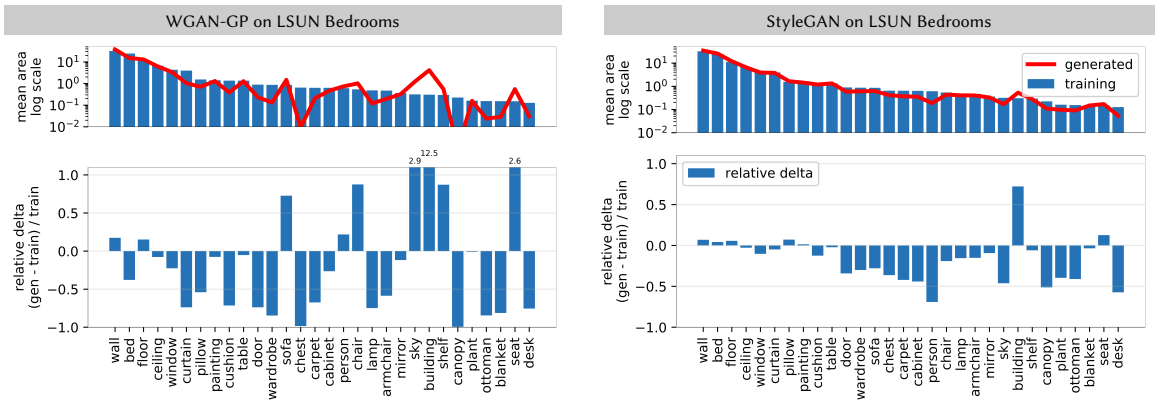


Figure 5-2: Using Generated Image Segmentation Statistics to understand the different behavior of the two models trained on LSUN bedrooms Yu et al. [2015]. The histograms reveal that WGAN-GP Gulrajani et al. [2017] (left) deviates from the true distribution much more than StyleGAN Karras et al. [2019] (right), identifying segmentation classes that are generated too little and others that are generated too much. For example, WGAN-GP does not generate enough pixels containing beds, curtains, or cushions compared to the true distribution of bedroom images, while StyleGAN correctly matches these statistics. StyleGAN is still not perfect, however, and does not generate enough doors, wardrobes, or people. Numbers above bars indicate clipped values beyond the range of the chart.

reveal many specific differences between recent state-of-the-art models. Both analyzed models are trained on the same image distribution (LSUN bedrooms Yu et al. [2015]), but WGAN-GP Gulrajani et al. [2017] departs from the true distribution much more than StyleGAN Karras et al. [2019].

It is also possible to summarize statistical differences in segmentation in a single number. To do this, we define the Fréchet Segmentation Distance (FSD), which is an interpretable analog to the popular Fréchet Inception Distance (FID) metric [Heusel et al., 2017]: $\text{FSD} \equiv ||\mu_g - \mu_t||^2 + \text{Tr}(\Sigma_g + \Sigma_t - 2(\Sigma_g\Sigma_t)^{1/2})$. In our FSD formula, μ_t is the mean pixel count for each object class over a sample of training images, and Σ_t is the covariance of these pixel counts. Similarly, μ_g and Σ_g reflect segmentation statistics for the generative model. In our experiments, we compare statistics between 10,000 generated samples and 10,000 natural images.

Generated Image Segmentation Statistics measure the entire distribution: for example, they reveal when a generator omits a particular object class. However, they do not single out specific images where an object should have been generated but was not. To gain further insight, we need a method to visualize omissions of the generator

for each image.

5.3.2 Quantifying instance-level mode collapse

To address the above issue, we compare image pairs $(\mathbf{x}, \mathbf{x}')$, where $\mathbf{x}$ is a real image that contains a particular object class dropped by a GAN generator G, and $\mathbf{x}'$ is a projection onto the space of all images that can be generated by a layer of the GAN model.

Defining a tractable inversion problem. In the ideal case, we would like to find an image that can be perfectly synthesized by the generator G and stay close to the real image $\mathbf{x}$. Formally, we seek $\mathbf{x}' = G(\mathbf{z}^*)$, where $\mathbf{z}^* = \arg\min_{\mathbf{z}} \ell(G(\mathbf{z}), \mathbf{x})$ and ℓ is a distance metric in image feature space. Unfortunately, as shown in Section 5.4.4, previous methods Donahue et al. [2017], Zhu et al. [2016] fail to solve this full inversion problem for recent generators due to the large number of layers in G. Therefore, we instead solve a tractable subproblem of full inversion. We decompose the generator G into layers

$$G = G_f(g_n(\cdots((g_1(\mathbf{z})))), \tag{5.1}$$

where $g_1, ..., g_n$ are several early layers of the generator, and G_f groups all the later layers of the G together.

Any image that can be generated by G can also be generated by G_f. That is, if we denote by range(G) the set of all images that can be output by G, then we have range(G) $\subset$ range(G_f). That implies, conversely, that any image that cannot be generated by G_f cannot be generated by G either. Therefore any omissions we can identify in range(G_f) will also be omissions of range(G).

Thus for layer inversion, we visualize omissions by solving the easier problem of

inverting the later layers G_f:

$$\mathbf{x}' = G_f(\mathbf{r}^*), \tag{5.2}$$
$$\text{where } \mathbf{r}^* = \arg\min_{\mathbf{r}} \ell(G_f(\mathbf{r}), \mathbf{x}).$$

Although we ultimately seek an intermediate representation $\mathbf{r}$, it will be helpful to begin with an estimated $\mathbf{z}$: an initial guess for $\mathbf{z}$ helps us regularize our search to favor values of $\mathbf{r}$ that are more likely to be generated by a $\mathbf{z}$. Therefore, we solve the inversion problem in two steps: First we construct a neural network E that approximately inverts the entire G and computes an estimate $\mathbf{z}_0 = E(\mathbf{x})$. Subsequently we solve an optimization problem to identify an intermediate representation $\mathbf{r}^* \approx \mathbf{r}_0 = g_n(\cdots(g_1(\mathbf{z}_0)))$ that generates a reconstructed image $G_f(\mathbf{r}^*)$ to closely recover $\mathbf{x}$. Figure 5-3 illustrates our layer inversion method.

Layer-wise network inversion. A deep network can be trained more easily by pre-training individual layers on smaller problems Hinton and Salakhutdinov [2006]. Therefore, to learn the inverting neural network E, we also proceed layer-wise. For each layer $g_i \in \{g_1, ..., g_n, G_f\}$, we train a small network e_i to approximately invert g_i. That is, defining $\mathbf{r}_i = g_i(\mathbf{r}_{i-1})$, our goal is to learn a network e_i that approximates the computation $\mathbf{r}_{i-1} \approx e_i(\mathbf{r}_i)$. We also want the predictions of the network e_i to well preserve the output of the layer g_i, so we want $\mathbf{r}_i \approx g_i(e_i(\mathbf{r}_i))$. We train e_i to minimize both left- and right-inversion losses:

$$\begin{aligned}
\mathcal{L}_{\text{L}} &\equiv \mathbb{E}_{\mathbf{z}}[||\mathbf{r}_{i-1} - e(g_i(\mathbf{r}_{i-1}))||_1] \\
\mathcal{L}_{\text{R}} &\equiv \mathbb{E}_{\mathbf{z}}[||\mathbf{r}_i - g_i(e(\mathbf{r}_i))||_1] \\
e_i &= \arg\min_{e} \quad \mathcal{L}_{\text{L}} + \lambda_{\text{R}}\mathcal{L}_{\text{R}},
\end{aligned} \tag{5.3}$$

To focus on training near the manifold of representations produced by the generator, we sample $\mathbf{z}$ and then use the layers g_i to compute samples of $\mathbf{r}_{i-1}$ and $\mathbf{r}_i$, so $\mathbf{r}_{i-1} = g_{i-1}(\cdots g_1(\mathbf{z})\cdots)$. Here $||\cdot||_1$ denotes an L1 loss, and we set $\lambda_R = 0.01$ to emphasize the reconstruction of $\mathbf{r}_{i-1}$.

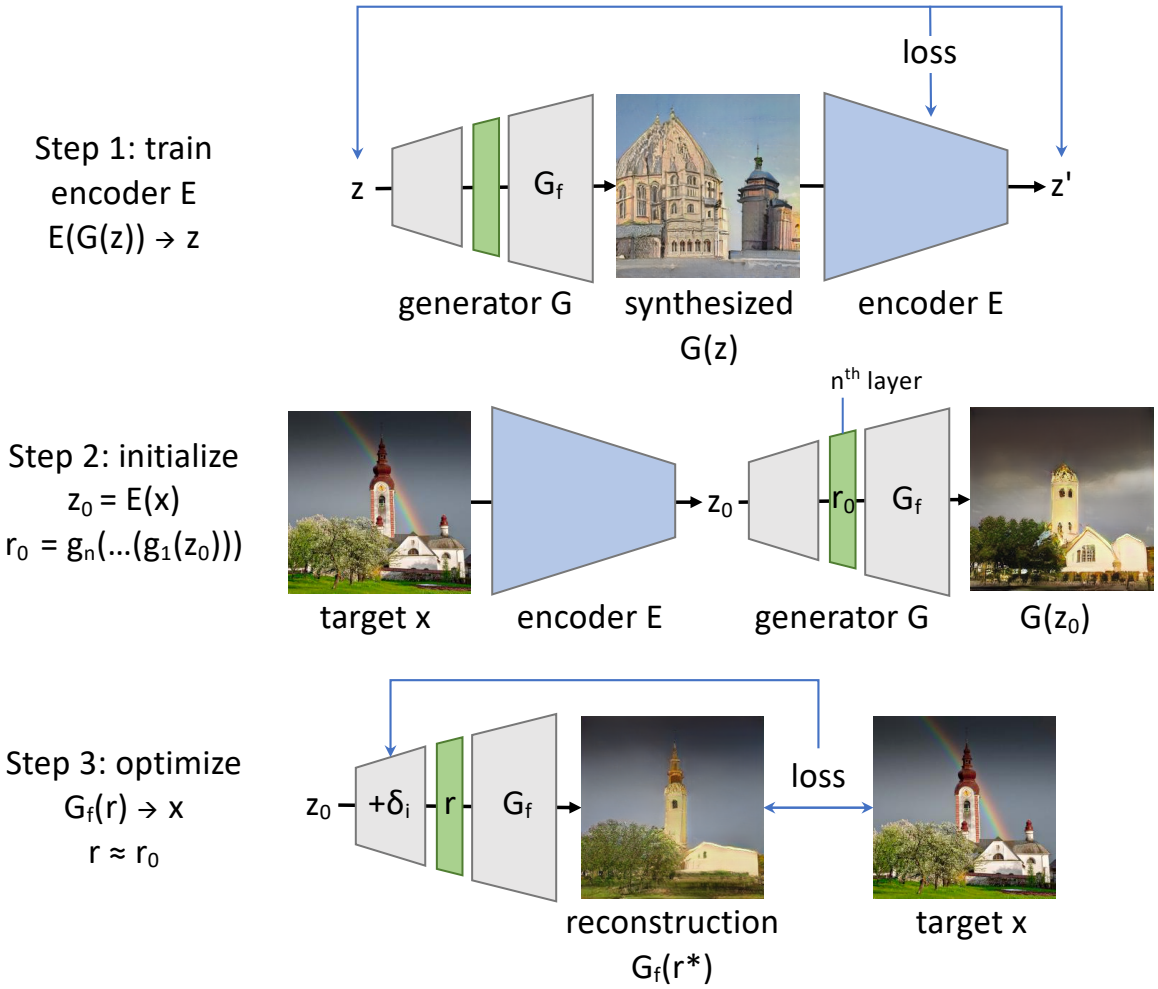


Figure 5-3: Overview of our layer inversion method. First, we train a network E to invert G; this is used to obtain an initial guess of the latent $\mathbf{z}_0 = E(\mathbf{x})$ and its intermediate representation $\mathbf{r}_0 = g_n(\cdots(g_1(\mathbf{z}_0)))$. Then $\mathbf{r}_0$ is used to initialize a search for $\mathbf{r}^*$ to obtain a reconstruction $\mathbf{x}' = G_f(\mathbf{r}^*)$ close to the target $\mathbf{x}$.

Once all the layers are inverted, we can compose an inversion network for all of G:

$$E^* = e_1(e_2(\cdots(e_n(e_f(\mathbf{x}))))). \tag{5.4}$$

The results can be further improved by jointly fine-tuning this composed network E^* to invert G as a whole. We denote this fine-tuned result as E.

Layer-wise image optimization. As described at the beginning of Section 5.3.2, inverting the entire G is difficult: G is non-convex, and optimizations over $\mathbf{z}$ are quickly trapped in local minima. Therefore, after obtaining a decent initial guess for $\mathbf{z}$, we

turn our attention to the more relaxed optimization problem of inverting the layers G_f; that is, starting from $\mathbf{r}_0 = g_n(\cdots(g_1(\mathbf{z}_0)))$, we seek an intermediate representation $\mathbf{r}^*$ that generates a reconstructed image $G_f(\mathbf{r}^*)$ to closely recover $\mathbf{x}$.

To regularize our search to favor $\mathbf{r}$ that are close to the representations computed by the early layers of the generator, we search for $\mathbf{r}$ that can be computed by making small perturbations of the early layers of the generator:

$$\begin{aligned} \mathbf{z}_0 &\equiv E(\mathbf{x}) \\ \mathbf{r} &\equiv \delta_n + g_n(\cdots(\delta_2 + g_2(\delta_1 + g_1(\mathbf{z}_0)))) \\ \mathbf{r}^* &= \arg\min_{\mathbf{r}} \left(\ell(\mathbf{x}, G_f(\mathbf{r})) + \lambda_{\text{reg}} \sum_i ||\delta_i||^2 \right) \\ \ell(\mathbf{x}, \mathbf{x}_g) &\equiv ||\mathbf{x} - \mathbf{x}_g||_1 + \lambda_{\text{V}} ||\text{V}(\mathbf{x}) - \text{V}(\mathbf{x}_g)||_1. \end{aligned} \tag{5.5}$$

That is, we begin with the guess $\mathbf{z}_0$ given by the neural network E, and then we learn small perturbations of each layer before the n-th layer, to obtain an $\mathbf{r}$ that reconstructs the image $\mathbf{x}$ well. For ℓ we sum image pixel loss and VGG perceptual loss Simonyan and Zisserman [2015], similar to existing reconstruction methods Dosovitskiy and Brox [2016], Zhu et al. [2016]; we set $\lambda_{\text{V}} = 1$. The hyper-parameter λ_{reg} determines the balance between image reconstruction loss and the regularization of $\mathbf{r}$. We set $\lambda_{\text{reg}} = 1$ in our experiments.

5.4 Results

Implementation details. We analyze three recent models: WGAN-GP [Gulrajani et al., 2017], Progressive GAN [Karras et al., 2018], and StyleGAN [Karras et al., 2019], trained on LSUN bedroom images [Yu et al., 2015]. In addition, for Progressive GAN we analyze a model trained to generate LSUN church images. To segment images, we use the Unified Perceptual Parsing network [Xiao et al., 2018], which labels each pixel of an image with one of 336 object classes. Segmentation statistics are computed over samples of 10,000 images.

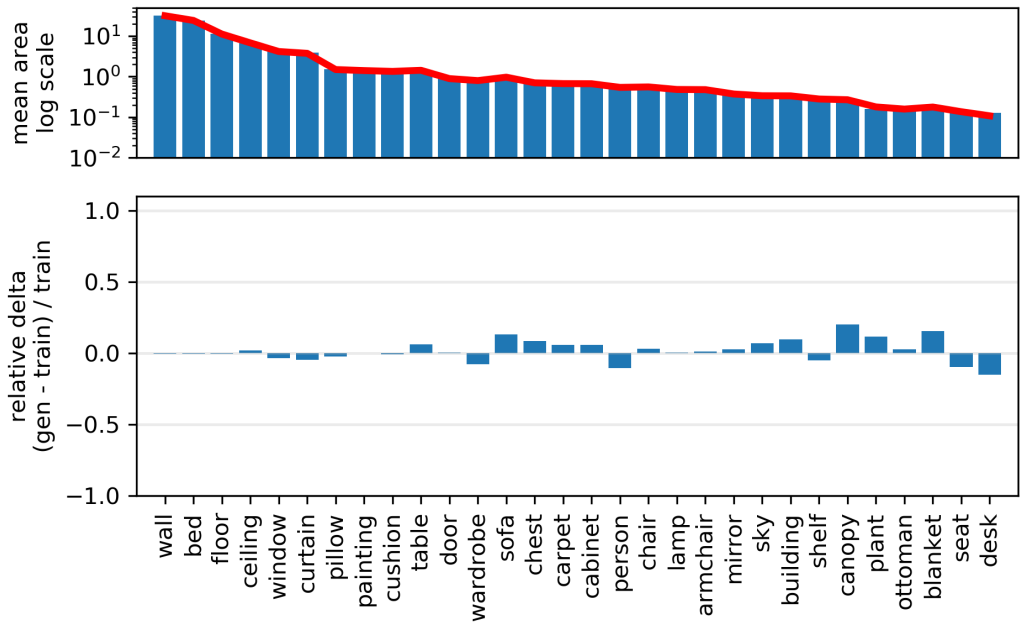


Figure 5-4: Sensitivity test for Generated Image Segmentation Statistics. This plot compares two different random samples of 10,000 images from the LSUN bedroom dataset. An infinite-sized sample would show no differences; the observed differences reveal the small measurement noise introduced by the finite sampling process.

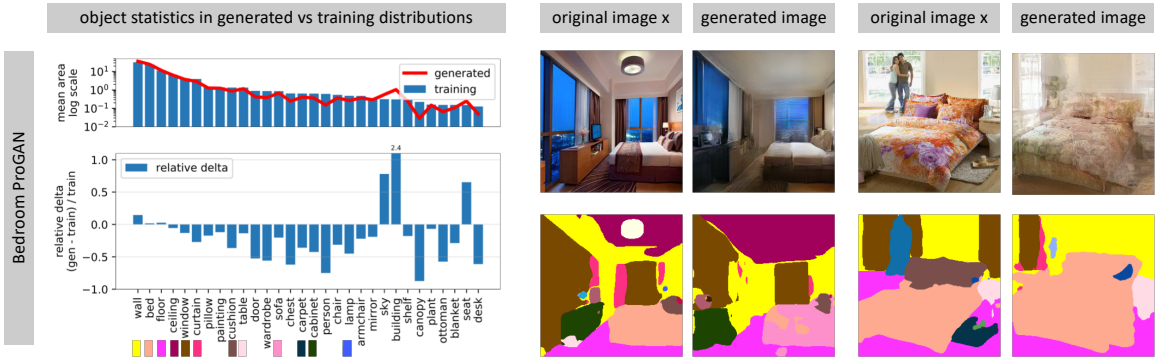


Figure 5-5: A visualization of the omissions of a bedroom generator; a Progressive GAN for LSUN bedrooms is tested. On top, a comparison of object distributions shows that many classes of objects are left out by the generator, including people, cushions, carpets, lamps, and several types of furniture. On the bottom, photographs are shown with their reconstructions $G(E(\mathbf{x}))$, along with segmentations. These examples directly reveal many object classes omitted by the bedroom generator.

5.4.1 Generated Image Segmentation Statistics

We first examine whether segmentation statistics correctly reflect the output quality of models across architectures. Figure 5-2 and Figure 5-5 show Generated Image Segmentation Statistics for WGAN-GP [Gulrajani et al., 2017], StyleGAN [Karras et al., 2019], and Progressive GAN [Karras et al., 2018] trained on LSUN bedrooms [Yu et al., 2015]. The histograms reveal that, for a variety of segmented object classes, StyleGAN matches the true distribution of those objects better than Progressive GAN, while WGAN-GP matches least closely. The differences can be summarized using Fréchet Segmentation Distance (Table 5.1), confirming that better models match the segmented statistics better overall.

Model	**FSD**
WGAN-GP Gulrajani et al. [2017] bedrooms (Figure 5-2)	428.4
ProGAN Karras et al. [2018] bedrooms (Figure 5-5)	85.2
StyleGAN Karras et al. [2019] bedrooms (Figure 5-2)	22.6

Table 5.1: FSD summarizes Generated Image Segmentation Statistics.

5.4.2 Sensitivity test

Figure 5-4 illustrates the sensitivity of measuring Generated Image Segmentation Statistics over a finite sample of 10,000 images. Instead of comparing a GAN to the true distribution, we compare two different randomly chosen subsamples of the LSUN bedroom data set to each other. A perfect test with infinite sample sizes would show no difference; the small differences shown reflect the sensitivity of the test and are due to sampling error.

5.4.3 Identifying dropped modes

Figure 5-1 and Figure 5-5 show the results of applying our method to analyze the generated segmentation statistics for Progressive GAN models of churches and bedrooms. Both the histograms and the instance visualizations provide insight into the limitations of the generators.

The histograms reveal that the generators partially skip difficult subtasks. For example, neither model renders as many people as appear in the target distribution. We use inversion to create reconstructions of natural images that include many pixels of people or other under-represented objects. Figure 5-1 and Figure 5-5 each shows two examples on the bottom. Our inversion method reveals the way in which the models fail. The gaps are not due to low-quality rendering of those object classes, but due to the wholesale omission of these classes. For example, large human figures and certain classes of objects are not included.

5.4.4 Layer-wise inversion vs. other methods

We compare our layer-wise inversion method to several previous approaches; we also benchmark it against ablations of key components of the method.

The first three columns of Figure 5-6 compare our method to prior inversion methods. We test each method on a sample of 100 images produced by the generator G, where the ground truth $\mathbf{z}$ is known, and the reconstruction of an example image is shown. In this case an ideal inversion should be able to perfectly reconstruct $\mathbf{x}' = \mathbf{x}$. In addition, a reconstruction of a real input image is shown at the bottom. While there is no ground truth latent and representation for this image, the qualitative comparisons are informative.

(a) Direct optimization of z. Smaller generators such as 5-layer DCGAN Radford et al. [2016] can be inverted by applying gradient descent on $\mathbf{z}$ to minimize reconstruction loss [Zhu et al., 2016]. In column (a), we test this method on a 15-layer Progressive GAN and find that neither $\mathbf{z}$ nor $\mathbf{x}$ can be constructed accurately.

(b): Direct learning of E. Another natural solution Donahue et al. [2017], Zhu et al. [2016] is to learn a deep network E that inverts G directly, without the complexity of layer-wise decomposition. Here, we learn an inversion network with the same parameters and architecture as the network E used in our method, but train it end-to-end by directly minimizing expected reconstruction losses over generated images, rather than learning it by layers. The method does benefit from the power of a deep

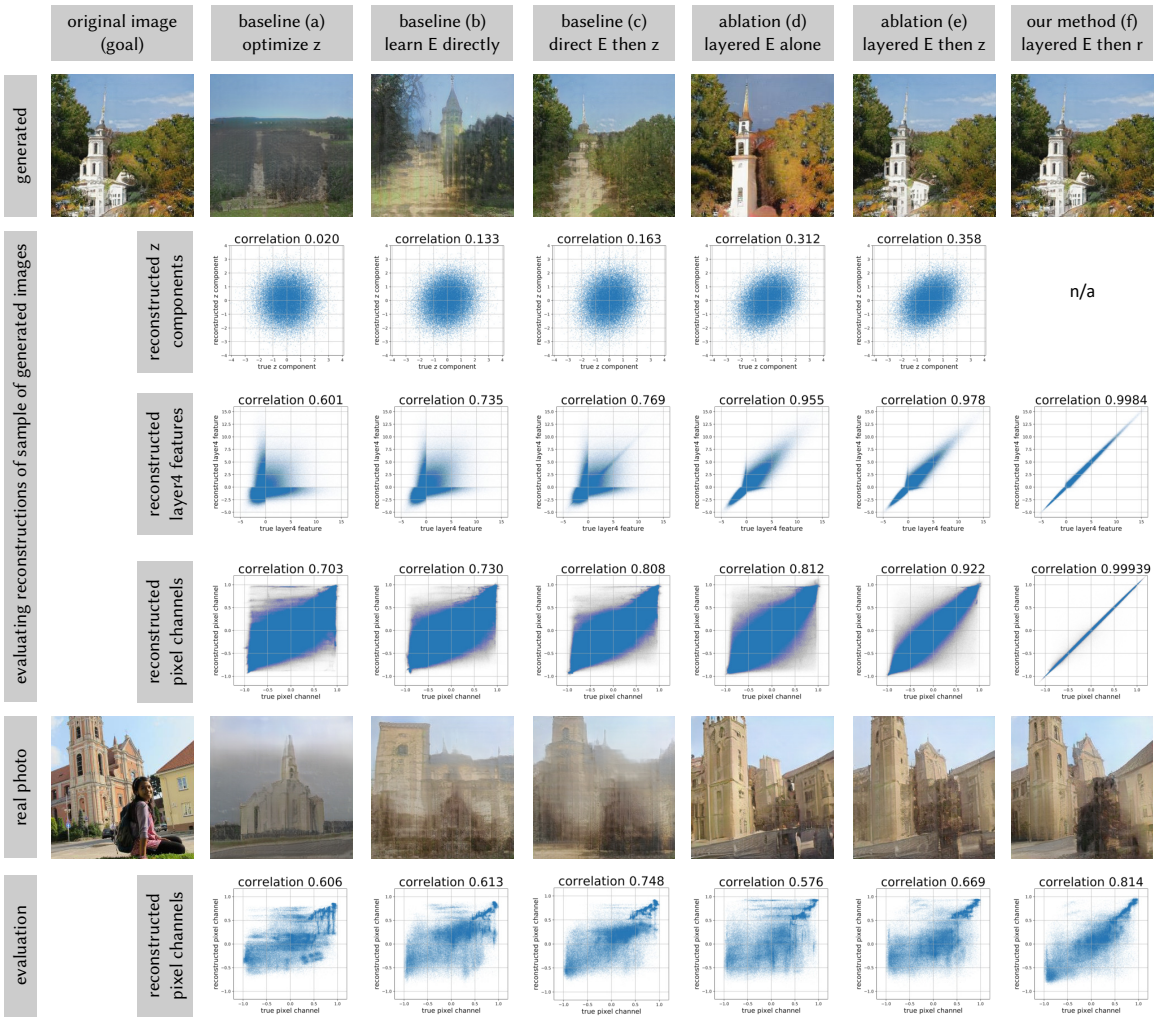


Figure 5-6: Comparison of methods to invert the generator of Progressive GAN trained to generate LSUN church images. Each method is described; (a) (b) and (c) are baselines, and (d), (e), and (f) are variants of our method. The first four rows show behavior given GAN-generated images as input. In the scatter plots, every point plots a reconstructed component versus its true value, with a point for every RGB pixel channel or every dimension of a representation. Reconstruction accuracy is shown as mean correlation over all dimensions for **z**, `layer4`, and image pixels, based on a sample of 100 images. Our method (f) achieves nearly perfect reconstructions of GAN-generated images. In the bottom rows, we apply each of the methods on a natural image.

network to learn generalized rules Gershman and Goodman [2014], and the results are marginally better than the direct optimization of $\mathbf{z}$. However, both qualitative and quantitative results remain poor.

(c): Optimization of z after initializing with $E(\mathbf{x})$. This is the full method used in Zhu et al. [2016]. By initializing method (a) using an initial guess from method (b), results can be improved slightly. For smaller generators, this method performs better than method (a) and (b). However, when applied to a Progressive GAN, the reconstructions are far from satisfactory.

Ablation experiments. The last three columns of Figure 5-6 compare our full method (f) to two ablations of our method.

(d): Layer-wise network inversion only. We can simply use the layer-wise-trained inversion network E as the full inverse, and simply use the initial guess $\mathbf{z}_0 = E(\mathbf{x})$, setting $\mathbf{x}' = G(\mathbf{z}_0)$. This fast method requires only a single forward pass through the inverter network E. The results are better than the baseline methods but far short of our full method.

Nevertheless, despite the inaccuracy of the latent code $\mathbf{z}_0$, the intermediate layer features are highly correlated with their true values; this method achieves 95.5% correlation versus the true $\mathbf{r}_4$. Furthermore, the qualitative results show that when reconstructing real images, this method obtains more realistic results despite being noticeably different from the target image.

(e): Inverting G without relaxation to G_f. We can improve the initial guess $\mathbf{z}_0 = E(\mathbf{x})$ by directly optimizing $\mathbf{z}$ to minimize the same image reconstruction loss. This marginally improves upon $\mathbf{z}_0$. However, the reconstructed images and the input images still differ signficantly, and the recovery of $\mathbf{z}$ remains poor. Although the qualitative results are good, the remaining error means that we cannot know if any reconstruction errors are due to failures of G to generate an image, or if those reconstruction errors are merely due to the inaccuracy of the inversion method.

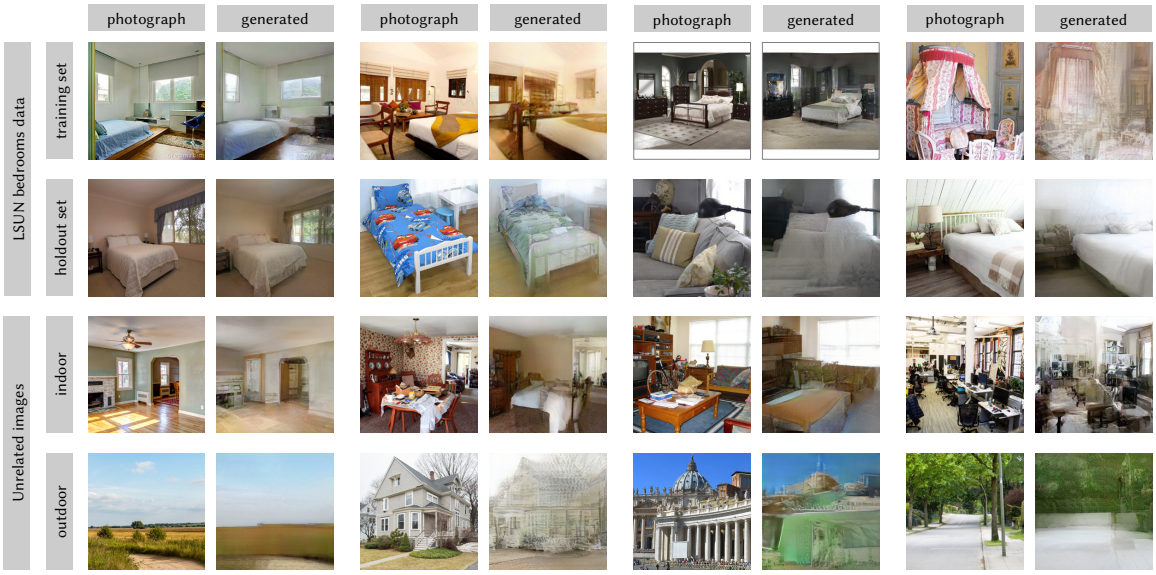


Figure 5-7: Inverting layers of a Progressive GAN bedroom generator. From top to bottom: uncurated reconstructions of photographs from the LSUN training set, the holdout set, and unrelated (non-bedroom) photographs, both indoor and outdoor.

(f): Our full method. By relaxing the problem and regularizing optimization of $\mathbf{r}$ rather than $\mathbf{z}$, our method achieves nearly perfect reconstructions of both intermediate representations and pixels. Denote the full method as $\mathbf{r}^* = E_f(\mathbf{x})$.

The high precision of E_f within the range of G means that, when we observe large differences between $\mathbf{x}$ and $G_f(E_f(\mathbf{x}))$, they are unlikely to be a failure of E_f. This indicates that G_f cannot render $\mathbf{x}$, which means that G cannot either. Thus our ability to solve the relaxed inversion problem with an accuracy above 99% gives us a reliable tool to visualize samples that reveal what G cannot do.

Note that the purpose of E_f is to show dropped modes, not positive capabilities. The range of G_f upper-bounds the range of G, so the reconstruction $G_f(E_f(\mathbf{x}))$ could be better than what the full network G is capable of. For a more complete picture, methods (d) and (e) can be additionally used as lower-bounds: those methods do not prove images are outside G's range, but they can reveal positive capabilities of G because they construct generated samples in range(G).

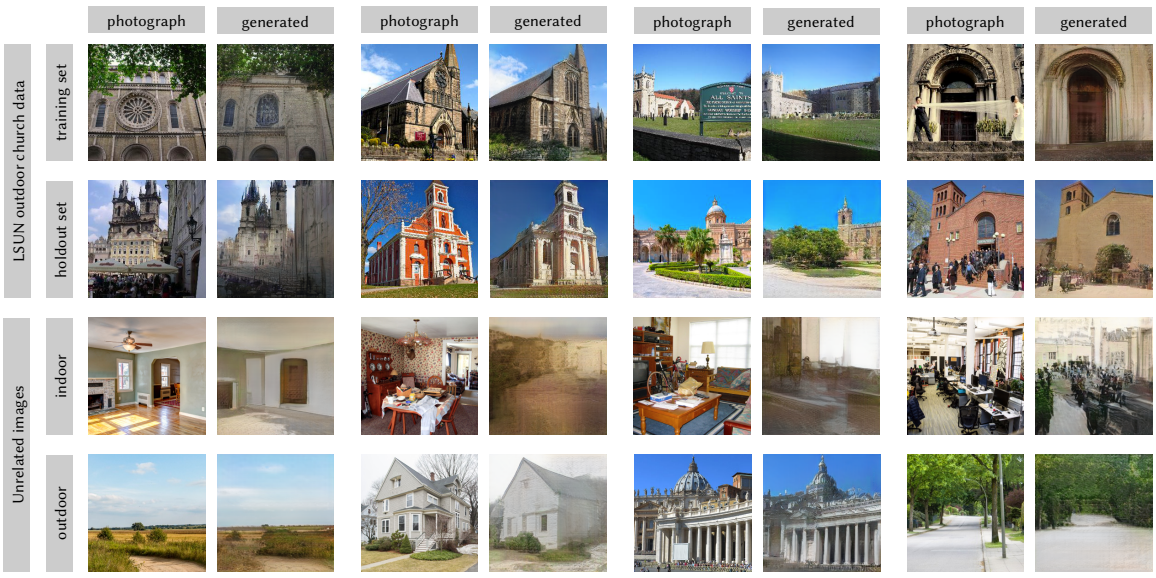


Figure 5-8: Inverting layers of a Progressive GAN outdoor church generator. From top to bottom: uncurated reconstructions of photographs from the LSUN training set, the holdout set, and unrelated (non-church) photographs, both indoor and outdoor.

5.4.5 Layer-wise inversion across domains

Next, we apply the inversion tool to test the ability of generators to synthesize images outside their training sets. Figure 5-7 shows qualitative results of applying method (f) to invert and reconstruct natural photographs of different scenes using a Progressive GAN trained to generate LSUN bedrooms. Reconstructions from the LSUN training and LSUN holdout sets are shown; these are compared to newly collected unrelated (non-bedroom) images taken both indoors and outdoors. Objects that disappear from the reconstructions reveal visual concepts that cannot be represented by the model. Some indoor non-bedroom images are rendered in a bedroom style: for example, a dining room table with a white tablecloth is rendered to resemble a bed with a white bed sheet. As expected, outdoor images are not reconstructed well.

Figure 5-8 shows similar qualitative results using a Progressive GAN for LSUN outdoor church images. Interestingly, some architectural styles are dropped even in cases where large-scale geometry is preserved. The same set of unrelated (non-church) images as shown in Figure 5-7 are shown. When using the church model, the indoor reconstructions exhibit lower quality and are rendered to resemble outdoor scenes; the reconstructions of outdoor images recover more details.

5.5 Discussion

We have proposed a way to measure and visualize mode-dropping in state-of-the-art generative models. Generated Image Segmentation Statistics can compare the quality of different models and architectures, and provide insights into the semantic differences of their output spaces. Layer inversions allow us to further probe the range of the generators using natural photographs, revealing specific objects and styles that cannot be represented. By comparing labeled distributions with one another, and by comparing natural photos with imperfect reconstructions, we can identify specific objects, parts, and styles that a generator cannot produce.

The methods we propose here constitute a first step towards analyzing and understanding the latent space of a GAN and point to further questions. Why does a GAN decide to ignore classes that are more frequent than others in the target distribution (e.g. "person" vs. "fountain" in Figure 5-1)? How can we encourage a GAN to learn about a concept without skewing the training set? What is the impact of architectural choices? Finding ways to exploit and address the mode-dropping phenomena identified by our methods are questions for future work.

Bibliography

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

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *CVPR*, 2017.

David Bau, Hendrik Strobelt, William Peebles, Jonas Wulff, Bolei Zhou, Jun-Yan Zhu, and Antonio Torralba. Semantic photo manipulation with a generative image prior. *SIGGRAPH*, 2019a.

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Zhou Bolei, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. Gan dissection: Visualizing and understanding generative adversarial networks. In *ICLR*, 2019b.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *NeurIPS*, 2007.

Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. In *ICLR*, 2017.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *ICLR*, 2019.

Antonia Creswell and Anil Anthony Bharath. Inverting the generator of a generative adversarial network. *IEEE transactions on neural networks and learning systems*, 2018.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *ICLR*, 2017.

Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *CVPR*, 2016.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. In *ICLR*, 2017.

Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, 2014.

Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *ECCV*, 2018.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. *arXiv preprint arXiv:1711.00867*, 2017.

Ke Li and Jitendra Malik. On the implicit assumptions of gans. *arXiv preprint arXiv:1811.12402*, 2018.

Zachary C Lipton and Subarna Tripathi. Precise recovery of latent vectors from generative adversarial networks. *arXiv preprint arXiv:1702.04782*, 2017.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NeurIPS*, 2017.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

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

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.

Irad Peleg and Lior Wolf. Structured gans. pages 719–728. IEEE, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *ICML*, 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *NeurIPS*, 2016.

Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. In *CVPR*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

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

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

Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. In *ICLR*, 2017.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *ICLR*, 2016.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *NeurIPS*, 2018.

Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *CVPR*, 2017.

Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models. In *ICLR*, 2017a.

Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models. In *ICLR*, 2017b.

Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, 2018.

Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *CVPR*, 2018.

Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *CVPR*, 2017.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017.

Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NeurIPS*, 2014.

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, 2016.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

Chapter 6

Rewriting a Deep Generative Model

David Bau, Steven Liu, Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba. ECCV 2020.

6.1 Introduction

We present the task of *model rewriting*, which aims to add, remove, and alter the semantic and physical rules of a pretrained deep network. While modern image editing tools achieve a user-specified goal by manipulating individual input images, we enable a user to synthesize an unbounded number of new images by editing a generative model to carry out modified rules.

For example in Figure 6-1, we apply a succession of rule changes to edit a StyleGANv2 model Karras et al. [2020] pretrained on LSUN church scenes Yu et al. [2015]. The first change removes watermark text patterns (a); the second adds crowds of people in front of buildings (b); the third replaces the rule for drawing tower tops with a rule that draws treetops (c), creating a fantastical effect of trees growing from towers. Because each of these modifications changes the generative model, every single change affects a whole category of images, removing all watermarks synthesized by the model, arranging people in front of many kinds of buildings, and creating tree-towers everywhere. The images shown are samples from an endless distribution.

But why is rewriting a deep generative model useful? A generative model enforces

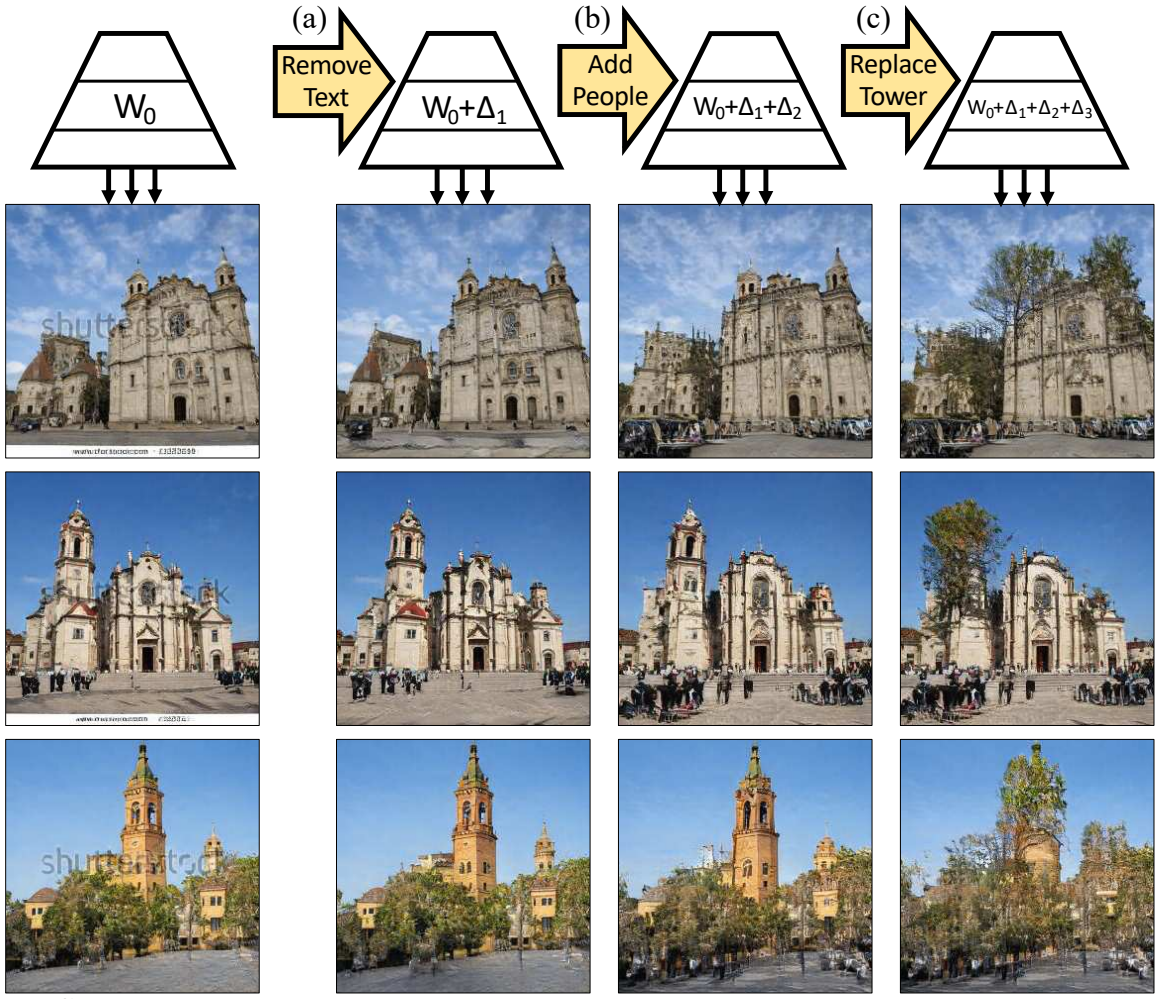


Figure 6-1: Rewriting the weights of a generator to change generative rules. Rules can be changed to (a) *remove* patterns such as watermarks; (b) *add* objects such as people; or (c) *replace* definitions such as making trees grow out of towers. Instead of editing individual images, our method edits the generator, so an infinite set of images can be potentially synthesized and manipulated using the altered rules.

many rules and relationships within the generated images Bau et al. [2019b], Jahanian et al. [2020]. From a purely scientific perspective, the ability to edit such a model provides insights about what the model has captured and how the model can generalize to unseen scenarios. At a practical level, deep generative models are increasingly useful for image and video synthesis Mathieu et al. [2016], Zhu et al. [2017], Isola et al. [2017], Chan et al. [2019]. In the future, entire image collections, videos, or virtual worlds could potentially be produced by deep networks, and editing individual images or frames will be needlessly tedious. Instead, we would like to provide authoring tools for modifying the models themselves. With this capacity, a set of similar edits could be *transferred* to many images at once.

A key question is how to edit a deep generative model. The computer vision community has become accustomed to training models using large data sets and expensive human annotations, but we wish to enable novice users to easily modify and customize a deep generative model *without* the training time, domain expertise, and computational cost of large-scale machine learning. In this paper, we present a new method that can locate and change a specific semantic relationship within a model. In particular, we show how to generalize the idea of a *linear associative memory* Kohonen and Ruohonen [1973] to a nonlinear convolutional layer of a deep generator. Each layer stores latent rules as a set of key-value relationships over hidden features. Our constrained optimization aims to add or edit one specific rule within the associative memory while preserving the existing semantic relationships in the model as much as possible. We achieve it by directly measuring and manipulating the model's internal structure, without requiring any new training data.

We use our method to create several visual editing effects, including the addition of new arrangements of objects in a scene, systematic removal of undesired output patterns, and global changes in the modeling of physical light. Our method is simple and fast, and it does not require a large set of annotations: a user can alter a learned rule by providing a single example of the new rule or a small handful of examples. We demonstrate a user interface for novice users to modify specific rules encoded in the layers of a GAN interactively. Finally, our quantitative experiments

on several datasets demonstrate that our method outperforms several fine-tuning baselines as well as image-based edit transfer methods, regarding both photorealism and desirable effects. Our code, data, and user interface are available at our website, `https://rewriting.csail.mit.edu`.

6.2 Related work

Deep image manipulation. Image manipulation is a classic problem in computer vision, image processing, and computer graphics. Common operations include color transfer Reinhard et al. [2001], Levin et al. [2004], image deformation Schaefer et al. [2006], Wolberg [1990], object cloning Pérez et al. [2003], Burt and Adelson [1983], and patch-based image synthesis Efros and Freeman [2001], Barnes et al. [2009], Hertzmann et al. [2001]. Recently, thanks to rapid advances of deep generative models Goodfellow et al. [2014], Kingma and Welling [2014], Hinton et al. [2006], learning-based image synthesis and editing methods have become widely-used tools in the community, enabling applications such as manipulating the semantics of an input scene Park et al. [2019], Bau et al. [2019a], Suzuki et al. [2018], Collins et al. [2020], image colorization Zhang et al. [2016], Iizuka et al. [2016], Larsson et al. [2016], Zhang et al. [2017], photo stylization Gatys et al. [2016], Johnson et al. [2016], Luan et al. [2017], Liao et al. [2017], image-to-image translation Isola et al. [2017], Zhu et al. [2017], Bousmalis et al. [2017], Taigman et al. [2017], Liu et al. [2017], Huang et al. [2018], and face editing and synthesis Fried et al. [2019], Nagano et al. [2018], Portenier et al. [2018]. While our user interface is inspired by previous interactive systems, our goal is *not* to manipulate and synthesize a single image using deep models. Instead, our work aims to manipulate the structural rules of the model itself, creating an altered deep network that can produce countless new images following the modified rules.

Edit transfer and propagation. Edit transfer methods propagate pixel edits to corresponding locations in other images of the same object or adjacent frames in the same video An and Pellacini [2008], Xu et al. [2009], Hasinoff et al. [2010], Chen et al. [2012, 2014], Yücer et al. [2012], Endo et al. [2016]. These methods achieve impressive

results but are limited in two ways. First, they can only transfer edits to images of the same instance, as image alignment between different instances is challenging. Second, the edits are often restricted to color transfer or object cloning. In contrast, our method can change context-sensitive rules that go beyond pixel correspondences (Section 6.5.3). In Section 6.5.1, we compare to an edit propagation method based on state-of-the-art alignment algorithm, Neural Best-Buddies Aberman et al. [2018].

Interactive machine learning systems aim to improve training through human interaction in labeling Cohn et al. [1994], Fails and Olsen Jr [2003], Settles and Craven [2008], or by allowing a user to to aid in the model optimization process via interactive feature selection Dy and Brodley [2000], Guo [2003], Raghavan et al. [2006], Krause et al. [2014] or model and hyperparameter selection Kapoor et al. [2010], Patel et al. [2011], Jiang and Canny [2017]. Our work differs from these previous approaches because rather than asking for human help to attain a fixed objective, we enable a user to solve novel creative modeling tasks, given a pre-trained model. Model rewriting allows a user to create a network with new rules that go beyond the patterns present in the training data.

Transfer learning and model fine-tuning. Transfer learning adapts a learned model to unseen learning tasks, domains, and settings. Examples include domain adaptation Saenko et al. [2010], zero-shot or few-shot learning Socher et al. [2013], Lake et al. [2015], model pre-training and feature learning Donahue et al. [2014], Zeiler and Fergus [2014], Yosinski et al. [2014], and meta-learning Bengio et al. [1992], Andrychowicz et al. [2016], Finn et al. [2017]. Our work differs because instead of extending the training process with more data or annotations, we enable the user to directly change the behavior of the existing model through a visual interface. Recently, several methods Ulyanov et al. [2018], Shocher et al. [2018], Bau et al. [2019a] propose to train or fine-tune an image generation model to a particular image for editing and enhancement applications. Our goal is different, as we aim to identify and change rules that can generalize to many different images instead of one.

6.3 Method

To rewrite the rules of a trained generative model, we allow users to specify a handful of model outputs that they wish to behave differently. Based on this objective, we optimize an update in model weights that generalizes the requested change. In Section 6.3, we derive and discuss this optimization. In Section 6.4, we present the user interface that allows the user to interactively define the objective and edit the model.

Section 6.3.1 formulates our objective on how to add or modify a specific rule while preserving existing rules. We then consider this objective for linear systems and connect it to a classic technique—associative memory Kohonen [1972], Anderson [1972], Kohonen [2012] (Section 6.3.2); this perspective allows us to derive a simple update rule (Section 6.3.3). Finally, we apply the solution to the nonlinear case and derive our full algorithm (Section 6.3.4).

6.3.1 Objective: Changing a rule with minimal collateral damage

Given a pre-trained generator $G(z;\theta_0)$ with weights θ_0, we can synthesize multiple images $x_i = G(z_i;\theta_0)$, where each image is produced by a latent code z_i. Suppose we have manually created desired changes x_{*i} for those cases. We would like to find updated weights θ_1 that change a computational rule to match our target examples $x_{*i} \approx G(z_i;\theta_1)$, while minimizing interference with other behavior:

$$\theta_1 = \arg\min_{\theta} \mathcal{L}_{\mathsf{smooth}}(\theta) + \lambda \mathcal{L}_{\mathsf{constraint}}(\theta), \tag{6.1}$$

$$\mathcal{L}_{\mathsf{smooth}}(\theta) \triangleq \mathbb{E}_z \left[\ell(G(z;\theta_0), G(z;\theta)) \right], \tag{6.2}$$

$$\mathcal{L}_{\mathsf{constraint}}(\theta) \triangleq \sum_i \ell(x_{*i}, G(z_i;\theta)). \tag{6.3}$$

A traditional solution to the above problem is to jointly optimize the weighted sum of $\mathcal{L}_{\mathsf{smooth}}$ and $\mathcal{L}_{\mathsf{constraint}}$ over θ, where $\ell(\cdot)$ is a distance metric that measures the perceptual distance between images Johnson et al. [2016], Dosovitskiy and Brox [2016], Zhang et al. [2018]. Unfortunately, this standard approach does not produce a

generalized rule within G, because the large number of parameters θ allow the generator to quickly overfit the appearance of the new examples without good generalization; we evaluate this approach in Section 6.5.

However, the idea becomes effective with two modifications: (1) instead of modifying all of θ, we reduce the degrees of freedom by modifying weights W at only one layer, and (2) for the objective function, we directly minimize distance in the output feature space of that same layer.

Given a layer L, we use k to denote the features computed by the first $L-1$ fixed layers of G, and then write $v = f(k; W_0)$ to denote the computation of layer L itself, with pretrained weights W_0. For each exemplar latent z_i, these layers produce features k_{*i} and $\boldsymbol{i} = f(k_{*i}; W_0)$. Now suppose, for each target example x_{*i}, the user has manually created a feature change v_{*i}. (A user interface to create target feature goals is discussed in Section 6.4.) Our objective becomes:

$$W_1 = \arg\min_W \mathcal{L}_{\mathsf{smooth}}(W) + \lambda \mathcal{L}_{\mathsf{constraint}}(W), \tag{6.4}$$

$$\mathcal{L}_{\mathsf{smooth}}(W) \triangleq \mathbb{E}_k \left[\, ||f(k; W_0) - f(k; W)||^2 \, \right], \tag{6.5}$$

$$\mathcal{L}_{\mathsf{constraint}}(W) \triangleq \sum_i ||v_{*i} - f(k_{*i}; W)||^2, \tag{6.6}$$

where $||\cdot||^2$ denotes the L2 loss. Even within one layer, the weights W contain many parameters. But the degrees of freedom can be further reduced to constrain the change to a specific direction that we will derive; this additional directional constraint will allow us to create a generalized change from a single (k_*, v_*) example. To understand the constraint, it is helpful to interpret a single convolutional layer as an associative memory, a classic idea that we briefly review next.

6.3.2 Viewing a convolutional layer as an associative memory

Any matrix W can be used as an associative memory Kohonen [2012] that stores a set of key-value pairs $\{(k_i, \boldsymbol{i})\}$ that can be retrieved by matrix multiplication:

$$\boldsymbol{i} \approx W k_i. \tag{6.7}$$

The use of a matrix as a *linear associative memory* is a foundational idea in neural networks Kohonen [1972], Anderson [1972], Kohonen [2012]. For example, if the keys $\{k_i\}$ form a set of mutually orthogonal unit-norm vectors, then an error-free memory can be created as

$$W_{\text{orth}} \triangleq \sum_i \boldsymbol{i} {k_i}^T. \tag{6.8}$$

Since ${k_i}^T k_j = 0$ whenever $i \neq j$, all the irrelevant terms cancel when multiplying by k_j, and we have $W_{\text{orth}}\ k_j = v_j$. A new value can be stored by adding $v_* {k_*}^T$ to the matrix as long as k_* is chosen to be orthogonal to all the previous keys. This process can be used to store up to N associations in an $M \times N$ matrix.

Figure 6-2 views the weights of one convolutional layer in a generator as an associative memory. Instead of thinking of the layer as a collection of convolutional filtering operations, we can think of the layer as a memory that associates keys to values. Here each key k is a single-location feature vector. The key is useful because, in our trained generator, the same key will match many semantically similar locations across different images, as shown in Figure 6-2c. Associated with each key, the map stores an output value v that will render an arrangement of output shapes. This output can be visualized directly by rendering the features in isolation from neighboring locations, as shown in Figure 6-2d.

For example, consider a layer that transforms a 512-channel featuremap into a 256-channel featuremap using a 3×3 convolutional kernel; the weights form a 256×512×3×3 tensor. For each key $k \in \mathbb{R}^{512}$, our layer will recall a value $v \in \mathbb{R}^{256\times3\times3} = \mathbb{R}^{2304}$ representing a 3×3 output pattern of 256-channel features, flattened to a vector, as

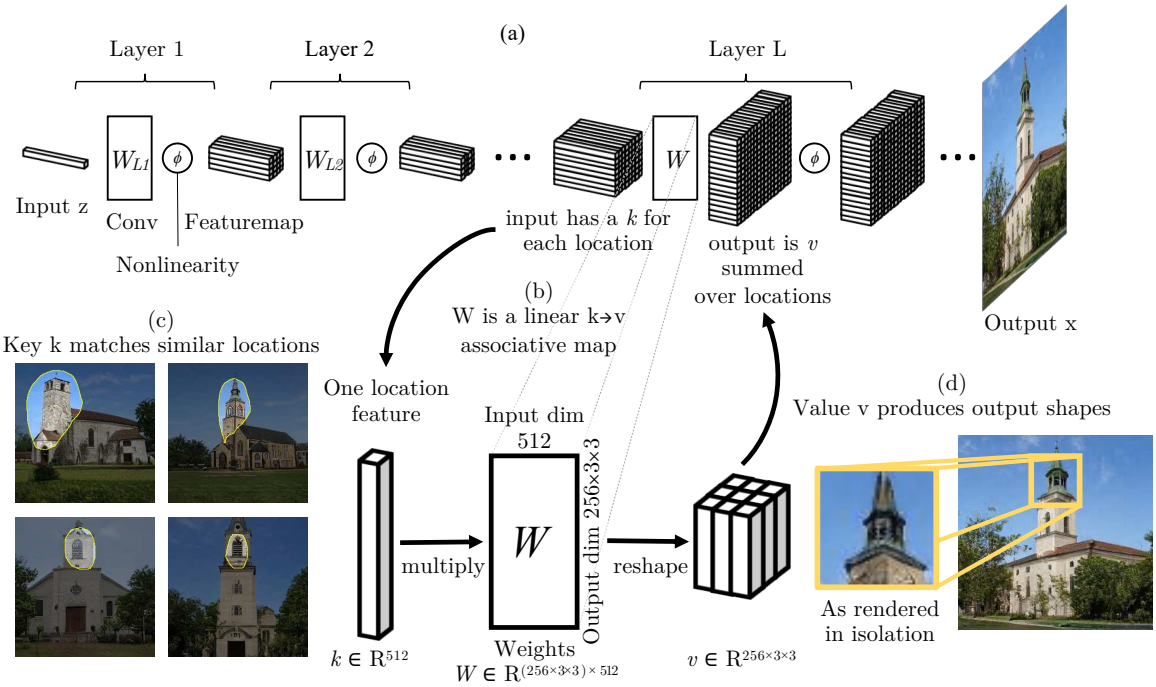


Figure 6-2: (a) A generator consists of a sequence of layers; we focus on one particular layer L. (b) The convolutional weights W serve an associative memory, mapping keys k to values v. The keys are single-location input features, and the values are patterns of output features. (c) A key will tend to match semantically similar contexts in different images. Shown are locations of generated images that have features that match a specific k closely. (d) A value renders shapes in a small region. Here the effect of a value v is visualized by rendering features at one location alone, with features at other locations set to zero. Image examples are taken from a StyleGANv2 model trained on LSUN outdoor church scenes.

$v = Wk$. Our interpretation of the layer as an associative memory does not change the computation: the tensor is simply reshaped and treated as a dense rectangular matrix $W \in \mathbb{R}^{(256\times3\times3)\times512}$, whose job is to map keys $k \in \mathbb{R}^{512}$ to values $v \in \mathbb{R}^{2304}$, via Eqn. 6.7.

Arbitrary Nonorthogonal Keys. In classic work, Kohonen Kohonen and Ruohonen [1973] observed that an associative memory can support more than N nonorthogonal keys $\{k_i\}$ if instead of requiring exact equality $\boldsymbol{i} = Wk_i$, we choose W_0 to minimize error:

$$W_0 \triangleq \arg\min_W \sum_i ||\boldsymbol{i} - Wk_i||^2. \tag{6.9}$$

To simplify notation, let us assume a finite set of pairs $\{(k_i, \boldsymbol{i})\}$ and collect keys and

values into matrices K and V whose i-th column is the i-th key or value:

$$K \triangleq [k_1|k_2|\cdots|k_i|\cdots], \tag{6.10}$$

$$V \triangleq [v_1|v_2|\cdots|v_i|\cdots]. \tag{6.11}$$

The minimization (Eqn. 6.9) is the standard linear least-squares problem. A unique minimal solution can be found by solving for W_0 using the normal equation $W_0 K K^T = V K^T$, or equivalently by using the pseudoinverse $W_0 = V K^+$.

6.3.3 Updating W to insert a new value

Now, departing from Kohonen Kohonen and Ruohonen [1973], we ask how to modify W_0. Suppose we wish to overwrite a single key to assign a new value $k_* \to v_*$ provided by the user. After this modification, our new matrix W_1 should satisfy two conditions:

$$W_1 = \arg\min_W ||V - WK||^2, \tag{6.12}$$

$$\text{subject to } v_* = W_1 k_*. \tag{6.13}$$

That is, it should store the new value; and it should continue to minimize error in all the previously stored values. This forms a constrained linear least-squares (CLS) problem which can be solved exactly as $W_1 K K^T = V K^T + \Lambda\, {k_*}^T$, where the vector $\Lambda \in \mathbb{R}^m$ is determined by solving the linear system with the constraint in Eqn. 6.13 (see Appendix B.1). Because W_0 satisfies the normal equations, we can expand $V K^T$ in the CLS solution and simplify:

$$W_1 K K^T = W_0 K K^T + \Lambda\, {k_*}^T \tag{6.14}$$

$$W_1 = W_0 + \Lambda (C^{-1} k_*)^T \tag{6.15}$$

Above, we have written $C \triangleq K K^T$ as the second moment statistics. (C is symmetric; if K has zero mean, C is the covariance.) The statistics C can be estimated beforehand by averaging kk^T on a sample of generated features; these can be gathered ahead of

time. Now Eqn. 6.15 has a simple form. Since $\Lambda \in \mathbb{R}^m$ and $(C^{-1}k_*)^T \in \mathbb{R}^n$ are simple vectors, the update $\Lambda(C^{-1}k_*)^T$ is a rank-one matrix with rows all multiples of the vector $(C^{-1}k_*)^T$.

Eqn. 6.15 is interesting for two reasons. First, it shows that enforcing the user's requested mapping $k_* \to v_*$ transforms the soft error minimization objective (6.12) into the hard constraint that the weights be updated in a particular straight-line direction $C^{-1}k_*$. Second, it reveals that the update direction is determined only by the overall key statistics and the specific targeted key k_*. The covariance C is a model constant that can be pre-computed and cached, and the update direction is determined by the key *regardless of any stored value*. Only Λ, which specifies the magnitude of each row change, depends on the target value v_*.

6.3.4 Generalize to a nonlinear neural layer

In practice, even a single network block contains several non-linear components such as a biases, ReLU, normalization, and style modulation. Below, we generalize our procedure to the nonlinear case where the solution to W_1 cannot be calculated in a closed form. We first define our update direction:

$$d \triangleq C^{-1}k_*. \tag{6.16}$$

Then suppose we have a non-linear neural layer $f(k; W)$ which follows the linear operation W with additional nonlinear steps. Since the form of Eqn. 6.15 is sensitive to the rowspace of W and insensitive to the column space, we can use the same rank-one update form to constrain the optimization of $f(k_*; W) \approx v_*$.

Therefore, in our experiments, when we update a layer to insert a new key $k_* \to v_*$, we begin with the existing W_0, and we perform an optimization over the rank-one subspace defined by the row vector d^T from Eqn. 6.16. That is, in the nonlinear case,

we update W_1 by solving the following optimization:

$$\Lambda_1 = \underset{\Lambda \in \mathbb{R}^M}{\arg\min} ||v_* - f(k_*; W_0 + \Lambda\, d^T)||. \tag{6.17}$$

Once Λ_1 is computed, we update the weight as $W_1 = W_0 + \Lambda_1 d^T$.

Our desired insertion may correspond to a change of more than one key at once, particularly if our desired target output forms a feature map patch V_* larger than a single convolutional kernel, i.e., if we wish to have $V_* = f(K_*; W_1)$ where K_* and V_* cover many pixels. To alter S keys at once, we can define the allowable deltas as lying within the low-rank space spanned by the $N \times S$ matrix D_S containing multiple update directions $d_i = C^{-1}K_{*i}$, indicating which entries of the associative map we wish to change.

$$\Lambda_S = \underset{\Lambda \in \mathbb{R}^{M \times S}}{\arg\min} ||V_* - f(K_*; W_0 + \Lambda\, {D_S}^T)||, \tag{6.18}$$

$$\text{where } D_S \triangleq [d_1|d_2|\cdots|d_i|\cdots|d_S]\,. \tag{6.19}$$

We can then update the layer weights using $W_S = W_0 + \Lambda_S {D_S}^T$. The change can be made more specific by reducing the rank of D_S; details are discussed Appendix B.3. To directly connect this solution to our original objective (Eqn. 6.6), we note that the constrained optimization can be solved using projected gradient descent. That is, we relax Eqn. 6.18 and use optimization to minimize $\arg\min_W ||V_* - f(K_*; W)||$; then, to impose the constraint, after each optimization step, project W into into the subspace $W_0 + \Lambda_S {D_S}^T$.

6.4 User interface

To make model rewriting intuitive for a novice user, we build a user interface that provides a three-step rewriting process: *Copy*, *Paste*, and *Context*.

Copy and Paste allow the user to copy an object from one generated image to another. The user browses through a collection of generated images and highlights an

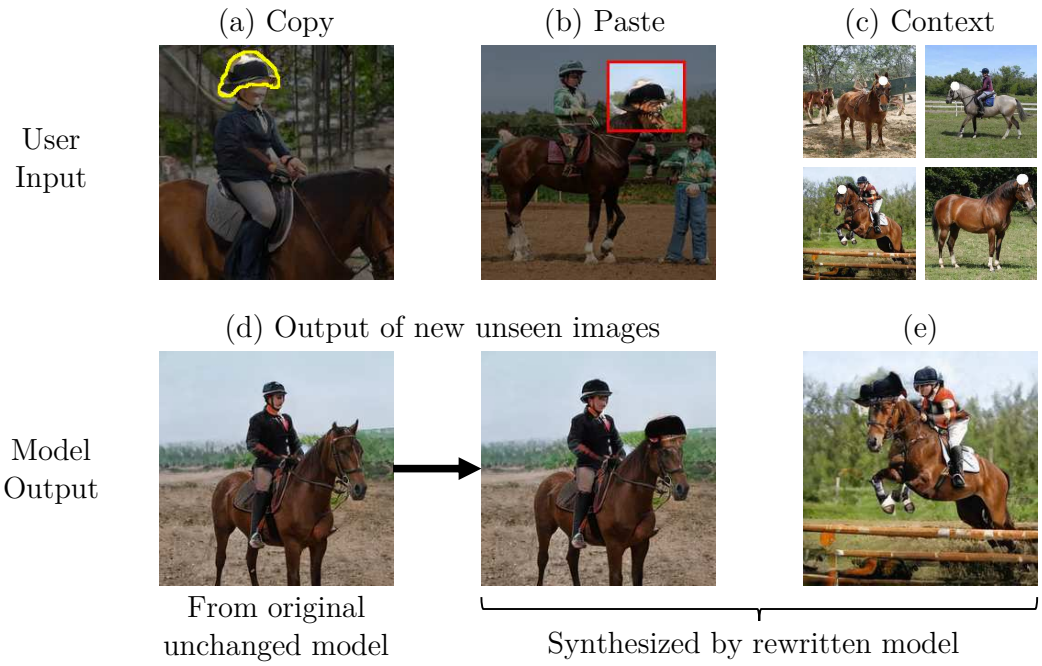


Figure 6-3: The *Copy-Paste-Context* interface for rewriting a model. (a) **Copy:** the user uses a brush to select a region containing an interesting object or shape, defining the target value V_*. (b) **Paste:** The user positions and pastes the copied object into a single target image. This specifies the $K_* \rightarrow V_*$ pair constraint. (c) **Context:** To control generalization, the user selects target regions in several images. This establishes the updated direction d for the associative memory. (d) The edit is applied to the model, not a specific image, so newly generated images will always have hats on top of horse heads. (e) The change has generalized to a variety of different types of horses and poses (see more in Appendix B).

area of interest to copy; then selects a generated target image and location for pasting the object. For example, in Figure 6-3a, the user selects a helmet worn by a rider and then pastes it in Figure 6-3b on a horse's head.

Our method downsamples the user's copied region to the resolution of layer L and gathers the copied features as the target value V_*. Because we wish to change not just one image, but the model rules themselves, we treat the pasted image as a new rule $K_* \rightarrow V_*$ associating the layer $L-1$ features K_* of the target image with the newly copied layer L values V_* that will govern the new appearance.

Context Selection allows a user to specify how this change will be generalized, by pointing out a handful of similar regions that should be changed. For example, in Figure 6-3b, the user has selected heads of different horses.

We collect the layer $L-1$ features at the location of the context selections as a set

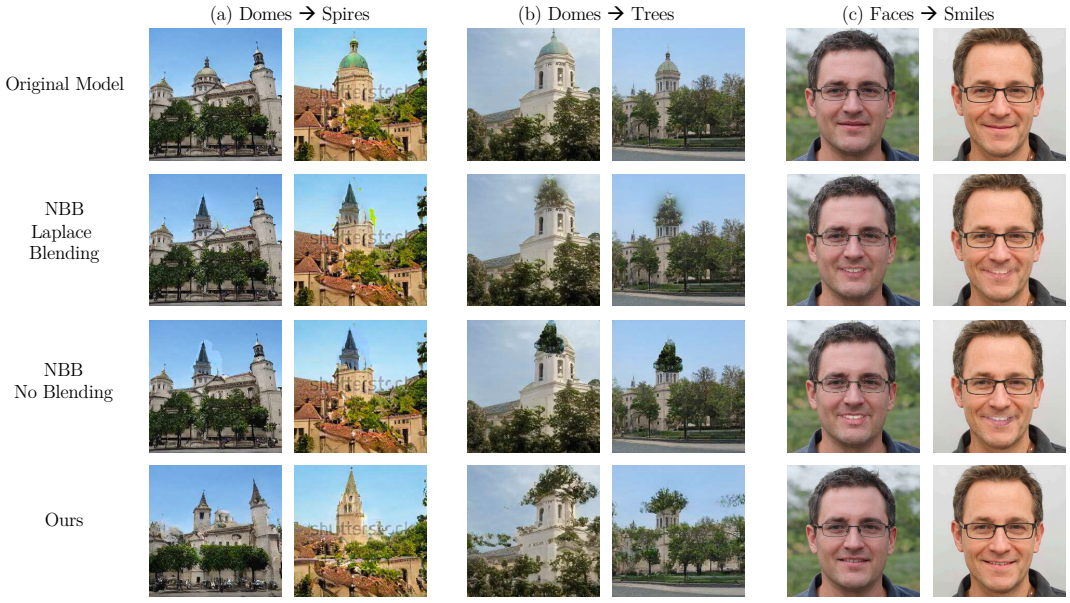


Figure 6-4: Adding and replacing objects in three different settings. (a) Replacing domes with an angular peaked spire causes peaked spires to be used throughout the model. (b) Replacing domes with trees can generate images unlike any seen in a training set. (c) Replacing closed lips with an open-mouth smile produces realistic open-mouth smiles. For each case, we show the images generated by an unchanged model, then the edit propagation results, with and without blending. Our method is shown in the last row.

of relevant K that are used to determine the weight update direction d via Eqn. 6.16. Generalization improves when we allow the user to select several context regions to specify the update direction (see Table 6.1); in Figure 6-3, the four examples are used to create a single d. Appendix B.3 discusses this rank reduction.

Applying one rule change on a StyleGANv2 model requires about eight seconds on a single Titan GTX GPU. Please check out the demo video of our interface.

6.5 Results

We test model rewriting with three editing effects. First, we add new objects into the model, comparing results to several baseline methods. Then, we use our technique to erase objects using a low-rank change; we test this method on the challenging watermark removal task. Finally, we invert a rule for a physical relationship between bright windows and reflections in a model.

	% smiling images ↑	LPIPS (masked) ↓	% more realistic than ours ↑
Our method (projected gradient descent)	84.37	**0.04**	–
With direct optimization of Λ	87.44	0.14	43.0
With single-image direction constraint	82.12	0.05	47.3
With single-layer, no direction constraint	90.94	0.30	6.8
Finetuning all weights	85.78	0.40	8.7
NBB + Direct copying	94.81	0.32	9.8
NBB + Laplace blending	**93.51**	0.32	8.6
Unmodified model	78.37	–	50.9

Table 6.1: Editing a StyleGANv2 Karras et al. [2020] FFHQ Karras et al. [2019] model to produce smiling faces in $n = 10,000$ images. To quantify the efficacy of the change, we show the percentage of smiling faces among the modified images, and we report the LPIPS distance on masked images to quantify undesired changes. For realism, workers make $n = 1,000$ pairwise judgements comparing images from other methods to ours.

	Dome → Spire			Dome → Tree	
	% `dome` pixels correctly modified ↑	LPIPS (masked) ↓	% more realistic than ours ↑	% `dome` pixels correctly modified ↑	LPIPS (masked) ↓
Our method (projected gradient descent)	**92.03**	**0.02**	–	48.65	**0.03**
With direct optimization of Λ	80.03	0.10	53.7	**59.43**	0.13
With single-image direction constraint	90.14	0.04	48.8	39.72	0.03
With single-layer, no direction constraint	80.69	0.29	38.1	41.32	0.45
Finetuning all weights	41.16	0.36	27.1	10.16	0.31
NBB + Direct copying	69.99	0.08	8.9	46.44	0.09
NBB + Laplace blending	69.63	0.08	12.2	31.18	0.09
Unmodified model	–	–	63.8	–	–

Table 6.2: We edit a StyleGANv2 Karras et al. [2020] LSUN church Yu et al. [2015] model to replace domes with spires/trees in $n = 10,000$ images. To quantify efficacy, we show the percentage of `dome` category pixels changed to the target category, determined by a segmenter Xiao et al. [2018]. To quantify undesired changes, we report LPIPS distance between edited and unchanged images, in non-dome regions. For realism, workers make $n = 1,000$ pairwise judgements comparing images from other methods to ours.

6.5.1 Putting objects into a new context

Here we test our method on several specific model modifications. In a church generator, the model edits change the shape of domes to spires, and change the domes to trees, and in a face generator, we add open-mouth smiles. Examples of all the edits are shown in Figure 6-4.

Quantitative Evaluation. In Tables 6.1 and 6.2,we compare the results to several baselines. We compare our method to the naive approach of fine-tuning all weights according to Eqn. 6.3, as well as the method of optimizing all the weights of a layer without constraining the direction of the change, as in Eqn. 6.6, and to a state-of-the-art image alignment algorithm, *Neural Best-Buddies* (NBB Aberman et al. [2018]),

Count of visible watermarks	middle	bottom
Zeroing 30 units (GAN Dissection)	**0**	6
Zeroing 60 units (GAN Dissection)	**0**	4
Rank-1 update (our method)	**0**	**0**
Unmodified model	64	26

Table 6.3: Visible watermark text produced by StyleGANv2 church model in $n = 1000$ images, without modification, with sets of units zeroed (using the method of GAN Dissection), and using our method to apply a rank-one update.

which is used to propagate an edit across a set of similar images by compositing pixels according to identified sparse correspondences. To transfer an edit from a target image, we use NBB and Moving Least Squares Schaefer et al. [2006] to compute a dense correspondence between the source image we would like to edit and the original target image. We use this dense correspondence field to warp the masked target into the source image. We test both direct copying and Laplace blending.

For each setting, we measure the efficacy of the edits on a sample of 10, 000 generated images, and we also quantify the undesired changes made by each method. For the smiling edit, we measure efficacy by counting images classified as smiling by an attribute classifier Sharma and Foroosh [2020], and we also quantify changes made in the images outside the mouth region by masking lips using a face segmentation model ZLL [2019] and using LPIPS Zhang et al. [2018] to quantify changes. For the dome edits, we measure how many dome pixels are judged to be changed to non-domes by a segmentation model Xiao et al. [2018], and we measure undesired changes outside dome areas using LPIPS. We also conduct a user study where users are asked to compare the realism of our edited output to the same image edited using baseline methods. We find that our method produces more realistic outputs that are more narrowly targeted than the baseline methods. For the smile edit, our method is not as aggressive as baseline methods at introducing smiles, but for the dome edits, our method is more effective than baseline methods at executing the change. Our metrics are further discussed in Appendix B.2.

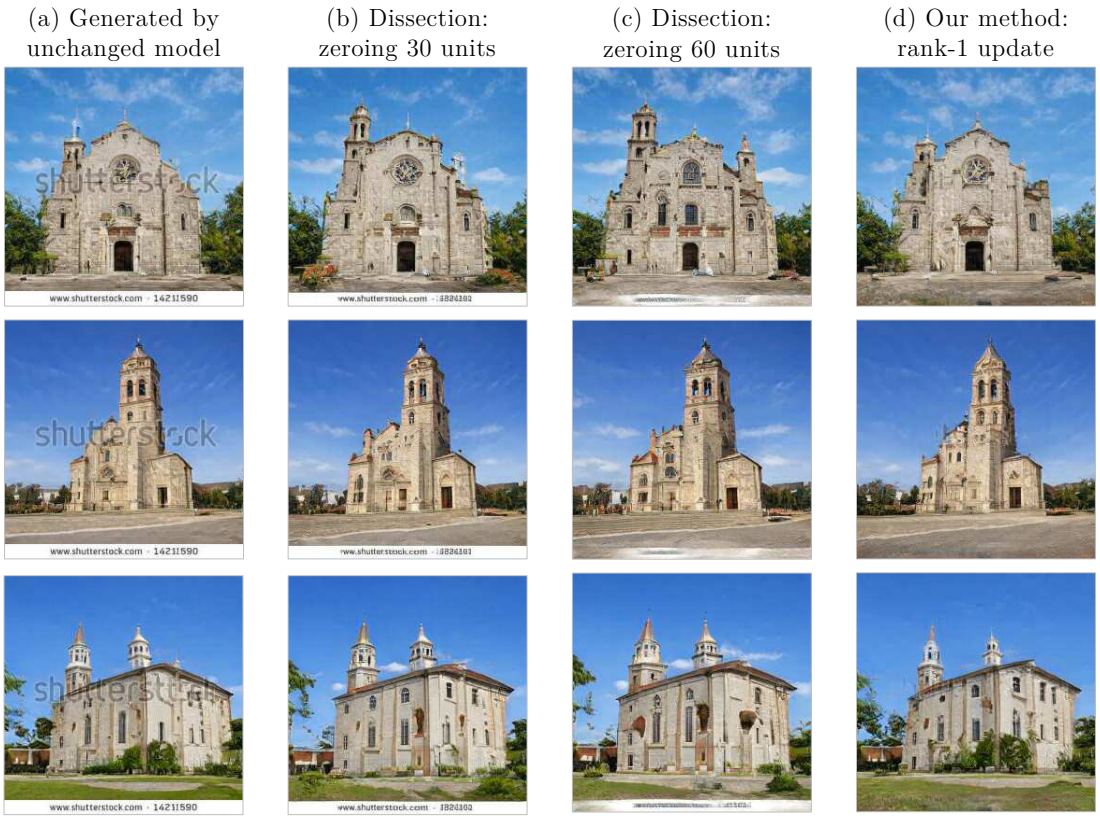


Figure 6-5:]
Removing watermarks from StyleGANv2 Karras et al. [2020] LSUN church Yu et al. [2015] model. (a) Many images generated by this model include transparent watermarks in the center or text on the bottom. (b) Using GAN Dissection Bau et al. [2019b] to zero 30 text-specific units removes middle but not bottom text cleanly. (c) Removing 60 units does not fully remove text, and distorts other aspects of the image. (b) Applying our method to create a rank-1 change erases both middle and bottom text cleanly.

6.5.2 Removing undesired features

Here we test our method on the removal of undesired features. Figure 6-5a shows several examples of images output by a pre-trained StyleGANv2 church model. This model occasionally synthesizes images with text overlaid in the middle and the bottom resembling stock-photo watermarks in the training set.

The GAN Dissection study Bau et al. [2019b] has shown that some objects can be removed from a generator by zeroing the units that best match those objects. To find these units, we annotated the middle and bottom text regions in ten generated images, and we identified a set of 60 units that are most highly correlated with features in

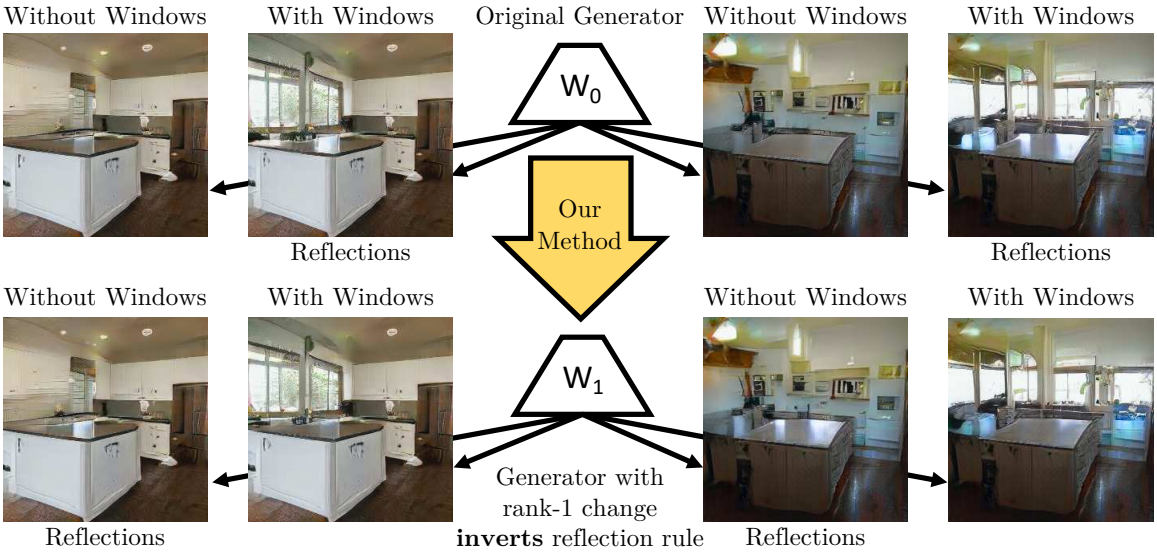


Figure 6-6: Inverting a single semantic rule within a model. At the top row, a Progressive GAN Karras et al. [2018] trained on LSUN kitchens Yu et al. [2015] links windows to reflections: when windows are added by manipulating intermediate features identified by GAN Dissection Bau et al. [2019b], reflections appear on the table. In the bottom row, one rule has been changed within the model to *invert* the relationship between windows and reflections. Now adding windows *decreases* reflections and vice-versa.

these regions. Zeroing the most correlated 30 units removes some of the text, but leaves much bottom text unremoved, as shown in Figure 6-5b. Zeroing all 60 units reduces more of the bottom text but begins to alter the main content of the images, as shown in Figure 6-5c.

For our method, we use the ten user-annotated images as a context to create a rank-one constraint direction d for updating the model, and as an optimization target $K_* \rightarrow V_*$, we use one successfully removed watermark from the setting shown in Figure 6-5b. Since our method applies a narrow rank-1 change constraint, it would be expected to produce a loose approximation of the rank-30 change in the training example. Yet we find that it has instead improved specificity and generalization of watermark removal, removing both middle and bottom text cleanly while introducing few changes in the main content of the image. We repeat the process for 1000 images and tabulate the results in Table 6.3.

6.5.3 Changing contextual rules

In this experiment, we find and alter a rule that determines the illumination interactions between two objects at different locations in an image.

State-of-the-art generative models learn to enforce many relationships between distant objects. For example, it has been observed Bau et al. [2019a] that a kitchen-scene Progressive GAN model Karras et al. [2018] enforces a relationship between windows on walls and specular reflections on tables. When windows are added to a wall, reflections will be added to shiny tabletops, and vice-versa, as illustrated in the first row of Figure 6-6. Thus the model contains a rule that approximates the physical propagation of light in a scene.

In the following experiment, we identified an update direction that allows us to change this model of light reflections. Instead of specifying an objective that copies an object from one context to another, we used a similar tool to specify a $K_* \rightarrow V_*$ objective that swaps bright tabletop reflections with dim reflections on a set of 15 pairs of scenes that are identical other than the presence or absence of bright windows. To identify a rank-one change direction d, we used projected gradient descent, as described in Section 6.3.4, using SVD to limit the change to rank one during optimization. The results are shown in the second row of Figure 6-6. The modified model differs from the original only in a single update direction of a single layer, but it inverts the relationship between windows and reflections: when windows are added, reflections are reduced, and vice-versa.

6.6 Discussion

Machine learning requires data, so how can we create effective models for data that do not yet exist? Thanks to the rich internal structure of recent GANs, in this paper, we have found it feasible to create such models by rewriting the rules within existing networks. Although we may never have seen a tree sprouting from a tower, our network contains rules for both trees and towers, and we can easily create a model that connects those compositional rules to synthesize an endless distribution of images containing

the new combination.

The development of sophisticated generative models beyond the image domain, such as the GPT-3 language model Brown et al. [2020] and WaveNet for audio synthesis Oord et al. [2016], means that it will be increasingly attractive to rewrite rules within other types of models as well. After training on vast datasets, large-scale deep networks have proven to be capable of representing an extensive range of different styles, sentiments, and topics. Model rewriting provides an avenue for using this structure as a rich medium for creating novel kinds of content, behavior, and interaction.

Bibliography

Kfir Aberman, Jing Liao, Mingyi Shi, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. Neural best-buddies: Sparse cross-domain correspondence. *ACM TOG*, 37(4):69, 2018.

Xiaobo An and Fabio Pellacini. Appprop: all-pairs appearance-space edit propagation. *ACM TOG*, 27(3):40, 2008.

James A Anderson. A simple neural network generating an interactive memory. *Mathematical biosciences*, 14(3-4):197–220, 1972.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 2016.

Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM TOG*, 28(3):24, 2009.

David Bau, Hendrik Strobelt, William Peebles, Jonas Wulff, Bolei Zhou, Jun-Yan Zhu, and Antonio Torralba. Semantic photo manipulation with a generative image prior. *ACM TOG*, 38(4), 2019a.

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Zhou Bolei, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. Gan dissection: Visualizing and understanding generative adversarial networks. In *ICLR*, 2019b.

Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Optimality in Artificial and Biological Neural Networks*, pages 6–8. Univ. of Texas, 1992.

Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *ICCV*, 2019.

Xiaowu Chen, Dongqing Zou, Qinping Zhao, and Ping Tan. Manifold preserving edit propagation. *ACM TOG*, 31(6):1–7, 2012.

Xiaowu Chen, Dongqing Zou, Jianwei Li, Xiaochun Cao, Qinping Zhao, and Hao Zhang. Sparse dictionary learning for edit propagation of high-resolution images. In *CVPR*, 2014.

David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine learning*, 15(2):201–221, 1994.

Edo Collins, Raja Bala, Bob Price, and Sabine Susstrunk. Editing in style: Uncovering the local semantics of gans. In *CVPR*, 2020.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *NeurIPS*, 2016.

Jennifer G Dy and Carla E Brodley. Visualization and interactive feature selection for unsupervised data. In *SIGKDD*, 2000.

Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH*. ACM, 2001.

Yuki Endo, Satoshi Iizuka, Yoshihiro Kanamori, and Jun Mitani. Deepprop: Extracting deep features from a single image for edit propagation. *Computer Graphics Forum*, 35(2):189–201, 2016.

Jerry Alan Fails and Dan R Olsen Jr. Interactive machine learning. 2003.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135. JMLR. org, 2017.

Ohad Fried, Ayush Tewari, Michael Zollhöfer, Adam Finkelstein, Eli Shechtman, Dan B Goldman, Kyle Genova, Zeyu Jin, Christian Theobalt, and Maneesh Agrawala. Text-based editing of talking-head video. *ACM TOG*, 38(4):1–14, 2019.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. *CVPR*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Diansheng Guo. Coordinating computational and visual approaches for interactive feature selection and multivariate clustering. *Information Visualization*, 2(4):232–246, 2003.

Samuel W Hasinoff, Martyna Jóźwiak, Frédo Durand, and William T Freeman. Search-and-replace editing for personal photo collections. In *ICCP*, 2010.

Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. Image analogies. In *SIGGRAPH*, 2001.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *ECCV*, 2018.

Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. *ACM TOG*, 35(4), 2016.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

Ali Jahanian, Lucy Chai, and Phillip Isola. On the "steerability" of generative adversarial networks. In *ICLR*, 2020.

Biye Jiang and John Canny. Interactive machine learning via a gpu-accelerated toolkit. pages 535–546, 2017.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1343–1352, 2010.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

Teuvo Kohonen. Correlation matrix memories. *IEEE transactions on computers*, 100 (4):353–359, 1972.

Teuvo Kohonen. *Associative memory: A system-theoretical approach*, volume 17. Springer Science & Business Media, 2012.

Teuvo Kohonen and Matti Ruohonen. Representation of associated data by matrix operators. *IEEE Transactions on Computers*, 100(7):701–702, 1973.

Josua Krause, Adam Perer, and Enrico Bertini. Infuse: interactive feature selection for predictive modeling of high dimensional data. *IEEE transactions on visualization and computer graphics*, 20(12):1614–1623, 2014.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266): 1332–1338, 2015.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. *ECCV*, 2016.

Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM TOG*, 23(3):689–694, 2004.

Jing Liao, Yuan Yao, Lu Yuan, Gang Hua, and Sing Bing Kang. Visual attribute transfer through deep image analogy. *ACM TOG*, 36(4):1–15, 2017.

Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *NeurIPS*, 2017.

Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *CVPR*, 2017.

Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *ICLR*, 2016.

Koki Nagano, Jaewoo Seo, Jun Xing, Lingyu Wei, Zimo Li, Shunsuke Saito, Aviral Agarwal, Jens Fursund, Hao Li, Richard Roberts, et al. pagan: real-time avatars using dynamic textures. In *SIGGRAPH Asia*, page 258, 2018.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.

Kayur Patel, Steven M Drucker, James Fogarty, Ashish Kapoor, and Desney S Tan. Using multiple models to understand data. In *IJCAI*, 2011.

Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *SIGGRAPH*, pages 313–318, 2003.

Tiziano Portenier, Qiyang Hu, Attila Szabó, Siavash Arjomand Bigdeli, Paolo Favaro, and Matthias Zwicker. Faceshop: Deep sketch-based face image editing. *ACM TOG*, 37(4):99:1–99:13, July 2018. ISSN 0730-0301.

Hema Raghavan, Omid Madani, and Rosie Jones. Active learning with feedback on features and instances. *JMLR*, 7(Aug):1655–1686, 2006.

Erik Reinhard, Michael Adhikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer graphics and applications*, 21(5):34–41, 2001.

Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *ECCV*, pages 213–226. Springer, 2010.

Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM TOG*, 25(3):533–540, July 2006. ISSN 0730-0301.

Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *EMNLP*, 2008.

Ankit Sharma and Hassan Foroosh. Slim-cnn: A light-weight cnn for face attribute prediction. In *International Conference on Automatic Face and Gesture Recognition*, 2020.

Assaf Shocher, Nadav Cohen, and Michal Irani. "zero-shot" super-resolution using deep internal learning. In *CVPR*, 2018.

Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *NeurIPS*, 2013.

Ryohei Suzuki, Masanori Koyama, Takeru Miyato, Taizan Yonetsuji, and Huachun Zhu. Spatially controllable image synthesis with internal representation collaging. *arXiv preprint arXiv:1811.10153*, 2018.

Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. In *ICLR*, 2017.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *CVPR*, 2018.

George Wolberg. *Digital image warping*. IEEE computer society press, 1990.

Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, 2018.

Kun Xu, Yong Li, Tao Ju, Shi-Min Hu, and Tian-Qiang Liu. Efficient affinity-based edit propagation using kd tree. *ACM TOG*, 28(5):1–6, 2009.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *NeurIPS*, 2014.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Kaan Yücer, Alec Jacobson, Alexander Hornung, and Olga Sorkine. Transfusive image manipulation. *ACM TOG*, 31(6):1–9, 2012.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.

Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. Real-time user-guided image colorization with learned deep priors. *ACM TOG*, 9(4), 2017.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

ZLL. Face-parsing pytorch. `https://github.com/zllrunning/face-parsing.PyTorch`, 2019.

Chapter 7

Epilogue

7.1 The end of algorithmics foretold

Since Turing laid the intellectual foundations of computing in the 1930s, computer science has centered on the study of algorithmics, a discipline about mathematics and engineering even more so than science.* Each algorithm we create has served as constructive proof† that demonstrates that there is a person who understands the problem well enough to turn every detail into lines of code. In our field, we are accustomed to having full knowledge of the goals and full control of the solutions. Instead of mere evidence, we prefer to seek out correctness.

And yet while Turing pioneered the mathematical principles behind our algorithmic computing discipline, he also foretold its end: in his seminal essay contemplating machine intelligence [Turing, 1950], he observed that "An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside." Seventy years later, machine learning is ascendant and Turing's prophecy has come to pass. Ignorance, it seems, is upon us.

*The question of whether the discipline of computing is a science has been repeatedly debated [Knuth, 1974, Denning et al., 1989]. Here we note that computing differs from science in how nearly every system we study has been created by a person whose goals we can take for granted.

†I refer not only to the formal Curry-Howard isomorphism [Hindley and Seldin, 1980] in which every program is known to be equivalent to a formal proof, but also the way in which constructing a program is proof of knowledge in the spirit of Feyman's saying, "What I cannot create, I do not understand" [Gleick, 1993].

7.2 Computing as a science

The message of this dissertation is that the end of the algorithmic era does not doom us to ignorance. Instead, it challenges us to develop a new kind of knowledge. Yes: machine learning puts us in the uncomfortable position of not knowing how our programs work. But not knowing does not mean there is nothing to know.

The simple methods that we demonstrate within these pages are evidence that, even when we work with large machine-learned neural networks, it is possible to discern their internal structure. Moreover, we have shown that there are benefits to understanding that structure. The insights unlock new applications.

Just as modern biologists devote their efforts to understanding the detailed molecular processes that spring from evolutionary development, future computer scientists will devote their effort to the hard work of understanding the detailed computations that arise from machine learning.

Yet this emerging science of computing has a different character than our field has had up to now. With a machine-learned program, we lack the familiarity that comes from knowing that a reasonable person has created a program. So we begin our investigations of machine-learned systems fully ignorant. We must build our knowledge and intuition with an open mind from the ground up, developing experience through observation, hypothesis-forming, and experimentation.

Computer scientists will need to prove our mettle as scientists.

Bibliography

Peter J. Denning, Douglas E Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R Young. Computing as a discipline. *Computer*, 22(2): 63–70, 1989.

James Gleick. *Genius: The life and science of Richard Feynman.* Vintage, 1993.

J Roger Hindley and Jonathan P Seldin. *To HB Curry: essays on combinatory logic, lambda calculus, and formalism*, volume 479490. Academic Press New York, 1980.

Donald E Knuth. Computer science and its relation to mathematics. *The American Mathematical Monthly*, 81(4):323–343, 1974.

Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

Appendix A

Supplementary Material on GAN Dissection

A.1 Automatic identification of artifact units

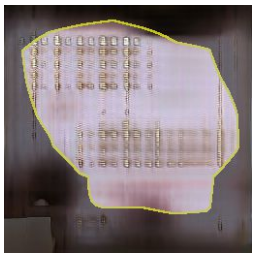
In Section 4.4.2, we have improved GANs by manually identifying and ablating artifact-causing units. Now we describe an automatic procedure to identify artifact units using unit-specific FID scores.

To compute the FID score [Heusel et al., 2017] for a unit u, we generate $200,000$ images and select the $10,000$ images that maximize the activation of unit u, and this subset of $10,000$ images is compared to the true distribution ($50,000$ real images) using FID. Although every such unit-maximizing subset of images represents a skewed distribution, we find that the per-unit FID scores fall in a wide range, with most units scoring well in FID while a few units stand out with bad FID scores: many of them were also manually flagged by humans, as they tend to activate on images with clear visible artifacts.

Figure A-1 shows the performance of FID scores as a predictor of manually flagged artifact units. The per-unit FID scores can achieve 50% precision and 50% recall. That is, of the 20 worst-FID units, 10 are also among the 20 units manually judged to have the most noticeable artifacts. Furthermore, repairing the model by ablating the highest-FID units works: qualitative results are shown in Figure A-2 and quantitative

(a) unit 435, FID=13.35

(b) unit 224, FID=30.80



(c) unit 231, FID=46.29

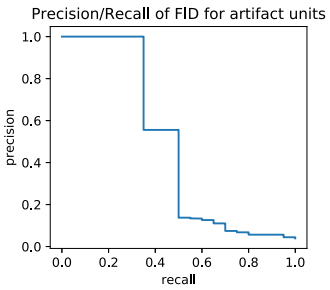


Figure A-1: At left, visualizations of the highest-activating image patches (from a sample of 1000) for three units. (a) the lowest-FID unit that is manually flagged as showing artifacts (b) the highest-FID unit that is not manually flagged (c) the highest-FID unit overall, which is also manually flagged. At right, the precision-recall curve for unit FID as a predictor of the manually flagged artifact units. A FID threshold selecting the top 20 FID units will identify 10 (of 20) of the manually flagged units.

Table A.1: We compare generated images before and after ablating "artifact" units. The "artifacts" units are found either manually, automatically, or both. We also report a simple baseline that ablates 20 randomly chosen units.

	Fréchet Inception Distance (FID)
original images	43.16
manually chosen "artifact" units ablated (as in Section 4.4.2)	**27.14**
highest-20 FID units ablated	27.6
union of manual and highest FID (30 total) units ablated	26.1
20 random units ablated	43.17

results are shown in Table A.1.

A.2 Human evaluation of dissection

As a sanity check, we evaluate the gap between human labeling of object concepts correlated with units and our automatic segmentation-based labeling, for one model, as follows.

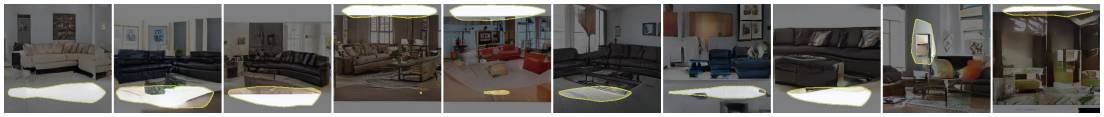
For each of 512 units of `layer4` of a "living room" Progressive GAN, 5 to 9 human annotations were collected (3728 labels in total). In each case, an AMT worker is asked to provide one or two words describing the highlighted patches in a set of top-activating images for a unit. Of the 512 units, 201 units were described by the same consistent word (such as "sofa", "fireplace" or "wicker") in 50% or more of the human labels. These units are interpretable to humans.

Figure A-2: The effects of ablating high-FID units compared to manually-flagged units: (a) generated images with artifacts, without intervention; (b) those images generated after ablating the 20-highest FID units; (c) those images generated after ablating the 20 manually-chosen artifact units.

Applying our segmentation-based dissection method, 154/201 of these units are also labeled with a confident label with IoU > 0.05 by dissection. In 104/154 cases, the segmentation-based model gave the same label word as the human annotators, and most others are slight shifts in specificity. For example, the segmentation labels "ottoman" or "curtain" or "painting" when a person labels "sofa" or "window" or "picture," respectively. A second AMT evaluation was done to rate the accuracy of both segmentation-derived and human-derived labels. Human-derived labels scored 100% (of the 201 human-labeled units, all of the labels were rated as consistent by most raters). Of the 154 segmentation-generated labels, 149 (96%) were rated by most AMT raters as accurate as well.

The five failure cases (where the segmentation is confident but rated as inaccurate by humans) arise from situations in which human evaluators saw one concept after observing only 20 top-activating images, while the algorithm, in evaluating 1000 images, counted a different concept as dominant. Figure A-3a shows one example: in the top images, mostly sofas are highlighted and few ceilings, whereas in the larger

(a) `unit118` in `layer4`



(b) `unit11` in `layer4`

Figure A-3: Two examples of generator units that our dissection method labels differently from humans. Both units are taken from `layer4` of a Progressive GAN of living room model. In (a), human label the unit as 'sofa' based on viewing the top-20 activating images, and our method labels as 'ceiling'. In this case, our method counts many ceiling activations in a sample of 1000 images beyond the top 20. In (b), the dissection method has no confident label prediction even though the unit consistently triggers on white letterbox shapes at the top and bottom of the image. The segmentation model we use has no label for such abstract shapes.

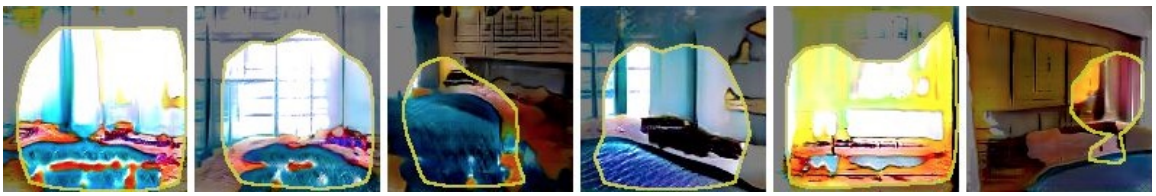
sample, mostly ceilings are triggered.

There are also 47/201 cases where the segmenter is not confident while humans have consensus. Some of these are due to missing concepts in the segmenter. Figure A-3b shows a typical example, where a unit is devoted to letterboxing (white stripes at the top and bottom of images), but the segmentation has no confident label to assign to these. We expect that as future semantic segmentation models are developed to be able to identify more concepts such as abstract shapes, more of these units can be automatically identified.

A.3 Protecting segmentation model against unrealistic images

Our method relies on having a segmentation function $s_c(\mathbf{x})$ that identifies pixels of class c in the output $\mathbf{x}$. However, the segmentation model s_c can perform poorly in the cases where $\mathbf{x}$ does not resemble the original training set of s_c. This phenomenon is visible when analyzing earlier GAN models. For example, Figure A-4 visualizes two units from a WGAN-GP model [Gulrajani et al., 2017] for LSUN bedrooms (this model was trained by Karras et al. [2018] as a baseline in the original paper). For these

(a) Unit 154, FID 63.8, "Floor" with IoU 0.20.



(b) Unit 371, FID 58.3, "Swimming Pool" with IoU 0.02.

Figure A-4: Two examples of units that correlate with unrealistic images that confuse a semantic segmentation network. Both units are taken from a WGAN-GP for LSUN bedrooms.

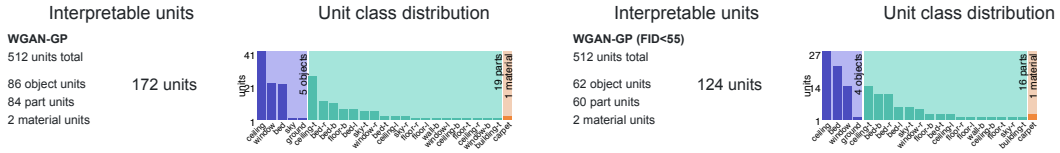


Figure A-5: Comparing a dissection of units for a WGAN-GP trained on LSUN bedrooms, considering all units (at left) and considering only "realistic" units with FID < 55 (at right). Filtering units by FID scores removes spurious detected concepts such as 'sky', 'ground', and 'building'.

two units, the segmentation network seems to be confused by the distorted images.

To protect against such spurious segmentation labels, we can use a technique similar to that described in Section A.1: automatically identify units that produce unrealistic images, and omit those "unrealistic" units from semantic segmentation. An appropriate threshold to apply will depend on the distribution being modeled: in Figure A-5, we show how applying a filter, ignoring segmentation on units with FID 55 or higher, affects the analysis of this base WGAN model. In general, fewer irrelevant labels are associated with units.

A.4 Computing causal units

In this section we provide more details about the ACE optimization described in Section 4.3.2.

Specifying the per-class positive intervention constant k. In Eqn. 4.3, the negative intervention is defined as zeroing the intervened units, and a positive intervention is defined as setting the intervened units to some big class-specific constant **k**. For interventions for class c, we set **k** to be mean featuremap activation conditioned on the presence of class c at that location in the output, with each pixel weighted by the portion of the featuremap locations that are covered by the class c. Setting all units at a pixel to **k** will tend to strongly cause the target class. The goal of the optimization is to find the subset of units that is causal for c.

Sampling c-relevant locations P. When optimizing the causal objective (Eqn. 4.5), the intervention locations P are sampled from individual featuremap locations. When the class c is rare, most featuremap locations are uninformative: for example, when class c is a door in church scenes, most regions of the sky, grass, and trees are locations where doors will not appear. Therefore, we focus the optimization as follows: during training, minibatches are formed by sampling locations P that are relevant to class c by including locations where the class c is present in the output (and are therefore candidates for removal by ablating a subset of units), and an equal portion of locations where class c is not present at P, but it would be present if all the units are set to the constant **k** (candidate locations for insertion with a subset of units). During the evaluation, causal effects are evaluated using uniform samples: the region P is set to the entire image when measuring ablations, and to uniformly sampled pixels P when measuring single-pixel insertions.

Initializing $\boldsymbol{\alpha}$ with IoU. When optimizing causal $\boldsymbol{\alpha}$ for class c, we initialize with

$$\alpha_u = \frac{\mathrm{IoU}_{u,c}}{\max_v \mathrm{IoU}_{v,c}} \tag{A.1}$$

That is, we set the initial α so that the largest component corresponds to the unit with the largest IoU for class c, and we normalize the components so that this largest component is 1.

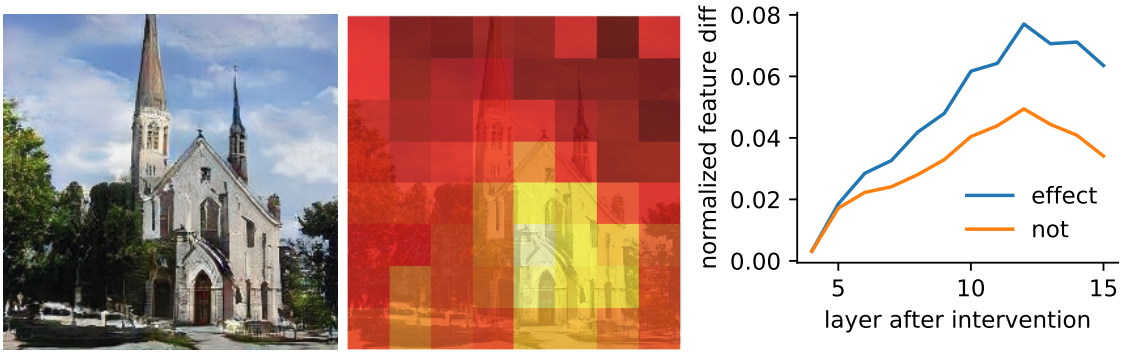


Figure A-6: Tracing the effect of inserting door units on downstream layers. An identical "door" intervention at `layer4` of each pixel in the featuremap has a different effect on later feature layers, depending on the location of the intervention. In the heatmap, brighter colors indicate a stronger effect on the `layer14` feature. A request for a door has a larger effect in locations of a building, and a smaller effect near trees and sky. At right, the magnitude of feature effects at every layer is shown, measured by the changes of mean-normalized features. In the line plot, feature changes for interventions that result in human-visible changes are separated from interventions that do not result in noticeable changes in the output.

Applying a learned intervention $\boldsymbol{\alpha}$ When applying the interventions, we clip $\boldsymbol{\alpha}$ by keeping only its top n components and zeroing the remainder. To compare the interventions of different classes an different models on an equal basis, we examine interventions where we set $n = 20$.

A.5 Tracing the effect of an intervention

To investigate the mechanism for suppressing the visible effects of some interventions seen in Section 4.4.4, in this section we insert 20 door-causal units on a sample of individual featuremap locations at `layer4` and measure the changes caused in later layers.

To quantify effects on downstream features, the change in each feature channel is normalized by that channel's mean L1 magnitude, and we examine the mean change in these normalized featuremaps at each layer. In Figure A-6, these effects that propagate to `layer14` are visualized as a heatmap: brighter colors indicate a stronger effect on the final feature layer when the door intervention is in the neighborhood of a building instead of trees or sky. Furthermore, we plot the average effect on every layer at right in Figure A-6, separating interventions that have a visible effect from those that do

not. A small identical intervention at `layer4` is amplified to larger changes up to a peak at `layer12`.

A.6 Monitoring GAN units during training

Dissection can also be used to monitor the progress of training by quantifying the emergence, diversity, and quality of interpretable units. For example, in Figure A-7 we show dissections of `layer4` representations of a Progressive GAN model trained on bedrooms, captured at a sequence of checkpoints during training. As training proceeds, the number of units matching objects increases, as does the number of object classes with matching units, and the quality of object detectors as measured by average IoU over units increases. During this successful training, dissection suggests that the model is gradually learning the structure of a bedroom, as increasingly units converge to meaningful bedroom concepts.

A.7 All layers of a GAN

In Section 4.4.1 we show a small selection of layers of a GAN; in Figure A-8 we show a complete listing of all the internal convolutional layers of that model (a Progressive GAN trained on LSUN living room images). As can be seen, the diversity of units matching high-level object concepts peaks at `layer4-layer6`, then declines in later layers, with the later layers dominated by textures, colors, and shapes.

Bibliography

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

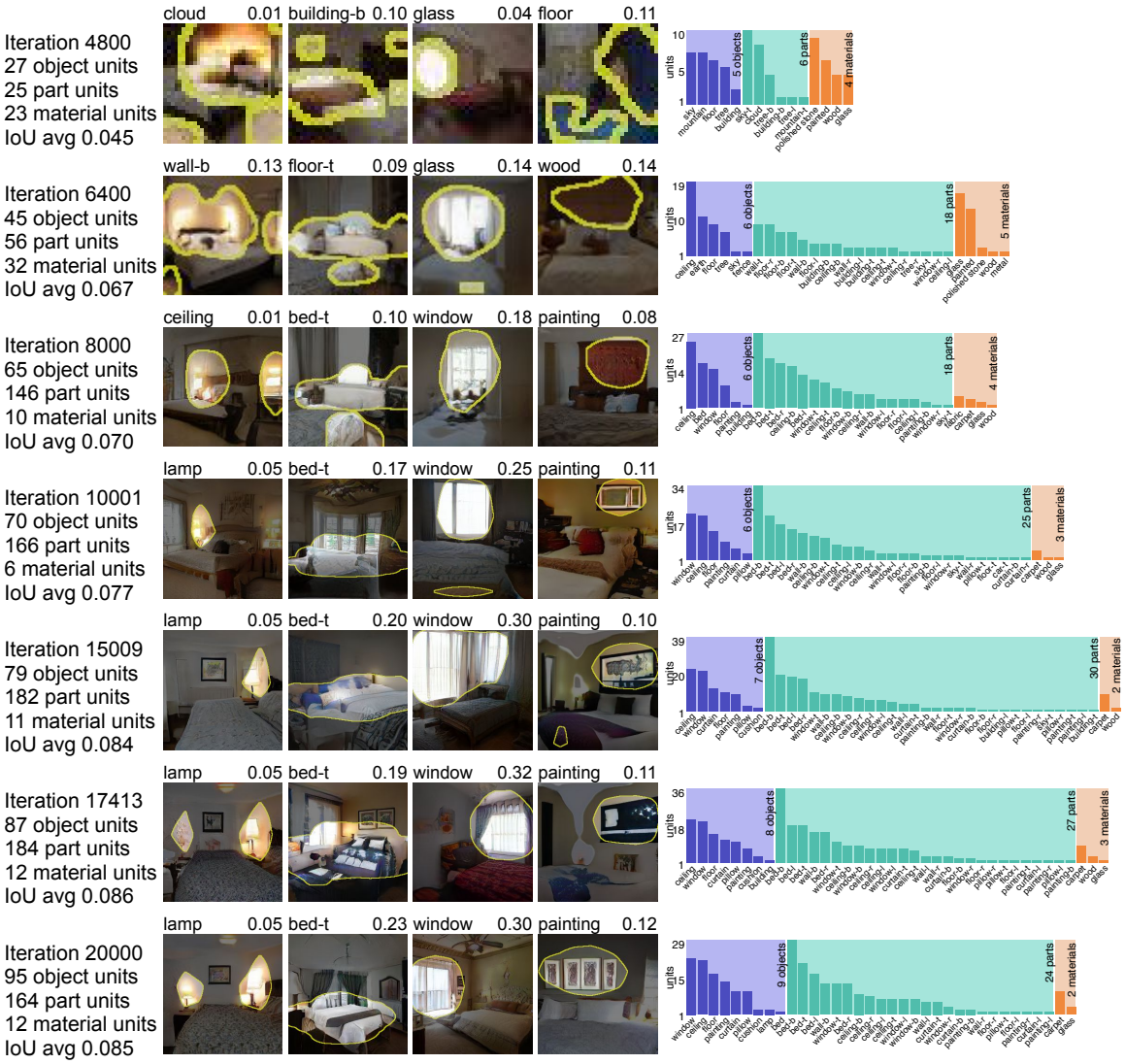


Figure A-7: The evolution of `layer4` of a Progressive GAN bedroom generator as training proceeds. The number and quality of interpretable units increases during training. Note that in early iterations, Progressive GAN generates images at a low resolution. The top-activating images for the same four selected units is shown for each iteration, along with the IoU and the matched concept for each unit at that checkpoint.

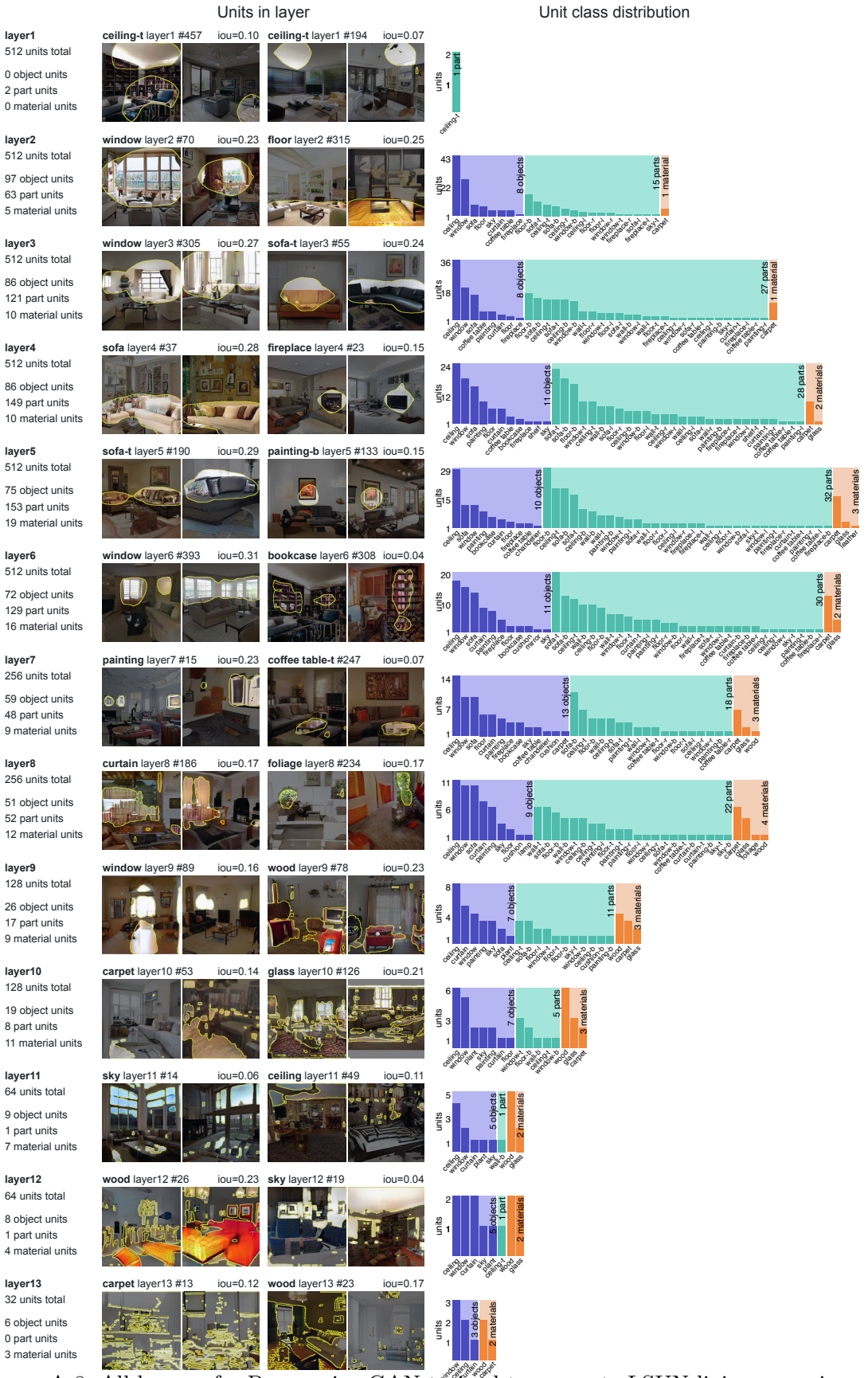


Figure A-8: All layers of a Progressive GAN trained to generate LSUN living room images.

Appendix B

Supplementary Material on Rewriting a Generative Model

Figures B-1, B-2, B-3, B-4, B-5, and B-6 show additional results of our editing method to change a model to achieve a variety of effects across an entire distribution of generated images. Each figure illustrates a single low-rank change of a StyleGAN v2 model derived from the user gestures shown in the top row. The twelve pairs of images shown below the top row of each figure are the images that score highest in the context direction d, out of a random sample of 1000: that is, these are images that are most relevant to the user's context selection. For each image, both the output of the unmodified original model and the modified model are shown. All changes are rank-one changes to the model, except Figure B-4, which is rank ten, and Figure B-6, which is rank three.

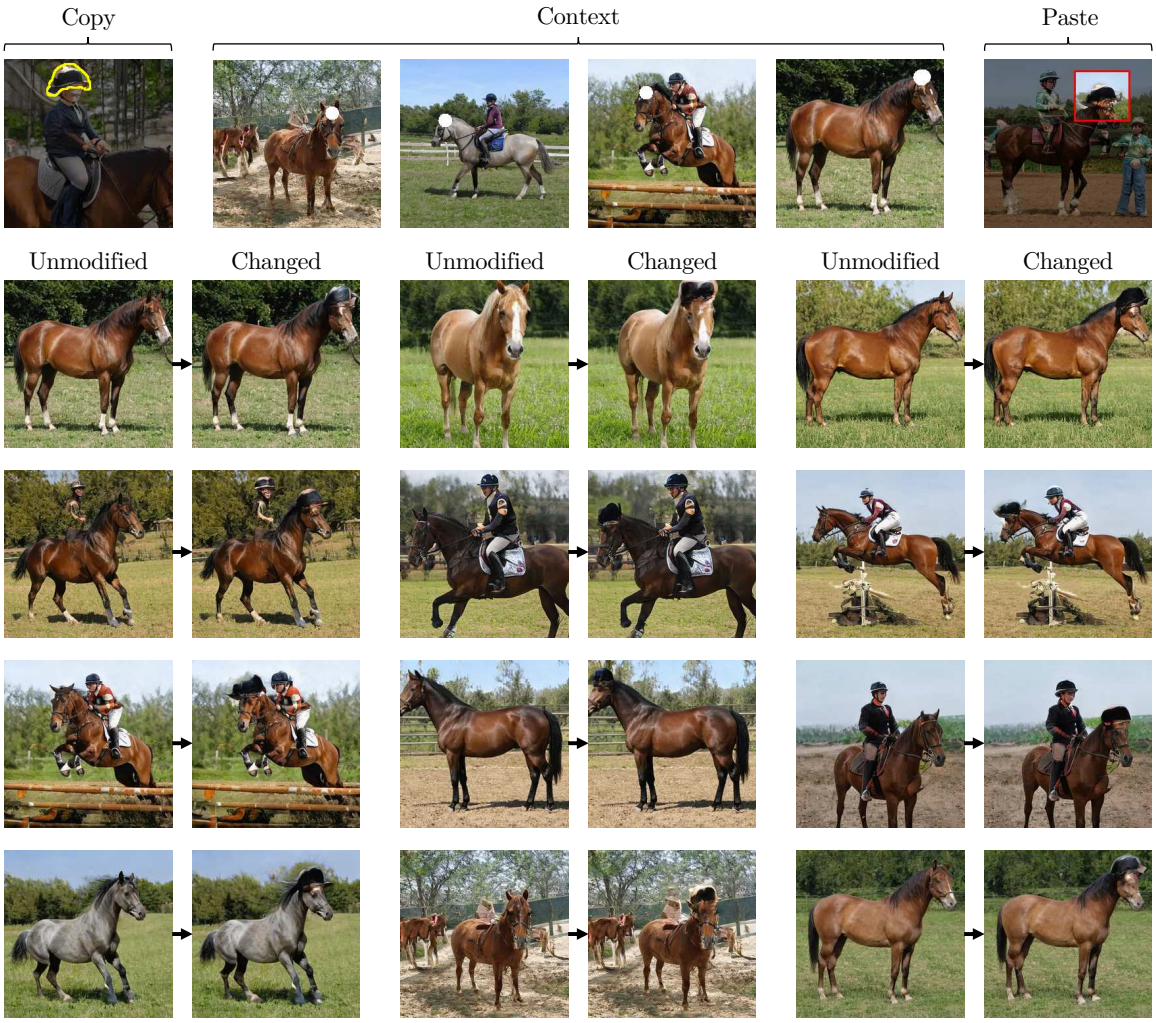


Figure B-1: **Giving horses a hat to wear.** After one hat is pasted onto an example of a horse, and after the user has pointed at four other horse heads, the model is changed so that horses in a variety of poses, settings, shapes, and sizes all get a hat on their head. This is not merely a re-balancing of the distribution of the model. This change introduces a new kind of image that was not generated before. The original training data does not include hats on horses, and the original pretrained StyleGANv2 does not synthesize hats on any horses.

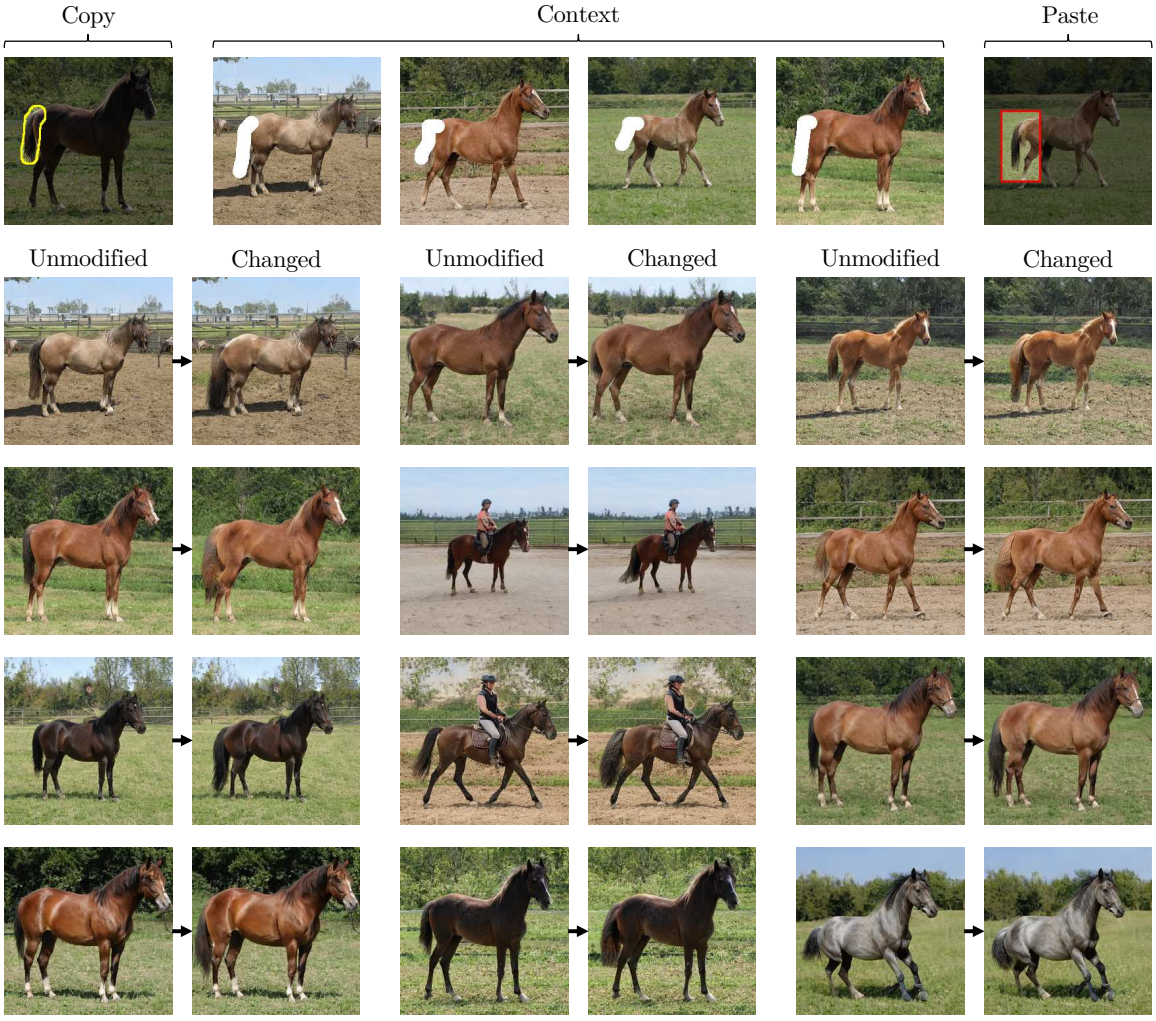


Figure B-2: **Giving horses a longer tail.** Notice that the color, shape, and occlusions of the tail vary to fit the specific horse, but in each case the tail is made longer, as demonstrated in the pasted example.

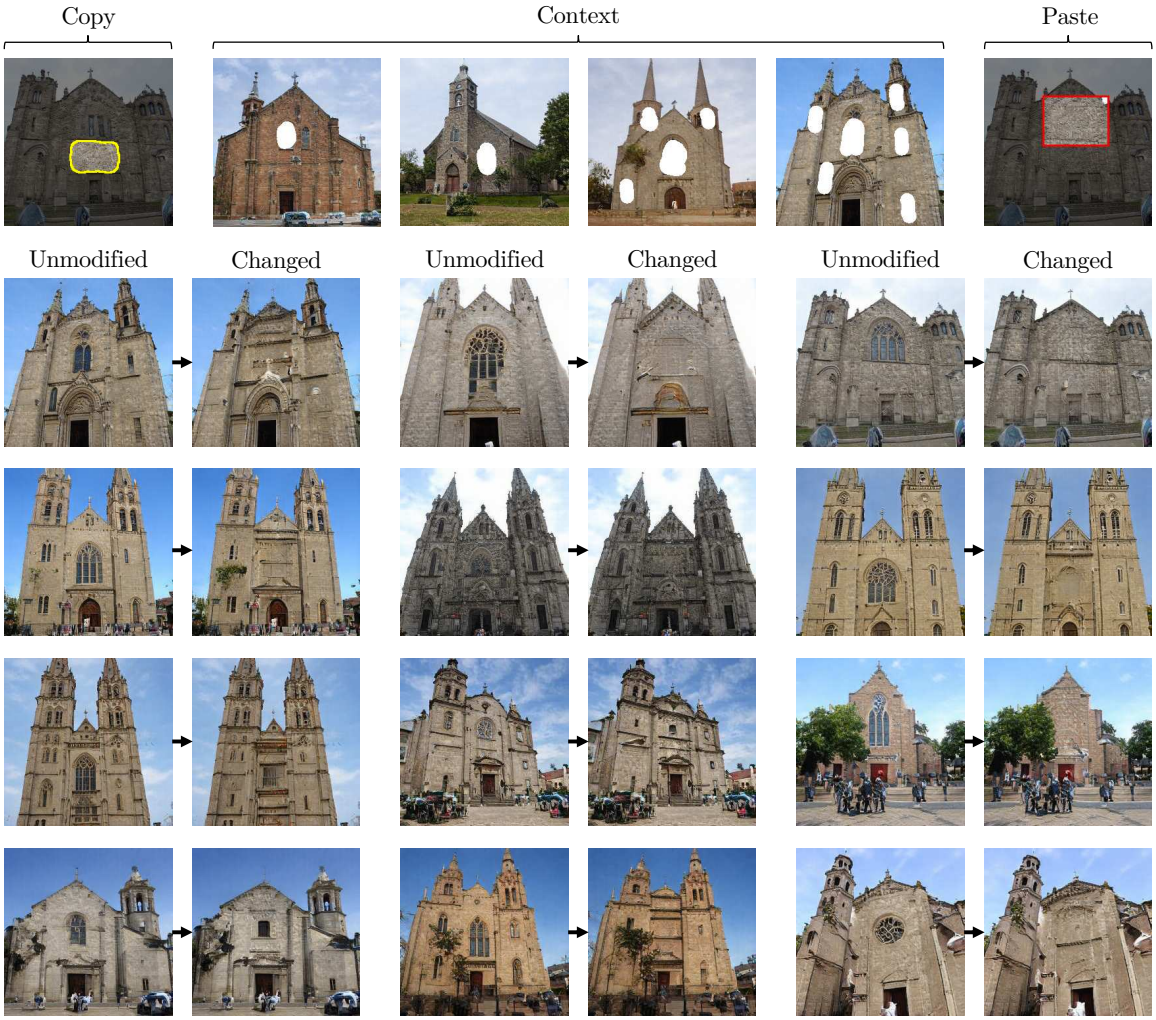


Figure B-3: **Removing main windows from churches.** The modified model will replace the central window with a blank wall, or with a wall with some different details.

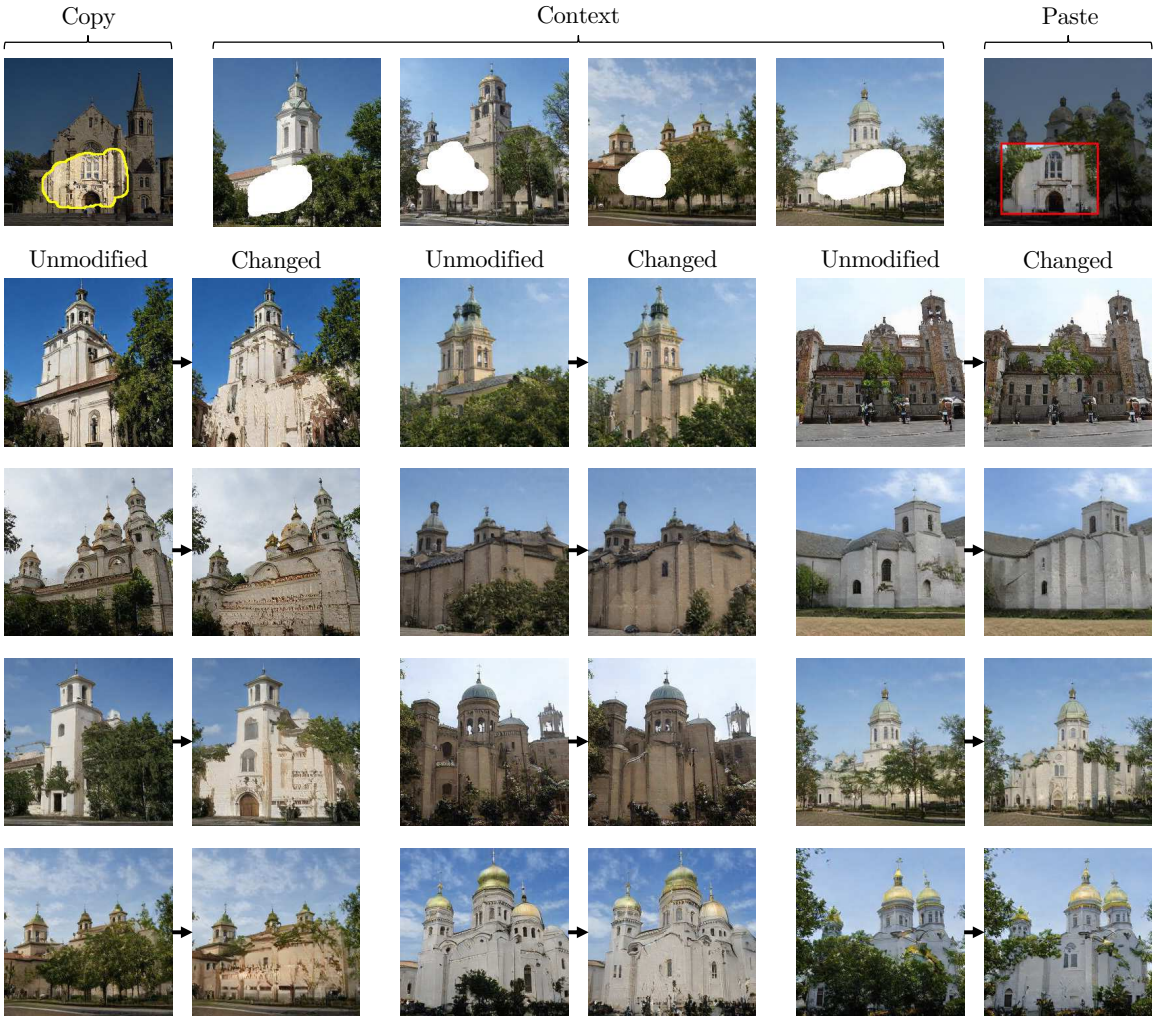


Figure B-4: **Reducing the occlusion of buildings by trees.** This edit removes the trees in front of buildings. Note that the model can still synthesize trees next to buildings.

Figure B-5: **Removing earrings.** Removing one set of earrings generalizes to many different types of earrings appearing in different poses.

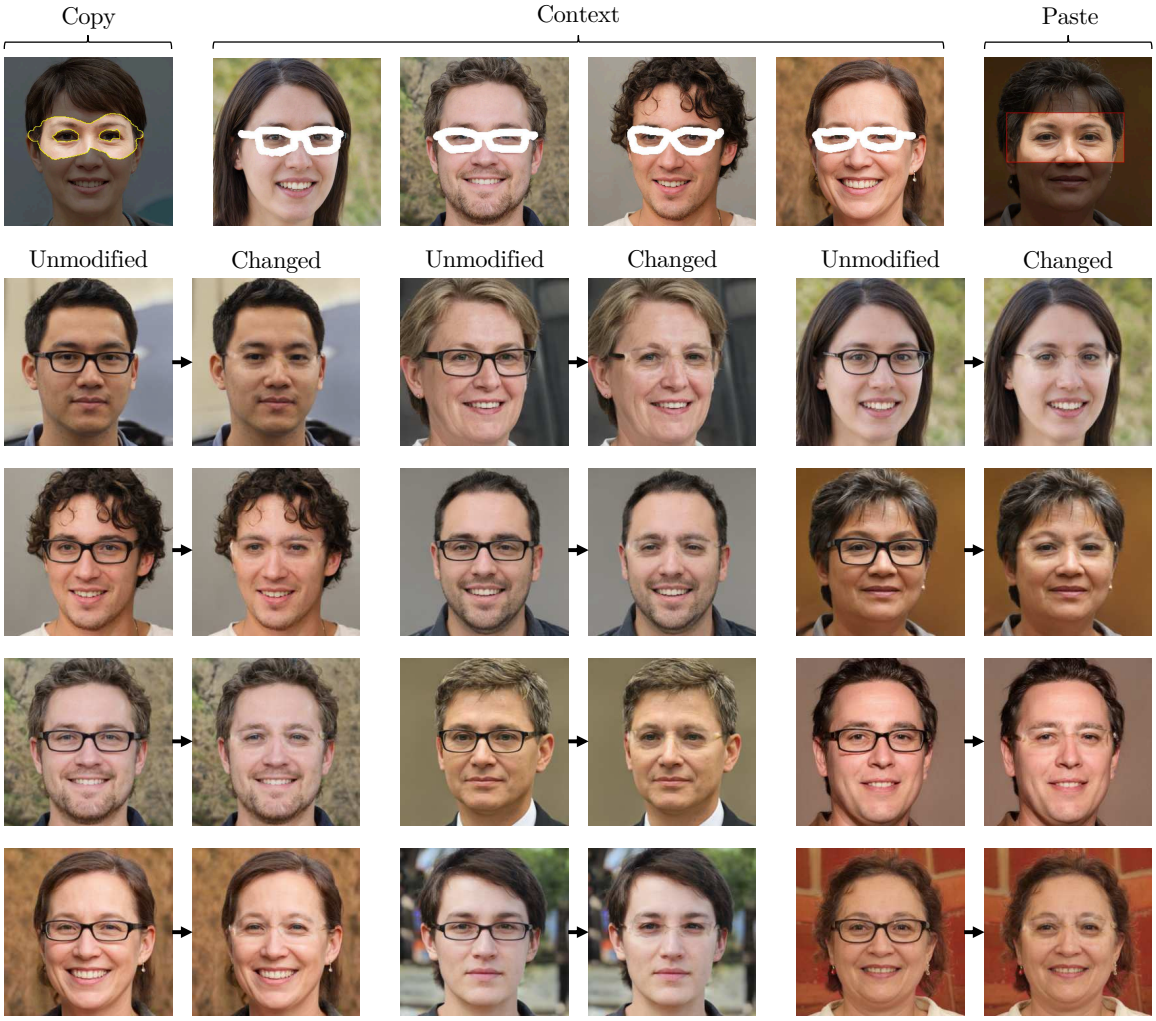


Figure B-6: **Removing glasses.** Note that glasses of different shapes are removed, and most facial structure is recovered. This is a rank-three change. Although most of the glasses have been removed, this edit did not remove the temples (side parts) of some glasses, and did not remove refraction effects.

B.1 Solving for Λ algebraically

To strengthen our intuition, here we describe the closed-form solution for Λ in the linear case. Recall from Equations 6.13 and 6.15:

$$W_1 k_* = v_* \tag{B.1}$$

$$W_1 = W_0 + \Lambda d^T \tag{B.2}$$

In the above we have written $d = C^{-1}k_*$ as in Eqn. 6.16 for berevity. Then we can solve for both W_1 and Λ simultaneously by rewriting the above system as the following matrix product in block form:

$$\left[\begin{array}{c|c} W_1 & \Lambda \end{array}\right] \left[\begin{array}{c|c} I & k_* \\ \hline -d^T & 0 \end{array}\right] = \left[\begin{array}{c|c} W_0 & v_* \end{array}\right] \tag{B.3}$$

$$\left[\begin{array}{c|c} W_1 & \Lambda \end{array}\right] = \left[\begin{array}{c|c} W_0 & v_* \end{array}\right] \left[\begin{array}{c|c} I & k_* \\ \hline -d^T & 0 \end{array}\right]^{-1} \tag{B.4}$$

In practice, we do not solve this linear system because a neural network layer is nonlinear. In the nonlinear case, instead of using matrix inversion, Λ is found using the optimization in Equation 6.17.

B.2 Implementation details

Datasets To compare identical model edits in different settings, we prepare a small set of saved editing sessions for executing an change. Each session corresponds to a set of masks that a user has drawn in order to specify a region to copy and paste, together with any number of context regions within generated images for a model.

Benchmark editing sessions are included with the source code.

Large-scale datasets are used only for pretraining the generative models. The generative models we use are trained on the following datasets. The face model is trained on Flickr-Faces-HQ (FFHQ) Karras [2019], a dataset of 70,000 1024×1024 face images. The outdoor church, horse, and kitchen models are trained on LSUN image datasets Yu et al. [2015]. LSUN provides 126,000 church images, 2.2 million kitchen images, and 2 million horse images at resolutions of 256×256 and higher.

Generators We rewrite two different generative model architectures: Progressive GAN and StyleGAN v2. The Progressive GAN generator has 18.3 million parameters and 15 convolutional layers; we edit a model pretrained on LSUN kitchens. We also edit StyleGAN v2 Karras et al. [2020]. StyleGAN v2 has 30 million parameters and 14 convolutional layers (17 layers for the higher-resolution faces model). We edit StyleGAN v2 models trained on FFHQ faces, LSUN churches, and LSUN horses. All the model weights were those published by the original GAN model authors. For StyleGAN v2, we apply the truncation trick with multiplier 0.5 when running the model.

Metrics To quantify undesired perceptual differences made in edits, we use the Learned Perceptual Image Patch Similarity (LPIPS) Zhang et al. [2018] metric to compare unedited images to edited images. We use the default Alexnet-based LPIPS network weights as published by the original LPIPS authors. To focus the measurement on undesired changes, we follow the method of the GAN projection work Huh et al. [2020] and mask out portions of the image that we intend to change, as identified by a semantic segmentation network. For faces, we segment the image using a modified BiSeNet Yu et al. [2018] as published by ZLL as faceparsing-Pytorch ZLL [2019]. For churches, we segment the image using the Unified Perceptual Parsing network Xiao et al. [2018].

To quantify the efficacy of the change, we also use pretrained networks. To detect whether a face image is similing, we use a Slim-CNN Sharma and Foroosh [2020] facial attribute classifier. To determine if domes have successfully been edited to other types

of objects, we again use the Unified Perceptual Parsing network, and we count pixels that have changed from being classified as domes to buildings or trees.

User studies Human realism measurements are done using Amazon Mechanical Turk (AMT). For each baseline editing method, 500 pairs of images are generated comparing an edited image using our approach to the same image edited using a baseline method, and two AMT workers are asked to judge which of the pair is more realistic, for a total of 1000 comparative judgements. We do not test the fantastical domes-to-trees edit, which is intended to be unrealistic.

B.3 Rank reduction for D_S

In this section we discuss the problem of transforming a user's *context* selection $K \in \mathbb{R}^{N \times T}$ (Section 6.4) into a constraint subspace $D_S \in \mathbb{R}^{N \times S}$, where the desired dimensionality $s \ll t$ is smaller than the number of given feature samples T provided in K.

We shall think of this as a lossy compression problem. Use P to denote the probability distribution of the layer $L-1$ features (unconditioned on any user selection), and think of K as a discrete distribution over the user's t context examples. We can then use cross-entropy $H(K, P)$ to quantify the information in K, measured as the message length in a code optimized for the distribution P. To express this information measure in the setting used in Section 6.3, we will model P as a zero-centered Gaussian distribution $P(k) = (2\pi)^{-n/2} \exp -k^T C^{-1} k/2$ with covariance C.

If we the normalize the basis using the ZCA whitening transform Z, we can express P as a spherical unit normal distribution in the variable $k' = Zk$. This yields a concise

matrix trace expression for cross entropy:

$$\text{Let } C = U\Sigma U^T \text{ be the eigenvector decomposition} \tag{B.5}$$

$$Z \triangleq C^{-1/2} = U\Sigma^{-1/2}U^T \tag{B.6}$$

$$k' \triangleq Zk \tag{B.7}$$

$$K' \triangleq ZK \tag{B.8}$$

$$P(k') = (2\pi)^{-n/2} \exp(-k'^T k'/2) \tag{B.9}$$

$$H(K', P) = \sum_{k' \in K'} \frac{1}{t} \log P(k') \tag{B.10}$$

$$= \frac{1}{2t} \sum_{k' \in K'} k'^T k' + \frac{n}{2t} \log 2\pi \tag{B.11}$$

$$= \frac{1}{2t} \operatorname{Tr}\left(K'^T K'\right) + \frac{n}{2t} \log 2\pi \tag{B.12}$$

In other words, by assuming a Gaussian model, the information in the user's context selection can be quantified the trace of a symmetric matrix given by inner products over the whitened context selection.

To reduce the rank of the user's context selection, we wish to project the elements of K' by discarding information along the $R = N - S$ most uninformative directions. Therefore, we seek a matrix $Q_R^* \in \mathbb{R}^{N \times R}$ that has R orthonormal columns, chosen so that the projection of the samples $Q_R Q_R^T K'$ minimize cross-entropy with P:

$$Q_R^* = \arg\min_{Q_R} H(Q_R Q_R^T K', P) \tag{B.13}$$

$$= \arg\min_{Q_R} \operatorname{Tr}\left(K'^T Q_R Q_R^T Q_R Q_R^T K'\right) \tag{B.14}$$

$$= \arg\min_{Q_R} \operatorname{Tr}\left(Q_R^T K' K'^T Q_R\right) \tag{B.15}$$

The trace minimization Eqn. B.15 is an instance of the well-studied trace optimization problem Kokiopoulou et al. [2011] that arises in many dimension-reduction settings. It can be solved by setting the columns of Q_R^* to a basis spanning the space of the

eigenvectors for the smallest R eigenvalues of $K'_{\mathsf{ctx}}K'^T$.

Denote by $Q_S^* \in \mathbb{R}^{N\times S}$ the matrix of orthonormal eigenvectors for the S *largest* eigenvalues of $K'_{\mathsf{ctx}}K'^T$. Then we have $(I - Q_R^*Q_R^{*T})k' = Q_S^*Q_S^{*T}k'$, i.e., erasing the uninteresting directions of Q_R^* is the same as preserving the directions Q_S^*. This is the S-dimensional subspace that we seek: it is the maximally informative low-dimensional subspace that captures the user's context selection.

Once we have Q_S^* within the whitened basis, the same subspace can be expressed in unwhitened row space coordinates as:

$$D_S = Z^T Q_S^* = ZQ_S^* \tag{B.16}$$

B.4 Axis-aligned rank reduction for D_S

The identification of axis-aligned units most relevant to a user's context selection can also be analyzed using the same rank-reduction objective as Section B.3, but with a different family for P. Instead of modeling P as a Gaussian with generic covariance C, we now model it as an axis-aligned Gaussian with diagonal covariance $\Sigma = \mathrm{diag}(\sigma_i)$. Then the optimal basis Q_S^* becomes the unit vectors for the unit directions e_i that maximize the expected ratio

$$\sum_{k\in K_{\mathsf{ctx}}} \frac{\left(e_i^T k\right)^2}{\sigma_i^2} \tag{B.17}$$

In Section 6.5.2 this scoring is used to identify the units most relevant to watermarks in order to apply GAN dissection unit ablation.

B.5 Experiment details and results

Table 6.2 shows the quantitative results of comparing our method with various baselines on editing a StyleGANv2 Karras et al. [2020] LSUN church Yu et al. [2015] model. For both edits, our method modifies the 7th convolution layer of the generator, with Adam

optimizer Kingma and Ba [2015], 0.05 learning rate, 2001 gradient iterations, and projecting to a low-rank change every 10 iterations (and also after the optimization loop). For `domes` → `trees`, a rank 1 edit is performed. (These settings are also the defaults provided in the user interface, and were used for video demos.) For `domes` → `spires`, a rank 10 edit is performed.

For the StyleGANv2 FFHQ Karras et al. [2019] edit shown in main paper 6.1, our method modifies the 9th convolution layer of the generator, also with Adam optimizer Kingma and Ba [2015], 0.05 learning rate, 2001 gradient iterations, and projecting to a low-rank change every 10 iterations (and also after the optimization loop).

For all experiments, the baseline that finetunes all weights uses the Adam optimizer Kingma and Ba [2015] with 2001 iterations and a learning rate of 10^{-4}.

B.6 Reflection experiment details

In Section 6.5.3, we found the rank-one rule reversal change for the abstract window lighting rule as follows.

1. **Generation**: we use the GAN to generate 15 images in two ways, one adding windows, and one removing windows, by activating and deactivating window-correlated units. The window correlated units are identified using dissection Bau et al. [2019].

2. **Annotation**: a user masks illuminated regions of the 15 images far from the windows that show reflected light that differs between the pairs.

3. **Optimization**: we optimize a change in the weights of the layer to reverse the behavior of the reflected light in the masked areas, to match dark output when there is a window and bright output when there is no window. This optimization is constrained to one direction by using an SVD reduction to rank one every 10 iterations.

The optimization is computed at each individual layer, and we use the layer that

achieves the lowest loss with a rank-one change: for this experiment, this is layer 6 of the model.

B.7 Selecting a layer for editing

There are two ways to view a convolutional layer: either as a computation in which information from neighboring locations is combined to detect or produce edges, textures, or shapes; or as a memory in which many independent feature mappings are memorized.

In our paper we have adopted the simple view that a layer acts as an associative memory that maps from one layer's local feature vectors to local patches of feature vectors in the next layer. This view is appropriate when layer representations have features in which neighboring locations are disentangled from one another. In practice, we find that both ProgressiveGAN and StyleGAN representations have this property. For example, if a feature patch is rendered in isolation from neighboring features, the network will usually render the same object as it does in the context of the full featuremap.

In Figures B-7 and B-10, we measure the similarity between patches rendered in isolation compared to same-sized patches cropped out of the full model, using Fréchet Inception Distance (FID) Heusel et al. [2017]. Lower FIDs indicate less dependence between neighboring patches, and higher FIDs indicate higher dependence between neighbors. These graphs show that layers 6-11 in StyleGANv2 and layers 4 and higher in Progressive GAN are most appropriate for editing as an associative memory. (Note that in StyleGANv2, the nth featuremap layer is the output of the $n-1$th convolutional layer, because the first featuremap layer is fixed. In Progressive GAN, the nth featuremap layer is the output of the nth convolutional layer.)

Figures B-8 and B-9 visualize individual patches rendered in isolation at various layers of StyleGANv2, and compare those to the entire image rendered together. Figures B-11 and B-12 visualize the same for Progressive GAN.

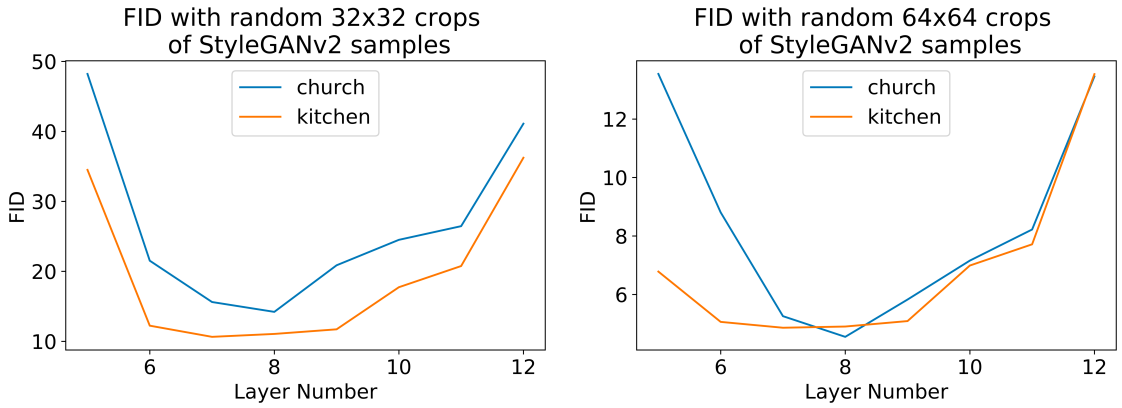


Figure B-7: FID of rendered cropped activations with respect to random crops of StyleGANv2 generated images. In StyleGANv2, the nth convolutional layer outputs the $n+1$th featuremap layer. The layer numbers above correspond to featuremap layers.

Figure B-8: Comparison of rendered cropped activations at various layers of StyleGANv2 generated LSUN church images.

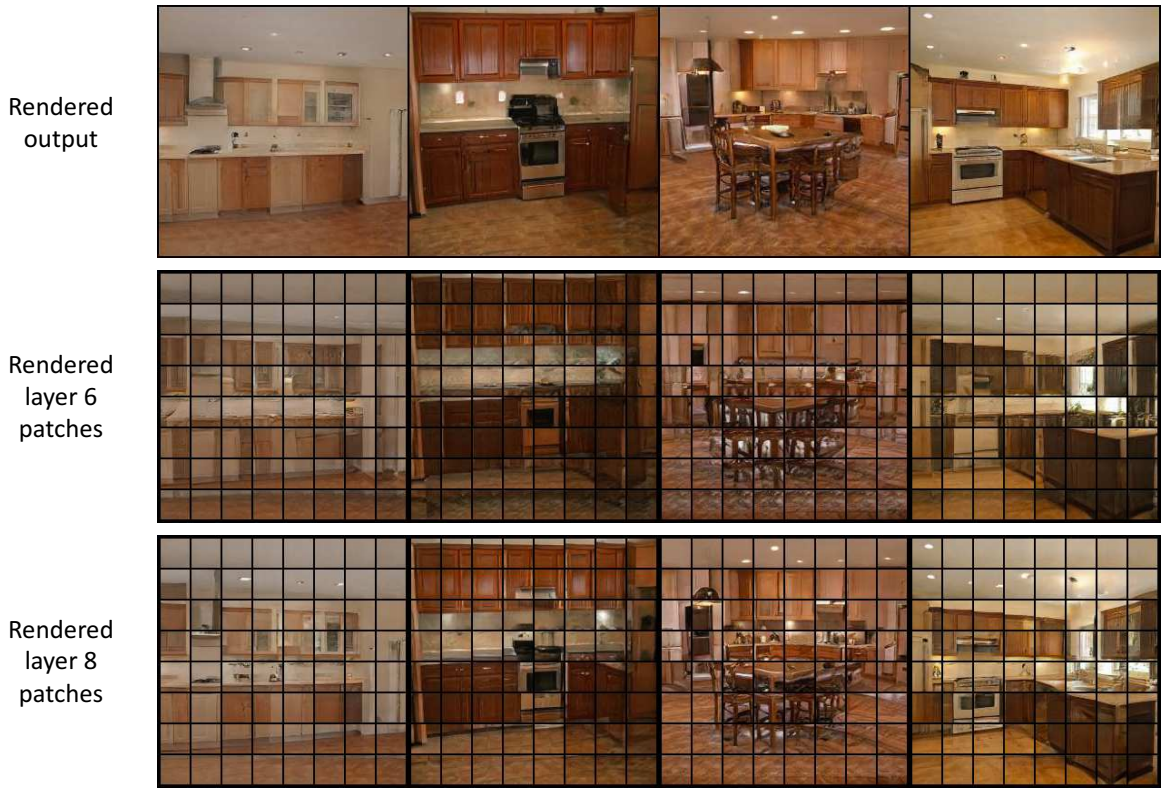


Figure B-9: Comparison of rendered cropped activations at various layers of StyleGANv2 generated LSUN kitchen images.

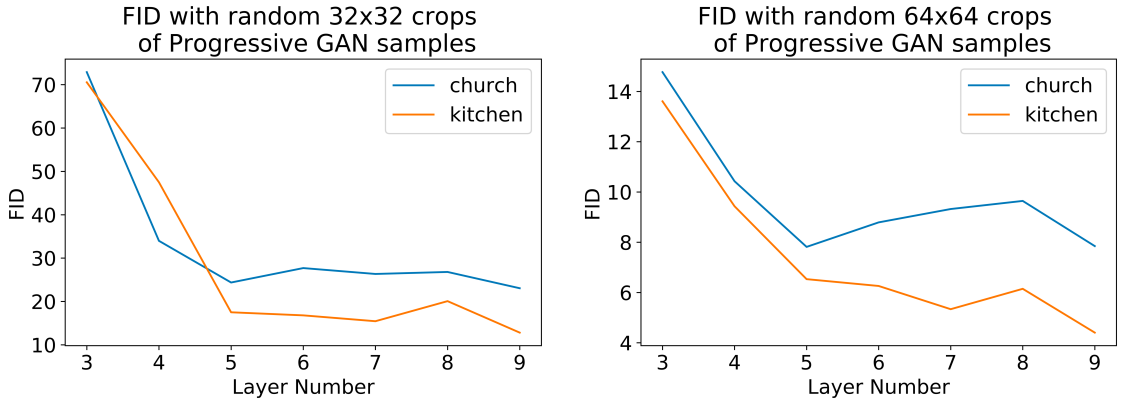


Figure B-10: FID of rendered cropped activations with respect to random crops of Progressive GAN generated images.

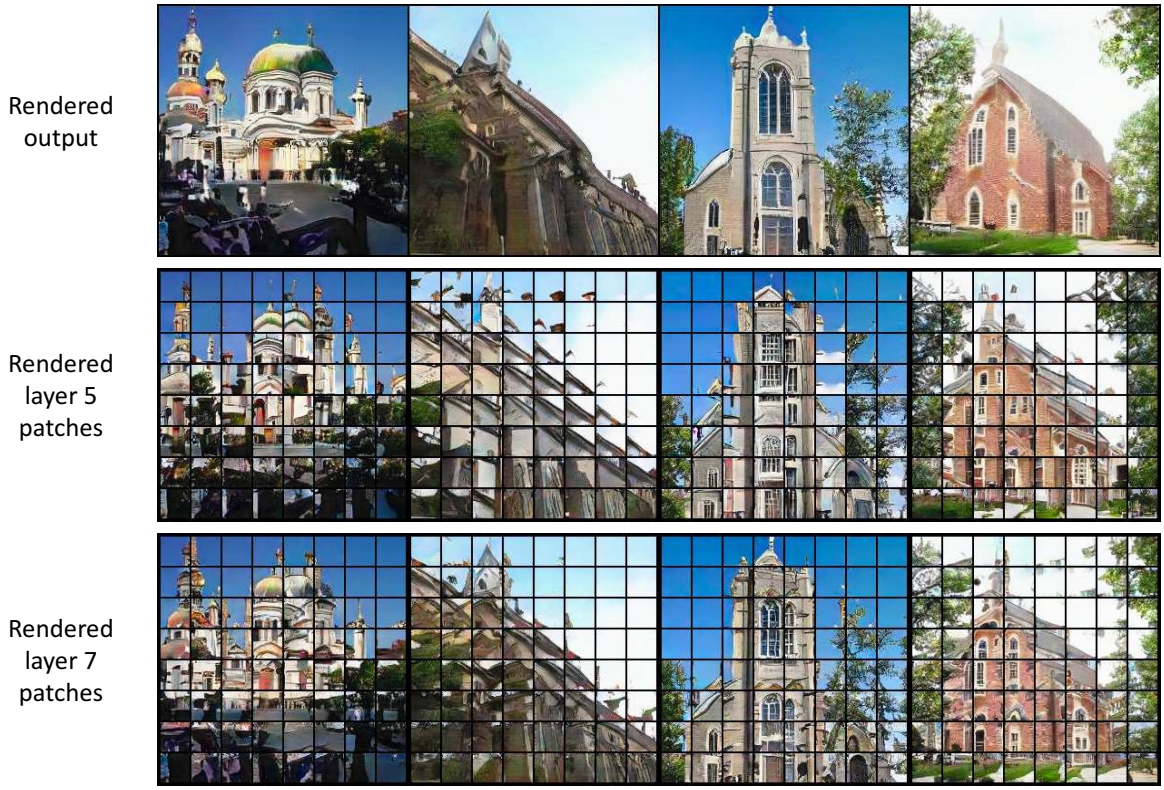


Figure B-11: Comparison of rendered cropped activations at various layers of Progressive GAN generated LSUN church images.

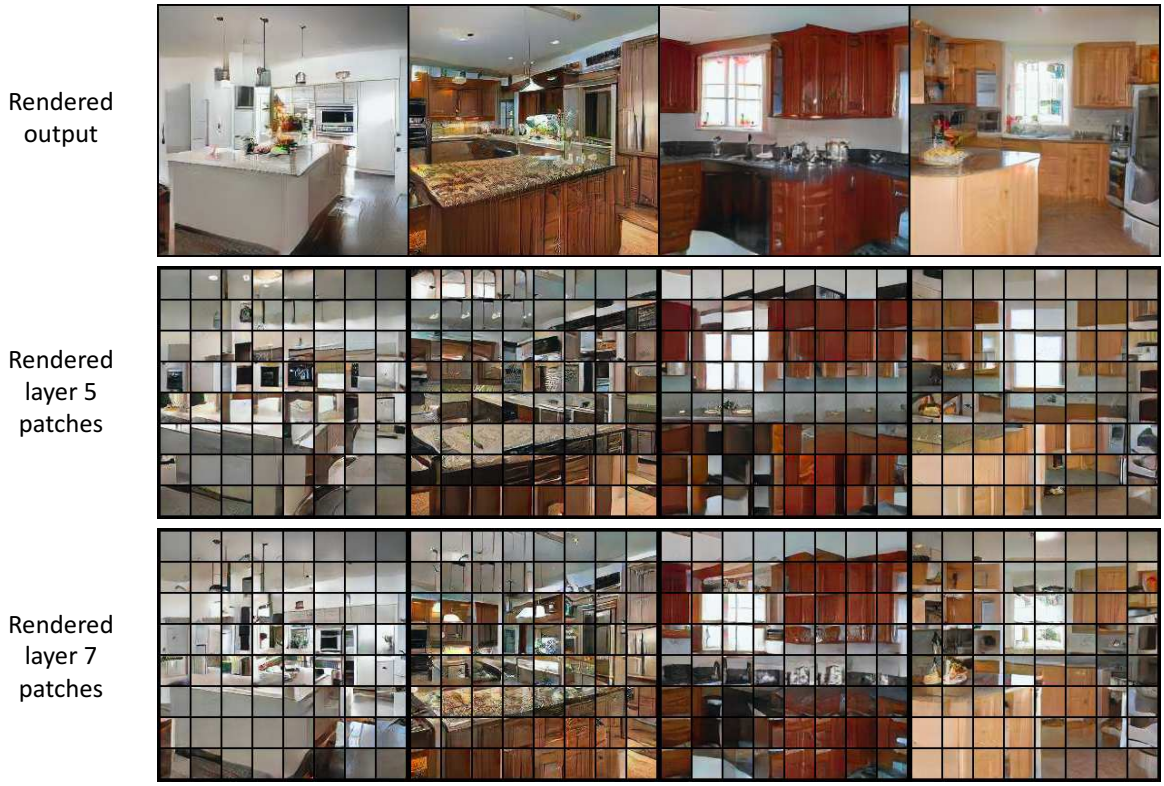


Figure B-12: Comparison of rendered cropped activations at various layers of Progressive GAN generated LSUN kitchen images.

Bibliography

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Zhou Bolei, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. Gan dissection: Visualizing and understanding generative adversarial networks. In *ICLR*, 2019.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

Minyoung Huh, Richard Zhang, Jun-Yan Zhu, Sylvain Paris, and Aaron Hertzmann. Transforming and projecting images to class-conditional generative networks. In *ECCV*, 2020.

Tero Karras. FFHQ dataset. `https://github.com/NVlabs/ffhq-dataset`, 2019.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Effrosini Kokiopoulou, Jie Chen, and Yousef Saad. Trace optimization and eigenproblems in dimension reduction methods. *Numerical Linear Algebra with Applications*, 18(3):565–602, 2011.

Ankit Sharma and Hassan Foroosh. Slim-cnn: A light-weight cnn for face attribute prediction. In *International Conference on Automatic Face and Gesture Recognition*, 2020.

Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, 2018.

Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *ECCV*, 2018.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

ZLL. Face-parsing pytorch. `https://github.com/zllrunning/face-parsing.PyTorch`, 2019.