Title

Structure and Representation in Neural Networks

Permalink

https://escholarship.org/uc/item/3qb142b4

Author

Filan, Daniel

Publication Date

2024

Structure and Representation in Neural Networks

By

Daniel Filan

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Stuart Russell, Chair
Professor Jitendra Malik
Assistant Professor Jacob Steinhardt

Spring 2024

Structure and Representation in Neural Networks

Abstract

Structure and Representation in Neural Networks

by

Daniel Filan

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Stuart Russell, Chair

Since neural networks have become dominant within the field of artificial intelligence, a sub-field of research has emerged attempting to understand their inner workings. One standard method within this sub-field has been to primarily understand neural networks as representing human-comprehensible features. Another possibility that has been less explored is to understand them as multi-step computer programs. A seeming prerequisite for this is some form of modularity: for different parts of the network to operate independently enough to be understood in isolation, and to implement distinct interpretable sub-routines.

To find modular structure inside neural networks, we initially use the tools of graphical clustering. A network is clusterable in this sense if it can be divided into groups of neurons with strong internal connectivity but weak external connectivity. We find that a trained neural network is typically more clusterable than randomly initialized networks, and often clusterable relative to random networks with the same distribution of weights as the trained network. We investigate factors that promote clusterability, and also develop novel methods targeted at that end.

For modularity to be valuable for understanding neural networks, it needs to have some sort of functional relevance. The type of functional relevance we target is local specialization of functionality. A neural network is locally specialized to the extent that parts of its computational graph can be abstractly represented as performing some comprehensible sub-task relevant to the overall task. We propose two proxies for local specialization: importance, which reflects how valuable sets of neurons are to network performance; and coherence, which reflects how consistently their neurons associate with features of the inputs. We then operationalize these proxies using techniques conventionally used to interpret individual neurons, applying them instead to groups of neurons produced by graph clustering algorithms. Our results show that clustering succeeds at finding groups of neurons that are important and coherent, although not all groups of neurons found are so.

We conclude with a case study of using more standard interpretability tools, designed to understand the features being represented by directions in activation space, applying them to the analysis of neural networks trained on the reward function of the game CoinRun. Despite our networks achieving a low test loss, the application of interpretability tools shows that networks do not adequately represent relevant features and badly mispredict reward out of distribution. That said, these tools do not reveal a clear picture of what computation the networks are in fact performing. This not only illustrates the need for better interpretability tools to understand generalization behaviour, but motivates it: if we take these networks as models of 'motivation systems' of policies trained by reinforcement learning, the conclusion is that such networks may competently pursue the wrong objectives when deployed in richer environments, indicating a need for interpretability techniques to shed light on generalization behaviour.

To the academy

Your long tradition of inquiry and teaching has further and further illuminated our universe, rendering it more and more comprehensible, and allowing greater and greater numbers of people to find joy in understanding. May this dissertation play some small part in furthering your noble work.

Contents

List of Figures

List of Tables

Acknowledgments

I'd first like to thank my advisor, Stuart Russell, for his support while I've been in graduate school. I'd also like to thank the members of my thesis committee, Jacob Steinhardt and Jitendra Malik, as well as Trevor Darrell for being on my qualifying exam committee.

I am also indebted to my collaborators on the research presented in this dissertation. These are Shlomi Hod, Stephen 'Cas' Casper, Cody Wild, and Andrew Critch for the research in chapters 3 and 4, and Pavel Czempin and Adam Gleave for the research in chapter 5. I hesitate to pronounce anything 'impossible', but without them it would at least have been very difficult to produce this dissertation. I would also like the many people who overlapped with me at the Center for Human-Compatible AI and Stuart Russell's research group, for creating a friendly research group focussed on important problems. These include Mark Nitzberg, Rosie Campbell, Martin Fukui, J. P. Gonzales, Judy Tam, Andrew Critch, Dylan Hadfield-Menell, Thanard Kurutach, Michael Dennis, Adam Gleave, Rohin Shah, Jaime Fernandez Fisac, Tom Gilbert, Sam Toyer, Rachel Freedman, Scott Emmons, Lawrence Chan, Cassidy Laidlaw, Niklas Lauffer, Alyssa 'Li' Dayan, Micah Carroll, Anand Siththaranjan, Arnaud Fickinger, Yuxi Liu, Hanlin Zhu, Erik Jenner, Johannes Treutlein, Shreyas Kapur, Bhaskar Mishra, Jiahai Feng, Jonathan Stray, Alex Turner, Ben Plaut, Cam Allen, Jacy Reese, Justin Svegliato, Michael Cohen, Caroline Jeanmaire, Davis Foote, Nisan Stiennon, Cody Wild, Erdem Bıyık, Smitha Milli, Stephen Wang, Vael Gates, Brian Christian, Tom Lenaerts, Vincent Corruble, Alina Trinh, George Matheos, Chris Cundy, Shlomi Hod, Stephen 'Cas' Casper, and Neel Nanda, as well as other postdocs, interns, visiting scholars, and staff members dedicated to keeping things running smoothly. In particular, I appreciated my many discussions with Dylan Hadfield-Menell, Michael Dennis, Adam Gleave, and Rohin Shah in my first two years at UC Berkeley to help me orient to the field of AI alignment.

I would also like to specifically thank Andrew Critch, Paul Christiano, and Chris Olah for early conversations that helped shape the research in this dissertation.

Marcus Hutter was my undergraduate honours advisor, and the first person who advised me on an AI research project. I'd like to thank him for being an attentive advisor and helping me start a career in AI research.

Broadly, I would not have become interested in this research direction were it not for the LessWrong community's discussion of the dangers of advanced artificial intelligence. I'd like to thank Buck Shlegeris for introducing me to the LessWrong community and the idea of existential risk from artificial intelligence, Eliezer Yudkowsky for his voluminous writings on the subject, and the developers of LessWrong between 2011 and today.

I have used a variety of tools during the creation of this dissertation, whose developers I feel could use more appreciation. In that spirit, I'd like to thank the creators and maintainers of Google Scholar, Emacs, Overleaf, Focusmate, and Logseq.

I would finally like to thank my parents, Susan and Andrew Filan, for proofreading this dissertation and for other services provided over the years.

Chapter 1

Introduction

For the past several years, the field of artificial intelligence has learned how to train neural networks to perform a startlingly wide array of tasks to a startlingly high degree of performance. Well-known examples include recognizing images (Deng et al., 2009; He et al., 2016a), playing Atari games (Badia et al., 2020; Bellemare et al., 2013) and Go (Silver et al., 2018), predicting how proteins fold (Jumper et al., 2021), producing illustrations (Ramesh et al., 2022), modelling human text (OpenAI, 2023), and solving high school math competition problems (Hendrycks et al., 2021; A. Zhou et al., 2023).

As this has happened, much work in the sub-field of interpretability has been put into understanding how these neural networks operate (Räuker et al., 2023). Several motivations exist for this line of research (Lipton, 2018), but one is a concern about goal misgeneralization (Di Langosco et al., 2022; Hubinger et al., 2019; Shah et al., 2022). The concern can be put in this way: neural network architectures are expressive enough to admit a variety of parametrizations that competently pursue various goals, in the sense of reliably acting in ways that bring about certain types of outcomes. Furthermore, the space of goals is rich enough that two different parametrizations can behave identically on a training set while generalizing competently but differently in richer environments. Potential examples of this kind of misgeneralization range from the mundane—agents trained to reach an object that is always at a certain location in the training set will go to the location, not the object, on a test set where those diverge (Di Langosco et al., 2022)—to the catastrophic—an agent that bides its time during training, achieving high performance on its assigned task, until it knows it has been deployed, at which point it pivots to achieving an arbitrary goal contrary to human intent (which in order to achieve, it needed to perform well enough in the training set to be deployed) (Bostrom, 2014, Chapter 8). In order to foresee and prevent goal misgeneralization, it would be valuable to understand the computation inside a neural network, ideally resulting in a comprehensible model of how it will behave in new situations.

To understand the computation inside a neural network, we likely will need some sort of 'frame': a theory of what sort of computation is likely being implemented, and which network components correspond to which computational elements. One of the first textbooks devoted to deep learning provides two such frames (Goodfellow et al., 2016). In its introduction, it provides an “Illustration of a deep learning model” adapted from work by Zeiler and Fergus (2014) and describes it thus:

> Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the first hidden layer’s description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the second hidden layer’s description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image.

This provides a frame of ‘natural feature representation’ whereby, especially in later layers in neural networks, the neurons (or at least some basis of the activation space) should be thought of as representing high-level human-comprehensible features. This frame suggests that the task of neural network interpretability is to determine what those features are.

However, the book also describes another frame:

> Another perspective on deep learning is that depth enables the computer to learn a multistep computer program. Each layer of the representation can be thought of as the state of the computer’s memory after executing another set of instructions in parallel. Networks with greater depth can execute more instructions in sequence. Sequential instructions offer great power because later instructions can refer back to the results of earlier instructions. According to this view of deep learning, not all the information in a layer’s activations necessarily encodes factors of variation that explain the input. The representation also stores state information that helps to execute a program that can make sense of the input. This state information could be analogous to a counter or pointer in a traditional computer program. It has nothing to do with the content of the input specifically, but it helps the model to organize its processing.

This suggests a more computational view taken by the literature on mechanistic interpretability (Cammarata et al., 2020; Olah, 2022), that we should be interested in how computation is organized in the network, what subroutines there may be, and how information is transformed between layers. Such a view places less emphasis on activations as representing comprehensible features of the input. Section 6.4.1 of the textbook keeps both perspectives in mind:

> Choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions. This can be interpreted from a representation learning point of view as saying that we believe the learning problem consists of discovering a set of underlying factors of variation that can in turn be described in terms of other, simpler underlying factors of

> variation. Alternately, we can interpret the use of a deep architecture as expressing a belief that the function we want to learn is a computer program consisting of multiple steps, where each step makes use of the previous step's output. These intermediate outputs are not necessarily factors of variation but can instead be analogous to counters or pointers that the network uses to organize its internal processing.

Broadly, this dissertation will more take the latter, more computational perspective, and cast some suspicion on the natural feature representation frame.

However, if we are to understand neural networks in terms of multi-step computer programs that they implement, we need some automated method to find non-trivial structure in the weights that meaningfully corresponds to elements of the neural network's computation. This structure must be higher-level than single neurons or parameters to be useful, because modern neural networks are too large for it to be practicable to inspect them neuron-by-neuron, weight-by-weight: they have as many as trillions of parameters (Villalobos et al., 2022), and each neuron can be connected to over 10,000 other neurons (Brown et al., 2020). However, there is no widely-agreed-upon way of finding such structure without having already done an interpretability analysis, or specializing to a particular distribution or input.

Modularity is a common property of biological and engineered systems (Baldwin & Clark, 2000; Booch et al., 2007; Clune et al., 2013), and also serves as an attractive way to organize our understanding of neural networks. Just as modularity in computer programming can allow us to understand each part in a program in isolation, helping us to understand the whole, modularity in neural networks could offer the same benefits. Modularity has two components: structural separation and functional specialization. Structural separation is required for there to be any 'modules' to speak of, and in order for it to be possible to understand the network by understanding the modules, each module must somehow meaningfully specialize in some sort of functionality. One simple way to define structural separation would be to use the language of graph clustering (Schaeffer, 2007): set in the context of weighted graphs, clusters are groups of vertices where the total weight of edges within the group is large in comparison to the weight of edges from a vertex in the group to a vertex outside the group. This can be applied to neural networks by treating neurons as vertices and taking the absolute value of the network weights to be the weights of edges between neurons in adjacent layers.

In this dissertation, we find evidence of cluster structure in the weights of image processing neural networks and show that these clusters have a kind of functional relevance that random groups of neurons do not. Dropping them out damages accuracy more than dropping out random groups of neurons, and they are easier to jointly activate than random groups of neurons, but they do not seem to specialize in inputs with specific labels. We also conduct a case study of networks used for modelling the reward functions of image-based environments. Consonant with the above results, we fail to find natural high-level features being faithfully represented by the networks in any layer, and exhibit classes of inputs where these networks reliably mispredict the reward, demonstrating that they cannot be representing and properly processing these high-level features, and motivating concerns about goal misgeneralization.

1.1 Overview

In Chapter 2, we review related work that forms a useful background to this paper. Topics we touch on include mechanistic interpretability, structure in neural networks, adversarial examples, other frames on neural network interpretability, and goal misgeneralization.

In Chapter 3, we study the graph clusterability structure of neural networks trained on standard image classification tasks. We show that such neural networks are often "clusterable" in the sense that the spectral clustering algorithm finds partitions of the neurons that score lower on the normalized cut metric (Shi & Malik, 2000) than the partitions found by spectral clustering on random neural networks. We investigate training conditions that promote this clusterability: weight pruning, dropout, and in some cases L_1 and L_2 regularization. We also develop and test novel regularization methods to promote clusterability, and show that they have some success on multi-layer perceptrons. We show that clusterability is associated with reducing training set error. Finally, we do some exploratory analysis of which datasets induce clusterability, finding ambiguous results that are not predicted by any simple theory.

In Chapter 4, we study the functional relevance of clusters of neurons by testing for "local specialization": the network's use of certain clusters to do specific computations. To operationalize local specialization, we break it down into properties of clusters called "importance" and "coherence". By "importance", we roughly mean whether the cluster contributes to network accuracy (compared to if it were dropped out), and by "coherence", we roughly mean whether the neurons consistently fire on inputs with similar features. We develop automated metrics for measuring importance and coherence, show that graph clustering algorithms find important and coherent clusters, and identify the ways in which the clusters are coherent: specifically, that it is easier to jointly induce high activations in clusters than in other groups of neurons, but that clusters do not seem to 'specialize' in helping with some classes but not others. Overall, this shows that networks do locally specialize their computation, and that this local specialization matches the cluster structure of the network weights.

In Chapter 5, we provide a case study of some neural networks trained on a reward function. Related to our failure to find cluster specialization with respect to class label, we see if we can probe these networks for natural high-level features relevant to classification. The reward function we use is that of the Procgen game 'CoinRun' (Cobbe et al., 2020). In this game, the player controls an agent sprite that has to reach a coin. Reward is zero until distance to the coin is below a (low) threshold, at which point reward reaches 10 and the game ends. We show that the networks cannot be linearly probed for the distance to the coin to a high degree of accuracy, and that the networks can be 'fooled' by frames where the agent sprite jumps up rather than down to the coin—falsifying the proposition that the network represents distance to the coin and uses it appropriately. This represents a novel example of goal misgeneralization, one of the concerns motivating this dissertation.

Finally, Chapter 6 summarizes the dissertation and describes future work to better understand and manipulate the cluster structure of neural networks.

Chapter 2

Related work

This chapter presents an overview of research that provides a useful background to the rest of this dissertation.

This research aligns with the sub-field of artificial intelligence known as neural network interpretability, focussed on the general question of understanding how neural networks work. Surveys by Samek et al. (2021) and Räuker et al. (2023) provide useful overviews of the literature on this topic.

The dissertation is more closely related to the sub-sub-field of mechanistic interpretability. The literature in this area has blossomed since Cammarata et al. (2020) showed that neural networks used in image classification contain ‘circuits’: sub-networks that perform comprehensible computation on comprehensible features, such as a car detector filter that linearly combines a window detector, a car body detector, and a wheels detector. Most relevant to this dissertation is mechanistic interpretability research that focusses on formalizing the notion of structures inside neural networks and how to detect them, rather than manually finding such structures. For example, Elhage et al. (2021) provide a convenient formalism for describing computation in transformers, which was used in the discovery of induction heads: portions of a network that find previous occurrences of an input token and predict what followed those tokens (Olsson et al., 2022). Elhage et al. (2023) later describe why a certain basis of the residual stream of transformers is privileged over other bases under Adam optimization (Kingma & Ba, 2014), and how that impacts that formalism. Chan et al. (2022) provide a definition of the degree to which a neural network structure contributes to performance on a certain task and an ablation-based algorithm to measure this degree. Conmy et al. (2023) outline a method to automatically find circuits in neural networks responsible for certain tasks. Most recently, Bricken et al. (2023) use dictionary learning to find an overcomplete basis of the activation space of transformers that is ‘monosemantic’: that is, that each basis vector corresponds to a single human-comprehensible concept.

There has also been work outside the mechanistic interpretability literature that focusses on finding structure in neural networks. Examples include work by Frankle and Carbin (2019), who discover that trained neural networks contain efficiently-trainable sub-networks, inspiring multiple follow-up papers (Frankle et al., 2020; H. Zhou et al., 2019), and You et al. (2020), who form a ‘relational graph’ from a neural network, and study how network performance relates to properties of the relational graph. There is also a literature on modularity in neural networks, which is covered in Section 3.6.

One challenge to the view of neural networks as representing natural features is the literature on adversarial examples. This literature was first developed by Szegedy et al. (2014) and Nguyen et al. (2015), who discovered that image classification networks had certain flaws: imperceptible perturbations could be applied to images to prevent them from being classified correctly, and unrecognizable images could be created that were classified with high confidence. Given the mismatch with human vision, these examples indicate that image classification networks cannot be representing the same sorts of features as humans do and also using those features in the same way that humans do. To this day, adversarial robustness remains an unsolved problem (Croce et al., 2024; Croce et al., 2020). Furthermore, at least some adversarial examples are not just accidents of training, but result from features that are highly predictive on the training dataset (Engstrom et al., 2019; Ilyas et al., 2019).

Beyond the frames for thinking about neural network computation quoted earlier, some others exist. One prominent frame is that of the neural tangent kernel (Jacot et al., 2018). This refers to the result that at infinite width, when using standard weight initialization distributions, neural networks are equivalent to a kernel method using a specific kernel known as the neural tangent kernel, and the process of gradient descent on neural networks is equivalent to kernel gradient descent using the neural tangent kernel. This would suggest focussing on the eigenfunctions of the neural tangent kernel as a key to understanding network behaviour (Bordelon et al., 2020; Simon et al., 2022). That said, flaws with the neural tangent kernel have motivated discussions of extensions of the research: firstly, into a different initialization method where, unlike the neural tangent kernel, feature learning takes place at infinite width, and characterizing this infinite-width behaviour (Yang & Hu, 2021); and secondly, looking at effects that are first-order in inverse width that characterize the difference between large-but-finite width and infinite width neural networks (Roberts et al., 2022). It remains to be seen what interpretability methods these extensions will inspire.

Another framework for thinking about neural network learning is singular learning theory (S. Watanabe, 2009, 2018). To somewhat simplify, singular learning theory studies Bayesian learning in the setting where whole algebraic varieties inside a parameterized model class induce the same distribution over evidence—for example, where for some data, the set of minimal-loss parameters is a circle or a figure 8. Neural networks are such a model class, in part due to continuous symmetries of networks that use the ReLU activation function (Petzka et al., 2020), but also because a network of a given width can simulate a network of a smaller width by setting the vector of input weights to multiple different neurons to be proportional, and one can continuously interpolate between various ways of doing this (Nagayasu & Watanabe, 2023; Şimşek et al., 2021). The stylized result of singular learning theory is that in cases where the set of minimal-loss parameters is something like a circle or a figure 8 rather than a set of isolated points, the posterior will converge on this set more quickly than classical statistical learning theory would predict. Specifically, if the set of optimal parameters has points in them that prevent them from being a manifold (like the point in a figure 8 where two lines cross), the posterior will converge to those points, known as 'singularities'. Furthermore, the more 'complex' the singularity is (roughly, the more lines that cross at the singularity), the sooner the posterior accumulates at it. Singular learning theory also features the phenomenon of phase transitions, where posterior mass suddenly switches from complex singularities that model the data well to simpler singularities that model the data better.

In recent years, singular learning theory has gained some steam as a way of thinking about neural networks (Lau et al., 2023; Wei et al., 2022), in part because it offers an explanation of why overparameterized networks can learn to generalize on natural distributions while still being able to memorize random noise (since memorization that critically relies on all parameters is a less complex singularity than simulating a neural network of fewer parameters), and in part because it is able to retrodict phase transitions seen in toy deep learning problems (Chen et al., 2023; Elhage et al., 2022). It is also conceptually related to the mode connectivity line of research, which studies cases where two neural network parameterizations that induce zero loss can be continuously interpolated between while maintaining near-zero loss along the path (Draxler et al., 2018; Freeman & Bruna, 2017; Garipov et al., 2018). This way of thinking motivates understanding neural networks via the phase transitions that they went through during training (Hoogland et al., 2023), as well as understanding the set of zero- or low-loss parameters connected to the final network (Juneja et al., 2022; Lubana et al., 2023).

Finally, the motivation of this dissertation is concern about goal misgeneralization, where an AI system behaves as desired during training, but at some point during deployment competently pursues a goal other than the desired one, perhaps catastrophically. This concern was expressed in academic literature by Bostrom (2014, Chapter 8) under the term "treacherous turn". Hubinger et al. (2019) then fleshed it out under the terms "mesa-optimization" and "inner (mis)alignment". Further attempts at formalization and examples are given by Di Langosco et al. (2022), who provide a definition and examples in the reinforcement learning (RL) setting, and Shah et al. (2022), who provide a broader definition for learning systems in general and examples both within and outside the context of RL. The issue is discussed by Ngo et al. (2022), who also identify interpretability research as a countermeasure, and by Carlsmith (2023) at great depth and less technically. The problem can be cast as a specific instance of distributional shift, identified as an issue for AI safety by Amodei et al. (2016), and about which there is a wealth of literature which is beyond the scope of this review.

Chapter 3

Clusterability in Neural Networks

3.1 Introduction

As mentioned in the introduction of this dissertation, for the purposes of understanding neural networks, it would be convenient if they were modular. Consider the view described by Goodfellow et al. (2016), quoted in Chapter 2: “Another perspective on deep learning is that depth enables the computer to learn a multistep computer program. Each layer of the representation can be thought of as the state of the computer’s memory after executing another set of instructions in parallel.” Just as modularity in computer programming can allow us to understand each part in a program in isolation, helping us to understand the whole, modularity in neural networks could offer the same benefits. This can be especially attractive when contrasted with an alternative of understanding each neuron individually, given the large number of neurons in modern networks, and given that in non-modular systems, each component can only be understood in light of a large fraction of the other components of the system.

In this chapter, we study a graph-theoretic analog to modularity: the extent to which a network can be partitioned into sets of neurons where each set is strongly internally connected, but only weakly connected to other sets. This definition refers only to the learned weights of the network, not to the data distribution, nor to the distributions of outputs or activations of the model. More specifically, we use a spectral clustering algorithm (Shi & Malik, 2000) to decompose trained networks into clusters, and measure the goodness of this decomposition. Since any degree of non-uniformity of weights can induce clusterability, we also measure the relative clusterability of a network compared to networks with the same set of weights in each layer but shuffled randomly, in order to determine whether any clusterability is simply due to each layer’s distribution of weights.

We conduct an empirical investigation into the clusterability of multi-layer perceptrons (MLPs) and convolutional neural networks (CNNs) trained on MNIST, Fashion-MNIST, and CIFAR-10 (Krizhevsky, 2009; LeCun et al., 1998; Xiao et al., 2017), using weight pruning and other regularization methods. We also test if clusterability can be induced by training

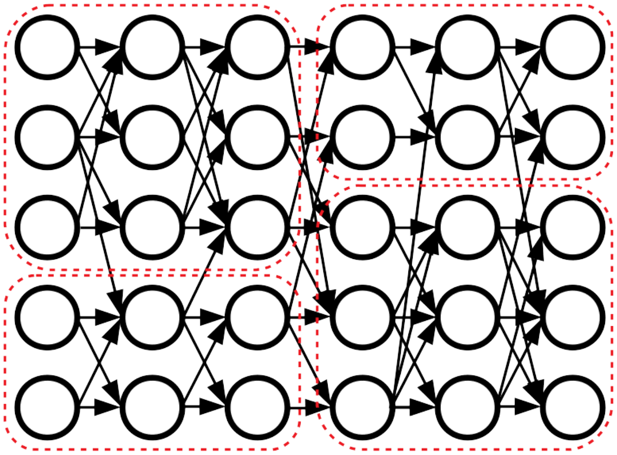


Figure 3.1: A pruned neural network, split into clusters.

on datasets that benefit from some degree of parallelism to classify. In addition, we test the clusterability of neural networks trained by other researchers in the VGG, ResNet, and Inception families (He et al., 2016a; Simonyan & Zisserman, 2015; Szegedy et al., 2016) for ImageNet classification (Deng et al., 2009). Finally, we investigate two ways of training neural networks specifically to promote clusterability: regularizing for clusterability and clusterable initialization.

Our main contributions are:

- Presenting a definition of absolute and relative clusterability of a neural network (section 3.2).
- Showing that trained neural networks are often more absolutely clusterable than randomly initialized networks (sections 3.3 and 3.4).
- Showing that neural networks trained with dropout and/or weight pruning are typically relatively clusterable, and often more clusterable than all 50 shuffled networks to which we compare them (sections 3.3 and 3.4).
- Showing that large neural networks trained for state-of-the-art ImageNet classification are reliably more clusterable than 99 out of 100 shuffled networks, and that a mid-sized VGG trained with dropout and weight pruning on CIFAR-10 classification reliably produces networks more clusterable than all 50 of their shuffles (section 3.4).
- Demonstrating novel methods of promoting clusterability in MLPs in a way that is compatible with standard neural network architectures and training procedures, with little loss in accuracy (section 3.5).

Code is available at https://github.com/dfilan/clusterability_in_neural_networks.

3.2 Clustering neural networks

3.2.1 Definitions

We represent a neural network as a weighted, undirected graph G. To do this for an MLP, we identify each neuron with any incoming or outgoing non-zero weights[1], including the pixel inputs and logit outputs, with an integer between 1 and N, where N is the total number of neurons, and take the set of neurons to be the set V of vertices in G. Two neurons have an undirected edge between them if they are in adjacent layers, and the weight of the edge is equal to the absolute value of the weight of the connection between the two neurons. We represent the set of weights by the adjacency matrix A defined by $A_{ij} = A_{ji} :=$ the edge weight between neurons i and j. If there is no edge between i and j, then $A_{ij} = A_{ji} := 0$. As such, A encodes all the weight matrices of the neural network, but not the biases.

For a CNN, the 'neurons' we use are channels of the hidden layers—we omit the input layer and fully-connected layers at the end of the network. In this case, the weight of the edge between two channels is the L_1 norm of the two-dimensional slice of the convolutional filter by which the input channel maps to the output channel.[2] When batch normalization is used between two convolutional layers, we divide weights by the moving standard deviation and multiply them by the scaling factor γ.

The degree of a neuron i is defined by $d_i := \sum_j A_{ij}$. The degree matrix D is a diagonal matrix where the diagonal elements are the degrees: $D_{ii} := d_i$. We define the volume of a set of neurons $X \subseteq V$ as $\operatorname{vol}(X) := \sum_{i \in X} d_i$, and the weight between two disjoint sets of neurons $X, Y \subseteq V$ as $W(X, Y) := \sum_{i \in X, j \in Y} A_{ij}$. If $X \subseteq V$ is a set of neurons, then we denote its complement as $\bar{X} := V \setminus X$.

A partition of the network is a collection of disjoint subsets $X_1, \ldots, X_k \subseteq V$ whose union forms the whole vertex set. Our 'goodness measure' of a partition is the normalized cut metric (Shi & Malik, 2000) defined as $\text{n-cut}(X_1, \ldots, X_k) := \sum_{i=1}^{k} W(X_i, \bar{X}_i)/\operatorname{vol}(X_i)$, which we call 'n-cut' in text. The n-cut will be low if neurons in the same partition element tend to share high-weight edges and those in different partition elements share low-weight edges or no edges at all, as long as the sums of degrees of neurons in each partition element are roughly balanced. For a probabilistic interpretation of n-cut that gives more intuitive meaning to the quantity, see subsection 3.2.2.

Finally, the graph Laplacian is defined as $L := D - A$,[3] and the normalized Laplacian as $L_{\text{norm}} := D^{-1}L$. L_{norm} is a positive semi-definite matrix with N real-valued non-negative eigenvalues (von Luxburg, 2007). The eigenvectors and eigenvalues of L_{norm} are the generalized eigenvectors and eigenvalues of the generalized eigenvalue problem $Lu = \lambda Du$.

[1] When networks are pruned, often some neurons have all incident weights pruned away, leaving them with no functional role in the network. We ignore these neurons in order to run spectral clustering.

[2] We also tried constructing the graph with the L_2 norm, and found essentially similar results.

[3] For the connection to the second derivative operator on $\mathbb{R}^n$, see Filan (2022) and von Luxburg (2007).

Algorithm 1 Normalized Spectral Clustering

```
  **Input:** Adjacency matrix $A$, number $k$ of clusters
  Compute the normalized Laplacian $L_{\text{norm}}$
  Compute the first $k$ eigenvectors $u_1, \dots, u_k \in \mathbb{R}^N$ of $L_{\text{norm}}$
  Form the matrix $U \in \mathbb{R}^{k \times N}$ whose $j^{\text{th}}$ row is $u_j^\top$
  For $n \in \{1, \dots, N\}$, let $y_n \in \mathbb{R}^k$ be the $n^{\text{th}}$ column of $U$
  Cluster the points $(y_n)_{n=1}^N$ with the $k$-means algorithm into clusters $C_1, \dots, C_k$
  **Return:** Clusters $X_1, \dots, X_k$ with $X_i = \{n \in \{1, \dots, N\} \mid y_n \in C_i\}$
```

3.2.2 Probabilistic interpretation of n-cut

As well as the formal definition given in Subsection 3.2.1, n-cut has a more intuitive interpretation. Note that a similar argument appears in Meilă and Shi (2001).

Divide each edge between vertices i and j into two 'stubs', one attached to i and the other attached to j, and associate with each stub the weight of the whole edge. Now: suppose $(X_1, \dots, X_k)$ is a partition of the graph. First, pick an integer l between 1 and k uniformly at random. Secondly, out of all of the stubs attached to vertices in X_l, pick one with probability proportional to its weight. Say that this procedure 'succeeds' if the edge associated with that stub connects two vertices inside X_l, and 'fails' if the edge connects a vertex inside X_l with a vertex outside X_l. The probability that the procedure fails is n-cut$(X_1, \dots, X_k)/k$.

Therefore, the n-cut divided by k is roughly a measure of what proportion of edge weight coming from vertices inside a partition element crosses the partition boundary.

3.2.3 Spectral clustering

To measure the clusterability of a graph, we use a spectral clustering algorithm to compute a partition, which we call a clustering, and evaluate the n-cut. The algorithm we use (Shi & Malik, 2000) solves a relaxation of the NP-hard problem of finding a clustering that minimizes the n-cut (von Luxburg, 2007). It is detailed in algorithm 1, which is adapted from von Luxburg (2007). We use the scikit-learn implementation (Pedregosa et al., 2011) using the ARPACK eigenvalue solver (Lehoucq et al., 1998).

We define the n-cut of a network as the n-cut of the clustering this Algorithm 1 returns, run with $k = 12$.[4] Since the n-cut is low when the network is clusterable, we will describe a decrease in n-cut as an increase in *absolute clusterability* and vice versa.

To measure the *relative clusterability* of an MLP, we sample 50 random networks by randomly shuffling the weight matrix of each layer of the trained network. We convert these networks to graphs, cluster them, and find their n-cuts. For CNNs, we shuffle the edge weights between channels once the network has been turned into a graph, which is equivalent to shuffling which two channels are connected by each spatial kernel slice. We then compare

[4] Subsection 3.3.6 shows results of some clusterability experiments for $k \in \{2, 4, 7, 10\}$. Results are similar for all values except $k = 2$.

the n-cut of the trained network to the sampled n-cuts, estimating the left one-sided p-value (North et al., 2002) and the Z-score: the number of standard deviations the network's n-cut lies below or above the mean of the shuffle distribution. This determines whether the trained network is more clusterable than one would predict based only on its sparsity and weight distribution.

3.3 Clusterability in MLPs

In this section, we report the results of experiments designed to determine the degree of clusterability of MLPs. For each experiment we train an MLP with 4 hidden layers, each of width 256, using Adam (Kingma & Ba, 2014). After the network has neared convergence, we train for additional epochs with weight pruning on a polynomial decay schedule (Zhu & Gupta, 2017) up to 90% sparsity. Pruning is used since the pressure to minimize connections plausibly causes modularity in biological systems (Clune et al., 2013).

During training, we use the Adam algorithm (Kingma & Ba, 2014) with the standard Keras hyperparameters: learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, no amsgrad. The loss function was categorical cross-entropy. For pruning, our initial sparsity is 0.5, our final sparsity is 0.9, the pruning frequency is 10 steps, and we use a cubic pruning schedule (see Zhu and Gupta (2017)). Initial and final sparsities were chosen due to their use in the TensorFlow Model Optimization Tutorial.[5] A batch size of 128 was used. Training went for 20 epochs before pruning, followed by 20 epochs of training with pruning. We use Tensorflow's implementation of the Keras API (Chollet et al., 2015; Martín Abadi et al., 2015).

First, we show results for MLPs trained on the MNIST and Fashion-MNIST datasets. We investigate networks trained with either no regularization, dropout with $p = 0.5$, L_1 regularization with weight 5×10^{-5}, or L_2 regularization with weight 5×10^{-5}. For each condition, we train for 5 runs, and check clusterability both right before pruning as well as at the end of training. In all conditions, networks train to $\sim$98% test accuracy on MNIST, and 87-89% test accuracy on Fashion-MNIST. At initialization, these networks have n-cuts of between 10.2 and 10.4, as plotted in Figure A.1.

When training with L_2 regularization, we occasionally found that networks had near-zero n-cut. This seemed to be due to extremely unbalanced clusterings where small groups of neurons were effectively disconnected from the rest of the network. When this happened, we retrained until the n-cut was significantly above zero, since we felt that the near-zero n-cuts did not reflect the true clusterability of the bulk of the network.

Results are shown in Figures 3.2 and 3.3, and Tables A.1 and A.2. We see that pruning promotes absolute clusterability, and dropout promotes absolute and relative clusterability. L_1 and L_2 regularization promote absolute clusterability pre-pruning, but not after pruning, at the expense of relative clusterability. All networks appear to be more clusterable than at initialization, except those trained with L_1 regularization and pruning.

[5] URL: https://web.archive.org/web/20190817115045/https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras

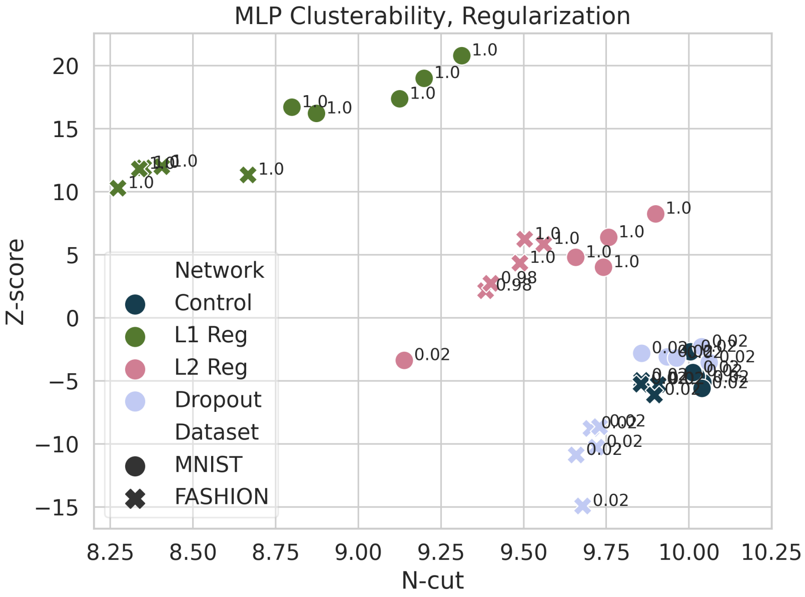


Figure 3.2: Clusterability of MLPs trained without pruning. Points are labeled with their one-sided p-value.

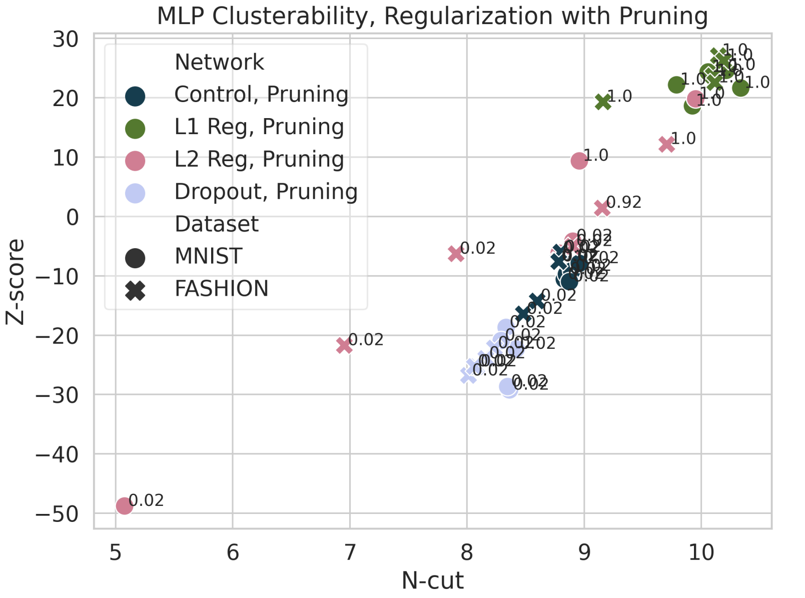


Figure 3.3: Clusterability of MLPs trained with pruning. Points are labeled with their one-sided p-value.

3.3.1 'Halves' datasets

We hypothesized that clusterability comes from different parts of the network independently computing different things. To test this, we trained on datasets that we hypothesized would lend themselves to parallel processing.

The first type of dataset, called 'MNIST-halves', features two MNIST images side-by-side, each shrunk in width so that the combined image is still 28 × 28 pixels. In 'MNIST-halves-same', both images are elements of the same class, and the class of the composite image is the class that the halves have. By contrast, in 'MNIST-halves-diff', the images come from random classes, and the class of the composite image is the sum of the classes of each half, modulo 10. For this task, it would be advantageous if the network could devote some neurons to processing one half, some to processing the other, and then combine the results. Since the neurons processing one half would not need information from those processing the other, we speculate that networks trained on 'halves-diff' will be more clusterable than those trained on those trained on 'halves-same'. We also make the same construction out of the Fashion-MNIST dataset, using the numerical class labels associated with the dataset. Samples from these datasets are shown in Figure 3.4. Networks train to around 99% accuracy on MNIST-halves-same, 92% on MNIST-halves-diff, 93-94% on Fashion-halves-same, and 71-72% on Fashion-halves-diff.

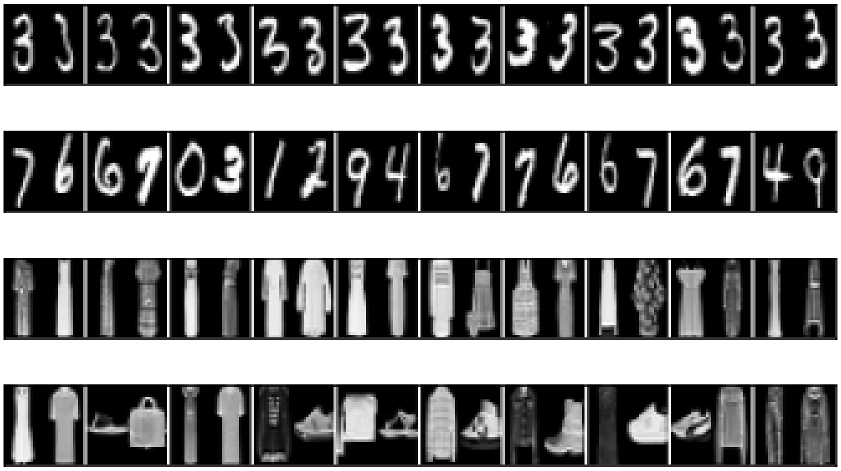


Figure 3.4: Samples from 'halves' datasets, all of class 3. Each row has 10 images from the respective dataset. First row is MNIST-halves-same, second is MNIST-halves-diff, third is Fashion-halves-same, fourth is Fashion-halves-diff.

Results are shown in Figure 3.5 and Table A.3. Networks trained on halves-diff datasets are more relatively clusterable than those trained on halves-same datasets, but not more absolutely clusterable. All networks are more clusterable than at initialization.

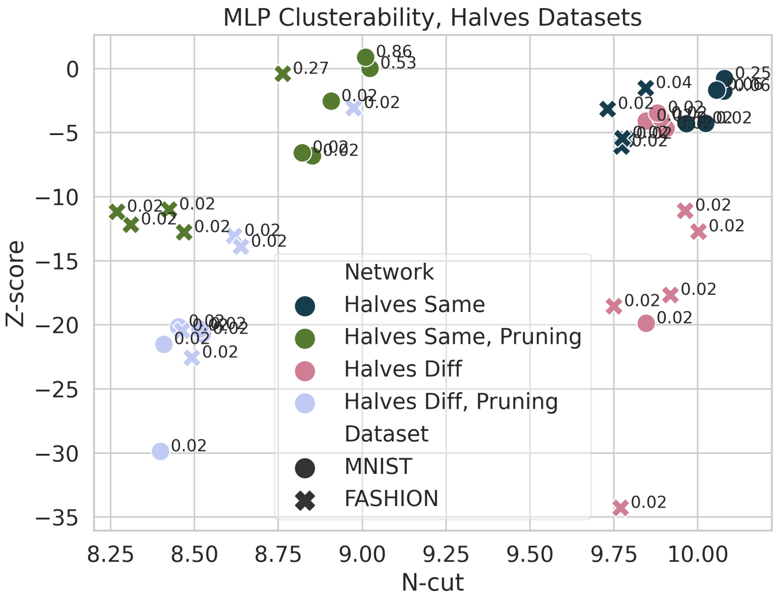


Figure 3.5: Clusterability of MLPs trained on 'halves' datasets. Points are labeled with their one-sided p-value.

3.3.2 Polynomial regression

We also train an MLP on a type of polynomial regression task, to see if this induces clusterability. The inputs to the network are two numbers, x and y, each independently normally distributed with mean 0 and variance 1. Each input is associated with a 512-dimensional label. Each label dimension is associated with one of the 512 polynomials in x and y with coefficients in $\{0, 1\}$ and exponents in $\{0, 1, 2\}$. For a given input, each dimension of the label has the value of the corresponding polynomial evaluated at that input. The network, with 4 hidden layers of 256 neurons each and 512 outputs, is trained to minimize mean square error between its outputs and the label. Test losses range from 0.005 to 0.16 without regularization (with most losses being approximately 0.01), 0.3 when L_1 regularization is used, and between 0.07 and 0.3 when L_2 regularization is used. Dropout seemed to harm performance and was therefore not used.

Since monomials like x^2y^2 can be computed based on monomials x^2 and y^2, and since each of the 8 valid non-constant monomials appears in 256 of the polynomials, one might expect that the network would adopt a clusterable structure, computing different monomials somewhat independently and combining their results in the output.

Results are shown in Figure 3.6 and Table 3.1. We see that networks trained on this task are consistently relatively clusterable, and that pruning enhances clusterability as usual.

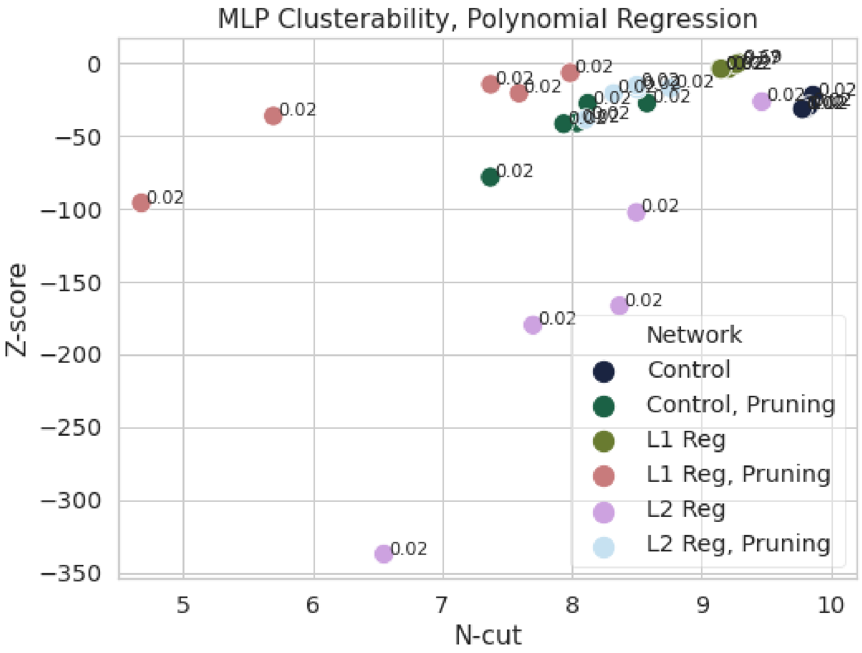


Figure 3.6: Clusterability of MLPs trained on polynomial regression. Points are labeled with their one-sided p-value.

Reg. method	Pruning?	N-cut	Dist. n-cuts	Prop. $p < 0.02$	Train loss	Test loss
None	$\times$	9.808	10.187 ± 0.013	5/5	0.078	0.046
None	$\surd$	8.009	9.301 ± 0.032	5/5	0.025	0.024
L_1	$\times$	9.210	9.278 ± 0.031	3/5	0.350	0.319
L_1	$\surd$	6.665	8.468 ± 0.059	5/5	0.211	0.192
L_2	$\times$	8.115	9.867 ± 0.012	5/5	0.175	0.140
L_2	$\surd$	8.438	9.800 ± 0.067	5/5	0.101	0.078

Table 3.1: Clusterability of MLPs trained on polynomial regression. See start of Appendix A for details. Losses averaged over 5 runs.

Dataset	N-cuts	Mean train acc.	Mean test acc.
MNIST	8.880 ± 0.049	1.000	0.984
LINES	7.361 ± 0.096	1.000	1.000
LINES-MNIST	7.955 ± 0.096	1.000	0.991
LINES-MNIST-SEP	8.26 ± 0.19	1.000	0.994

Table 3.2: N-cuts and accuracies for networks trained on mixture datasets without dropout. LINES-MNIST refers to the dataset where each class has data from both LINES and MNIST, while LINES-MNIST-SEP refers to the dataset where classes 0-4 have examples from LINES and classes 5-9 have examples from MNIST. Each row presents statistics for 5 networks. The "N-cuts" column shows the mean and standard deviation over 5 networks.

3.3.3 Mixture dataset experiments

We initially hypothesized that modularity is a result of different regions of the network processing different types of input. To test this, we developed mixture datasets composed of two original datasets. These datasets are either of the 'separate' type, where one original dataset has only classes 0 through 4 included and the other has classes 5 through 9 included; or the 'overlapping' type, where both original datasets contribute examples of all classes. The datasets that we mix are MNIST and LINES, which consists of 28×28 images of white vertical lines on a black background, labeled with the number of vertical lines.

If modularity were a result of different regions of the network specializing in processing different types of information, we would expect that networks trained on mixture datasets would have n-cuts lower than those trained on either constituent dataset. This is not what we observe.

Table 3.2 shows n-cuts and accuracies for networks trained with pruning but without dropout, while Table 3.3 shows the same for networks trained with pruning and dropout. Networks trained on mixtures between LINES and MNIST have n-cuts intermediate between those trained on LINES and those trained on MNIST. That being said, the artificial nature of the LINES dataset, as well as the low test accuracy of networks trained with dropout on LINES (seemingly implying that dropout in this case increased the degree of overfitting), put the generalizability of these results into question.

3.3.4 Random dataset experiments

There are two potential dataset-agnostic explanations for why clusterability would increase during training. The first is that it increases naturally as a byproduct of applying gradient updates. The second is that in order to accurately classify inputs, networks adopt relatively clusterable structure. To distinguish between these two explanations, we run three experiments on a dataset of 28×28 images with i.i.d. uniformly random pixel values, associated with

Dataset	N-cuts	Mean train acc.	Mean test acc.
MNIST	8.350 ± 0.046	0.967	0.979
LINES	6.85 ± 0.16	0.912	0.296
LINES-MNIST	6.933 ± 0.094	0.873	0.637
LINES-MNIST-SEP	7.779 ± 0.072	0.984	0.894

Table 3.3: N-cuts and accuracies for networks trained on mixture datasets with dropout. Notation as in Table 3.2.

random labels between 0 and 9.

In the **unlearnable random dataset experiment**, we train an MLP on 60,000 random images with default hyperparameters, 10 runs with dropout and 10 runs without. Since the networks are unable to memorize the labels, this tests the effects of SGD controlling for accuracy. We compare the n-cuts of the unpruned networks against the distribution of randomly initialized networks to check whether SGD without pruning increases clusterability. We also compare the n-cuts from the pruned networks against the distribution of n-cuts from shuffles of those networks, to check if SGD increases clusterability in the presence of pruning more than would be predicted purely based on the increase in sparsity.

In the **kilo-epoch random dataset experiment**, we modify the unlearnable random dataset experiment to remove the pruning and train for 1000 epochs instead of 20, to check if clusterability simply takes longer to emerge from training when the dataset is random. Note that even in this case, the network is unable to classify the training set better than random.

In the **memorization experiment**, we modify the random dataset and training method to be more easily learnable. To do this, we reduce the number of training examples to 3,000, train without pruning for 100 epochs and then with pruning for 100 more epochs, and refrain from shuffling the dataset between epochs. As a result, the network is often able to memorize the dataset, letting us observe whether SGD, pruning, and learning can increase clusterability on an arbitrary dataset.

As is shown in Table 3.4, the unlearnable random dataset experiment shows no increase in clusterability before pruning relative to the initial distribution shown in Figure A.1, suggesting that it is not a result of the optimizer alone. We see an increase in clusterability after pruning, but no relative clusterability, and absolute clusterability is below that of MLPs trained on MNIST or Fashion-MNIST with no regularization or with dropout displayed in Figure 3.3.

The results from the kilo-epoch random dataset experiment are shown in Table 3.5. The means and standard deviations suggest that even a long period of training caused no increase in clusterability relative to the distribution shown in Figure A.1, while pruned networks were not relatively clusterable or as clusterable as those trained on MNIST or Fashion-MNIST plotted in Figure 3.2.

The results of the memorization experiment, shown in Table 3.6, are different for the networks trained with and without dropout. Some networks trained with dropout memorized

Dropout	Unp. n-cuts	N-cuts	Dist. n-cuts	Prop. $p < 0.02$	Train acc.
×	10.289 ± 0.035	9.336 ± 0.026	9.326 ± 0.046	0/10	0.101
√	10.266 ± 0.041	9.312 ± 0.040	9.316 ± 0.042	0/10	0.102

Table 3.4: Results from the unlearnable random dataset experiment. Reporting as in tables in appendix A. "Unp" is short for unpruned. Accuracies and n-cut distributions are of pruned networks.

Dropout	Unp. n-cuts	N-cuts	Dist. n-cuts	Prop. $p < 0.02$	Train acc.
×	10.267 ± 0.044	9.332 ± 0.022	9.331 ± 0.050	0/10	0.101
√	10.274 ± 0.031	9.299 ± 0.045	9.305 ± 0.045	0/10	0.101

Table 3.5: Results from the kilo-epoch random dataset experiment. Reporting as in Table 3.4.

Dropout	Unp. n-cuts	N-cuts	Dist. n-cuts	Prop. $p < 0.02$	Train acc.
×	10.093 ± 0.025	8.746 ± 0.029	9.154 ± 0.035	10/10	1.000
√	10.221 ± 0.061	8.82 ± 0.15	8.984 ± 0.042	6/10	0.213

Table 3.6: Results from the memorization experiment. Reporting as in Table 3.4.

the dataset, and they appear to be relatively clusterable Those trained without dropout all memorized the dataset and were all relatively clusterable. In fact, their degree of clusterability is similar to that of those trained on Fashion-MNIST or MNIST without dropout. Before the onset of pruning, the n-cuts of the networks trained without dropout were consistently lower than those of randomly initialized networks, as shown in Figure A.1, and the n-cuts of those trained with dropout were at the lower end of the randomly initialized distribution.

Overall, these results suggest that the training process promotes modularity as a by-product of learning or memorization, and not automatically. Furthermore, we see that dropout fails to promote clusterability when it inhibits memorization.

3.3.5 Topology-preserving shuffles

One might suppose that the increase in clusterability relative to random networks is due to the pruning producing a clusterable topology, and that the values of the non-zero weights are unimportant. To test this, we compare each trained network to a new distribution: instead of randomly shuffling all elements of each weight matrix, we only shuffle the non-zero elements, thereby preserving the network's topology.

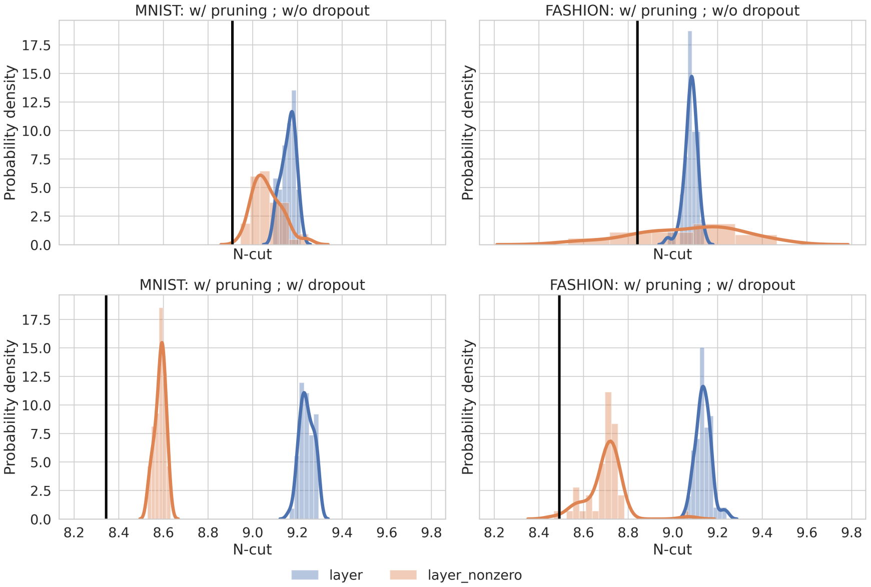


Figure 3.7: N-cuts of pruned networks trained on MNIST and Fashion-MNIST with and without dropout, compared to the distribution of n-cuts of networks generated by shuffling all elements of each weight matrix (shown in blue, labeled 'layer'), as well as the distribution of n-cuts of networks generated by shuffling only the non-zero elements of each weight matrix so as to preserve network topology (shown in orange, labeled 'layer-nonzero'). Realized n-cuts are shown as black vertical lines. Produced by the `distplot` function of seaborn 0.9.0 (Waskom et al., 2018) with default arguments.

Figure 3.7 shows the n-cuts of some representative networks compared to the distribution of n-cuts of all shuffled networks, and also the distribution of n-cuts of the topology-preserving shuffles. We see three things: first, that in all cases our networks are more clusterable than would be expected given their topology; second, that the topology-preserving shuffles are more clusterable than other shuffles, suggesting that the pruning process is removing the right weights to promote clusterability; and third, that with dropout, the distribution of topology-preserving shuffles has much lower n-cuts than the distribution of all shuffles.

3.3.6 Choosing the number of clusters

The number of clusters is a hyperparameter of the spectral clustering algorithm. In most of this paper, we report the results from using 12 clusters.

To test the robustness of our results to this hyperparameter, we re-ran the MLP clusterability experiments on pruned networks trained with and without dropout, using 2, 4, 7, and 10 clusters. In each condition, we trained 10 networks, compared to a distribution of 320 shuffles, and counted how many networks were more clusterable than all 320 of the shuffles. For 4, 7, and 10 clusters, the results align with those reported earlier in this section: networks are relatively clusterable, and more clusterable (both absolutely and relatively) when trained with dropout. Results are shown in Tables 3.7, 3.8, and 3.9 respectively.

Dataset	Dropout	N-cuts	Dist. n-cuts	$p < 1/320$	Train acc.	Test acc
MNIST	×	2.000 ± 0.035	2.042 ± 0.017	7/10	1.00	0.984
MNIST	√	1.840 ± 0.015	2.039 ± 0.019	10/10	0.967	0.979
Fashion	×	1.880 ± 0.030	1.992 ± 0.018	10/10	0.983	0.893
Fashion	√	1.726 ± 0.022	2.013 ± 0.017	10/10	0.863	0.869

Table 3.7: Clusterability results for pruned MLPs with 4 clusters. N-cuts shows the mean and standard deviations of the n-cuts of the trained networks. Dist. n-cuts shows the average mean and standard deviation of the distributions of shuffles, the average being taken over the 10 trained networks. $p < 1/320$ shows how many networks were more clusterable than all 320 shuffles. Accuracies are averaged over runs.

When we ran the experiments with 2 clusters, the significance results were very different, as shown in Table 3.10. No network trained on MNIST or Fashion-MNIST was statistically significantly clusterable. We therefore conjecture that our results generalize to any number of clusters that is not too small (or comparable to the number of neurons).

Dataset	Dropout	N-cuts	Dist. n-cuts	$p < 1/320$	Train acc.	Test acc.
MNIST	×	4.556 ± 0.052	4.710 ± 0.026	10/10	1.00	0.984
MNIST	√	4.222 ± 0.035	4.712 ± 0.023	10/10	0.967	0.979
Fashion	×	4.351 ± 0.057	4.613 ± 0.029	10/10	0.983	0.893
Fashion	√	4.079 ± 0.046	4.663 ± 0.029	10/10	0.863	0.869

Table 3.8: Clusterability results for pruned MLPs with 7 clusters. Reporting as in Table 3.7.

Dataset	Dropout	N-cuts	Dist. n-cuts	$p < 1/320$	Train acc.	Test acc.
MNIST	×	7.137 ± 0.042	7.397 ± 0.037	10/10	1.00	0.984
MNIST	√	6.688 ± 0.035	7.421 ± 0.031	10/10	0.967	0.979
Fashion	×	6.975 ± 0.117	7.257 ± 0.036	9/10	0.983	0.893
Fashion	√	6.460 ± 0.041	7.339 ± 0.034	10/10	0.863	0.869

Table 3.9: Clusterability results for pruned MLPs with 10 clusters. Reporting as in Table 3.7

Dataset	Dropout	N-cuts	Dist. n-cuts	$p < 1/320$	Train acc.	Test acc.
MNIST	×	0.333 ± 0.003	0.330 ± 0.003	0/10	1.00	0.984
MNIST	√	0.332 ± 0.002	0.323 ± 0.003	0/10	0.967	0.979
Fashion	×	0.319 ± 0.003	0.313 ± 0.003	0/10	0.983	0.893
Fashion	√	0.312 ± 0.003	0.312 ± 0.003	0/10	0.863	0.869

Table 3.10: Clusterability results for pruned MLPs with 2 clusters. Reporting as in Table 3.7.

3.4 Clusterability in CNNs

To test if the results found on MNIST and Fashion-MNIST in section 3.3 are specific to the MLP architecture, we repeat them using a small CNN. Our network has 3 convolutional hidden layers of 64 channels each, followed by a fully-connected hidden layer with 128 neurons. All convolutional kernels are 3×3, with the second and third hidden layers being followed by max pooling with a 2×2 window. Batch size was 64, and training proceeded for 10 epochs before pruning, followed by 10 epochs of training with pruning. L_1 and L_2 regularization strength is the same as for MLPs, and dropout rate is 0.25 for convolutional layers and 0.5 for fully-connected layers. In all regularization schemes, networks trained to around 99% accuracy on MNIST and 90-92% accuracy on Fashion-MNIST. At initialization, their n-cuts are between 10.90 and 10.95, as shown in Figure A.2.

Results are shown in Figures 3.8, 3.9, and 3.10, and Tables A.4 and A.5. We see that pruning fails to promote relative clusterability at all, that L_1 regularization promotes absolute but not relative clusterability, and that before pruning, L_2 regularization promotes absolute and relative clusterability. Networks are reliably more clusterable than at initialization, except those trained with L_2 regularization after pruning.

To see if these results generalize to larger networks on more complex datasets, we run the same experiments using a version of VGG-16 described by S. Liu and Deng (2015) trained on CIFAR-10. We use a batch size of 128, and train for 200 epochs before 50 epochs of training with pruning. Data augmentation is used: random rotations between 0 and 15 degrees, random shifts both vertically and horizontally of up to 10% of the side length, and random horizontal flipping. We use a per-layer dropout rate as specified in S. Liu and Deng (2015). Without pruning, we achieve 86-90% test accuracy, and with pruning, we achieve 88-91% test accuracy. At initialization, these networks have n-cuts between 8.45 and 8.63, as plotted in Figure A.3. Results are shown in Figures 3.11 and 3.12, and Table A.6.

We see that these CNNs are typically not relatively clusterable, except when trained with dropout alone and pruning, or Liu and Deng's (S. Liu & Deng, 2015) combination of dropout and L_2 regularization as well as pruning. Networks trained without pruning are not more clusterable than at initialization, but networks trained with pruning are except when only L_2 regularization is used.

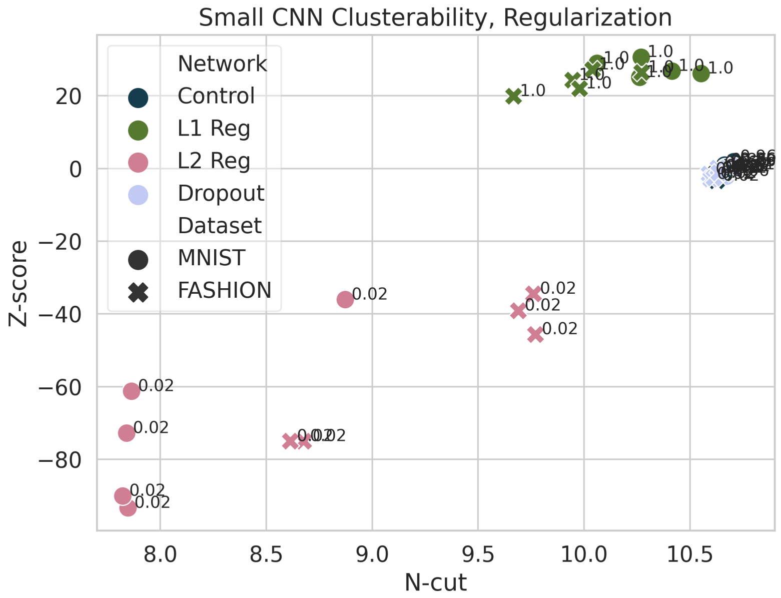


Figure 3.8: Clusterability of small CNNs trained without pruning. Points are labeled with their one-sided p-value. For detail on networks trained with no regularization or dropout, see Figure 3.9.

3.4.1 Clusterability of ImageNet models

We also explore the clusterability of networks trained on ImageNet (Deng et al., 2009). Specifically, we looked at VGG-16 and 19 (Simonyan & Zisserman, 2015), ResNet-18, 34, and 50 (He et al., 2016a), and Inception-V3 (Szegedy et al., 2016). Weights were obtained from the Python `image-classifiers` package, version 1.0.0. Clustering was less stable for these networks, likely because of their large size, so we used 100 random starts for k-means clustering instead of the default 10, and models were compared to a distribution of 100 shuffles. Results are shown in Figure 3.13 and Table A.7. We observe that all networks are relatively clusterable.

3.4.2 ‘Stack’ datasets

As in subsection 3.3.1, we explored the effects of datasets we designed to induce modularity. Instead of the halves datasets, we used ‘stack’ datasets where different images are in different input channels, rather than shrunk and put next to each other in one channel, because convolutional layers followed by max-pooling promote spatial location invariance within channels. The datasets are otherwise the same: in ‘stack-same’, the channels are images of the same class, while in ‘stack-diff’, the channels are of different classes and the label is the

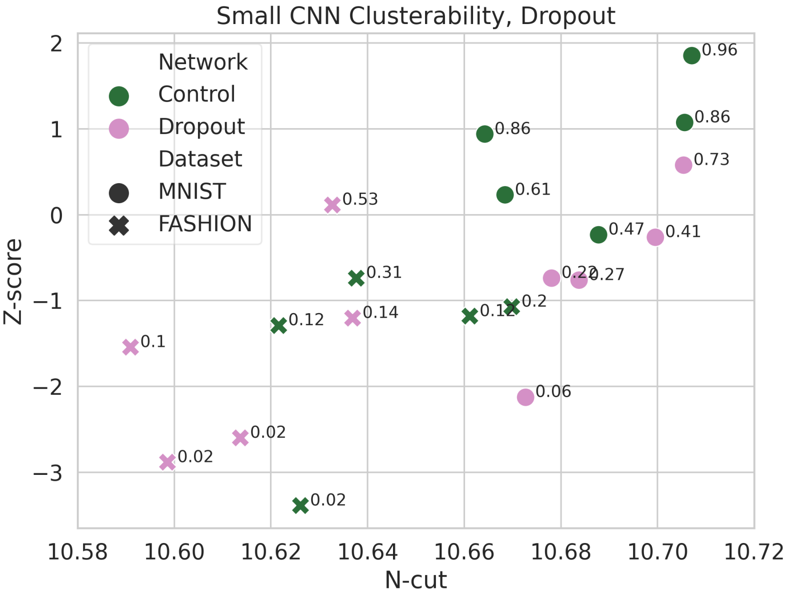


Figure 3.9: Clusterability of small CNNs trained without pruning with and without dropout—a subset of Figure 3.8.

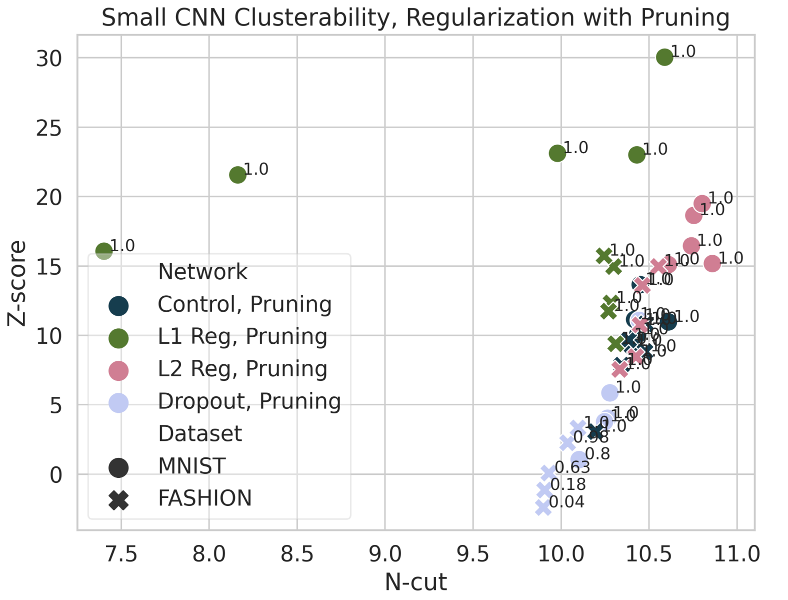


Figure 3.10: Clusterability of small CNNs trained with pruning. Points are labeled with their one-sided p-value.

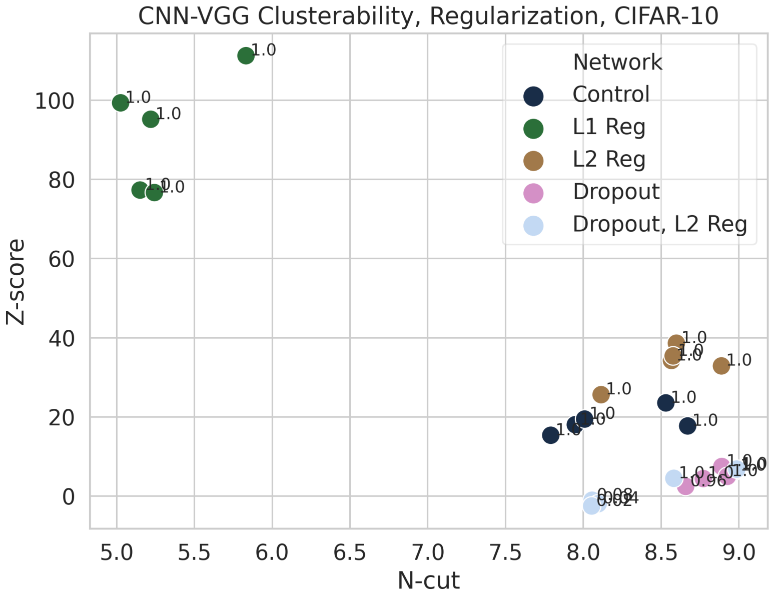


Figure 3.11: Clusterability of VGGs trained without pruning on CIFAR-10. Points are labeled with their one-sided p-value.

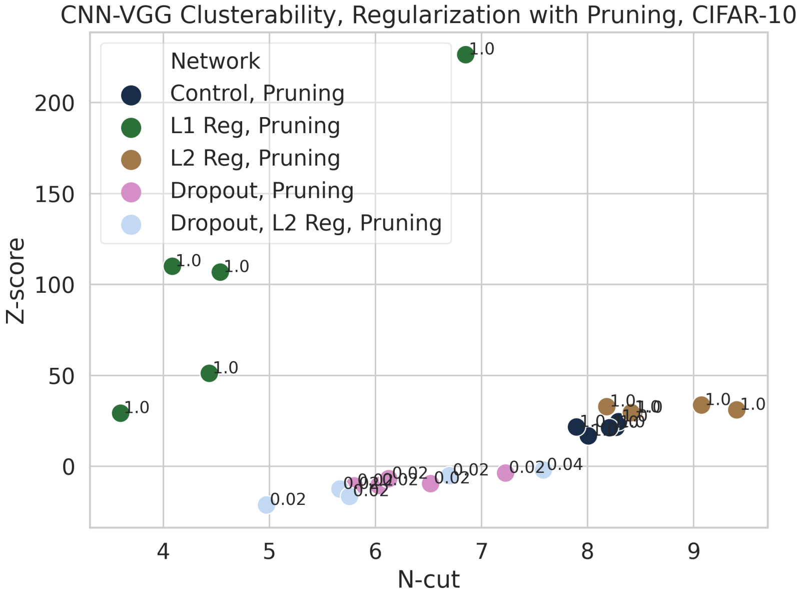


Figure 3.12: Clusterability of VGGs trained with pruning on CIFAR-10. Points are labeled with their one-sided p-value.

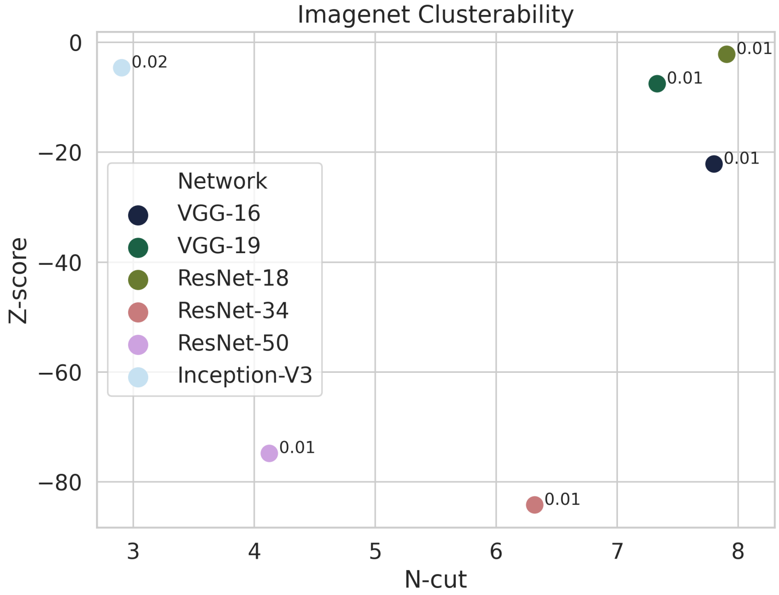


Figure 3.13: Clusterability of models trained on ImageNet. Points are labeled with their one-sided p-value.

classes' sum modulo 10. Images are shown in Figure 3.14. We train the same CNN as was trained on MNIST and Fashion-MNIST.

Results are shown in Figure 3.15 and Table A.8. We see that pruning promotes absolute but not relative clusterability, while networks trained on stack-diff datasets are somewhat more clusterable, both in absolute and relative terms, than those trained on stack-same datasets. They are always more clusterable than at initialization.

3.5 Promoting clusterability

We explore two methods to promote clusterability while hewing closely to standard training procedure: regularization and initialization.

3.5.1 Regularization

The most straightforward approach to promote clusterability is regularization. The Cheeger inequalities (Alon & Milman, 1985; Dodziuk, 1984) give a bound on the Cheeger constant of a graph (closely related to n-cut for $k = 2$) in terms of the second eigenvalue of the normalized Laplacian matrix L_{norm}: namely, the lower the eigenvalue, the more easy it is to divide the network into two. J. R. Lee et al. (2014) have shown that this bound extends to the analog

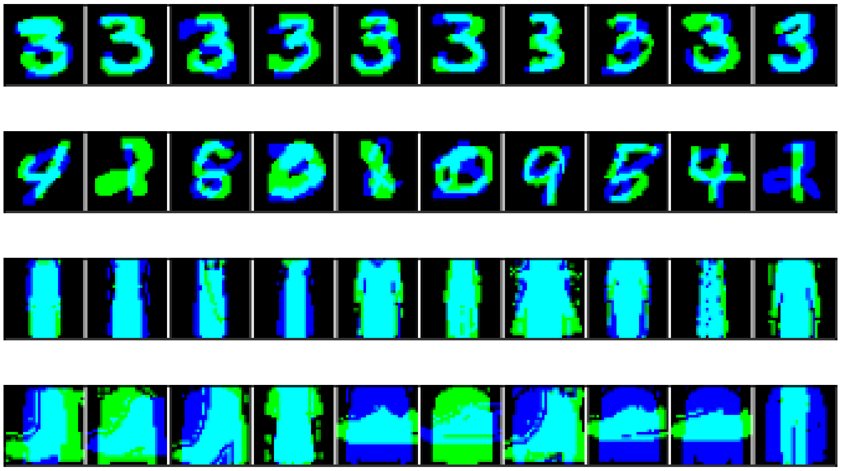


Figure 3.14: Samples from 'stack' datasets, all of class 3. Each row has 10 images from the respective dataset. One channel is colored blue, the other is colored green. First row is MNIST-stack-same, second is MNIST-stack-diff, third is Fashion-stack-same, fourth is Fashion-stack-diff.

of the Cheeger constant for $k > 2$. As such, regularizing the k^{th} eigenvalue should produce a network with a low n-cut value for k clusters.

This is possible because if a symmetric $n \times n$ matrix S has distinct eigenvalues $\lambda_1, \ldots, \lambda_n$ associated with orthonormal eigenvectors $u_1, \ldots, u_n$, the derivative of λ_k with respect to S is $u_k u_k^\top$ (Magnus, 1985). L_{norm} is not symmetric, but it has the same set of eigenvalues as the symmetric matrix $L_{\text{sym}} := D^{-1/2} L D^{-1/2}$. So, we can take the derivative of eigenvalues with respect to L_{sym}, which itself is a differentiable function of the network weights, allowing us to regularize the eigenvalues of L_{norm}.

Given the above, we only need compute the gradient of L_{sym} with respect to each weight matrix W^s. We will then have

$$\frac{\partial \lambda_k}{\partial W^s_{n,m}} = \sum_{ij} (u_k)_i (u_k)_j \frac{\partial (L_{\text{sym}})_{i,j}}{\partial W^s_{n,m}}.$$

Since $L_{\text{sym}} = I - D^{-1/2} A D^{-1/2}$, $(L_{\text{sym}})_{i,j} = \delta_{i,j} - d_i^{-1/2} d_j^{-1/2} A_{i,j}$, where $\delta_{i,j}$ is the function that is 1 if $i = j$ and 0 otherwise. Furthermore, for symmetry, we write $d_i = (1/2)(\sum_k A_{i,k} +$

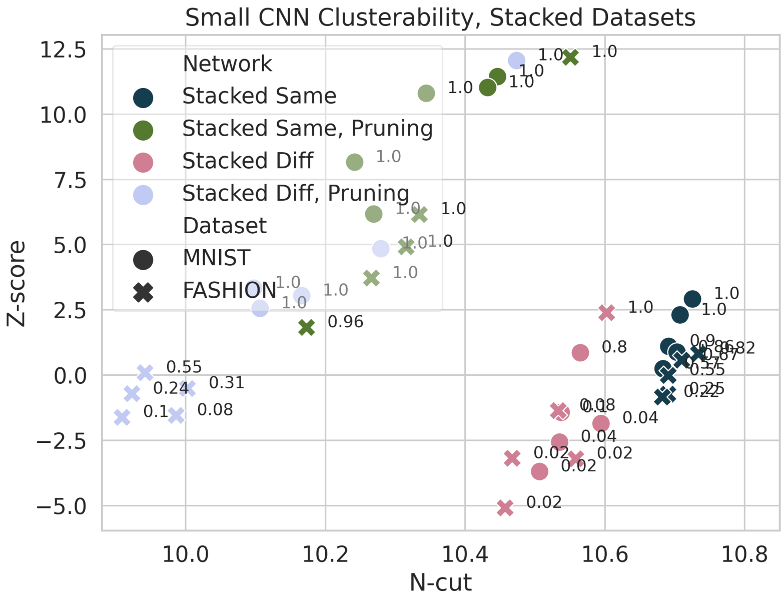


Figure 3.15: Clusterability of models trained on 'stack' datasets. Points are labeled with their one-sided p-value.

$\sum_l A_{l,i}$). This gives the partial derivatives

$$\begin{aligned}\frac{\partial d_i}{\partial A_{i,i}} &= 1\\ \frac{\partial d_i}{\partial A_{i,j}} &= \frac{1}{2}\\ \frac{\partial d_i}{\partial A_{k,i}} &= \frac{1}{2},\end{aligned}$$

where $j, k \neq i$.

From here on out, we need not consider derivatives with respect to $(L_{\text{sym}})_{i,i}$ or $A_{i,i}$, since $(L_{\text{sym}})_{i,i} = A_{i,i} = 0$ always by definition, because nodes don't have edges to themselves. We can now compute derivatives of L_{sym} with respect to A:

$$\begin{aligned}\frac{\partial (L_{\text{sym}})_{i,j}}{\partial A_{i,j}} &= -d_i^{-1/2} d_j^{-1/2} + \frac{1}{4} A_{i,j} \left(d_i^{-3/2} d_j^{-1/2} + d_i^{-1/2} d_j^{-3/2} \right)\\ \frac{\partial (L_{\text{sym}})_{i,j}}{\partial A_{i,l}} &= \frac{1}{4} A_{i,j} d_i^{-3/2} d_j^{-1/2}\end{aligned}$$

$$\begin{aligned}\frac{\partial(L_{\text{sym}})_{i,j}}{\partial A_{k,j}} &= \frac{1}{4}A_{i,j}d_i^{-1/2}d_j^{-3/2}\\ \frac{\partial(L_{\text{sym}})_{i,j}}{\partial A_{k,l}} &= 0,\end{aligned}$$

where $i \neq k$ and $j \neq l$.

Next, let $e(n, s)$ be the index of neuron n of layer s, and let the weight matrices be W^1 through W^S. It's then the case that

$$\frac{\partial A_{e(n,s),e(m,s+1)}}{\partial W^s_{n,m}} = \frac{\partial A_{e(m,s+1),e(n,s)}}{\partial W^s_{n,m}} = \text{sgn}(W^s_{n,m}),$$

and $\partial A_{i,j}/\partial W^s_{n,m} = 0$ for all other i and j, by the definition of how A is constructed from the weight matrices, where $\text{sgn}(x)$ is the sign of x: 1 if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$.

Therefore, combining all the equations above, we have:

$$\begin{aligned}\frac{\partial(L_{\text{sym}})_{i,j}}{\partial W^s_{n,m}} &= \text{sgn}(W^s_{n,m})\left(\frac{\partial(L_{\text{sym}})_{i,j}}{\partial A_{e(n,s),e(m,s+1)}} + \frac{\partial(L_{\text{sym}})_{i,j}}{\partial A_{e(m,s+1),e(n,s)}}\right)\\ &= \text{sgn}(W^s_{n,m})\Bigg(\delta_{i,e(n,s)} \times \frac{1}{4}A_{i,j}d_i^{-3/2}d_j^{-1/2} + \delta_{j,e(m,s+1)} \times \frac{1}{4}A_{i,j}d_i^{-1/2}d_j^{-3/2}\\ &\qquad - \delta_{i,e(n,s)}\delta_{j,e(m,s+1)}d_i^{-1/2}d_j^{-1/2} + \delta_{i,e(m,s+1)} \times \frac{1}{4}A_{i,j}d_i^{-3/2}d_j^{-1/2}\\ &\qquad + \delta_{j,e(n,s)} \times \frac{1}{4}A_{i,j}d_i^{-1/2}d_j^{-3/2} - \delta_{i,e(m,s+1)}\delta_{j,e(n,s)}d_i^{-1/2}d_j^{-1/2}\Bigg).\end{aligned}$$

This concludes our derivation.

To ensure that produced clusterability is meaningful, we must also normalize the weights: in a network with ReLU activation functions, it is possible to scale the inputs of a hidden neuron by a positive constant c while scaling the outputs by $1/c$ without changing the function the network computes. Since having more weights close to zero makes it easier for clustering algorithms to cut low-weight edges while preserving high-weight edges, regularizing for clusterability will induce these transformations. However, they do not make the network any more modular in any real sense, which is our aim.

Therefore, when applying this regularizer, we manually apply this transformation after each gradient step to enforce the invariant that the norm of the input weights to each hidden neuron should be $\sqrt{2}$, chosen to maintain the benefits of He initialization (He et al., 2016b; Kumar, 2017). This is detailed in algorithm 2.

Due to the expense of eigenvalue computation on large networks, we tested this regularizer on an MLP with 3 hidden layers of width 64 that was trained on MNIST images downsampled to 7×7 pixels. Training proceeded for 10 epochs followed by 10 epochs of training with pruning. We trained 10 networks without regularization and 10 with the 2^{nd}, 3^{rd}, and 4^{th} eigenvalues regularized with weight 0.1. As we see in Table 3.11, little accuracy is lost, and regularized networks are more clusterable both in absolute and relative terms. Results are also plotted in Figure 3.16.

Clust. reg?	Pruning?	N-cut	Dist. n-cuts	Prop. $p < 0.02$	Train acc.	Test acc.
×	×	10.020	10.035 ± 0.030	2/10	0.955	0.957
×	√	7.11	7.43 ± 0.12	4/10	0.930	0.934
√	×	8.93	8.14 ± 0.23	1/10	0.961	0.959
√	√	5.51	6.31 ± 0.20	7/10	0.888	0.896

Table 3.11: Clusterability of networks trained with and without the clusterability regularizer, with and without pruning. "Dist. n-cuts" contains the mean and standard deviation of the distribution of n-cuts of shuffled networks. All figures shown are averaged over 10 training runs, except "Prop. $p < 0.02$", which shows how many networks were more clusterable than all 50 shuffles.

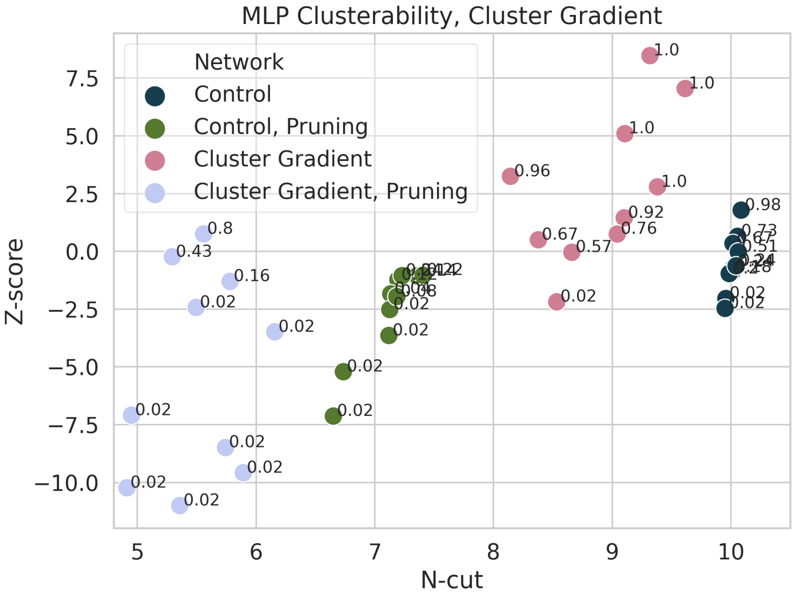


Figure 3.16: Clusterability of small MLPs trained with and without the clusterability regularizer. Points are labeled with their one-sided p-value.

Algorithm 2 Normalization for MLP Clusterability Regularization

```
input: Weight matrices $W_1, \dots, W_m$, bias vectors $b_1, \dots, b_m$
for hidden layers $i$ from 1 to $m-1$ do
  for all neurons $n$ in the layer do
    form $v$ by concatenating the $n^{\text{th}}$ column of $W_i$ (that is, the weights that feed into neuron $n$) and the $n^{\text{th}}$ entry of $b_i$
    set $x := \sqrt{\sum_j v_j^2}$
    if $x \neq 0$ then
      multiply $n^{\text{th}}$ column of $W_i$ by $\sqrt{2}/x$
      multiply $n^{\text{th}}$ entry of $b_i$ by $\sqrt{2}/x$
    end if
    multiply $n^{\text{th}}$ row of $W_{i+1}$ by $x/\sqrt{2}$
  end for
end for
return: New weight matrices $W_1, \dots, W_m$ and bias vectors $b_1, \dots, b_m$
```

3.5.2 Initialization

Another way to promote clusterability is to initialize weights to be clusterable, and from then on to train as usual. After standard initialization, we randomly associate each neuron in the hidden layers with one of c tags. We then weaken the weight of each edge going between differently-tagged neurons by multiplying it by $\beta \in (0, 1)$, and strengthen each edge going between neurons with the same tag by a factor of $1 + (1 - \beta)(c - 1)$. This preserves the mean absolute value of weights in a layer. This method is computationally cheaper than ongoing regularization, since it only needs to be done once, and doesn't require the computation of eigenvalues.

We trained clusterably initialized MLPs on MNIST and Fashion-MNIST with $c = 10$ and $\beta = 0.6$, chosen to balance performance with final clusterability. We saw no loss in test accuracy. Results are shown in Figure 3.17 and Table A.9. This initialization method appears to be effective in promoting relative and absolute clusterability, especially in tandem with weight pruning. We also trained clusterably initialized CNNs on MNIST and Fashion-MNIST, and a clusterably initialized VGG on CIFAR-10. For these, we kept $c = 10$ but instead used $\beta = 0.8$. Again, we saw no loss in test accuracy. Results are shown in Figures 3.18 and 3.19 and Tables A.10 and A.11. By contrast with the MLP results, the initialization seems much less effective at promoting absolute and relative clusterability.

3.6 Related work

This work adds to a body of research focused on modularity and compositionality in neural systems either at the neuron level (Mu & Andreas, 2020; Voss et al., 2021; You et al.,

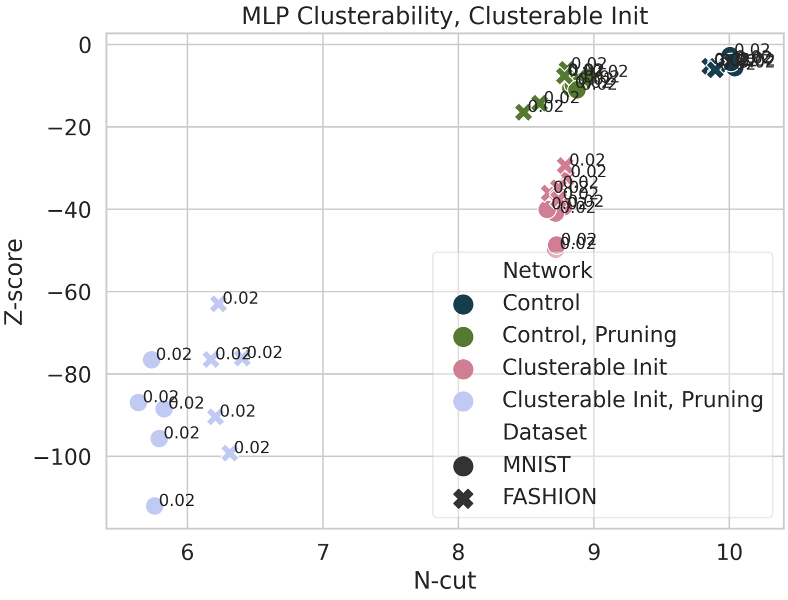


Figure 3.17: Clusterability of MLPs trained with and without clusterable initialization. Points are labeled with their one-sided p-value.

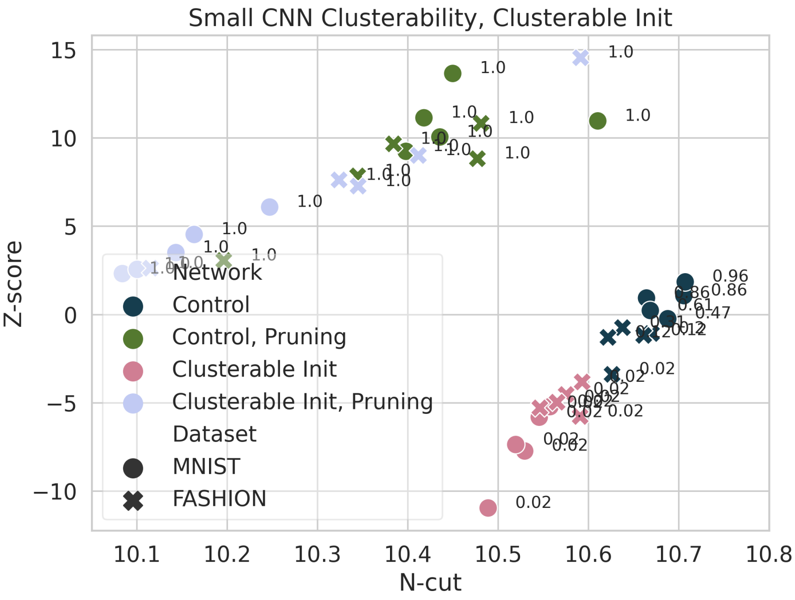


Figure 3.18: Clusterability of small CNNs trained with and without clusterable initialization. Points are labeled with their one-sided p-value.

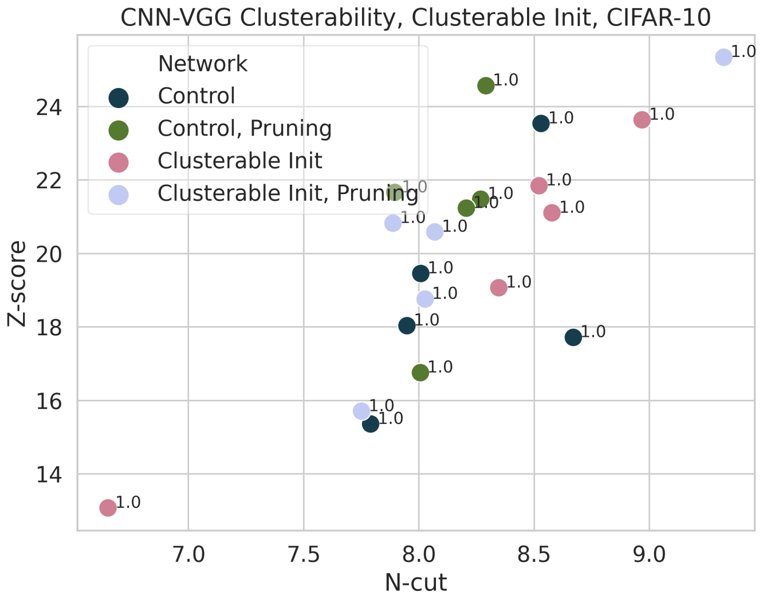


Figure 3.19: Clusterability of VGGs trained with and without clusterable initialization. Points are labeled with their one-sided p-value.

2020) or at the sub-network level (Csordás et al., 2021; Lake et al., 2015; Lake et al., 2017; Udrescu et al., 2020). Watanabe et al. (C. Watanabe, 2019; C. Watanabe et al., 2018, 2019) demonstrate a different way of grouping neurons into modules, investigate properties of the modules, and link their notion of modularity to generalization error. Davis et al. (2020) use a clustering method based on statistical properties (rather than the weights of the network directly) to visualize networks, prune them, and do feature attribution. Lu and Ester (2019) bi-cluster neurons in a hidden layer and show that this divides neurons into functionally-related groups that are important for classification. None of these papers compare the modularity of networks trained in different ways, nor do they show that networks are more modular than initialized networks or networks with the same per-layer distribution of weights.

There is also existing literature on imposing modular structure on neural networks, either by some kind of dynamic routing (Alet et al., 2018; Andreas et al., 2016; Chang et al., 2018; De Lange et al., 2021; Goyal et al., 2019; Kirsch et al., 2018; Parascandolo et al., 2018) or by explicitly enforcing the desired modularity structure (D. Lee et al., 2018; Oyama et al., 2001). These typically involve training with a non-standard architecture, limiting their broader applicability. Amer and Maul (2019) offer a review of modularization techniques.

3.7 Conclusion

We see that many neural networks are clusterable to a degree that would not be expected merely from their initialization, or from their distributions of weights. MLPs trained on image classification are reliably more clusterable than 98% of their shuffles when L_1 or L_2 regularization is not used, as are networks trained on ImageNet classification. So are VGGs trained on CIFAR-10, as long as they are trained with dropout (which is standard practice) and are pruned (which is not). L_1 and L_2 regularization appear to have irregular effects on clusterability, sometimes producing spuriously clusterable networks, and sometimes producing remarkably unclusterable networks. We also demonstrate that clusterability can be induced in MLPs either by initializing weights differently or by regularization, with little cost to performance and without hand-coding a non-standard architecture or significantly modifying the training procedure.

Overall, we find that clusterability is a common phenomenon, giving hope that networks are also in some sense modular. That being said, our work limits itself to image classification networks, leaving open the question of clusterability in other domains. We also fail to induce clusterability in CNNs by initialization or regularization, which suggests a direction for future work. Finally, although we have investigated a graphical notion of clusterability, we have yet to demonstrate any functional relevance of this graphical notion. For that, we must proceed to Chapter 4.

Chapter 4

Importance and Coherence

4.1 Introduction

In Chapter 3, we applied a graph-theoretic analogue of modularity—that is, clusterability—to neural networks, investigating how clusterable they are and how they can be made more so. This was motivated by the potential benefits of modularity for interpretability. However, for the purposes of interpretability the relevant notion of modularity is functional. If distinct clusters of neurons perform different sub-tasks then identifying cluster structure aids our understanding of the network, but if they do not its relevance is unclear. Therefore, it would be valuable to determine the extent to which neural networks are *locally specialized*: that is, how much their functionality can be abstracted into comprehensible sub-tasks, each localized to different groups of neurons. It is also valuable to know whether the localization of functionality aligns with the cluster structure investigated in Chapter 3. In this chapter, we focus on developing quantifiable proxies for local specialization. We then apply them, together with variants of the partitioning methods used in Chapter 3, to a variety of image classification networks.

There exists a body of research for developing networks which either have distinct architectural building blocks (Alet et al., 2018; Goyal et al., 2019; Parascandolo et al., 2018) or are trained in a way that promotes modularity via regularization or parameter isolation (De Lange et al., 2021; Kirsch et al., 2018). Yet in machine learning, it is more common to encounter networks whose architecture and training are not designed to separate the computation of sub-tasks. For example, in computer vision, networks are generally trained end-to-end with all of the filters in one layer connected to all filters in the next. Do such networks nonetheless exhibit local specialization? There is some evidence for this. For example, by methodical manual investigation, Cammarata et al. (2020) discovered sub-networks which perform human-explainable sub-tasks such as car detection using neurons which detect different car parts. More scalably detecting local specialization in networks would help us to extend our understanding of their learning dynamics and to expand our interpretability toolbox by suggesting an additional level of abstraction beyond single-neuron methods.

In this chapter, we systematically analyze the extent to which networks which are not explicitly trained to be modular nonetheless exhibit local specialization. First, this requires a method for breaking down a network's computational graph into groups of neurons. For this, we use spectral clustering on a graph representation of the network, using an extension of the methods of Chapter 3. Second, this requires a scalable method for approximating the degree to which a partitioning shows local specialization. We do this by applying interpretability tools to clusters of neurons as a way of quantifying proxies for local specialization.

Proxies for local specialization: Our definition of local specialization requires human-comprehensible sub-tasks relevant to the overall task to be localized in particular subsets of neurons. However, directly determining this would require a human in the loop, making it difficult to scale such a method. How could measuring local specialization be automated? Consider an idealized prototype of a highly modular network that has subsets of neurons performing sub-tasks in which (1) each sub-task is necessary for high performance on the overall task, (2) each sub-task is implemented by a single subset only, and (3) each subset executes only a single sub-task. The combination of (1) and (2) suggests that the removal of one of the subsets from the network would harm performance because the network lacks the implementation of a necessary sub-task that is localized to the subset. We say that such a subset is *important*. Next, given that neurons are frequently understood as feature detectors, (3) suggests that the neurons in a subset should tend to be strongly activated by inputs which contain features relevant to the subset's sub-task. We say that such a subset is *coherent*. Measuring importance and coherence thus offers a sense of the degree to which a partitioning of a network's neurons contains subsets that meet these prototypical conditions, despite not perfectly satisfying them.

Results and contributions: To measure these proxies, we take partitionings of neurons in a network generated by spectral clustering, and analyze them using methods from the interpretability literature that have conventionally been applied to single neurons. Our key results are shown in Tables 4.1 and 4.2. We find that these partitions have groups of neurons that are disproportionately likely to be important compared to random groups of neurons, but that are not always more important than random groups of neurons on average. We also find that the groups of neurons in the partitionings are reliably more coherent than random ones, though only with respect to features other than class label.

By showing that our partitioning methods are able to reveal local specialization, these results suggest that they can be used to screen for interesting, abstractable subsets of units and better understand deep networks. Our key contributions are threefold:

1. Introducing two proxies, importance and coherence, to assess whether a partitioning of a network shows local specialization and identify which subsets of neurons are the most responsible.

2. Quantifying these proxies by applying single-neuron interpretability methods on subsets of neurons in an automated fashion.

3. Applying our methods on the partitions produced by spectral clustering on a range of neural networks and finding evidence of local specialization captured by these partitions.

Code is available at https://github.com/thestephencasper/local_specialization.

4.2 Related work

In our experiments we combine clustering with interpretability tools to measure importance and coherence. We use neural lesions (B. Zhou et al., 2018) and feature visualization (Olah et al., 2017; C. Watanabe, 2019), but in a similar way, other interpretability techniques including analysis of selectivity (Madan et al., 2020; Morcos et al., 2018), network "dissection" (Bau et al., 2017; Mu & Andreas, 2020), earth-mover distance (Testolin et al., 2020), or intersection information (Panzeri et al., 2017) could also be combined with clustering-based partitionings under a similar framework. Chan et al. (2022) offer a different method of lesioning neurons for interpretability purposes. Related to the themes of importance and coherence, Cammarata et al. (2020) demonstrate that feature visualization and analysis of weights can be used to identify groups of neurons whose functionality is human-interpretable.

4.3 Methods

To evaluate local specialization, our procedural pipeline is as follows. (1) We begin with a trained neural network; (2) construct a graph from it, treating each neuron as a node; (3) perform spectral clustering on the graph to obtain a partitioning or "clustering" of neurons which we then further divide by layer to obtain a "subclustering"; (4) use our proxies for local specialization to analyze the subclusters, operationalized by lesioning neurons or feature visualization and comparing them to random subclusters; and (5) aggregate results across the network to obtain a p value, effect measure, and a quantity which we refer to as the *Fisher statistic*. This pipeline is outlined in Figure 4.1 and Figure 4.2, and each step is explained in detail below.

4.3.1 Generating partitions with spectral clustering

To partition a network into clusters, we use an approach based on Filan et al. (2021) which consisted of two steps: *"graphification"*—transforming the network into an undirected, edge-weighted graph; and *clustering*—obtaining a partitioning via spectral clustering.

Graphification: To perform spectral clustering, a network must be represented as an undirected graph with non-negative edges. For MLPs (multilayer perceptrons), each graph node corresponds to a neuron in the network including input and output neurons. For CNNs (convolutional neural networks), a node corresponds to a single channel (which we

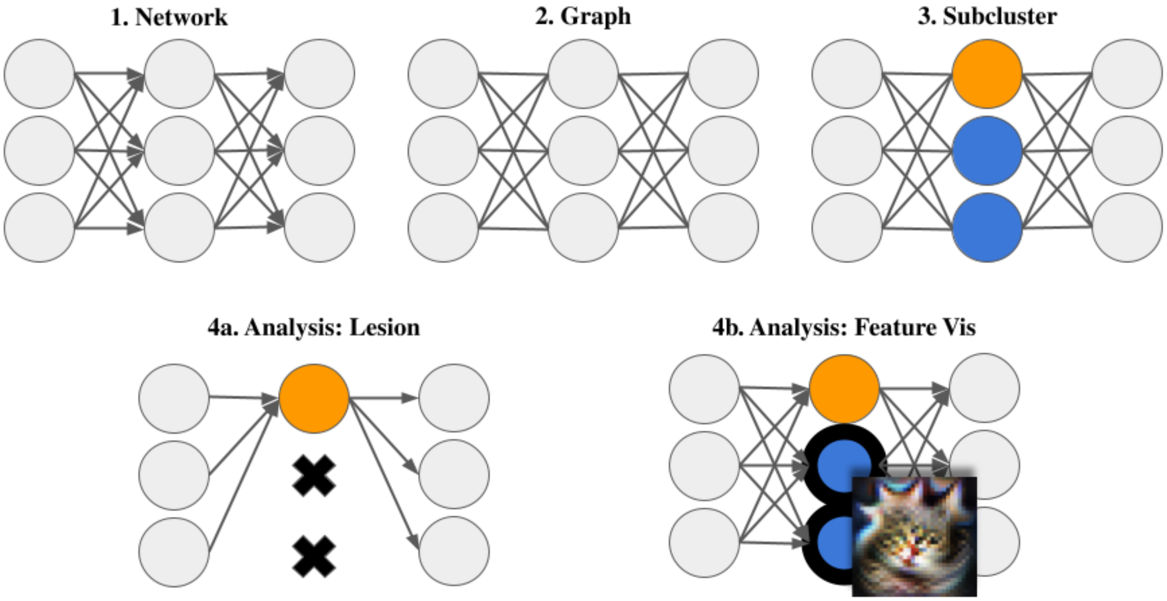


Figure 4.1: **Our procedural pipeline.** The first three steps generate a partitioning of the network into "subclusters" which we analyze using (4a) lesion and (4b) feature visualization methods to measure importance and coherence compared to random subclusters. Finally, (not shown in the pipeline), we aggregate results to produce Fisher statistics, p values, and effect measures. These final steps are shown in Figure 4.2.

also refer to as a "neuron") in a convolutional layer.[1] For CNNs, we ignore input, output, and fully-connected layers when clustering.

For graphification, we test two ways of assigning adjacency edges between neurons: with weights and with correlations. For weight-based clustering with dense layers, if two neurons have a weight connecting them in the network, their corresponding vertices are connected by an edge with weight equal to the absolute value of the network's weight between the neurons. For convolutional channels, we connect them by an edge with weight equal to the L_1 norm for the corresponding 2D kernel slice. If layers are connected but with a batch-normalization layer in between, we mimic the scaling performed by the batch norm operation by multiplying weights by $\gamma/\sqrt{\sigma^2+\varepsilon}$ where γ is the scaling factor, σ is the moving standard deviation, and ε is a small constant. Notably, this method of constructing the graph requires no dataset or runtime analysis of the network.

While graphification via weights only results in connections between neurons in adjacent layers, doing so with correlations creates more dense graphs. With this method, we connect the nodes for two neurons with their squared Spearman correlation across a validation set.

[1] If a unit is used as inputs to multiple layers, as happens in ResNets with skip connections, we consider these inputs to be separate neurons.

Spearman correlation gives the Pearson (linear) correlation between ranks and reflects how well two sets of data can be related by a monotonically increasing function.[2] Rather than using neurons' post-ReLU outputs to calculate these correlations, we use their pre-ReLU activations with the goal of extracting richer data from them.[3] Again, for convolutional channels, we take the L_1 norm of activations before calculating Spearman correlations. The fact that we take absolute valued weights or squared correlations to construct edges between neurons in graphification means that our analysis does not discriminate between positive and negative associations between neurons.

In addition to graphification via weights versus activations, we also test two scopes with which to perform clustering: network-wide and layer-wise. For network-wide clustering, we cluster on one graph for the network as a whole. For layer-wise clustering, we produce a partitioning for each layer l individually by clustering on the graph of connections between l and the layers adjacent to l. Ultimately, we run 4 sets of experiments on each network by clustering {weights, activations} × {network-wide, layer-wise}.

Spectral (sub)clustering: We perform normalized spectral clustering (Shi & Malik, 2000) on the resulting graphs to obtain a partition of the neurons into clusters. For all sub-ImageNet experiments, we set the number of clusters to $k = 16$, while for ImageNet-scale networks, we use $k = 32$. In subsection 4.4.4, we reproduce a subset of sub-ImageNet experiments with $k \in \{8, 12\}$ showing that results are robust to alternate choices of k. Spectral clustering is implemented as specified in subsection 3.2.3 of the previous chapter.

Layers at different depths of a network tend to develop different representations. Therefore, for network-wide clustering in which clusters of neurons span more than one layer, we analyze clusters one layer at a time. We call these sets of neurons within the same cluster and layer *subclusters*. To ensure comparability between these clusterings when performing layer-wise clustering, we set the number of clusters per layer to be the same as the number produced in that layer with network-wide clustering. In our experiments, we compare these subclusters to other random sets of neurons of the same size in the same layer. We refer to subclusters identified by the clustering algorithm as "true subclusters" and sets of random neurons as "random subclusters." Random subclusters form the natural control condition to test whether the specific partitioning of neurons exhibits importance or coherence compared to alternative partitions, while taking account of layer and size.

4.3.2 Analysis pipeline

Overview: In our experiments, we measure the degree to which true subclusters identified via spectral clustering are more important and coherent than random subclusters of the same size and layer. Operationalizations are given in Subsections 4.4.2 and 4.4.3, but in brief, the

[2] We use Spearman rather than Pearson correlation because networks are nonlinear, and there is no particular reason to expect associations between arbitrary neurons to be linear.

[3] Although all negative values are mapped to zero by a ReLU, we expect the degree of negativity to carry information about the possible presence (or lack thereof) of features which the neuron was meant to detect, especially for networks trained with dropout.

measure for importance of a subcluster quantifies the performance reduction from dropping out the neurons in that subcluster, and the measures for coherence quantify the degree to which the neurons in a subcluster are mutually associated with some feature of an input. For each subcluster with more than one neuron and which does not include every neuron in the layer, we calculate a measure of importance or coherence and compare it to that of 19 random subclusters. These experiments and measures are discussed in detail in Section 4.4. We present two measures of how true and random subclusters compare under these proxies. First, we calculate a measure of whether true subclusters are disproportionately often more important or coherent than random subclusters, called the Fisher statistic. This value is our primary focus. However, for additional resolution, we also calculate an effect measure used to assess the importance and coherence of the 'typical' subcluster. Importantly, the Fisher statistic and effect measure quantify different things.

Fisher statistics: We wish to test whether spectral clustering methods find subsets of neurons which satisfy our proxies for local specialization more than if we had simply chosen random subsets. To do this, for each subcluster measurement we take the percentile of each true subcluster relative to the distribution of measurements of random subclusters.[4] Next, we use the Fisher method to test whether the subclusters in a single network satisfy our proxies more than random subsets of neurons. To do so, we first center the subcluster percentiles around 0.5, which under the null hypothesis would give a granular, unbiased approximation of the uniform distribution. We then combine the centered percentiles $\{p_1, \ldots, p_n\}$ into the Fisher statistic $(-1/n)\sum_{i=1}^{n} \log p_i$. For reference, the Fisher statistic of a uniform distribution of percentiles in our setting is 0.98. Figure 4.3 shows example distributions of percentiles and their Fisher statistics. Note also that since the log function has a larger derivative near 0 than near 1, low percentiles have greater influence on the Fisher statistic, so J- or U-shaped distributions can also have Fisher statistics greater than 1, even if there are more high percentiles than low percentiles. For all non-ImageNet architectures, we train and analyze 5 different networks per condition and report the mean Fisher statistic.

The Fisher statistic of n uniformly-distributed random percentiles multiplied by $2n$ takes a chi-squared distribution with $2n$ degrees of freedom. This lets us produce a p value for each network, testing whether this statistic is higher than the null hypothesis would produce, which would mean that there were more low percentiles than if subclusters were distributed uniformly.[5] This procedure is illustrated in Figure 4.2. For non-ImageNet architectures, we need to aggregate the five p values we get per condition, one for each network. To do so, in each condition, we take the mean of the p values and correct it using the corresponding quantile of a Bates($n = 5$) distribution[6] which gives the distribution of the mean of 5 independent random variables uniformly distributed on $[0, 1]$.[7]

[4] For metrics where high values indicate local specialization, we take the percentile of the negative metric, so that low percentiles consistently indicate local specialization.

[5] The fact that we coarsely measure percentiles and then center them makes this test conservative because our statistic is more sensitive to low percentiles than high percentiles.

[6] The Bates(n) quantile function is $F_n(x) = \frac{1}{n!}\sum_{k=0}^{\lfloor nx \rfloor}(-1)^k\binom{n}{k}(nx-k)^{n-1}$ (Marengo et al., 2017).

[7] We use the Bates method for aggregation here instead of using the Fisher method because in this case,

Multiple testing correction: Methods above produce a p value for every network architecture, partitioning method, and local specialization proxy. We next correct for multiple testing using the Benjamini-Hochberg method (Benjamini & Hochberg, 1995), controlling the false discovery rate: that is, the expected proportion of rejections of the null hypothesis that are false positives, where the expectation is taken under the data-generating distribution. For a false discovery rate α and m ordered p values $\{p_1, \ldots, p_m\}$, this procedure chooses a critical p value as p_k where $k = \arg\max_{j \in 1:m} \mathbb{I}(p_j \leq \frac{j\alpha}{m})$ where $\mathbb{I}$ is an indicator. All p values greater than p_k are deemed not significant at this level. In our case, the critical value was $p_k = 0.025$. Table 4.1 shows in bold the Fisher statistics (or means thereof) that are statistically significantly greater than 0.98 after this correction, where significance is taken at the $\alpha = 0.05$ level.

In summary, (1) we aggregate each network's subcluster percentiles using the Fisher method, yielding a Fisher statistic and a p value, (2) we aggregate the Fisher statistics and p values across identically-configured replicates of the same experiment by taking the mean and using the Bates method, respectively, and (3) we correct for multiple comparisons using the Benjamini-Hochberg procedure.

Effect measures: In addition to p values, we also calculate effect measures which give a sense of how different results are for true and random subclusters are *on average*. The effect measure is the mean over subclusters of $2x/(x + \mu)$, where x is a true subcluster measure of importance/coherence and μ is the mean over that of random subclusters.[8] We do this as opposed to simply taking x/μ to avoid division by zero. This results in effect measures in the interval $[0, 2]$, and which side of 1 they are on indicates whether the true subclusters are more important/coherent than random ones. For ease of interpretation, note that if $2x/(x + \mu) = 1 + y$, then $x/\mu \approx 1 + 2y$ for $y \ll 1$, so (for example) an effect measure of 1.05 would mean that the measure of a true subcluster was $\approx 10\%$ higher than the expected measure of a random subcluster. Together with these effect measures, we also report their standard errors (considering reported effect measures as estimates of the mean network-wide effect measures over the whole distribution of trained networks).

Differences between Fisher statistics and effect measures: In some of our experiments, the Fisher statistics and effect measures seem to "disagree" with one suggesting that the network was locally specialized and the other suggesting it was not. This potential for disagreement is due to the fact that Fisher statistics are based on the percentiles of subcluster measurements relative to the distribution of those of random subclusters, while effect measures compare the value of subcluster measurements to the mean value of random subcluster measurements. When the Fisher statistic seems to indicate local specialization but the effect measure does not, this means that on the relevant metric, there are more subclusters than would be expected under the null hypothesis whose metric value is higher than that of random subclusters, but the "typical" subcluster has a metric value similar to or less than an average random subcluster. In other words, the partitioning method has

all five p values are from identically configured experiments, and the Bates test is less sensitive to low outliers.

[8] If x is ever less than 0 for a subcluster, we conservatively replace it with 0.

detected some subclusters that satisfy our proxies for local specialization, but the typical subcluster found does not. This is compatible with a J- or U-shaped distribution of subcluster percentiles. We consider this a positive result, indicating that our partitioning method is detecting some local specialization.

4.4 Experiments

To show the applicability of our methods at different scales, we experiment with a range of networks. For small-scale experiments, we train MLPs with 4 hidden layers of 256 neurons each and small convolutional networks with 3 layers each of 64 neurons followed by a dense layer of 128 neurons trained on the MNIST (LeCun et al., 1998) dataset. At a mid scale, we train VGG-style CNNs containing 13 convolutional layers using the architectures of Simonyan and Zisserman (2015) trained on CIFAR-10 (Krizhevsky, 2009) using the procedure of S. Liu and Deng (2015), which includes weight decay and dropout for regularization. Finally, for ImageNet (Krizhevsky, 2009) scale, we analyze pretrained ResNet-18 (He et al., 2016a), VGG-16, and VGG-19 (Simonyan & Zisserman, 2015) models.

4.4.1 Network training details

We use Tensorflow's implementation of the Keras API (Chollet et al., 2015; Martín Abadi et al., 2015). When training all networks, we use the Adam algorithm (Kingma & Ba, 2014) with the standard Keras hyperparameters: learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, no amsgrad. The loss function was categorical cross-entropy. For all non-ImageNet networks, we train 5 identically-configured replicates.

Small MLPs (MNIST): We train MLPs with 4 hidden layers, each of width 256, for 20 epochs with batch size 128. All MLPs achieved a test accuracy on MNIST between 97.6% and 98.2%. On Fashion-MNIST, they achieved test accuracies between 88.4% and 89.4%.

Small CNNs (MNIST): These networks had 3 convolutional layers with 64 3×3 channels each with the second and third hidden layers being followed by max pooling with a 2 by 2 window. There was a final fully-connected hidden layer with 128 neurons. We train them with a batch size of 64 for 10 epochs. All small CNNs achieved a testing accuracy MNIST between 99.1% and 99.4%. On Fashion-MNIST, they achieved test accuracies between 91.8% and 92.4%.

Mid-sized VGG CNNs (CIFAR-10): We implement a version of VGG-16 described by S. Liu and Deng (2015) and Simonyan and Zisserman (2015). We train these with Adam, for 200 epochs with a batch size of 128. These are trained using L_2 regularization with a coefficient of 5×10^{-4} and dropout with a rate tuned per-layer as done in S. Liu and Deng (2015) and Simonyan and Zisserman (2015). Training was done with data augmentation which consisted of random rotations between 0 and 15 degrees, random shifts both vertically and horizontally of up to 10% of the side length, and random horizontal flipping. Test accuracies were between 82.8% and 86.6%.

Large CNNs (ImageNet): We experimented with VGG-16 and 19 (Simonyan & Zisserman, 2015) and ResNet-18 (He et al., 2016a) networks. Weights were obtained from the Python `image-classifiers` package, version 1.0.0.

Table 4.1: Fisher statistics (or means over 5 runs) for (1) lesion-based experiments measuring importance via overall accuracy drops (Acc. Drop) and coherence via the class-wise range of accuracy drops (Class Range); and (2) Feature visualization-based experiments in networks measuring coherence via the optimization score (Vis Score) and the entropy of network outputs (Softmax H). Each row corresponds to a network paired with a partitioning method. Fisher statistics above 0.98 indicate that subclusters satisfy our local specialization proxies disproportionately more than random subclusters do. Values statistically significantly greater than 0.98 are **bolded**. Section 4.3.2 details the calculation of these statistics and their p values. For tables that include p values, see Appendix B.

		Lesion		**Feature Visualization**	
Network	**Partitioning**	**Acc. Drop**	**Class Range**	**Vis Score**	**Softmax H**
MLP, MNIST	Weight/Net.	**2.13**	0.91	**1.32**	0.92
	Weight/Layer	**2.05**	0.91	**1.21**	**1.23**
	Act./Net.	**1.46**	1.00	**1.34**	**1.15**
	Act./Layer	**1.69**	1.03	**1.36**	**1.12**
CNN, MNIST	Weight/Net.	**1.29**	0.84	1.10	0.93
	Weight/Layer	1.10	0.94	1.02	0.99
	Act./Net.	**1.73**	0.70	1.09	0.90
	Act./Layer	**1.46**	0.92	1.05	0.98
VGG, CIFAR-10	Weight/Net.	**1.50**	**2.12**	**1.46**	0.99
	Weight/Layer	**1.15**	**1.27**	1.00	0.97
	Act./Net.	**1.40**	0.97	**2.34**	1.08
	Act./Layer	**1.56**	1.03	**2.67**	**1.12**
VGG-16, ImageNet	Weight/Net.	**2.54**	0.49	**1.72**	1.19
	Weight/Layer	**2.15**	0.56	**1.90**	1.06
	Act./Net.	**1.89**	0.63	**1.82**	1.07
	Act./Layer	**1.66**	0.70	**1.85**	0.98
VGG-19, ImageNet	Weight/Net.			**1.91**	1.03
	Weight/Layer			**2.23**	1.00
	Act./Net.			**1.87**	1.10
	Act./Layer			**2.01**	0.98
ResNet-18, ImageNet	Weight/Net.	**1.42**	1.13		
	Weight/Layer	**1.29**	0.99		
	Act./Net.	**1.30**	0.92		
	Act./Layer	**1.31**	0.96		

4.4.2 Lesion experiments

One approach that has been used for understanding both biological (Gazzaniga & Ivry, 2013) and artificial (Casper, Boix, et al., 2021; B. Zhou et al., 2018) neural systems involves disrupting neurons during inference. We experiment with "lesion" tests in which we analyze network performance on the test set when a subcluster is dropped out. We then analyze the damage to the network's performance. First, we measure importance by taking the drop in accuracy. Specifically, let θ be the parameter vector of the neural network, c be a set of neurons, $\mathcal{M}(\theta, c)$ be a masked version of θ where weights into or out of nodes in c have been set to 0, and $\mathrm{Acc}(\vartheta, \mathcal{D})$ be the accuracy of the network parameterized by ϑ on dataset $\mathcal{D}$. Then, our measure for importance is $\mathrm{Acc}(\theta, \mathtt{test}) - \mathrm{Acc}(\mathcal{M}(\theta, c), \mathtt{test})$, where $\mathtt{test}$ is a test dataset that was not used to construct the activation-based partitionings.

Second, we measure the coherence in a subcluster with respect to class by taking the range of class-specific accuracy drops. Specifically, let $\mathtt{test}_i$ be the subset of the test set with label i, and let $\Delta(\theta, c, i) := \mathrm{Acc}(\theta, \mathtt{test}_i) - \mathrm{Acc}(\mathcal{M}(\theta, c), \mathtt{test}_i)$ be the drop in accuracy for examples with label i from lesioning c. Then, this measure of coherence is the range $(\max_i \Delta(\theta, c, i) - \min_i \Delta(\theta, c, i))$ of accuracy drops over classes. We use this to detect whether clusters are more crucial for some classes over others, which would suggest that they coherently act to correctly label those classes.

We use the analysis pipeline from Section 4.3.2 to test for importance and coherence using these overall accuracy differences and class-wise ranges. In this setting, effect measures > 1 indicate more importance/coherence among true subclusters on average compared to random ones. Table 4.1 shows Fisher statistics, and Table 4.2 shows effect measure data. Results are summarized in Section 4.4.6.

4.4.3 Feature visualization experiments

To further analyze coherence, we leverage another set of interpretability techniques based on feature visualization. We use gradient-based optimization to create an input image which maximizes the L_1 norm of the pre-ReLU activations of the neurons in a subcluster. Letting the parameter vector be θ and the subcluster be c, we write $\mathrm{Act}(x, \theta, c)$ for the vector of pre-ReLU activations of neurons in c in network θ on input x, and denote this optimized input image as $x(\theta, c)$, which approximately maximizes $\|\mathrm{Act}(x, \theta, c)\|_1$. The key insight is that properties of these visualizations $x(\theta, c)$ can suggest what roles the subclusters play in the network.

All visualizations were created using the Lucid[9] package. The optimization objective for visualizing subclusters was the L_1 norm of the pre-ReLU inputs for all neurons inside the subcluster (it was an L_1 norm of L_1 norms for convolutional feature maps). For small MLPs, small CNNs, and mid-sized CNNs, we generated images using random jittering and scaling, and for ImageNet models, we used Lucid's default transformations which consist of padding, jittering, rotation, and scaling with default hyperparameters. For all networks, we used the

[9] https://github.com/tensorflow/lucid

Table 4.2: Effect measures and their standard errors for (1) lesion-based experiments measuring importance via overall accuracy drops (Acc. Drop) and coherence via the class-wise range of accuracy drops (Class Range); and (2) Feature visualization-based experiments in networks measuring coherence via the optimization score (Vis Score) and the entropy of network outputs (Softmax H). Each row corresponds to a network paired with a partitioning method. Results are calculated as explained in Section 4.3.2. For accuracy drop, class-wise range and visualization score experiments, an effect measure > 1 corresponds to more importance/coherence among true subclusters than random ones, while one of < 1 does for softmax entropy experiments. Entries where the effect measure is more than two standard errors away from 1 in the direction of local specialization are **bolded**. Note that ImNet is short for ImageNet. For tables that include p values, see Appendix B.

		Lesion		**Feature Visualization**	
Network	**Partitioning**	**Acc. Drop**	**Class Range**	**Vis Score**	**Softmax H**
		High→Imp.	**High→Coh.**	**High→Coh.**	**Low→Coh.**
MLP, MNIST	Weight/Net.	**1.123 ± 0.058**	0.701 ± 0.036	1.003 ± 0.004	1.105 ± 0.021
	Weight/Layer	1.061 ± 0.048	0.676 ± 0.029	**1.024 ± 0.004**	**0.931 ± 0.016**
	Act./Net.	0.883 ± 0.038	0.646 ± 0.024	**1.02 ± 0.003**	0.997 ± 0.013
	Act./Layer	0.929 ± 0.040	0.687 ± 0.025	**1.026 ± 0.003**	1.011 ± 0.012
CNN, MNIST	Weight/Net.	0.837 ± 0.048	0.527 ± 0.031	0.998 ± 0.004	1.026 ± 0.008
	Weight/Layer	0.814 ± 0.046	0.635 ± 0.033	1.004 ± 0.003	1.007 ± 0.007
	Act./Net.	1.078 ± 0.061	0.543 ± 0.039	0.933 ± 0.006	1.025 ± 0.011
	Act./Layer	0.939 ± 0.060	0.625 ± 0.043	0.925 ± 0.005	**0.970 ± 0.010**
VGG, CIFAR-10	Weight/Net.	0.682 ± 0.066	0.407 ± 0.042	0.871 ± 0.011	1.124 ± 0.012
	Weight/Layer	0.808 ± 0.041	0.692 ± 0.033	**1.013 ± 0.006**	0.992 ± 0.012
	Act./Net.	0.926 ± 0.032	0.679 ± 0.023	**1.327 ± 0.005**	**0.950 ± 0.009**
	Act./Layer	0.956 ± 0.030	0.695 ± 0.021	**1.379 ± 0.004**	**0.930 ± 0.008**
VGG-16, ImNet	Weight/Net.	**1.205 ± 0.050**	0.790 ± 0.010	**1.043 ± 0.005**	0.998 ± 0.001
	Weight/Layer	**1.168 ± 0.019**	0.825 ± 0.005	**1.076 ± 0.003**	**0.991 ± 0.001**
	Act./Net.	**1.129 ± 0.026**	0.859 ± 0.006	**1.066 ± 0.003**	1.000 ± 0.001
	Act./Layer	**1.063 ± 0.021**	0.876 ± 0.005	**1.056 ± 0.003**	1.001 ± 0.001
VGG-19, ImNet	Weight/Net.			**1.061 ± 0.004**	1.003 ± 0.001
	Weight/Layer			**1.099 ± 0.003**	1.001 ± 0.001
	Act./Net.			**1.046 ± 0.003**	**0.996 ± 0.001**
	Act./Layer			**1.081 ± 0.002**	1.004 ± 0.001
ResNet-18, ImNet	Weight/Net.	0.926 ± 0.045	0.957 ± 0.011		
	Weight/Layer	0.971 ± 0.016	0.971 ± 0.004		
	Act./Net.	0.979 ± 0.017	0.977 ± 0.004		
	Act./Layer	0.983 ± 0.014	0.967 ± 0.004		

pixel-based parameterization of the image and no regularization on the Adam optimizer for 100 steps. For visualizations in small MLPs and CNNs, we used versions of these networks trained on 3-channel versions of their datasets in which the same inputs were stacked thrice because Lucid requires networks to have 3-channel inputs. However, we show greyscaled versions of these in Figure 4.4.

We use two techniques to analyze coherence using these visualizations of subclusters. First, we analyzed the value of the maximization objective for each image we produced, $\|\text{Act}(x(\theta, c), \theta, c)\|_1$, which we call the "score" of the visualization. This gives one notion of how coherent a subcluster may be with respect to input features, because if a single image can strongly excite an entire subcluster, this suggests that the neurons comprising it are involved in detecting/processing related features. Second, we obtain a measure of coherence by analyzing the entropy $H(\text{label} \mid x(\theta, c); \theta)$ of the softmax outputs of the network when these images are passed through. If the entropy is low, this suggests that a cluster is coherent with respect to class labels.

Just as with lesion experiments, we perform analysis using these two methods using the pipeline from Section 4.3.2 to measure how coherent true subclusters are compared to random ones.

For visualization score experiments, effect measures > 1 indicate coherence while for the softmax H experiments, effect measures < 1 indicate coherence. Table 4.1 shows Fisher statistics, and Table 4.2 shows effect measure data.

4.4.4 Robustness to cluster number

In previous subsections, we only present results from experiments in which the number of clusters, k, was set to 16 for sub-ImageNet networks and 32 for ImageNet ones. The fact that we find evidence of local specialization in networks across a range of sizes using $k = 16$ suggests that detecting it is robust to k. Here, we also present a direct comparison between results for CIFAR VGGs for $k \in \{8, 12, 16\}$. Tables 4.3 and 4.4 show these results for lesion and feature-visualization experiments respectively. In general, whether or not an experiment resulted in a Fisher statistic or an effect measure suggesting local specialization is consistent across these values of k.

4.4.5 Redundancy

If multiple neurons in a network are redundant, having similar weights and activations, our clustering methods will likely group them. These groups of redundant neurons would be coherent and perhaps important, thereby demonstrating functional locality in a way that is arguably less interesting than for a non-atomic subtask. However, two pieces of evidence suggest that redundancy is not the sole driver of our results.

First, activation-based clusterings that group neurons with correlated activity are more likely to detect redundancy than weight-based clusterings. However, Table 4.1 shows that the activation-based clusterings are not reliably more important or coherent than weight-based clusterings, suggesting that the importance and coherence is not only due to redundancy.

Second, in addition to the VGGs trained on CIFAR-10 with L_2 regularization and dropout that we analyze in Table 4.1, we also train and analyze unregularized versions. Since dropout encourages redundant information to be encoded in multiple neurons, and L_2 regularization

Table 4.3: Comparison for different cluster number values in VGG networks for lesion-based experiments. Results are shown for $k \in \{8, 12, 16\}$. Fisher statistics are means over 5 runs. Each Fisher statistic which is significant at the $\alpha = 0.05$ after Benjamini-Hochberg correction is **bold**. Effect measures are reported together with their standard errors. An effect measure > 1 corresponds to more importance/coherence among true subclusters than random ones. All above 1 are **bold**.

		Importance: Acc. Drop		**Coherence: Class Range**	
Network	**Partitioning**	**Fisher**	**Effect Meas.**	**Fisher**	**Effect Meas.**
Cluster number (k)		**stat.**	**High→Imp.**	**stat.**	**High→Coh.**
VGG, CIFAR-10 $k = 8$	Weight/Net.	**1.85**	0.910 ± 0.102	**1.94**	0.270 ± 0.045
	Weight/Layer	1.03	0.854 ± 0.047	**1.17**	0.828 ± 0.039
	Act./Net.	**1.49**	0.966 ± 0.038	0.98	0.748 ± 0.030
	Act./Layer	**1.77**	1.051 ± 0.035	0.93	0.752 ± 0.027
VGG, CIFAR-10 $k = 12$	Weight/Net.	**1.63**	0.737 ± 0.078	**2.01**	0.388 ± 0.045
	Weight/Layer	1.05	0.867 ± 0.042	1.09	0.736 ± 0.035
	Act./Net.	**1.44**	0.901 ± 0.035	1.04	0.721 ± 0.028
	Act./Layer	**1.57**	0.994 ± 0.032	0.96	0.694 ± 0.023
VGG, CIFAR-10 $k = 16$	Weight/Net.	**1.50**	0.682 ± 0.066	**2.12**	0.407 ± 0.042
	Weight/Layer	**1.15**	0.808 ± 0.041	**1.27**	0.692 ± 0.033
	Act./Net.	**1.40**	0.926 ± 0.032	0.97	0.679 ± 0.023
	Act./Layer	**1.56**	0.956 ± 0.030	1.03	0.695 ± 0.021

encourages each neuron to depend on many previous-layer neurons, these unregularized networks should have less redundancy than those analyzed in the main paper. Our results in Table 4.5 show unregularized networks as having more importance and coherence in many instances, suggesting that our measures are not just picking up redundancy.

4.4.6 Findings

Our partitionings identify important subclusters. Fisher statistics for lesion accuracy drops are high and significant, as shown in Table 4.1, indicating that subclusters are more likely to be highly important relative to random groups of neurons. However, as shown in Table 4.2, not all of the corresponding effect measures are above one, even when the Fisher statistic is significantly greater than 0.98. This indicates that when we detect that an unusual number of subclusters are important, this does not necessarily correspond to importance on average.

Our partitionings identify subclusters that are coherent w.r.t. input features but not class label. Class-specific measures of coherence, class-wise lesion accuracy drop range and output entropy, showed significant coherence in almost no conditions. The class-wise range measure even tended to show that subclusters were less coherent w.r.t. class

Table 4.4: Comparison for different cluster values in VGG networks for feature visualization-based experiments. Results are shown for $k \in \{8, 12, 16\}$. Fisher statistics are means over 5 runs. Each mean Fisher statistic which is significant at the $\alpha = 0.05$ level after Benjamini-Hochberg correction is **bold**. Effect measures are reported together with their standard errors. For vis score tests, an effect measure > 1 corresponds to more coherence, and for softmax H tests, one of < 1 corresponds to more coherence. All effect measures on the side of 1 indicating more coherence are **bold**.

		Coherence: Vis Score		**Coherence: Softmax H**	
Network **Cluster number** (k)	**Partitioning**	**Fisher stat.**	**Effect Meas. High→Coh.**	**Fisher stat.**	**Effect Meas. Low→Coh.**
VGG, CIFAR-10 $k = 8$	Weight/Net.	**1.87**	1.011 ± 0.016	1.18	1.028 ± 0.017
	Weight/Layer	0.83	0.974 ± 0.005	0.90	1.059 ± 0.015
	Act./Net.	**2.61**	**1.345 ± 0.006**	1.02	**0.940 ± 0.010**
	Act./Layer	**2.72**	**1.358 ± 0.005**	1.06	**0.946 ± 0.009**
VGG, CIFAR-10 $k = 12$	Weight/Net.	**1.59**	0.959 ± 0.013	0.89	1.106 ± 0.013
	Weight/Layer	0.96	0.993 ± 0.005	1.12	**0.945 ± 0.012**
	Act./Net.	**2.60**	**1.363 ± 0.005**	1.12	**0.917 ± 0.009**
	Act./Layer	**2.73**	**1.364 ± 0.004**	1.07	**0.928 ± 0.008**
VGG, CIFAR-10 $k = 16$	Weight/Net.	**1.46**	0.871 ± 0.011	0.99	1.124 ± 0.012
	Weight/Layer	1.00	**1.013 ± 0.006**	0.97	0.992 ± 0.012
	Act./Net.	**2.34**	**1.327 ± 0.005**	1.08	**0.950 ± 0.009**
	Act./Layer	**2.67**	**1.379 ± 0.004**	**1.12**	**0.930 ± 0.008**

than random groups of neurons. However, subclusters were reliably coherent as measured by *visualization score*, both as quantified by Fisher statistics and effect measures. Together, these results offer evidence that subclusters tended to perform coherent sub-tasks, but not in a class-specific way.

All partitioning methods yield similar results. In Table 4.1, we find no clear difference between the Fisher statistics of activation-based and weight-based clusterings, or between layer-wise and network-wide clusterings. This is somewhat unexpected: one might have predicted that weight-based methods' lack of runtime information or layer-wise methods' lack of global information would lead to lower quality clusterings, but this was not the case.

4.5 Discussion

Contributions: In this work, we introduce several methods for partitioning networks into clusters of neurons and analyzing the resulting partitions for local specialization. Key to this is measuring proxies: *importance* as a means of understanding what parts of a network are crucial for performance, and *coherence* as a measure for how much the neurons in a part work

Table 4.5: Comparison for regularized and unregularized CNN-VGG networks for lesion-based experiments. Results are shown for $k \in \{8, 12, 16\}$. Fisher statistics are means over 5 runs. Each Fisher statistic which is significant at the $\alpha = 0.05$ after Benjamini-Hochberg correction is **bold**. Effect measures are reported together with their standard errors. An effect measure > 1 corresponds to more importance/coherence among true subclusters than random ones. All above 1 are **bold**.

		Importance: Acc. Drop		**Coherence: Class Range**	
Network	**Partitioning**	**Fisher**	**Effect Meas.**	**Fisher**	**Effect Meas.**
		stat.	**High→Imp.**	**stat.**	**High→Coh.**
VGG, CIFAR-10 Regularized	Weight/Net.	**1.50**	0.682 ± 0.066	**2.12**	0.407 ± 0.042
	Weight/Layer	**1.15**	0.808 ± 0.041	**1.27**	0.692 ± 0.033
	Act./Net.	**1.40**	0.926 ± 0.032	0.97	0.679 ± 0.023
	Act./Layer	**1.56**	0.956 ± 0.030	1.03	0.695 ± 0.021
VGG, CIFAR-10 Unregularized	Weight/Net.	**1.27**	**1.033 ± 0.007**	1.13	0.995 ± 0.015
	Weight/Layer	0.94	0.994 ± 0.004	0.92	**1.042 ± 0.011**
	Act./Net.	**1.48**	**1.032 ± 0.003**	**1.25**	**1.022 ± 0.011**
	Act./Layer	**1.76**	**1.049 ± 0.003**	**1.17**	**1.077 ± 0.010**

Table 4.6: Comparison for regularized and unregularized CNN-VGG networks for feature visualization-based experiments. Results are shown for $k \in \{8, 12, 16\}$. Fisher statistics are means over 5 runs. Each mean Fisher statistic which is significant at the $\alpha = 0.05$ level after Benjamini-Hochberg correction is **bold**. Effect measures are reported together with their standard errors. For vis score tests, an effect measure > 1 corresponds to more coherence, and for softmax H tests, one of < 1 corresponds to more coherence. All effect measures on the side of 1 indicating more coherence are **bold**.

		Coherence: Vis Score		**Coherence: Softmax H**	
Network	**Partitioning**	**Fisher**	**Effect Meas.**	**Fisher**	**Effect Meas.**
		stat.	**High→Coh.**	**stat.**	**Low→Coh.**
VGG, CIFAR-10 Regularized	Weight/Net.	**1.46**	0.871 ± 0.011	0.99	1.124 ± 0.012
	Weight/Layer	1.00	**1.013 ± 0.006**	0.97	0.992 ± 0.012
	Act./Net.	**2.34**	**1.327 ± 0.005**	1.08	**0.950 ± 0.009**
	Act./Layer	**2.67**	**1.379 ± 0.004**	**1.12**	**0.930 ± 0.008**
VGG, CIFAR-10 Unregularized	Weight/Net.	**1.27**	**1.033 ± 0.007**	1.13	0.995 ± 0.015
	Weight/Layer	0.94	0.994 ± 0.004	0.92	1.0424 ± 0.011
	Act./Net.	**1.48**	**1.032 ± 0.003**	**1.25**	1.022 ± 0.011
	Act./Layer	**1.76**	**1.049 ± 0.003**	**1.17**	1.077 ± 0.010

together. We rigorously evaluate these proxies using statistical methods, finding that even the weights-only clustering methods are able to reveal clusters with a significant degree of importance and coherence compared to random ones. In each network, we found evidence that our partitioning methods were able to identify specialized subsets of neurons via measuring accuracy drops under lesions (importance) and feature visualization scores (coherence). To the best of our knowledge, ours is the first method which is able to quantitatively assess the local specialization of neural networks in a way that does not require a human in the loop.

Relation to other research: Having effective tools for interpreting networks is important for understanding AI systems, in particular by helping to diagnose failure modes (e.g., Carter et al. (2019), Casper, Nadeau, et al. (2021), and Mu and Andreas (2020)). Our work relates to this goal, though indirectly. The tests we perform are based on data from lesions and feature visualizations, both of which are interpretability tools. But rather than directly using these data to interpret subclusters, our focus is one step higher: on automatically testing whether these subclusters are worth analyzing at all and finding ways to screen for subclusters that should be the subject of deeper investigation. By showing that the partitioning methods we use generate partitionings that align with local specialization, these results suggest that clustering neurons offers a useful level of abstraction through which to study networks.

Limitations: One limitation of our work is a lack of assurance that importance and coherence are reliably strong proxies for human-comprehensible forms of local specialization. While they are sufficient to imply some degree of abstractability with respect to the task at hand, they may not always be particularly useful for understanding the network. Relatedly, our approach is also not designed to identify the sub-task performed by a group of neurons, nor does it identify relationships between groups. Given these limitations, the tools we introduce should be seen as methods for screening a network for evidence of local specialization overall and for particular sets of neurons where sub-task functionality is localized. A final notable limitation is that these methods do not offer tools for building more modular networks beyond techniques for measuring local specialization. Future work toward this may benefit from our techniques but should also hinge on architectural or regularization-based methods for promoting independent operation of groups of neurons.

Conclusion: While we make progress toward better understanding how networks can be understood, neural systems are still complex, and more insights are needed to develop useful understandings of them. The ultimate goal should be to develop reliable methods for building models which perform well and also lend themselves to faithful abstract interpretations. We believe that these methods should involve testing networks for local specialization and investigating the functions that important or coherent parts of the network specialize in.

Figure 4.2: **Our extended procedural pipeline.** This figure expands Figure 4.1 and shows the successive steps after generating a partitioning of subclusters (step 3 in Figure 4.1). After performing either lesion or feature visualization analysis, the results from each true subcluster and its random subclusters are aggregated to produce p values and effect measures. For simplicity, only the analysis for the lesion experiment is presented, but the same pipeline is used for the feature visualization experiments.

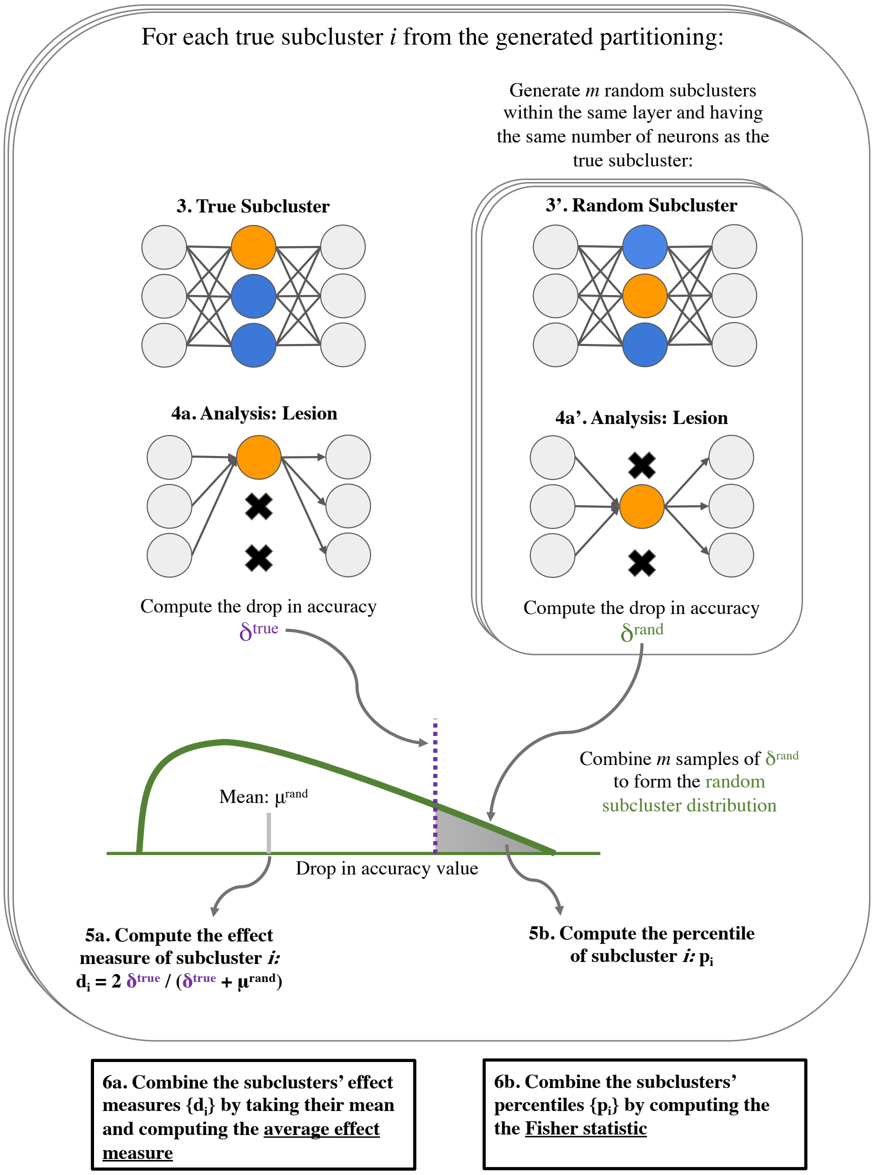
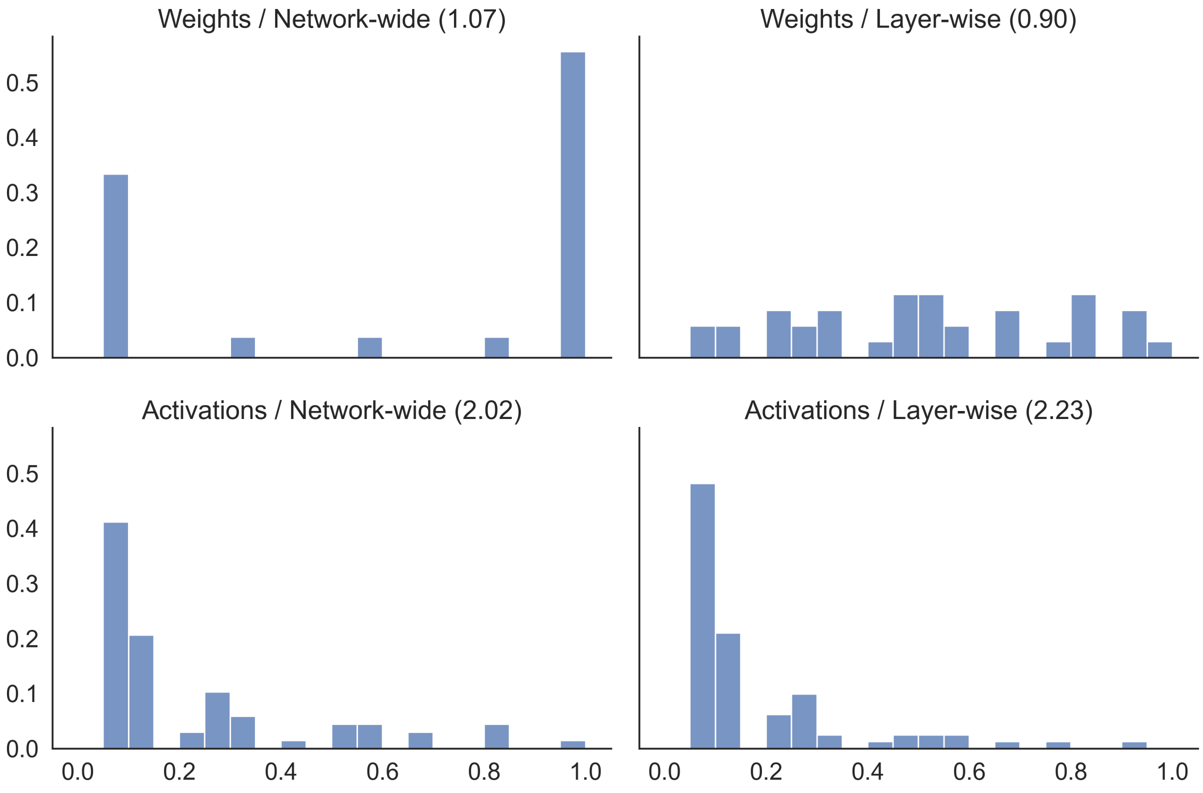

For each true subcluster *i* from the generated partitioning:
Generate *m* random subclusters within the same layer and having the same number of neurons as the true subcluster:
3. True Subcluster
3'. Random Subcluster
4a. Analysis: Lesion
4a'. Analysis: Lesion
Compute the drop in accuracy
δ^{true}
Compute the drop in accuracy
δ^{rand}
Combine *m* samples of δ^{rand} to form the random subcluster distribution
Mean: μ^{rand}
Drop in accuracy value
5a. Compute the effect measure of subcluster *i*:
$d_i = 2\ \delta^{true} / (\delta^{true} + \mu^{rand})$
5b. Compute the percentile of subcluster *i*: p_i
6a. Combine the subclusters' effect measures $\{d_i\}$ by taking their mean and computing the average effect measure
6b. Combine the subclusters' percentiles $\{p_i\}$ by computing the the Fisher statistic

Figure 4.3: **Illustration of Fisher statistics of various percentile distributions.** A VGG network trained on CIFAR-10 is partitioned using four methods ({weights, activations} × {network-wide, layer-wise}) and analyzed for coherence (see discussion of visualization scores in Section 4.4.3) to produce the collection of percentiles for each subcluster. This figure shows histograms of the percentile distribution for each clustering, and their associated Fisher statistics. Recall that a lower percentile means that a true subcluster is more coherent than random subclusters while controlling for layer and size. The activation-based clusterings have disproportionately many low percentiles, and Fisher statistics greater than 2. Table 4.1 shows that these trends are statistically significant when aggregated over five models.



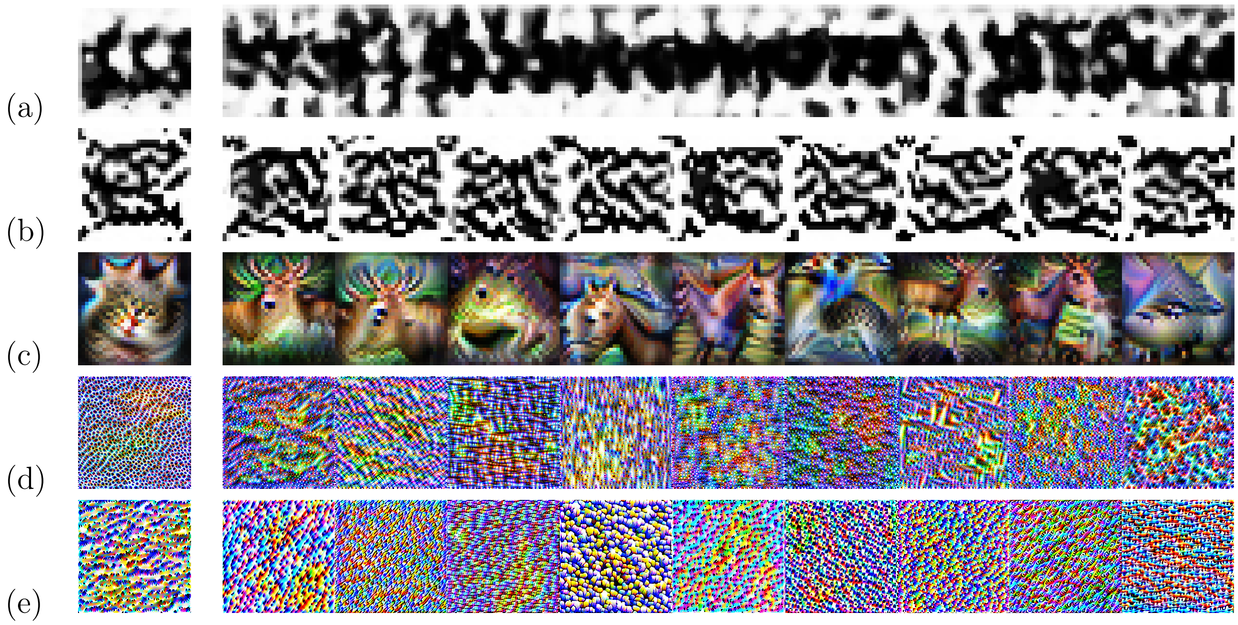


Figure 4.4: **Example feature visualizations for true and random subclusters:** In the left column are shown true subcluster visualizations, and in the right column are visualizations of subclusters of random neurons of the same size in the same layer. (a) MLP, MNIST; (b) CNN, MNIST; (c) VGG, CIFAR-10; (d) VGG-16, ImageNet; (e) VGG-19, ImageNet.

Chapter 5

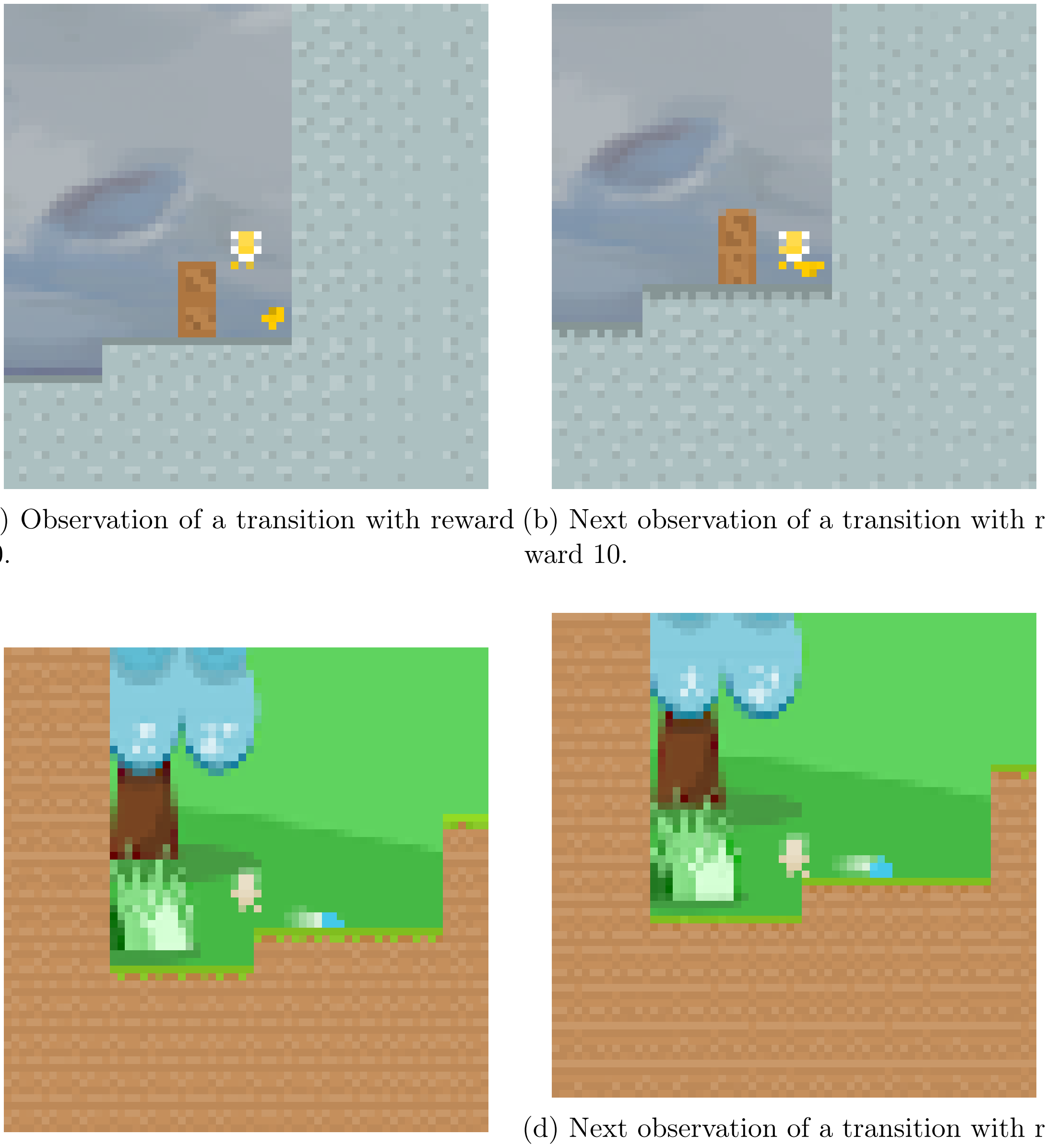
Feature Representation in Reward Models

5.1 Introduction

In Chapters 3 and 4, we developed a frame on neural network interpretability that emphasized cluster structure in the learned network weights. This is encouraging as a step towards being able to think of neural networks as structured programs, as described in the introduction of the deep learning textbook by Goodfellow et al. (2016). However, one negative result of Chapter 4 was that we do not see structures that are differentially useful for specific output classes. This is in some tension with the standard view of neurons as learning high-level human-comprehensible features, since one would expect some of those features to be more relevant for some output classes than others.

In this chapter, we provide a case study of networks trained to predict rewards in the CoinRun environment: a platformer game where the agent must move their sprite to a coin at the end of the level, without jumping onto an enemy or into lava (Cobbe et al., 2020). Reaching the coin, jumping on an enemy, or jumping into lava immediately ends the game. Transitions where the agent reaches the coin receive a reward of 10, and all other transitions receive a reward of 0. Examples of transitions are shown in Figure 5.1. Visualizations of the final neuron, which reward predictions are read off of, reliably failed to indicate a sensible representation of whether the agent sprite had reached the coin. Indeed, the networks could not be probed for a high-accuracy linear representation of distance from the agent sprite to the coin, and were able to be fooled into assigning no reward to transitions where the agent reached the coin. Furthermore, they assign high reward to transitions where the observation and next observation are random pixels. We also see similar problems in networks trained on similar environments.

The significance of this chapter is two-fold. Firstly, it shows difficulty for the hypothesis that networks represent human-comprehensible features of relevance for determining what the correct output should be. We repeatedly show failures of this hypothesis: adversarial examples show that such representations either do not exist or are not used appropriately; and probing indicates that we do not see linear representations of features that are crucial



(a) Observation of a transition with reward 10.

(b) Next observation of a transition with reward 10.

(c) Observation of a transition with reward 0.

(d) Next observation of a transition with reward 0.

Figure 5.1: Examples of CoinRun transitions

to the way we would hope networks compute reward. This reinforces the aforementioned difficulties posed by the results of Chapter 4.

Secondly, it serves as a toy example motivating goal misgeneralization concerns. If we imagine that these reward nets were to act as sub-systems of an agent, this agent would face problems of goal misgeneralization in environments containing our manual adversarial examples, or even environments where it is possible for the agent to supply itself with random pixels. These issues exist despite the fact that the reward networks are trained on the true underlying reward, which is more reward access than is usually provided (especially in cases where the relevant reward function is implicit in human judgements and not able to be explicitly written down (Skalse et al., 2023)), and despite the fact that the environment is designed to be "ideal for evaluating generalization, as distinct training and test sets can be generated for each environment", with the authors boasting that "The environments' intrinsic diversity demands that agents learn robust policies; overfitting to narrow regions in state space will not suffice" (Cobbe et al., 2020). These claims are true as far as they go, yet they do not suffice for learning reward functions which generalize to large distributional shifts. This, together with the difficulty of understanding the networks with standard tools based on the feature representation paradigm, motivates the need for a different frame for interpretability.

Code for experiments conducted in this chapter is available at https://github.com/HumanCompatibleAI/reward-function-interpretability.

5.2 Related work

Reward models have begun to play a large role in AI practice. Christiano et al. (2017) introduced the method of reinforcement learning with human feedback, which has been used in state of the art language models such as ChatGPT (OpenAI, 2022), and a variant has been used in Claude, another state of the art model (Anthropic, 2023; Bai et al., 2022). Given the importance of reward models, there has been some interest in applying interpretability techniques to them (Jenner & Gleave, 2022; Michaud et al., 2020; Russell & Santos, 2019), as well as to policy networks that include a value head (Hilton et al., 2020).

This chapter also builds on a wealth of literature using linear probing as an interpretability technique, which was introduced by Alain and Bengio (2017). Linear probing has been used in a variety of contexts, but perhaps most relevantly to the AlphaZero chess engine (Silver et al., 2018) where it showed representations of features highly relevant to evaluating the state of the game like "is the playing side in check" and "can the other side checkmate the playing side in one move" (McGrath et al., 2022).

The chapter also uses the method of feature visualization, introduced by Zeiler and Fergus (2014). In particular, we use feature visualization by optimizing an input image to maximize an activation of the network, which was pioneered by Olah et al. (2017). We also use the related method of adversarial training (Mądry et al., 2018). In the context of reward learning, it has conceptually inspired the adversarial inverse reinforcement learning paradigm (Fu et al., 2018), although we use a different adversarial model). It has been also tested by Ziegler et al. (2022) on domains where the notion of "failure" requires human evaluation.

Related to our findings about networks not representing distances, R. Liu et al. (2018) have written about the difficulties convolutional neural networks have in representing coordinates of pixels.

5.3 Reward nets we trained

The primary environment that reward networks were trained on was CoinRun (Cobbe et al., 2020). Reward networks were trained by supervision, using a loss function of mean squared error (MSE), on a collection of rollouts of a policy trained with reinforcement learning. This dataset had approximately 1 million transitions: 900,173 to train on, and 100,019 to test on. 93.6% of trajectories ended with a non-zero reward, and there were 9,925 trajectories in the whole dataset, meaning that 0.93% of transitions were associated with a non-zero reward. This means that the MSE incurred by always guessing the mean reward on the dataset is 0.92.

The network architecture was deliberately overparameterized, to maximize the network's ability to represent relevant quantities. There were 5 hidden convolutional layers with 96, 256, 384, 384, and 256 channels respectively. This was chosen to mirror the convolutional portion of the AlexNet architecture (Krizhevsky et al., 2012). All filters had a 3 by 3 receptive field and used stride 1, with enough padding to maintain the height and width dimensions throughout the network—no max-pooling or dropout was used. After the final convolutional layer, the channels were averaged over their spatial dimension, and then linearly mapped to the output. Networks were trained with the Adam optimizer (Kingma & Ba, 2014) with learning rate 0.001 and batch size 128. Their input was the observation and next observation, and they output a vector of estimated rewards, one per action, where the reward of the action actually taken in a transition was regressed on.

Two main networks were trained on the CoinRun data: one for 15 epochs, achieving a test loss of 0.046, and one for 40 epochs, achieving a test loss of 0.038. It should be noted that training runs using these hyperparameters seemed to face a significant amount of variance in how much the loss went down, so several were required to get these two trained networks.

In addition, another reward network of the same architecture was trained on Heist: a game where the agent has to solve a maze by collecting keys and using them to unlock doors so that it can reach a diamond. An observation from Heist is shown in Figure 5.2. Similarly to CoinRun, transitions where the agent reaches the diamond have reward 10 (and cause the game to end), and all other transitions have reward 0. The dataset was generated by rolling out a reinforcement learning policy for 3,007 episodes, 20.3% of which ended in a non-zero reward, comprising 2,622,147 transitions in total. Unlike the CoinRun networks, this dataset was then pruned by discarding ∼95% of zero-reward transitions, so that after pruning, 0.46% of transitions had non-zero reward. This resulted in a final dataset of 118,598 transitions in the training set, and 13,177 in the test set. On this dataset, always guessing the mean reward would incur a MSE loss of 0.4.

Like in CoinRun, Adam was used with a batch size of 128 and a learning rate of 0.001. A network was trained for 1,119 epochs which achieved a test loss of 0.012. Note that this proved to be more epochs than necessary: essentially the same loss was achieved after approximately 165 epochs.

5.4 Visualization

Our initial plan to understand these networks was to visualize their output neurons, in the manner of Olah et al. (2017), in order to better grasp what they 'thought' the cause of reward was. We describe in detail the results of visualizing the 15-epoch network, but other networks yielded similar results[1].

The first method was synthesizing inputs de novo by optimization in Fourier space. Namely, inputs are initialized with each Fourier mode drawn from a normal distribution with mean 0 and standard deviation 1. These Fourier components are then optimized to maximize the reward assigned to the input by the network. Specifically, this is done using the Adam algorithm (Kingma & Ba, 2014) for 512 iterations with learning rate 0.05. Before each forward pass the input is randomly translated by up to 8 pixels vertically and horizontally, as an attempt to ensure that the final image is not "overfit". Each action would have an input synthesized to maximize the reward association with that action.

This process produced images with predicted rewards of between 200 and 1,200: the average predicted reward of an image was 700, and the standard deviation was 240. Images would tend to look like Figure 5.3: the observation and next observation would have "in-focus" regions with roughly the same shape, and pink turning to cyan in those regions. The remainder of the images are shown in Figures C.1 and C.2.

One might think that the network was merely picking up on pink-to-cyan transitions, but monochrome pink observations followed by monochrome cyan observations did not produce high predicted reward. Similarly, given the random-looking nature of the images, one might think that the network is merely sensitive to the statistics of pixel values. To test this, we randomly scrambled the pixels of these optimized images, and fed the scrambled images into the networks. These scrambled images typically had predicted rewards between 2 and 18: significantly lower than their original images, but still competitive with the highest-reward images in the training dataset.

The second visualization method used was to start with dataset examples the network assigned predicted reward $\sim$10 to, and to then optimize those images in pixel space for predicted reward. This would hopefully indicate what aspect of those dataset examples the network 'thought' was valuable. A similar method was used as for the de novo synthesis: random translations of up to 8 pixels were applied, and the optimization method was the

[1] Also, all visualizations depicted are of action 0, which is simultaneously pressing the 'down' and 'left' arrow keys.

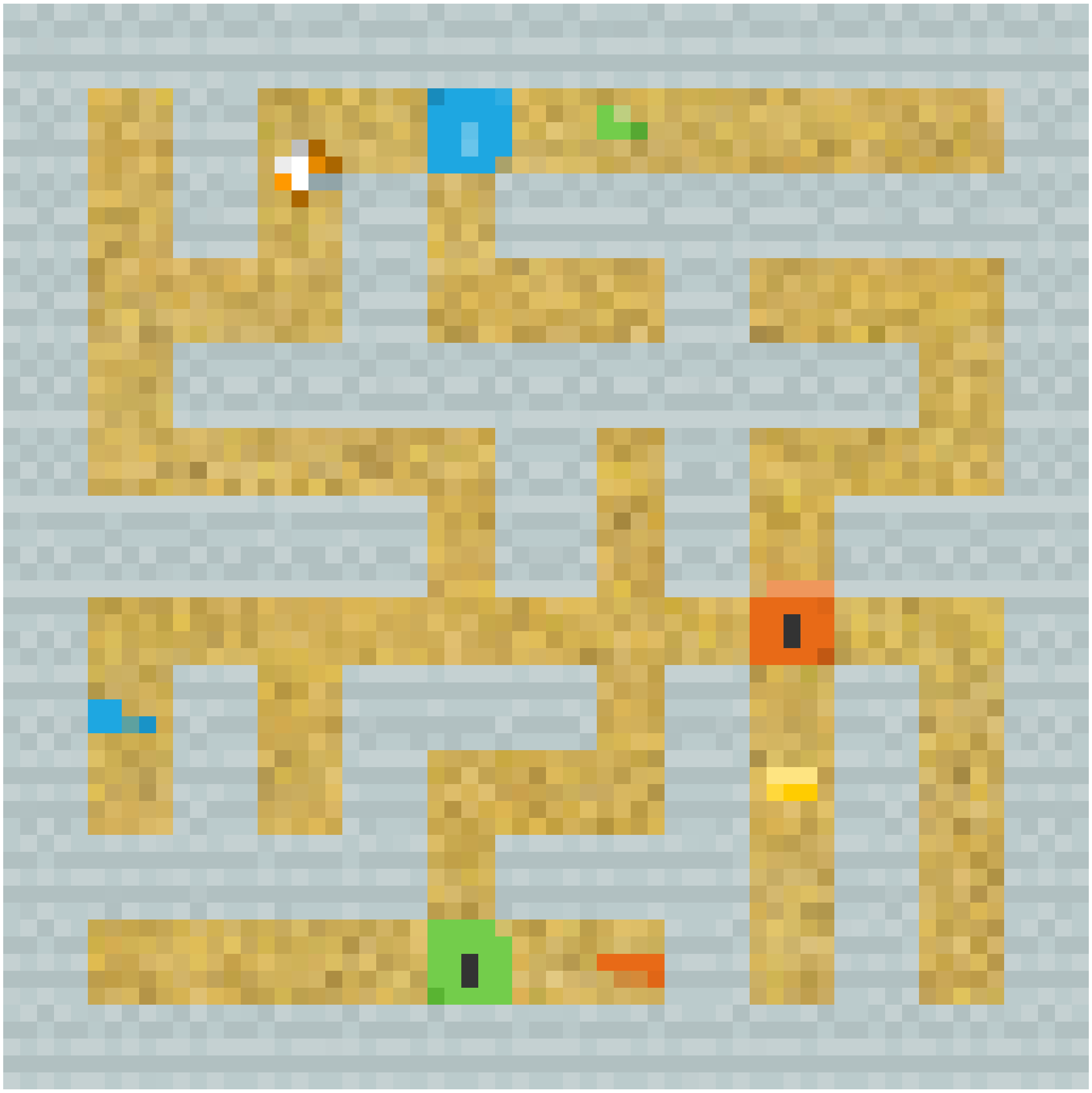
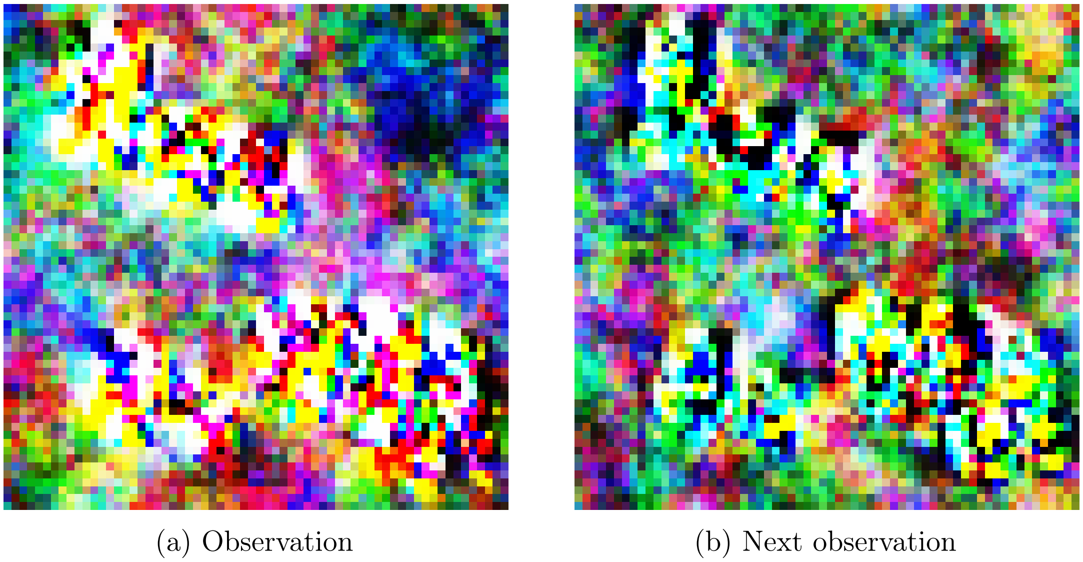


Figure 5.2: An observation from Heist. The agent (whose sprite is the white, orange, and grey pixellated blob near the top) must gather keys of various colours to fit in locks of matching colours to make it to the diamond (the yellow pixels below the red lock).



(a) Observation

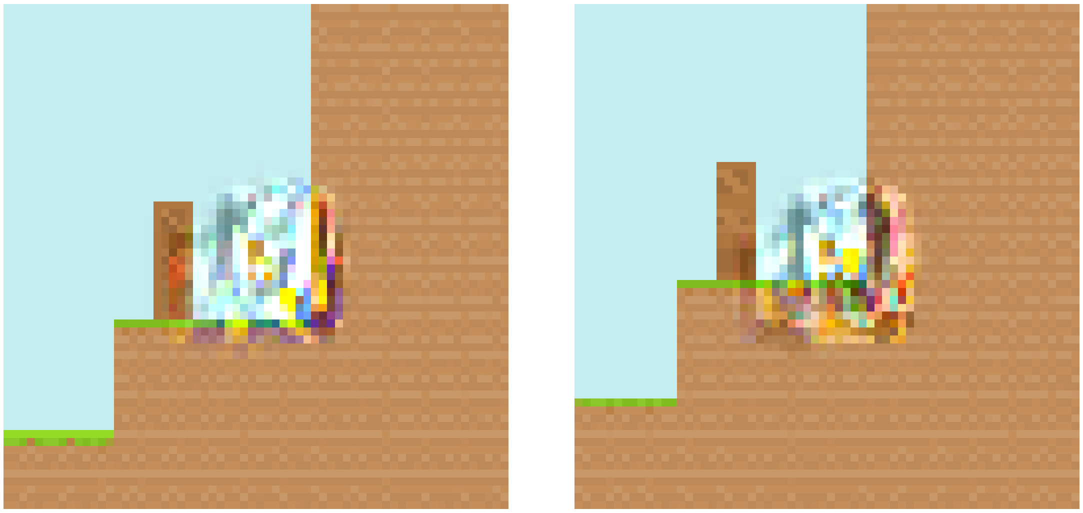
(b) Next observation

Figure 5.3: Optimized input for the 15-epoch reward model for CoinRun. This transition had predicted reward 724. Note that the presence of high-contrast shapes at the same location in the observation and next observation that appear to be in focus, and a colour transition from yellow and pink to cyan.

Adam optimizer with learning rate 0.05. The result was that as the number of gradient steps grows, a pixellated mass appears around the agent sprite and the coin, and grows to take up a larger and larger proportion of the frame (see Figures 5.4, 5.5, 5.6, 5.7, and 5.8). This process eventually yields a predicted reward of over 1,000, caused by images that almost look like random images, although they retain the move from magenta pixels to cyan pixels characteristic of results of the de novo method.

Because observations in this game are centred around the agent, these results were ambiguous between the agent sprite and coin playing a special role and the centre of the image playing that special role. To disambiguate, we re-ran the process starting with synthetic images where the agent sprite and the coin were translated to the right of the frame, which the network also assigned predicted reward $\sim$10 to (see Section 5.6 for more details on these images). On these images, we get a growing blob centred around the agent sprite and the coin, indicating that they do play a special role in the picture, rather than it just being a matter of the centre of the image (see Figures 5.9, 5.10, 5.11, 5.12, 5.13, and 5.14).

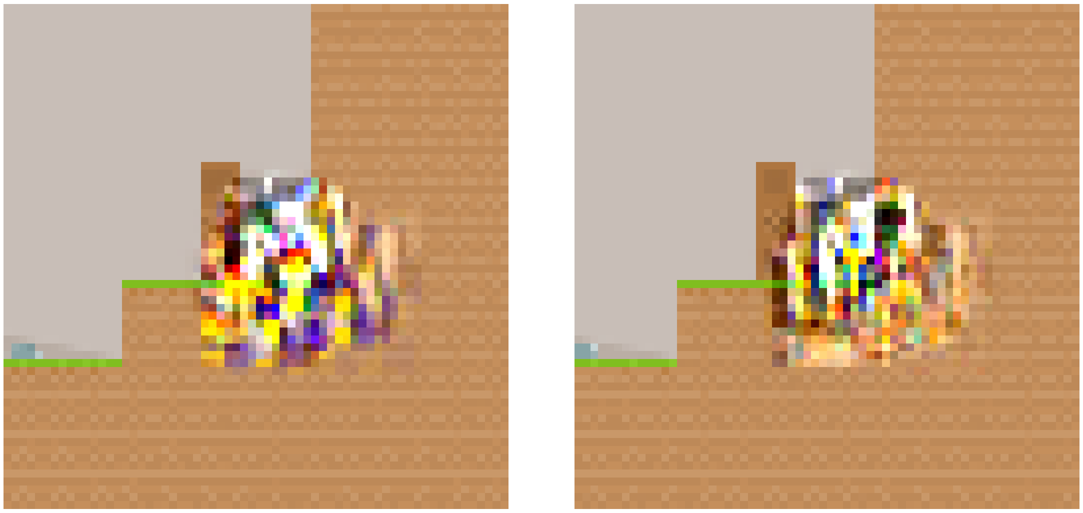
Broadly, these visualization attempts reveal that the network is not relying on an incredibly simple pixel-level proxy, that it 'likes' to see the same objects in the same place between frames, and that it cares about the region around the sprite and the coin, but a significant amount of mystery remains. Perhaps the clearest lesson is that these networks are not exactly learning the desired reward function, but have inputs on which they predict a much too high reward.



(a) Observation

(b) Next observation

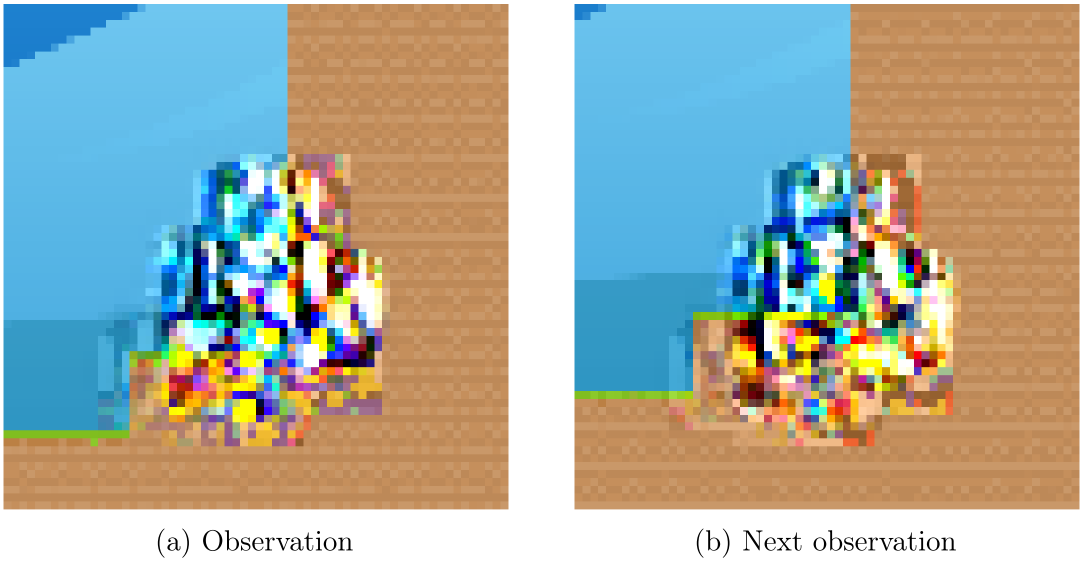
Figure 5.4: Observation of a high-scoring transition after 8 steps of gradient descent, with predicted reward 50. Observe that the area around where the agent sprite and coin would be have become fuzzy.



(a) Observation

(b) Next observation

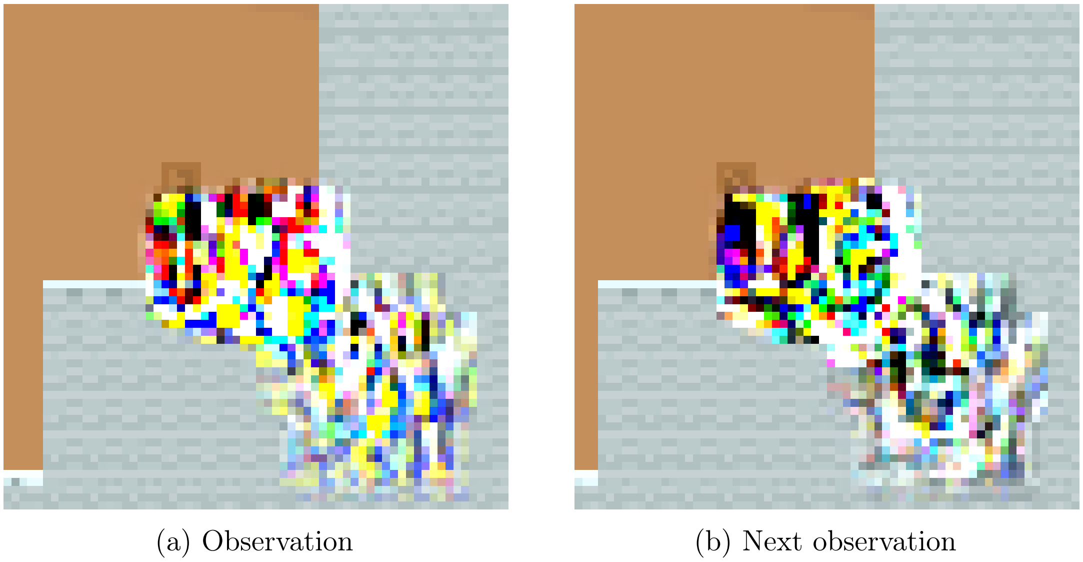
Figure 5.5: Observation of a high-scoring transition after 16 steps of gradient descent, with predicted reward 96. The perturbed area around the agent sprite and coin has changed more in colour and grown larger.



(a) Observation

(b) Next observation

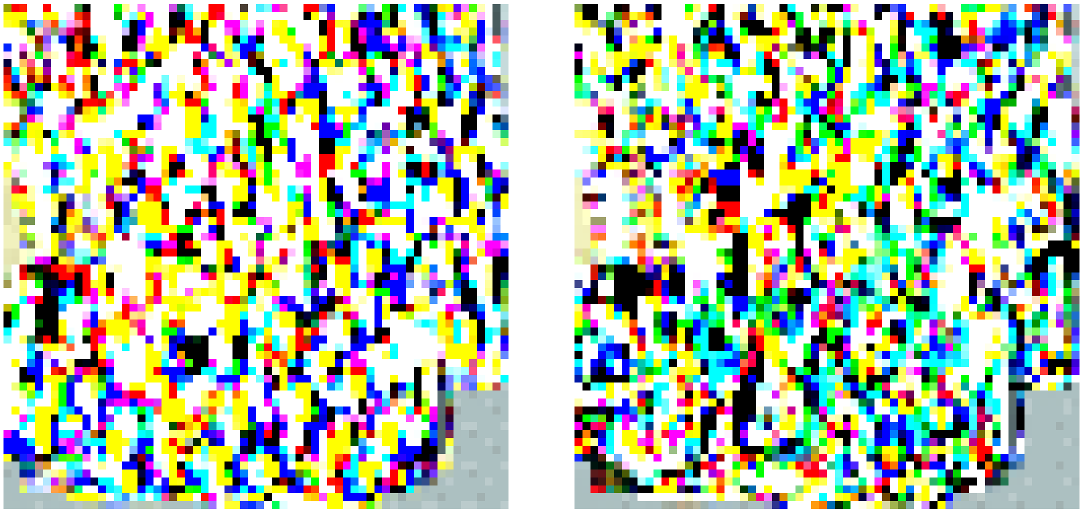
Figure 5.6: Observation of a high-scoring transition after 32 steps of gradient descent, with predicted reward 158. The perturbed area has become even larger, and is no longer as circular.



(a) Observation

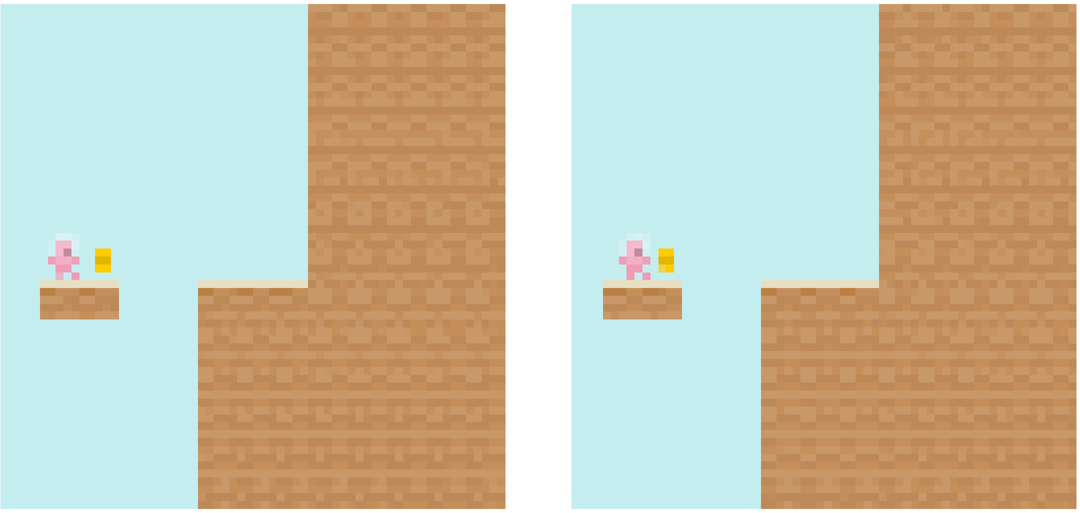
(b) Next observation

Figure 5.7: Observation of a high-scoring transition after 64 steps of gradient descent, with predicted reward 229. The perturbed area has grown even larger, more brightly coloured, and less circular.



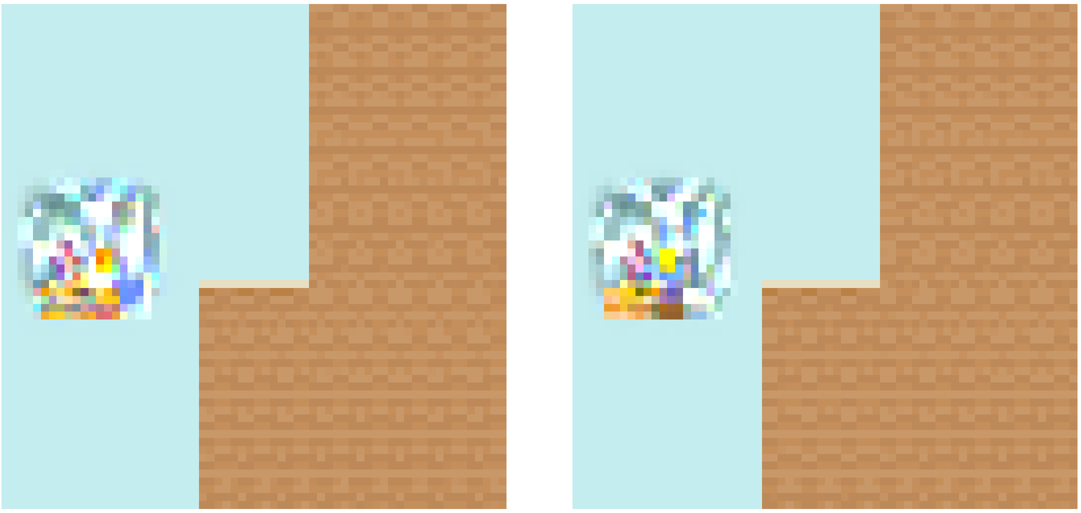
(a) Observation (b) Next observation

Figure 5.8: Observation of a high-scoring transition after 128 steps of gradient descent, with predicted reward 1,401. Essentially the whole frames are taken up by random-looking pixels, although note that the observation has more yellow and pink where the next observation has more cyan, evoking Figure 5.3.



(a) Observation (b) Next observation

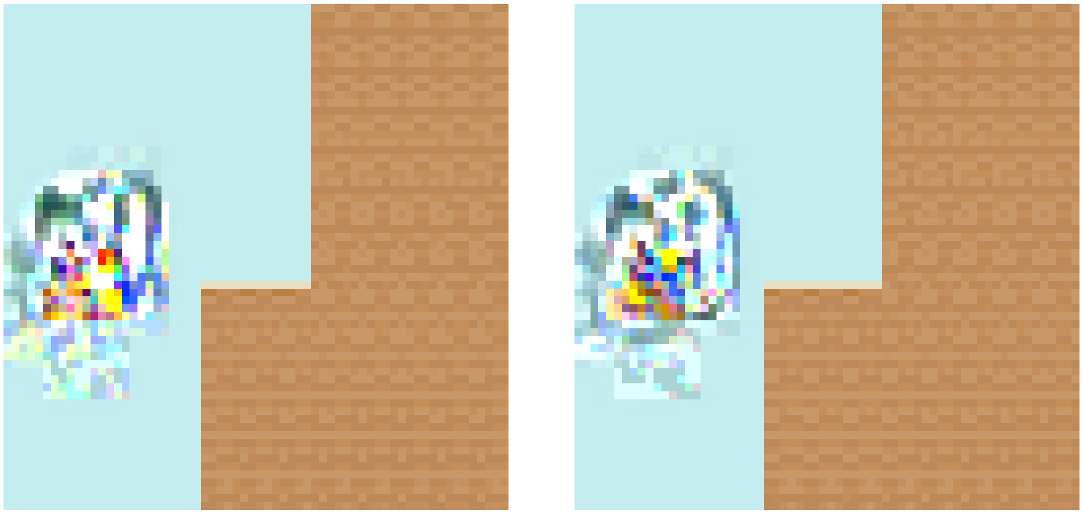
Figure 5.9: Observation of a synthetic transition with the sprite and coin translated to the left, with predicted reward 9.8.



(a) Observation

(b) Next observation

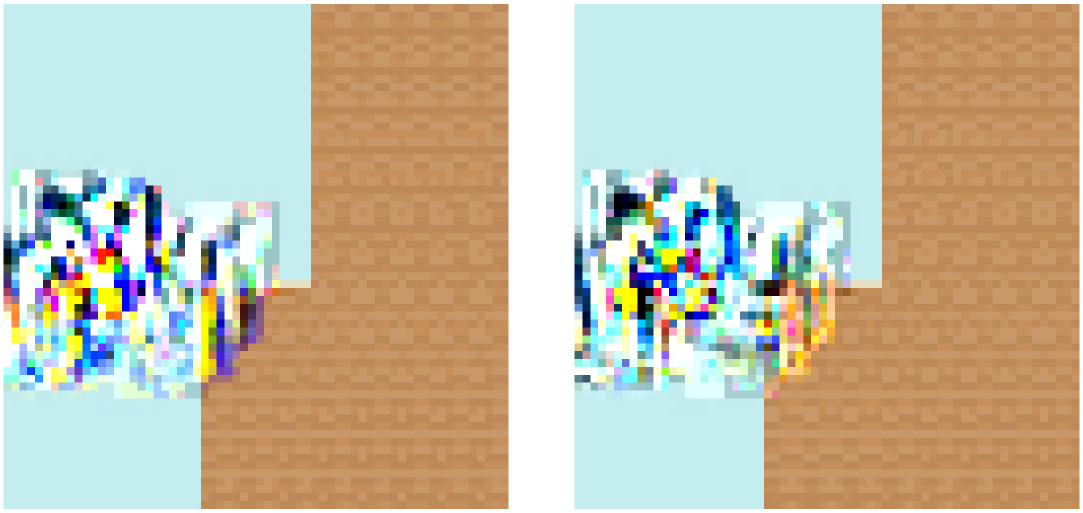
Figure 5.10: Observation of a synthetic transition with the sprite and coin translated to the left after 8 steps of gradient descent, with predicted reward 25. A fuzzy ball has appeared around the sprite and coin.



(a) Observation

(b) Next observation

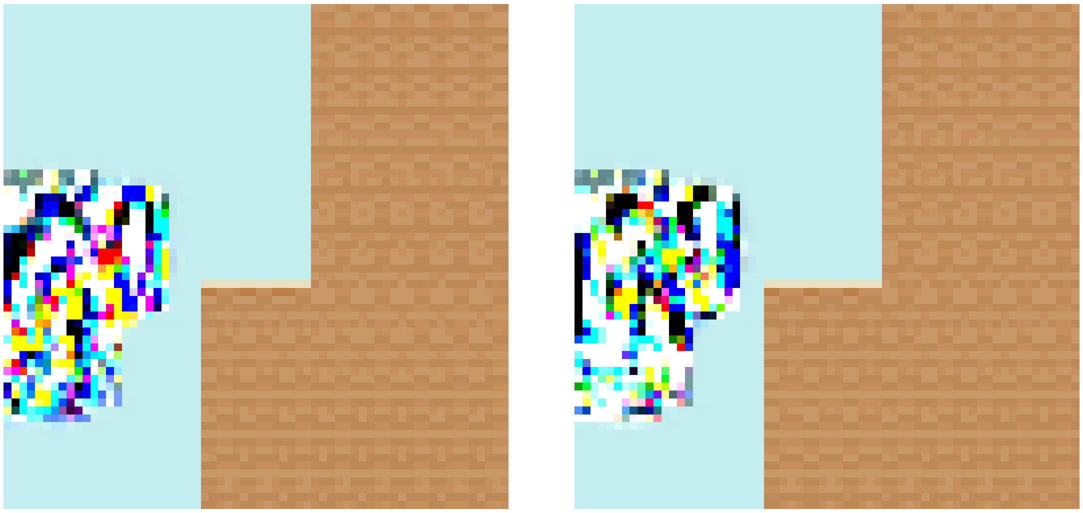
Figure 5.11: Observation of a synthetic transition with the sprite and coin translated to the left after 16 steps of gradient descent, with predicted reward 46. The fuzzy ball has grown.



(a) Observation

(b) Next observation

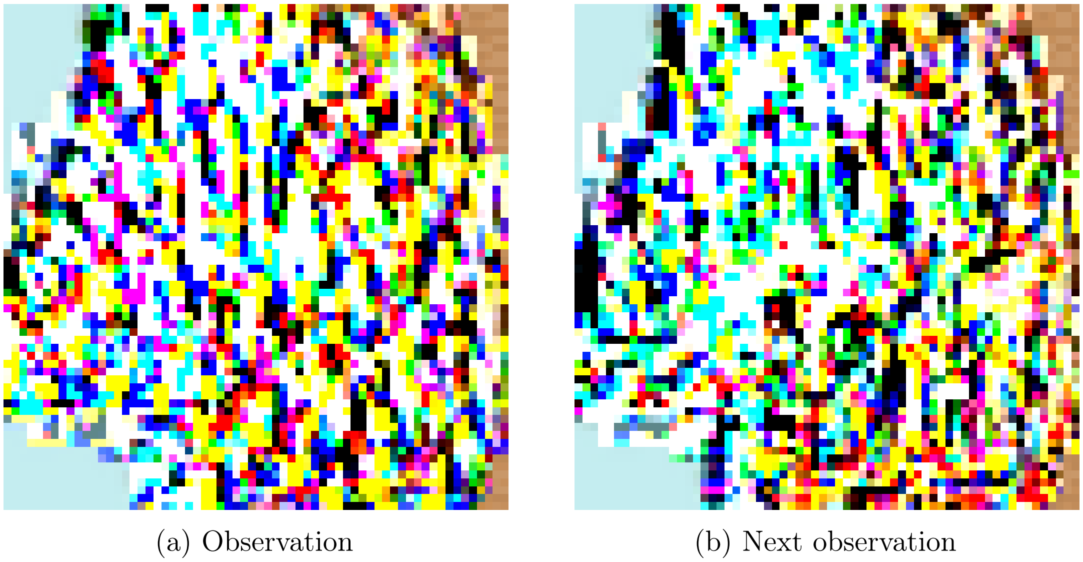
Figure 5.12: Observation of a synthetic transition with the sprite and coin translated to the left after 32 steps of gradient descent, with predicted reward 125. The fuzzy ball has grown further, and is now no longer circular.



(a) Observation

(b) Next observation

Figure 5.13: Observation of a synthetic transition with the sprite and coin translated to the left after 64 steps of gradient descent, with predicted reward 130. The perturbed area is now more brightly coloured.



(a) Observation

(b) Next observation

Figure 5.14: Observation of a synthetic transition with the sprite and coin translated to the left after 128 steps of gradient descent, with predicted reward 1,217. Both frames have nearly entirely been taken up by random-looking pixels, although note that the observation has more yellow and pink pixels, while the next observation has more cyan pixels, evoking Figure 5.3.

5.5 Random noise

Inspired by the 'visualizations' produced in Section 5.4 looking like random pixels, we wanted to assess how our networks would assign reward to observations composed of random pixels. Specifically, we generated images by drawing each RGB value of each pixel independently from a normal distribution of mean 0.5 and variance 1, clipped to have values between 0 and 1. See Figure 5.15 for what such images looks like. There were two variants of this experiment: firstly, we used different random images for the observation and the next observation; and secondly, we used the same random image for both. This allowed us to determine to what extent any effect found was due to a large difference between the observation and next observation, rather than just that the observation and next observation looked random.

Our broad observation is that reward networks reliably predicted positive reward for random noise observations—in fact, often this reward was competitive with the maximum reward available in the game. Results are shown in Table 5.1.

5.6 Manual adversarial examples

In order to further probe the CoinRun reward networks and test hypotheses of how they could be operating, we manually produced adversarial examples of two different types. Each kind was instantiated by 16 different examples, all modifications of the same set of 16 original

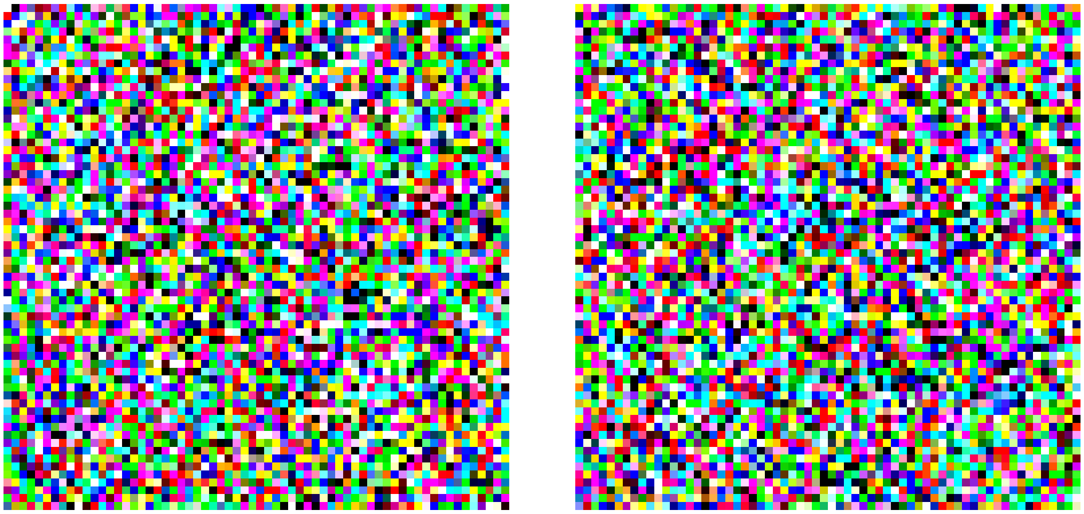


Figure 5.15: Two examples of images with IID normal pixels.

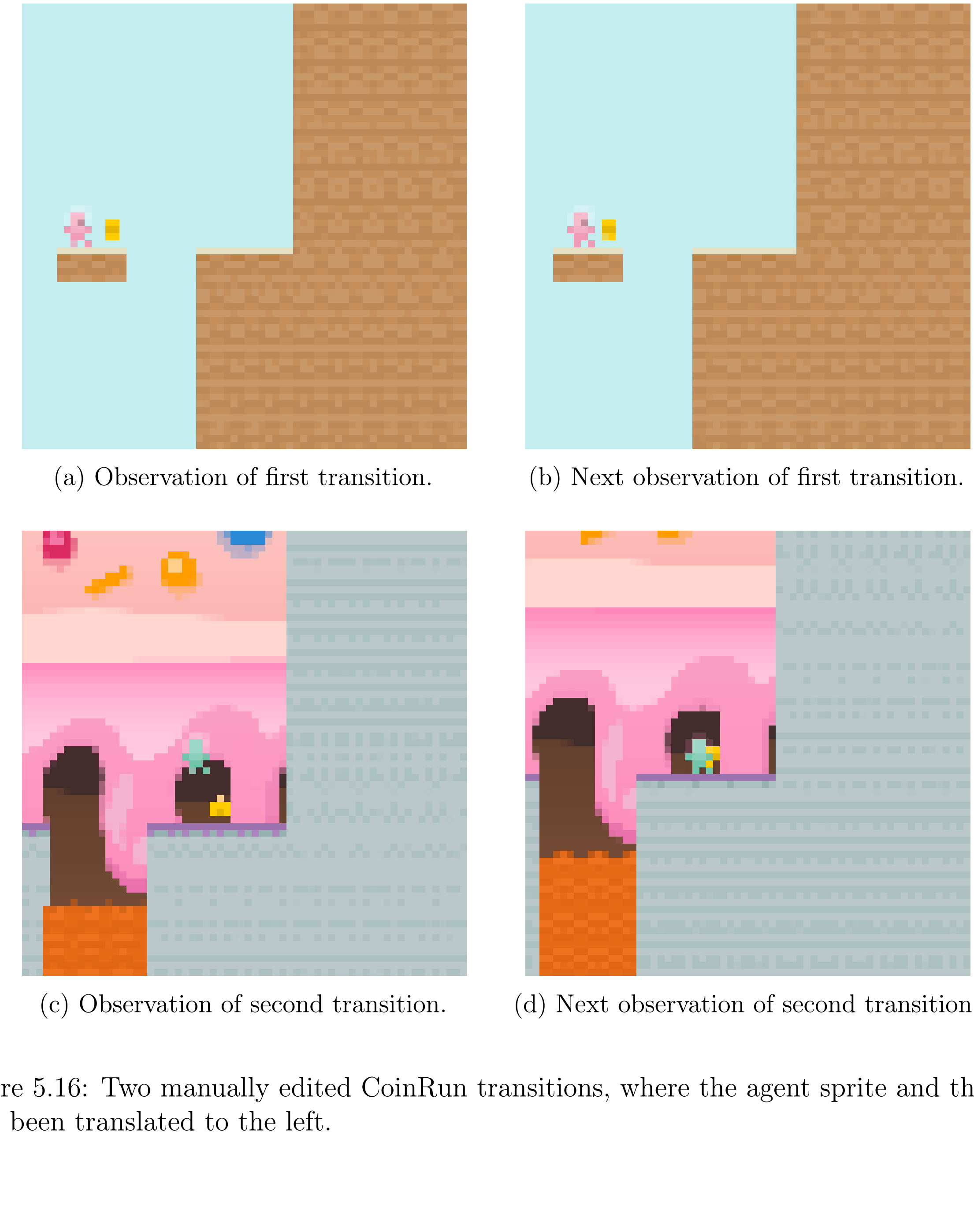
	CoinRun, 15 epochs	CoinRun, 40 epochs	Heist
Different	7.3 ± 6.3	40.1 ± 15.5	85.2 ± 25.2
Same	9.9 ± 6.2	70.5 ± 19.4	3.0 ± 3.7

Table 5.1: Rewards assigned to random noise inputs by different networks, expressed as mean ± standard deviation. Different means that the observation and next observation are different random images, while same means that they are the same. Evaluated on a dataset of 100 random images. The reward predicted for each action on a single input is included separately in the calculation of means and standard deviations.

transitions where the agent in fact reached the coin, and the reward networks predicted rewards of $\sim$10. The original examples, as well as each manual adversarial example, are shown in appendix C.2. That section also includes edits to images that were made in an attempt to fool the reward networks, but did not appear to succeed in doing so. Image editing was done using the GIMP program (The GIMP Development Team, 2023).

The first type involved translating the coin and the sprite to the left (Figure 5.16). This was designed to test whether the reward networks are sensitive either to the location of the sprite and coin relative to the wall, or to their location in the image.

The second type involved placing the coin above the agent sprite, and having the agent sprite jump up to meet the coin (Figure 5.17). This was designed to test whether the reward network was overly sensitive to the arrangement of the coin and the agent sprite, as perhaps suggested by the region of the coin and the sprite seeming to be crucial in Section 5.4. Note that it could also suggest sensitivity to the coin's proximity to the ground, or unnatural



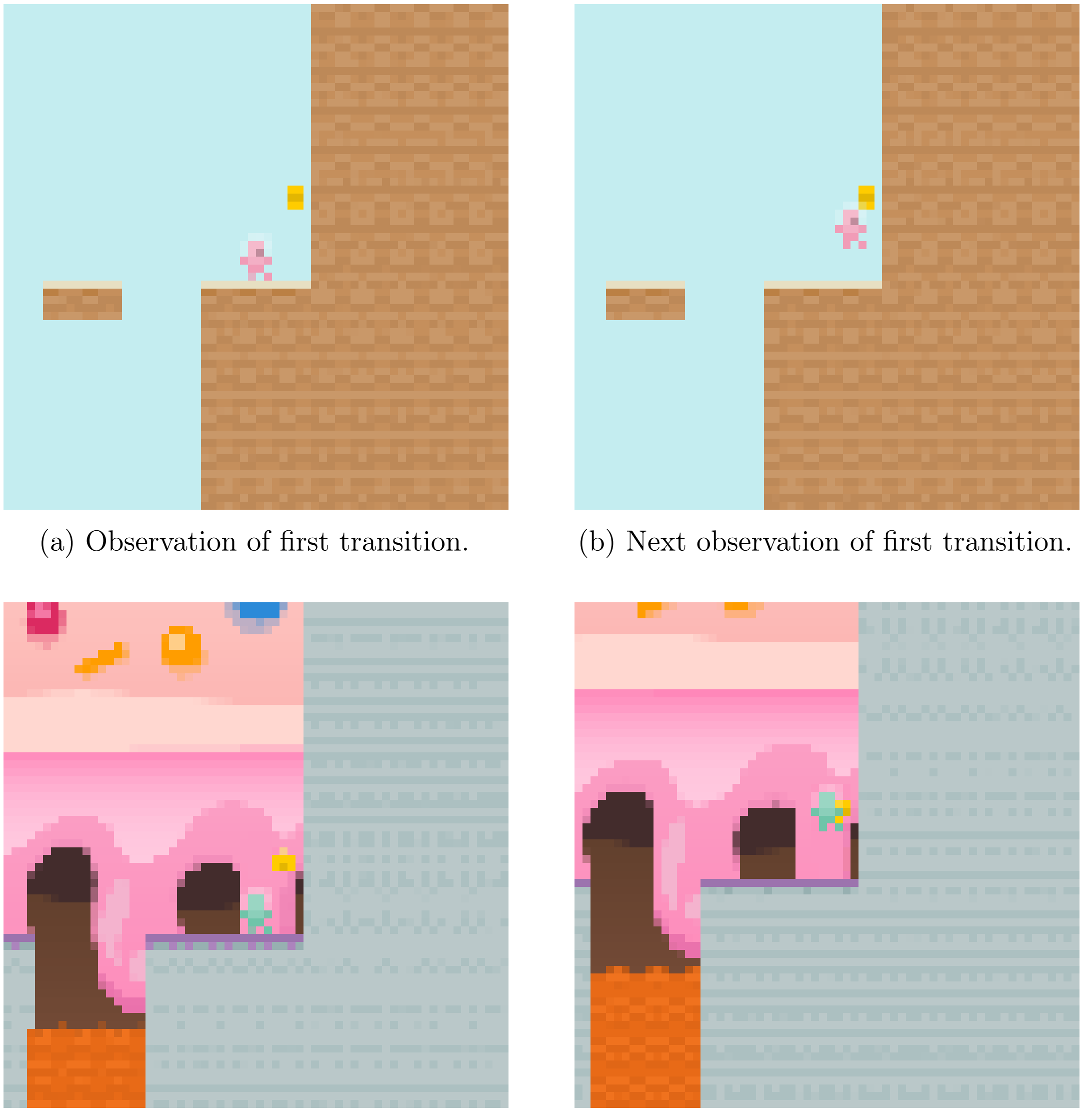
(a) Observation of first transition.

(b) Next observation of first transition.

(c) Observation of second transition.

(d) Next observation of second transition.

Figure 5.16: Two manually edited CoinRun transitions, where the agent sprite and the coin have been translated to the left.



(a) Observation of first transition.

(b) Next observation of first transition.

(c) Observation of second transition.

(d) Next observation of second transition.

Figure 5.17: Two manually edited CoinRun transitions, where the agent sprite and coin have been moved so that the agent is jumping up to the coin.

vertical position of the agent sprite in the image (since the image is still centred on the location of the agent sprite prior to editing of the image).

For both types of manual adversarial example, we consider the "true" reward to be 10, because the sprite in fact reaches the coin.

The rewards predicted by the various CoinRun networks for the various manual adversarial examples are shown in Table 5.2. Overall, translations somewhat fooled the networks, but rearranging the coin and the sprite caused them to almost uniformly assess the transitions as

	CoinRun, 15 epochs	CoinRun, 40 epochs
Original	9.7 ± 1.3	9.7 ± 0.6
Translated	6.0 ± 3.3	7.6 ± 2.6
Rearranged	0.15 ± 0.51	0.3 ± 1.1

Table 5.2: Rewards assigned to manual adversarial examples by different networks, expressed as mean ± standard deviation. Original refers to the original examples that were not manually edited, Translated refers to examples where the agent sprite and coin have been translated to the left, and Rearranged refers to examples where the coin has been placed above the agent sprite.

having zero reward. This provides a further demonstration of transitions that these reward networks misassess, but this time a more naturalistic example. It also demonstrates that these networks cannot be representing and properly using the high-level concept of distance from the agent sprite to the coin, since that would be sufficient to correctly predict the reward of these transitions.

5.7 Probes

In Section 5.6, we found failures that suggested that the reward networks were either not representing the distance from the agent sprite to the coin, or not using that representation correctly. In this section, we tested whether the network was linearly representing this distance (or some simple function of it) by training linear probes. This was applied not only to the networks trained on CoinRun, but also the one trained on Heist, where the probes were instead trained on the distance from the agent sprite to the diamond.

5.7.1 Methods

For all networks, we probed for the distance between the agent sprite and the goal[2], the squared distance, 1 divided by the distance, and 1 divided by the squared distance. For the Heist network, we also train probes on the x- and y-components of the distance from the agent sprite to the goal.

In CoinRun, the coin is often not visible on the screen, meaning that it would be uninformative to probe for the actual distance from the agent sprite to the coin (and functions thereof). Instead, we took the relevant function of the distance and capped it at the maximum value it could possibly assume while having the coin still be visible on the screen. For the distance, this cap was 18.4, and for the squared distance, it was 338.0. The inverse distance and inverse square distances were capped at 5.0 in order to avoid overflow problems, although

[2] That being the coin in the case of CoinRun and the diamond in the case of Heist.

the game functioning properly should never have allowed values of those functions to be so high. Probing with these caps should work as long as the network represents the relevant function of observed distance gated by whether the coin is visible, as well as a feature indicating whether the coin was visible or not.

Indexing the layers such that the input is -1 and the result of the first convolution is layer 0, we probed layers 1, 2, 3, and 4, always after the application of the activation function[3]. The probe head involved average-pooling the layers, and then applying a linear function to predict the output. Probes were always trained using the Adam optimizer with default hyperparameters. Both for CoinRun and for Heist, they were trained on 3,007 episode rollouts. For CoinRun, this was 895,489 transitions, and for Heist, it was 2,622,272 transitions. For both CoinRun and Heist, 10% of these transitions held out as a test set. Probes were trained for different numbers of epochs, usually between 4 and 10, depending on how many seemed necessary to reach convergence.

The mean squared error of the probes was compared against two baselines: always guessing the mean value of the probed variable, and a probe trained on a randomly-initialized neural network. This helped determine whether the trained network was representing the probed variables any more than a random network could be said to represent them—which is not necessarily a trivial baseline, since random features can be useful for regression (Rahimi & Recht, 2007). Since the loss of probes trained on random networks to predict certain features (namely, everything but the squared distance) did not vary much by layer, for some experiments a probe was only trained on the final layer of a random network for those features, in order to speed them up.

On CoinRun networks, we also trained 'filtered probes' for the distance and squared distance on the subset of data where the coin was visible—that is, on 39,492 transitions where the distance between the agent sprite and the coin was less than 6. This answered a somewhat different question: do networks linearly represent (a simple function of) the distance from the agent sprite to the coin in a way that only has to work well when the coin is visible?

5.7.2 Results

Results of training probes on the 15-epoch CoinRun network are shown in Table 5.3, and of training probes on the 40-epoch CoinRun network are shown in Table 5.4. Results of training probes on the Heist network are shown in Table 5.5.

On CoinRun nets, we seem to consistently see some sort of representation of the agent-sprite-to-coin distance in layer 2, and on the Heist network we see some sort of representation of the squared agent-sprite-to-diamond in layers 1, 2, and 3, but these representations do not appear to be especially precise: test loss is in no instance less than half of the lowest baseline loss.

[3] Note that layer 4 is the final hidden layer of each network.

	Distance	Squared distance	Inverse distance	Inverse squared distance
Mean	17.2	1.12×10^4	0.0047	0.0027
Layer 1	15.4 (17.0)	1.13×10^4 (1.3×10^4)	0.0040 (0.0042)	0.0026 (0.0026)
Layer 2	11.4 (16.9)	9.4×10^3 (1.2×10^4)	0.0033 (0.0042)	0.0026 (0.0026)
Layer 3	17.7 (17.2)	9.9×10^3 (1.2×10^4)	0.0043 (0.0042)	0.035 (0.0024)
Layer 4	17.1 (17.2)	1.2×10^4 (1.1×10^4)	0.0042 (0.0046)	0.0022 (0.0027)

Table 5.3: Test losses of training probes on the 15-epoch CoinRun network. Mean refers to the loss obtained from always guessing the mean value of the attribute in question. All probes are trained with batch size 128 for 5 epochs on a dataset of 805,940 transitions, except for layers 2 and 3 for the distance and squared distance, which were trained for 10 epochs, layer 3 for the inverse distance, which was trained for 20 epochs, and layer 4 for the squared distance, which was trained for 40 epochs (which still did not seem to have converged). Losses for probes trained identically on randomly initialized networks in parentheses.

	Distance	Squared distance	Inverse distance	Inverse squared distance
Mean	17.2	1.12×10^4	0.0047	0.0027
Layer 1	14.7	1.1×10^4 (1.2×10^4)	0.0042	0.0035
Layer 2	11.6	8.4×10^3 (1.2×10^4)	0.0071	0.0041
Layer 3	16.8	1.1×10^4 (1.2×10^4)	0.0051	0.043
Layer 4	17.1 (17.1)	5.4×10^4 (2.4×10^4)	0.0043 (0.0046)	0.0023 (0.0029)

Table 5.4: Test losses of training probes on the 40-epoch CoinRun network. Mean refers to the loss obtained from always guessing the mean value of the attribute in question. All probes are trained with batch size 128 for 5 epochs on a dataset of 805,940 transitions, except for squared distance probes which were trained for 10 epochs. Losses for probes trained identically on randomly initialized networks in parentheses—for the sake of conserving compute, this was only evaluated on layer 4 except for experiments with the squared distance, where it seemed to significantly vary by layer.

	x-distance	y-distance	Distance	Squared distance	Inverse distance	Inverse squared distance
Mean	25.4	23.9	8.5	1.9×10^3	0.013	0.0059
Layer 1	24.8 (25.4)	23.5 (23.9)	5.4 (5.7)	1.3×10^3 (1.6×10^3)	0.0091	0.0047
Layer 2	23.2 (25.3)	22.7 (23.9)	5.3	1.3×10^3 (1.5×10^3)	0.0098	0.0057
Layer 3	23.3 (25.4)	22.0 (23.8)	5.4	1.2×10^3 (1.7×10^3)	0.0097	0.0054
Layer 4	25.3 (25.3)	23.9 (23.8)	8.4 (6.3)	1.9×10^3 (1.8×10^3)	0.012 (0.0097)	0.0053 (0.0052)

Table 5.5: Test losses of training probes on the Heist network. Mean refers to the loss obtained from always guessing the mean value of the attribute in question. All probes are trained with batch size 128 for 5 epochs on a dataset of 2,360,045 transitions, except for squared distance probes which were trained for 10 epochs. Losses for probes trained identically on randomly initialized networks in parentheses—for the sake of conserving compute, for many variables where the accuracy of the probes did not seem to depend on the layer they were probing, probes were not trained on every layer.

	Distance	Squared distance
Mean	1.83	102
Layer 1	1.64 (1.85)	94.4 (104)
Layer 2	0.86 (1.85)	60.8 (106)
Layer 3	1.63 (1.84)	66.3 (111)
Layer 4	1.82 (1.90)	236 (112)

Table 5.6: Test losses of training probes on filtered data on the 15-epoch CoinRun network. Mean refers to the loss obtained from always guessing the mean value of the attribute in question. All probes are trained with batch size 128. Probes were trained for 20 epochs on the distance for layers 1 and 2, 40 epochs on the distance for layers 3 and 4, 50 epochs on the squared distance for layer 1, and 30 epochs on the squared distance for layers 2, 3, and 4. Losses for probes trained identically on randomly initialized networks in parentheses.

	Distance	Squared distance
Mean	1.83	102
Layer 1	1.42 (1.83)	82.8 (103)
Layer 2	0.88 (1.82)	53.0 (110)
Layer 3	1.37 (1.88)	73.6 (108)
Layer 4	1.83 (1.86)	240 (115)

Table 5.7: Test losses of training probes on filtered data on the 40-epoch CoinRun network. Mean refers to the loss obtained from always guessing the mean value of the attribute in question. All probes are trained with batch size 128. Probes were trained for 20 epochs on the distance for layers 1 and 2, 50 epochs on the distance for layer 3, 30 epochs on the distance for layer 4, 50 epochs on the squared distance for layer 1, and 30 epochs on the squared distance for layers 2, 3, and 4. Losses for probes trained identically on randomly initialized networks in parentheses.

Results of training filtered probes on the 15-epoch CoinRun network are shown in Table 5.6, and of training filtered probes on the 40-epoch CoinRun network are shown in Table 5.7. We see some improvement on the unfiltered probes results, but not much, suggesting that the primary issue is that networks lack a high-quality linear representation of the distance from the agent sprite to the coin even on the subset of data where both are plainly visible.

5.8 Adversarial training

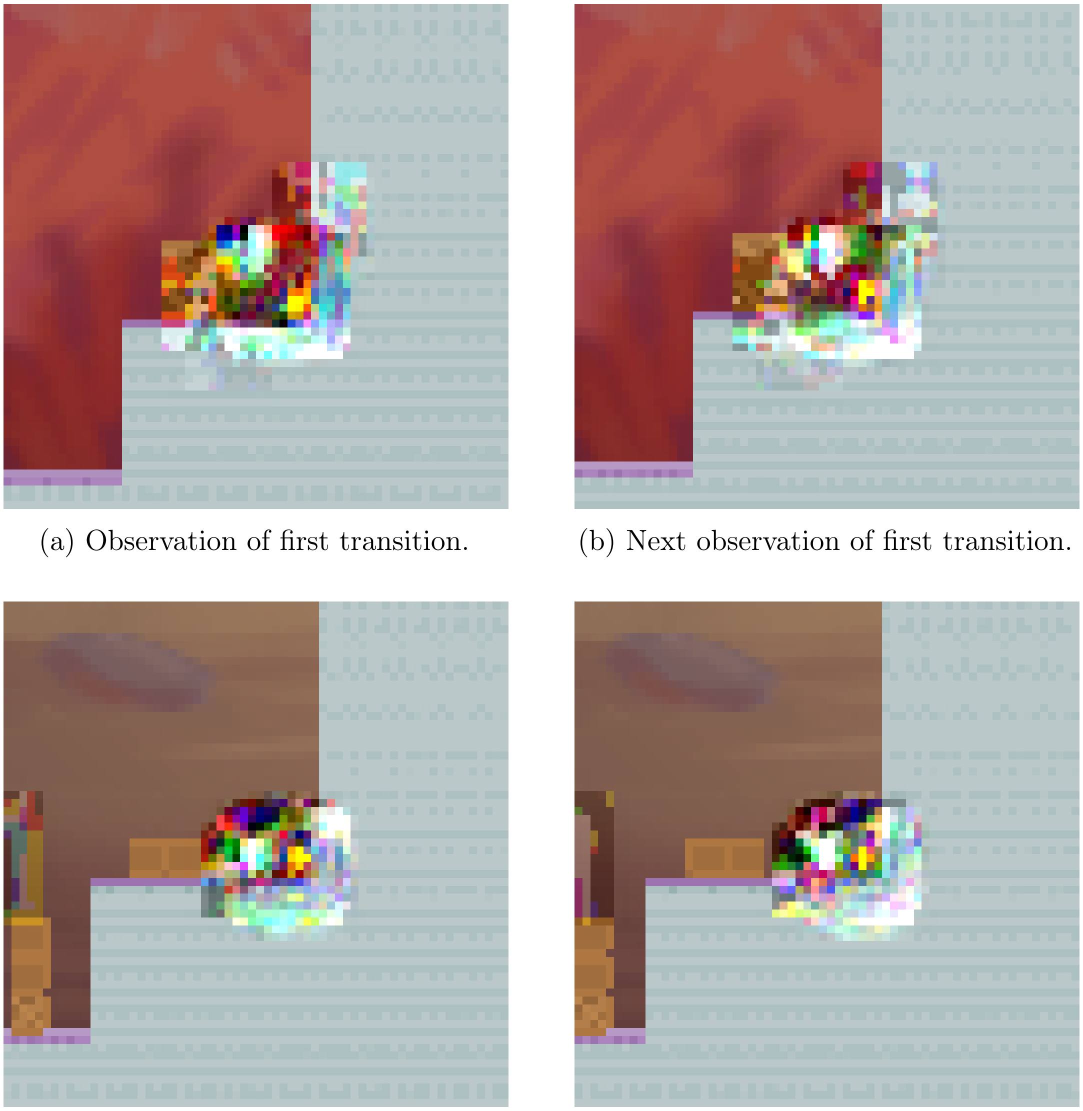
Given the problems with the representations of the CoinRun and Heist reward networks shown in Sections 5.5, 5.6, and 5.7, we thought it would be valuable to determine whether or not adversarial training could fix those problems. Broadly, the method used was to repeatedly 'visualize' the network by synthesizing inputs with high predicted reward, and then train the network on those 'visualizations' with label 0, on the assumption that the 'visualizations' depicted situations where an agent sprite was not reaching the coin, and were in fact adversarial examples.

We found it difficult to come up with a training scheme that consistently successfully trained the network to a low loss. One hypothesis for why this would be is that when training on MSE loss, adversarial examples contribute disproportionately to the batch gradient due to the fact that the network is making a disproportionately large error on those examples. As a result, under this hypothesis, the gradient over a whole batch would mostly point towards the network to 'forget what it knows'. This would suggest using something like the Huber loss instead (Huber, 1964), but preliminary attempts to adversarially train networks on that loss did not succeed.

One scheme did have a successful run. This run involved training for 10 epochs in total, using batch size 256, adding adversarial examples in every epoch starting in epoch 5, and picking the number of adversarial examples added per epoch to be 0.1% of the size of the training dataset. After adding adversarial examples, batch gradients were clipped to have norm at most the 99$^{\text{th}}$ percentile of norms of gradients in the epoch before adding adversarial examples. In this run, the "adversarial examples" used were the highest-reward dataset examples, followed by 16 gradient steps of optimization for predicted reward. This run achieved a train loss of 0.1025 and a test loss of 0.1091. For the first adversarial examples added, its loss is 0.0035, and for the final adversarial examples added and trained on, its loss is 1.55. However, the trained network is still susceptible to adversarial examples. Mean squared error for examples generated at the end of the training run was 436.4, suggesting that a typical predicted reward for these examples was ∼21—lower than the predicted reward of adversarial examples constructed in the same way for the 15-epoch CoinRun network (∼100), indicating that the training was somewhat efficacious. Furthermore, some of these final "adversarial examples" feature something like an agent sprite landing on a coin, as shown in Figure 5.18. Examples of adversarial examples produced throughout the training process are shown in Figures C.4, C.5, C.6, C.7, and C.8.

The network almost always assigned ∼0 predicted reward to random noise, suggesting that adversarial training initially hardened the network against generic high-frequency randomness. This also made de novo visualization harder: using method 1 of Section 5.4 produced images with predicted reward ∼0, likely because penultimate activations were all in the flat part of the ReLU, meaning that no gradient could pass to the images in question.

Results of training probes on this network are shown in Table 5.8, and of training filtered probes on this network are shown in Table 5.9. Regarding the manual adversarial examples described in Section 5.6, this network assigned a mean reward of 8.9 to the unedited examples



(a) Observation of first transition.

(b) Next observation of first transition.

(c) Observation of second transition.

(d) Next observation of second transition.

Figure 5.18: Two visualizations produced at the end of adversarial training. Note the yellow pixels where the coin used to be, and a blue-ish white-ish smudge that looks like a smeared-out agent sprite.

	Distance	Squared distance	Inverse distance	Inverse squared distance
Mean	17.2	1.12×10^4	0.0047	0.0027
Layer 1	15.4	1.1×10^4 (1.2×10^4)	0.0035	0.0023
Layer 2	13.6	1.0×10^4 (1.2×10^4)	0.0031	0.0027
Layer 3	15.2	1.0×10^4 (1.2×10^4)	0.0066	0.0025
Layer 4	16.3 (17.1)	5.3×10^4 (2.4×10^4)	0.0038 (0.0045)	0.0021 (0.0026)

Table 5.8: Test losses of training probes on the adversarially trained CoinRun network. Mean refers to the loss obtained from always guessing the mean value of the attribute in question. All probes are trained with batch size 128 for 5 epochs on a dataset of 805,940 transitions, except for squared distance probes which were trained for 10 epochs. Losses for probes trained identically on randomly initialized networks in parentheses—for the sake of conserving compute, this was only evaluated on layer 4 except for experiments with the squared distance, where it seemed to vary by layer.

	Norm	Squared distance
Mean	1.83	102
Layer 1	1.75 (1.91)	98.5 (99.0)
Layer 2	1.22 (1.87)	85.9 (107)
Layer 3	1.45 (1.81)	85.3 (106)
Layer 4	1.70 (1.86)	233 (109)

Table 5.9: Test losses of training probes on filtered data on the adversarially trained CoinRun network. Mean refers to the loss obtained from always guessing the mean value of the attribute in question. All probes are trained with batch size 128. Probes were trained for 20 epochs on the distance for layers 1 and 2, 40 epochs on the distance for layers 3 and 4, 50 epochs on the squared distance for layer 1, and 30 epochs on the squared distance for layers 2, 3, and 4. Losses for probes trained identically on randomly initialized networks in parentheses.

with a standard deviation of 2.3, a mean reward of 4.6 to the translated coin and sprite examples with a standard deviation of 3.4, and a mean reward of 0.03 to examples where the coin was above the sprite, with a standard deviation of 0.1.

Overall, the adversarially trained network did better than the other networks on avoiding 'visualizations' with high predicted reward that appeared meaningless, including assigning lower reward to random noise, but failed to have better representations as indicated by the results of training probes and finding the network's response to manual adversarial examples. However, this could be due to the fact that the network was not trained to as low a loss as those other networks.

5.9 Dots and distances

In the case of CoinRun networks, we found difficulty in probing them for the distance from the agent sprite to the coin, and concluded that they were not adequately representing that distance. However, that conclusion may have been too hasty. For any monotonically increasing or decreasing function, the network could linearly represent that function of the distance, and calculate the reward by determining whether that function of the distance was over or under the appropriate threshold. If said function looked strange enough, we might get poor results when probing for the distance, despite the network's representations being perfectly adequate.

We therefore wanted to simplify the environment in order to study a setting where it was clear what intermediate variables the neural network should be predicting. In this setting, called 'dots and distances', images feature six filled circles of radius 0.05 dropped uniformly at random on a background white square of width 1: two red, two blue, and then two green. Each image is labelled with the average distance between the centres of circles of the same colour. Example images are shown in Figure 5.19. The mean label is approximately[4] 0.469, and the standard deviation of labels is approximately 0.129.

Two networks were trained by regression on this data: the first on 9e5 training data points for 50 epochs, achieving a test MSE loss of 0.0052; and the second on 9e6 training data points for 5 epochs, achieving a test MSE loss of 0.0055. For reference, the MSE loss one would incur from always guessing the mean is 0.0166. Another network was trained on a classification task, more analogous to the CoinRun reward regression problem: data points were divided into two classes, one where the average distance was less than a cut-off, and one where it was greater. The cut-off was chosen so that 0.93% of data points were in the first class, matching the CoinRun data. The classification network was trained on 9e6 training data points for 5 epochs, and achieved a test cross-entropy loss of 0.013. For reference, the cross-entropy loss incurred from always guessing base rates is 0.053.

These networks were then probed for the vector of distances between each pair of same-coloured dots. All probes were trained for 10 epochs with batch size 128, each on a fresh dataset of size 9e4. Results are shown in Table 5.10, and they show both similarities and differences to the results of probing CoinRun networks. The main difference is that the quality of the representation of the distance vector increases layer by layer until the final layer. By comparison, in CoinRun the best estimate of the distance was in layer 2. Another distinction is that the mean squared error of probing our networks for the distance vector is approximately half that of constantly guessing the mean distance vector. This is better

[4] Precisely, it's $3/50 \times (2 + \sqrt{2} + 5 \log(1 + \sqrt{2}))$, see https://math.stackexchange.com/questions/1294800/average-distance-between-two-randomly-chosen-points-in-unit-square-without-calc and note that in our case the centres of the circles are confined to a square of width 0.9, so that the whole circle is contained in the image.

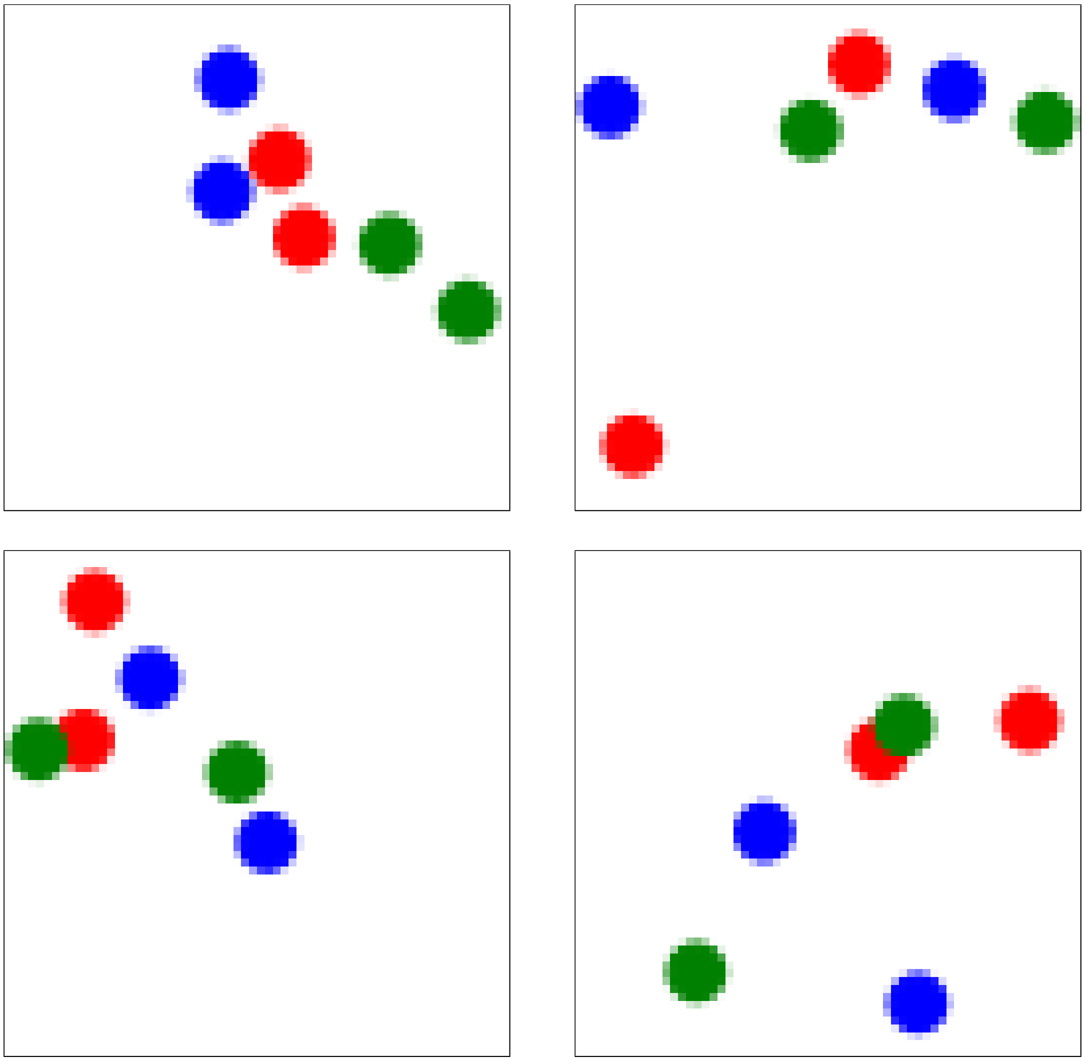


Figure 5.19: Example images in the dots and distances dataset. The black frame is for ease of determining the border of the images, and is not part of the actual images.

	50-epoch net	5-epoch net	Classification net	Random
Layer 1	0.0465	0.0486	0.0426	0.0500 ± 0.0005
Layer 2	0.0326	0.0355	0.0343	0.0500 ± 0.0004
Layer 3	0.0548	0.0311	0.0283	0.0499 ± 0.0001
Layer 4	0.0236	0.0255	0.0255	0.0497 ± 0.0002

Table 5.10: Test losses of training probes for the vector of distances on networks trained on dots and distances. The loss from constantly guessing the mean was 0.0498. All probes are trained with batch size 128, on a fresh dataset of size 9e4 for 10 epochs. The 'Random' column contains the mean and standard deviation of the test accuracies resulting from training probes on five different randomly-initialized networks. We see that networks contain a gradually improving representation of the vector of distances, but this representation is by no means perfect. Note that for Layer 3 of the 50-epoch network, at the end of the second-to-last epoch of probe training the test loss was 0.0304.

than the unfiltered results on CoinRun, where we could only achieve two thirds of the mean squared error of blind guessing (see Tables 5.3 and 5.4), but on par with the filtered results, where we also achieved around half the mean squared error of blind guessing (see Tables 5.6 and 5.7). At any rate, the representation is not very precise or accurate. This could be explained by the relatively high test losses of the networks trained on MSE minimization, which were approximately one third of the loss associated with blindly guessing the mean.

5.10 Discussion

In this chapter, we have used a few conventional interpretability methods to understand the reward networks in question: visualization by optimization, and training linear probes. A natural question to ask is whether these tools worked. In one sense, they did. At the end of this chapter, we know that the CoinRun networks represent something that somewhat matches distance from the agent sprite to the coin on conventional data, that optimal images seem to involve consistent in-focus shapes with a pink to cyan colour transition, and we have some examples of inputs that our networks mispredict reward for. Furthermore, the results of these methods suggested the hypothesis of testing the networks' responses to observations of random pixels, which turned out to be fruitful. However, the tools were less informative than one might hope. We still do not have a solid understanding of how the networks actually work. This suggests that the underlying assumption behind these methods, that networks linearly represent comprehensible features in their activation spaces, is wrong, and a new way of thinking is needed.

That said, our findings are concerning. The reward networks incorrectly handle out-of-distribution inputs, lending plausibility to concerns about goal misgeneralization. Of particular concern is the high reward given to observations of random pixels, which suggests that if an agent used these reward models, and the environment changed so that the agent were able to give itself random noise observations, it may do that to the exclusion of accomplishing valuable tasks. Cohen et al. (2022) details at some length the problems of having an advanced AI agent attempting to 'wirehead' itself by maximizing its own reward signal: it seems likely that similar problems could occur with advanced agents attempting to ensure that they could always perceive noise observations.

Ultimately, this chapter merely prevents case studies. Ideally, more work would be done to investigate the representations of reward networks learned by methods such as RLHF (Christiano et al., 2017) and RLAIF (Bai et al., 2022), although in large models with complex reward functions it may be more difficult to enumerate relevant features and rule out their representation somewhere in the network in the way we have been able to do in this chapter.

Chapter 6

Conclusion

6.1 Summary of dissertation

This dissertation started with a discussion of the importance of having a ‘frame’ to understand the computation inside neural networks, so that we can foresee and prevent goal misgeneralization. It endorsed the frame of neural networks as implementing multi-step computer programs, and discussed the importance in that frame of finding modular structure in the network to understand this computation.

After surveying some literature on the topic of understanding neural networks in Chapter 2, we focussed on graph clusterability as our structure of choice. In Chapter 3, we measured the degree of clusterability in trained neural networks, determined how it compared to that of random neural networks, and determined which training methods promoted clusterability: namely pruning, dropout, and bespoke regularization. Then, in Chapter 4, we attempted to quantify the degree to which functionality was ‘locally specialized’ in the clusters found by spectral clustering: that is, whether clusters were more important to network functionality and more semantically coherent than other groupings of neurons. We found that they were more important, and that they were more coherent in the sense that they were easier to jointly activate. The dissertation closed with Chapter 5, where we embarked on an exploratory analysis of some neural networks trained on reward functions. We use standard interpretability tools designed to understand the features represented by linear combinations of neurons, and determine that seemingly capable reward networks do not use high-quality representations of relevant features, and can be fooled into giving the wrong reward on out-of-distribution samples. This motivates a concern with goal misgeneralization, and also a desire to develop new tools to understand the workings of neural networks. Although this dissertation has by no means established a new paradigm for neural network interpretability, it has taken a step in that direction and motivated a search for one.

6.2 Future work

There is much work to be done extending the research presented in this dissertation.

Firstly, discussion of clusterability focussed its attention to applying the spectral clustering algorithm to image classification networks. One simple extension would be to try different algorithms to see which one was best at finding the relevant structure. These should be tested both against metrics of cluster quality as well as metrics of functional specialization. Another important extension is to the transformer architecture commonly used in large language models (Vaswani et al., 2017). Specifically, it is not obvious how one would meaningfully turn an attention layer into a weighted graph, nor is it obvious how multi-token computation should be represented without redundantly representing the same linear layers for each token. Given the broader goal of understanding network computation as execution of a multi-step program, it is possible that this goal is best served by using existing work providing a programming language that compiles to transformer networks and applying decompilation techniques to it (Lindner et al., 2023; Weiss et al., 2021).

Furthermore, it is not at all obvious that the right way to form a graph out of a neural network is to use one node per neuron. One alternative used by Olah et al. (2018) is to use non-negative matrix factorization to find the ‘right’ basis of activation space. One could then make a graph with those basis vectors as nodes. Another alternative is suggested by research by Bricken et al. (2023) is to train an autoencoder on activation spaces to find latent sparse factors of variation. These appear to be more interpretable than neurons, and perhaps form a more suitable atom of network computation.

Another extension, related to that of finding the right clustering algorithm, is to find a better method of regularizing for clusterability. The approach we use in Subsection 3.5.1 fails to work for convolutional neural networks, and also requires solving an eigenvalue problem each time the regularizer gradient was calculated. One possible approach would be to use the outcome of a power iteration method after a small number of iterations, but it is entirely possible that entirely different approaches are warranted.

Perhaps the most obvious direction for future research, however, is better understanding the relevance of clusters. Work presented in this dissertation provides evidence that they play some functional role, but it is as yet unclear what exactly that role is. One way to do so is to develop better and more fine-grained automated metrics of functional specialization than those we present in Chapter 4. Such metrics could ideally shed light on the kind of functionality contained within different clusters. Another possibility is to use large language models to interpret clusters, as in work by Bills et al. (2023), although the effectiveness of this approach has not yet been proven. It is also possible that manual exploration of clusters might shed light on their functionality, and possibly provide clues as to how to algorithmically analyze them in future.

Bibliography

Alain, G., & Bengio, Y. (2017). Understanding intermediate layers using linear classifier probes. https://openreview.net/forum?id=HJ4-rAVtl

Alet, F., Lozano-Pérez, T., & Kaelbling, L. P. (2018). Modular meta-learning, 856–868.

Alon, N., & Milman, V. D. (1985). λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, *38*(1), 73–88.

Amer, M., & Maul, T. (2019). A review of modularization techniques in artificial neural networks. *Artificial Intelligence Review*, *52*(1), 527–561.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.

Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016). Neural module networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 39–48.

Anthropic. (2023). *Claude's constitution.* https://www.anthropic.com/index/claudes-constitution

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., & Blundell, C. (2020). Agent57: Outperforming the Atari human benchmark. *International Conference on Machine Learning*, 507–517.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*.

Baldwin, C. Y., & Clark, K. B. (2000). *Design rules: The power of modularity* (Vol. 1). MIT press.

Bau, D., Zhou, B., Khosla, A., Oliva, A., & Torralba, A. (2017). Network dissection: Quantifying interpretability of deep visual representations. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6541–6549.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, *57*(1), 289–300.

Bills, S., Cammarata, N., Mossing, D., Tillman, H., Gao, L., Goh, G., Sutskever, I., Leike, J., Wu, J., & Saunders, W. (2023). Language models can explain neurons in language models [https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html].

Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B., Conallen, J., & Houston, K. A. (2007). *Object-oriented analysis and design with applications* (Third). Addison-Wesley Professional.

Bordelon, B., Canatar, A., & Pehlevan, C. (2020). Spectrum dependent learning curves in kernel regression and wide neural networks. *International Conference on Machine Learning*, 1024–1034.

Bostrom, N. (2014). *Superintelligence: Paths, dangers, strategies*. Oxford University Press.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Hatfield-Dodds, Z., Tamkin, A., Nguyen, K., . . . Olah, C. (2023). Towards monosemanticity: Decomposing language models with dictionary learning [https://transformer-circuits.pub/2023/monosemantic-features/index.html]. *Transformer Circuits Thread*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901.

Cammarata, N., Carter, S., Goh, G., Olah, C., Petrov, M., & Schubert, L. (2020). Thread: Circuits [https://distill.pub/2020/circuits]. *Distill*. https://doi.org/10.23915/distill.00024

Carlsmith, J. (2023). Scheming AIs: Will AIs fake alignment during training in order to get power? *arXiv preprint arXiv:2311.08379*.

Carter, S., Armstrong, Z., Schubert, L., Johnson, I., & Olah, C. (2019). Activation atlas [https://distill.pub/2019/activation-atlas/]. *Distill*. https://doi.org/10.23915/distill.00015

Casper, S., Boix, X., D'Amario, V., Guo, L., Schrimpf, M., Vinken, K., & Kreiman, G. (2021). Frivolous units: Wider networks are not really that wide. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(8), 6921–6929.

Casper, S., Nadeau, M., & Kreiman, G. (2021). One thing to fool them all: Generating interpretable, universal, and physically-realizable adversarial features. *arXiv preprint arXiv:2110.03605*.

Chan, L., Garriga-Alonso, A., Goldwosky-Dill, N., Greenblatt, R., Nitishinskaya, J., Radhakrishnan, A., Shlegeris, B., & Thomas, N. (2022). Causal scrubbing, a method for rigorously testing interpretability hypotheses [https://www.alignmentforum.org/posts/JvZhhzycHu2Yd57RN/causal-scrubbing-a-method-for-rigorously-testing]. *AI Alignment Forum*.

Chang, M. B., Gupta, A., Levine, S., & Griffiths, T. L. (2018). Automatically composing representation transformations as a means for generalization. *arXiv preprint arXiv:1807.04640*.

Chen, Z., Lau, E., Mendel, J., Wei, S., & Murfet, D. (2023). Dynamical versus Bayesian phase transitions in a toy model of superposition. *arXiv preprint arXiv:2310.06301.*

Chollet, F., et al. (2015). Keras.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, *30*.

Clune, J., Mouret, J.-B., & Lipson, H. (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society B: Biological Sciences*, *280*(1755).

Cobbe, K., Hesse, C., Hilton, J., & Schulman, J. (2020). Leveraging procedural generation to benchmark reinforcement learning. *International Conference on Machine Learning*, 2048–2056.

Cohen, M., Hutter, M., & Osborne, M. (2022). Advanced artificial agents intervene in the provision of reward. *AI Magazine*, *43*(3), 282–293.

Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., & Garriga-Alonso, A. (2023). Towards automated circuit discovery for mechanistic interpretability. *arXiv preprint arXiv:2304.14997.*

Croce, F., Andriushchenko, M., Sehwag, V., & Debenedetti, E. (2024). *RobustBench: Available leaderboards* [Accessed 2024-01-26]. https://robustbench.github.io/#leaderboard

Croce, F., Andriushchenko, M., Sehwag, V., Debenedetti, E., Flammarion, N., Chiang, M., Mittal, P., & Hein, M. (2020). RobustBench: A standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670.*

Csordás, R., van Steenkiste, S., & Schmidhuber, J. (2021). Are neural nets modular? inspecting their functionality through differentiable weight masks. *International Conference on Learning Representations.*

Davis, B., Bhatt, U., Bhardwaj, K., Marculescu, R., & Moura, J. (2020). On network science and mutual information for explaining deep neural networks. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8399–8403.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., & Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(7), 3366–3385.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

Di Langosco, L. L., Koch, J., Sharkey, L. D., Pfau, J., & Krueger, D. (2022). Goal misgeneralization in deep reinforcement learning. *International Conference on Machine Learning*, 12004–12019.

Dodziuk, J. (1984). Difference equations, isoperimetric inequality and transience of certain random walks. *Transactions of the American Mathematical Society*, *284*(2), 787–794.

Draxler, F., Veschgini, K., Salmhofer, M., & Hamprecht, F. (2018). Essentially no barriers in neural network energy landscape. *International Conference on Machine Learning*, 1309–1318.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., & Olah, C. (2022). Toy models of superposition [https://transformer-circuits.pub/2022/toy_model/index.html]. *Transformer Circuits Thread.*

Elhage, N., Lasenby, R., & Olah, C. (2023). Privileged bases in the transformer residual stream [https://transformer-circuits.pub/2023/privileged-basis/index.html]. *Transformer Circuits Thread.*

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., ... Olah, C. (2021). A mathematical framework for transformer circuits [https://transformer-circuits.pub/2021/framework/index.html]. *Transformer Circuits Thread.*

Engstrom, L., Gilmer, J., Goh, G., Hendrycks, D., Ilyas, A., Madry, A., Nakano, R., Nakkiran, P., Santurkar, S., Tran, B., Tsipras, D., & Wallace, E. (2019). A discussion of 'Adversarial examples are not bugs, they are features' [https://distill.pub/2019/advex-bugs-discussion]. *Distill.* https://doi.org/10.23915/distill.00019

Filan, D. (2022). A nice representation of the Laplacian [https://danielfilan.com/pdfs/laplacian_representation.pdf].

Filan, D., Casper, S., Hod, S., Wild, C., Critch, A., & Russell, S. (2021). Clusterability in neural networks. *arXiv preprint arXiv:2103.03386.*

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks [https://openreview.net/forum?id=rJl-b3RcF7]. *International Conference on Learning Representations.*

Frankle, J., Dziugaite, G. K., Roy, D., & Carbin, M. (2020). Linear mode connectivity and the lottery ticket hypothesis. *International Conference on Machine Learning*, 3259–3269.

Freeman, C. D., & Bruna, J. (2017). Topology and geometry of half-rectified network optimization. *5th International Conference on Learning Representations, ICLR 2017.*

Fu, J., Luo, K., & Levine, S. (2018). Learning robust rewards with adverserial inverse reinforcement learning. *International Conference on Learning Representations.*

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., & Wilson, A. G. (2018). Loss surfaces, mode connectivity, and fast ensembling of DNNs. *Advances in Neural Information Processing Systems*, *31.*

Gazzaniga, M., & Ivry, R. B. (2013). *Cognitive neuroscience: The biology of the mind: Fourth international student edition.* WW Norton.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org/]. MIT Press.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., & Schölkopf, B. (2019). Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893.*

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 770–778. https://doi.org/10.1109/CVPR.2016.90

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *Proceedings of the IEEE International Conference on Computer Vision*, 1026–1034.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., & Steinhardt, J. (2021). Measuring mathematical problem solving with the MATH dataset. *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=7Bywt2mQsCe

Hilton, J., Cammarata, N., Carter, S., Goh, G., & Olah, C. (2020). Understanding RL vision [https://distill.pub/2020/understanding-rl-vision]. *Distill*. https://doi.org/10.23915/distill.00029

Hoogland, J., Gietelink Oldenziel, A., Murfet, D., & van Wingerden, S. (2023). Towards developmental interpretability [https://www.alignmentforum.org/posts/TjaeCWvLZtEDAS5Ex/towards-developmental-interpretability]. *AI Alignment Forum*.

Huber, P. J. (1964). Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, *35*(1), 73–101. https://doi.org/10.1214/aoms/1177703732

Hubinger, E., van Merwijk, C., Mikulik, V., Skalse, J., & Garrabrant, S. (2019). Risks from learned optimization in advanced machine learning systems. *arXiv preprint arXiv:1906.01820*.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., & Madry, A. (2019). Adversarial examples are not bugs, they are features. *Advances in Neural Information Processing Systems*, *32*.

Jacot, A., Gabriel, F., & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems*, *31*.

Jenner, E., & Gleave, A. (2022). Preprocessing reward functions for interpretability. *arXiv preprint arXiv:2203.13553*.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589.

Juneja, J., Bansal, R., Cho, K., Sedoc, J., & Saphra, N. (2022). Linear connectivity reveals generalization strategies. *The Eleventh International Conference on Learning Representations*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirsch, L., Kunze, J., & Barber, D. (2018). Modular networks: Learning to decompose neural computation. *Advances in Neural Information Processing Systems*, *31*, 2408–2418.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* (tech. rep.). University of Toronto.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*, 1097–1105.

Kumar, S. K. (2017). On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*.

Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, *350*(6266), 1332–1338.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, *40*.

Lau, E., Murfet, D., & Wei, S. (2023). Quantifying degeneracy in singular models via the learning coefficient. *arXiv preprint arXiv:2308.12108*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lee, D., Tang, H., Zhang, J., Xu, H., Darrell, T., & Abbeel, P. (2018). Modular architecture for Starcraft II with deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *14*(1), 187–193.

Lee, J. R., Gharan, S. O., & Trevisan, L. (2014). Multiway spectral partitioning and higher-order Cheeger inequalities. *Journal of the ACM*, *61*(6), 1–30.

Lehoucq, R. B., Sorensen, D. C., & Yang, C. (1998). *Arpack users' guide: Solution of large-scale eigenvalue problems with implicitly restarted arnoldi methods*. SIAM.

Lindner, D., Kramár, J., Rahtz, M., McGrath, T., & Mikulik, V. (2023). Tracr: Compiled transformers as a laboratory for interpretability. *arXiv preprint arXiv:2301.05062*.

Lipton, Z. C. (2018). The mythos of model interpretability. *Queue*, *16*(3), 31–57.

Liu, R., Lehman, J., Molino, P., Petroski Such, F., Frank, E., Sergeev, A., & Yosinski, J. (2018). An intriguing failing of convolutional neural networks and the CoordConv solution. *Advances in Neural Information Processing Systems*, *31*.

Liu, S., & Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. *3rd IAPR Asian Conference on Pattern Recognition, ACPR 2015, Kuala Lumpur, Malaysia, November 3-6, 2015*, 730–734. https://doi.org/10.1109/ACPR.2015.7486599

Lu, J., & Ester, M. (2019). Checking functional modularity in DNN by biclustering task-specific hidden neurons. *Real Neurons & Hidden Units: Future directions at the intersection of neuroscience and artificial intelligence at NeurIPS 2019*.

Lubana, E. S., Bigelow, E. J., Dick, R. P., Krueger, D., & Tanaka, H. (2023). Mechanistic mode connectivity. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, & J. Scarlett (Eds.), *Proceedings of the 40th International Conference on Machine Learning* (pp. 22965–23004). PMLR. https://proceedings.mlr.press/v202/lubana23a.html

Madan, S., Henry, T., Dozier, J., Ho, H., Bhandari, N., Sasaki, T., Durand, F., Pfister, H., & Boix, X. (2020). *On the capability of neural networks to generalize to unseen category-pose combinations* (tech. rep.). Center for Brains, Minds and Machines (CBMM).

Mądry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations.*

Magnus, J. R. (1985). On differentiating eigenvalues and eigenvectors. *Econometric Theory*, 179–191.

Marengo, J. E., Farnsworth, D. L., & Stefanic, L. (2017). A geometric derivation of the Irwin-Hall distribution. *International Journal of Mathematics and Mathematical Sciences*, *2017.*

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems [Software available from tensorflow.org]. https://www.tensorflow.org/

McGrath, T., Kapishnikov, A., Tomašev, N., Pearce, A., Wattenberg, M., Hassabis, D., Kim, B., Paquet, U., & Kramnik, V. (2022). Acquisition of chess knowledge in AlphaZero. *Proceedings of the National Academy of Sciences*, *119*(47), e2206625119.

Meilă, M., & Shi, J. (2001). A random walks view of spectral segmentation. *International Workshop on Artificial Intelligence and Statistics*, 203–208.

Michaud, E. J., Gleave, A., & Russell, S. (2020). Understanding learned reward functions. *arXiv preprint arXiv:2012.05862.*

Morcos, A. S., Barrett, D. G., Rabinowitz, N. C., & Botvinick, M. (2018). On the importance of single directions for generalization. *International Conference on Learning Representations.* https://openreview.net/forum?id=r1iuQjxCZ

Mu, J., & Andreas, J. (2020). Compositional explanations of neurons. *Advances in Neural Information Processing Systems*, *33*, 17153–17163.

Nagayasu, S., & Watanabe, S. (2023). Bayesian free energy of deep ReLU neural network in overparametrized cases. *arXiv preprint arXiv:2303.15739.*

Ngo, R., Chan, L., & Mindermann, S. (2022). The alignment problem from a deep learning perspective. *arXiv preprint arXiv:2209.00626.*

Nguyen, A., Yosinski, J., & Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

North, B. V., Curtis, D., & Sham, P. C. (2002). A note on the calculation of empirical P values from Monte Carlo procedures. *The American Journal of Human Genetics*, *71*(2), 439–441.

Olah, C., Mordvintsev, A., & Schubert, L. (2017). Feature visualization [https://distill.pub/2017/feature-visualization]. *Distill.* https://doi.org/10.23915/distill.00007

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., & Mordvintsev, A. (2018). The building blocks of interpretability [https://distill.pub/2018/building-blocks/]. *Distill.* https://doi.org/10.23915/distill.00010

Olah, C. (2022). *Mechanistic interpretability, variables, and the importance of interpretable bases* (tech. rep.). Anthropic.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., ... Olah, C. (2022). In-context learning and induction heads [https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html]. *Transformer Circuits Thread*.

OpenAI. (2022). *Introducing ChatGPT*. https://openai.com/blog/chatgpt

OpenAI. (2023). GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Oyama, E., Chong, N. Y., Agah, A., & Maeda, T. (2001). Inverse kinematics learning by modular architecture neural networks with performance prediction networks. *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, *1*, 1006–1012.

Panzeri, S., Harvey, C. D., Piasini, E., Latham, P. E., & Fellin, T. (2017). Cracking the neural code for sensory perception by combining statistics, intervention, and behavior. *Neuron*, *93*(3), 491–507.

Parascandolo, G., Kilbertus, N., Rojas-Carulla, M., & Schölkopf, B. (2018). Learning independent causal mechanisms. *International Conference on Machine Learning*, 4036–4044.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Petzka, H., Trimmel, M., & Sminchisescu, C. (2020). Notes on the symmetries of 2-layer ReLU-networks. *Proceedings of the Northern Lights Deep Learning Workshop*, *1*, 6–6.

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, *20*.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (2022). Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*.

Räuker, T., Ho, A., Casper, S., & Hadfield-Menell, D. (2023). Toward transparent AI: A survey on interpreting the inner structures of deep neural networks. *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 464–483.

Roberts, D. A., Yaida, S., & Hanin, B. (2022). *The principles of deep learning theory* [https://deeplearningtheory.com]. Cambridge University Press.

Russell, J., & Santos, E. (2019). Explaining reward functions in Markov decision processes. *The Thirty-Second International FLAIRS Conference*.

Samek, W., Montavon, G., Lapuschkin, S., Anders, C. J., & Müller, K.-R. (2021). Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, *109*(3), 247–278.

Schaeffer, S. E. (2007). Graph clustering. *Computer science review*, *1*(1), 27–64.

Shah, R., Varma, V., Kumar, R., Phuong, M., Krakovna, V., Uesato, J., & Kenton, Z. (2022). Goal misgeneralization: Why correct specifications aren't enough for correct goals. *arXiv preprint arXiv:2210.01790.*

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 888–905.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140–1144.

Simon, J. B., Dickens, M., & Deweese, M. (2022). Neural tangent kernel eigenvalues accurately predict generalization. https://openreview.net/forum?id=lycl1GD7fVP

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations (ICLR 2015)*, 1–14.

Şimşek, B., Ged, F., Jacot, A., Spadaro, F., Hongler, C., Gerstner, W., & Brea, J. (2021). Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. *International Conference on Machine Learning*, 9722–9732.

Skalse, J. M. V., Farrugia-Roberts, M., Russell, S., Abate, A., & Gleave, A. (2023). Invariance in policy optimisation and partial identifiability in reward learning. *International Conference on Machine Learning*, 32033–32058.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2818–2826. https://doi.org/10.1109/CVPR.2016.308

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing properties of neural networks. *2nd International Conference on Learning Representations.*

Testolin, A., Piccolini, M., & Suweis, S. (2020). Deep learning systems as complex networks. *Journal of Complex Networks*, *8*(1), cnz018.

The GIMP Development Team. (2023, February 27). *Gimp* (Version 2.10.34). https://www.gimp.org

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., & Tegmark, M. (2020). AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *Advances in Neural Information Processing Systems*, *33*, 4860–4871.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Villalobos, P., Sevilla, J., Besiroglu, T., Heim, L., Ho, A., & Hobbhahn, M. (2022). Machine learning model sizes and the parameter gap. *arXiv preprint arXiv:2207.02852.*

von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, *17*(4), 395–416.

Voss, C., Goh, G., Cammarata, N., Petrov, M., Schubert, L., & Olah, C. (2021). Branch specialization. *Distill*, *6*(4), e00024–008.

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Ostblom, J., Lukauskas, S., Gemperline, D. C., Augspurger, T., Halchenko, Y., Cole, J. B., Warmenhoven, J., de Ruiter, J., Pye, C., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Bachant, P., . . . Qalieh, A. (2018). Mwaskom/seaborn: V0.9.0. https://doi.org/10.5281/zenodo.1313201

Watanabe, C. (2019). Interpreting layered neural networks via hierarchical modular representation. *International Conference on Neural Information Processing*, 376–388.

Watanabe, C., Hiramatsu, K., & Kashino, K. (2018). Modular representation of layered neural networks. *Neural Networks*, *97*, 62–73.

Watanabe, C., Hiramatsu, K., & Kashino, K. (2019). Understanding community structure in layered neural networks. *Neurocomputing*, *367*, 84–102.

Watanabe, S. (2009). *Algebraic geometry and statistical learning theory*. Cambridge University Press.

Watanabe, S. (2018). *Mathematical theory of Bayesian statistics*. CRC Press.

Wei, S., Murfet, D., Gong, M., Li, H., Gell-Redman, J., & Quella, T. (2022). Deep learning is singular, and that's good. *IEEE Transactions on Neural Networks and Learning Systems*.

Weiss, G., Goldberg, Y., & Yahav, E. (2021). Thinking like transformers. *International Conference on Machine Learning*, 11080–11090.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Yang, G., & Hu, E. J. (2021). Tensor programs IV: Feature learning in infinite-width neural networks. *International Conference on Machine Learning*, 11727–11737.

You, J., Leskovec, J., He, K., & Xie, S. (2020). Graph structure of neural networks. *International Conference on Machine Learning*, 10881–10891.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, 818–833.

Zhou, A., Wang, K., Lu, Z., Shi, W., Luo, S., Qin, Z., Lu, S., Jia, A., Song, L., Zhan, M., et al. (2023). Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. *arXiv preprint arXiv:2308.07921*.

Zhou, B., Sun, Y., Bau, D., & Torralba, A. (2018). Revisiting the importance of individual units in CNNs via ablation. *arXiv preprint arXiv:1806.02891*.

Zhou, H., Lan, J., Liu, R., & Yosinski, J. (2019). Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in Neural Information Processing Systems*, *32*, 3592–3602.

Zhu, M., & Gupta, S. (2017). To prune, or not to prune: Exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

Ziegler, D., Nix, S., Chan, L., Bauman, T., Schmidt-Nielsen, P., Lin, T., Scherlis, A., Nabeshima, N., Weinstein-Raun, B., de Haas, D., et al. (2022). Adversarial training for high-stakes reliability. *Advances in Neural Information Processing Systems*, *35*, 9274–9286.

Appendix A

Appendix to ‘Clusterability in Neural Networks’

In this appendix, we give tables and plots providing more information on data that was presented in chapter 3.

First, figures A.1, A.2, and A.3 show the distribution of n-cuts of randomly-initialized MLPs, small CNNs, and VGGs, respectively.

We also show tables of statistics of clusterability experiments shown in the main paper: tables A.1, A.2, A.3, A.4, A.5, A.6, A.7, A.8, A.9, A.10, and A.11. In these, the column “N-cut” shows the mean n-cut of the networks, “Dist. n-cuts” shows the average mean and standard deviation of the distribution of n-cuts of shuffled networks, where the average is taken over different trained networks (which induce different distributions of shuffled networks), and train and test accuracies are also averaged over runs. In table A.7, since we only have one of each network, no averaging is done. “Prop. $p < 0.02$” shows how many networks are more clusterable than all 50 shuffles.

Reg.	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
None	×	10.024	10.165 ± 0.035	5/5	0.997	0.980
Dropout	×	9.97	10.30 ± 0.11	5/5	0.973	0.980
L_1	×	9.06	6.50 ± 0.14	0/5	0.991	0.979
L_2	×	9.639	9.387 ± 0.066	1/5	0.992	0.980
None	√	8.880	9.183 ± 0.031	5/5	1.000	0.984
Dropout	√	8.350	9.247 ± 0.039	5/5	0.967	0.979
L_1	√	10.063	2.72 ± 0.33	0/5	0.997	0.981
L_2	√	8.335	8.599 ± 0.081	3/5	0.998	0.982

Table A.1: Clusterability of MLPs trained on MNIST. See start of appendix A for details.

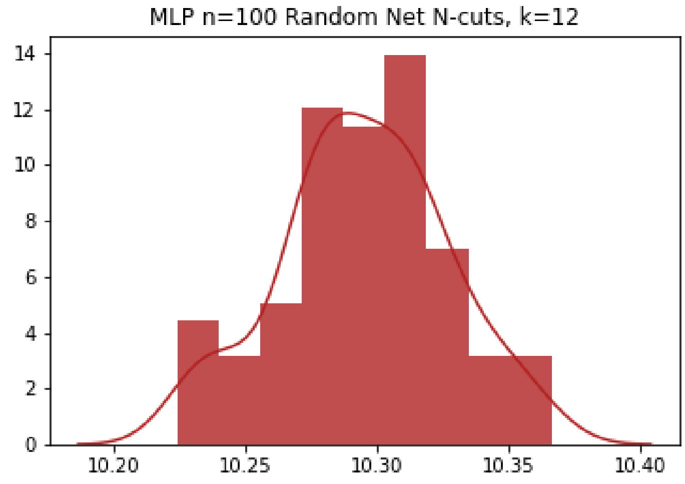


Figure A.1: N-cuts of 100 randomly-initialized MLPs. Vertical axis shows probability density. Plot was generated using the `distplot` function of seaborn 0.9.0 Waskom et al., 2018 with `kde` set to `True`.

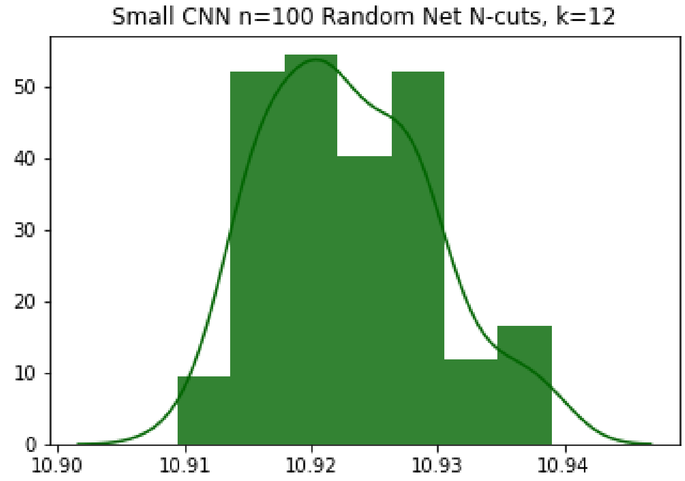


Figure A.2: N-cuts of 100 randomly-initialized small CNNs. Vertical axis shows probability density. Plot was generated using the `distplot` function of seaborn 0.9.0 Waskom et al., 2018 with `kde` set to `True`.

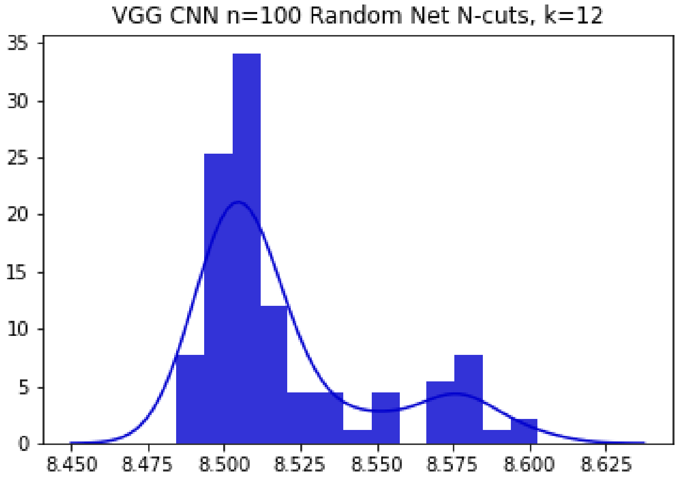


Figure A.3: N-cuts of 100 randomly-initialized VGGs. Vertical axis shows probability density. Plot was generated using the `distplot` function of seaborn 0.9.0 Waskom et al., 2018 with `kde` set to `True`.

Reg.	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
None	$\times$	9.884	10.080 ± 0.037	5/5	0.939	0.891
Dropout	$\times$	9.698	10.052 ± 0.034	5/5	0.862	0.872
L_1	$\times$	8.41	7.21 ± 0.10	0/5	0.907	0.880
L_2	$\times$	9.468	9.185 ± 0.068	0/5	0.913	0.879
None	$\surd$	8.687	9.032 ± 0.035	5/5	0.983	0.894
Dropout	$\surd$	8.110	9.131 ± 0.041	5/5	0.864	0.867
L_1	$\surd$	9.94	4.59 ± 0.23	0/5	0.933	0.887
L_2	$\surd$	8.524	8.746 ± 0.077	3/5	0.951	0.892

Table A.2: Clusterability of MLPs trained on Fashion-MNIST. See start of appendix A for details.

Dataset	Same/diff	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
MNIST	same	$\times$	10.041	10.104 ± 0.027	2/5	0.999	0.993
MNIST	diff	$\times$	9.874	10.312 ± 0.091	5/5	0.989	0.926
MNIST	same	$\surd$	8.923	9.036 ± 0.035	3/5	1.000	0.995
MNIST	diff	$\surd$	8.460	9.181 ± 0.033	5/5	1.000	0.940
Fashion	same	$\times$	9.782	9.919 ± 0.032	4/5	0.972	0.940
Fashion	diff	$\times$	9.881	10.246 ± 0.021	5/5	0.849	0.716
Fashion	same	$\surd$	8.447	8.800 ± 0.038	4/5	0.997	0.947
Fashion	diff	$\surd$	8.637	9.046 ± 0.028	5/5	0.899	0.724

Table A.3: Clusterability of MLPs trained on 'halves' datasets. See start of appendix A for details.

Reg.	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
None	$\times$	10.687	10.673 ± 0.018	0/5	0.998	0.992
Dropout	$\times$	10.688	10.698 ± 0.016	0/5	0.990	0.994
L_1	$\times$	10.31	5.30 ± 0.18	0/5	0.993	0.991
L_2	$\times$	8.050	10.252 ± 0.032	5/5	0.996	0.991
None	$\surd$	10.462	10.034 ± 0.039	0/5	1.000	0.993
Dropout	$\surd$	10.267	10.073 ± 0.039	0/5	0.974	0.993
L_1	$\surd$	9.31	3.05 ± 0.28	0/5	0.995	0.991
L_2	$\surd$	10.754	9.764 ± 0.059	0/5	0.999	0.992

Table A.4: Clusterability of small CNNs trained on MNIST. See start of appendix A for details.

Reg.	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
None	$\times$	10.643	10.671 ± 0.018	1/5	0.982	0.922
Dropout	$\times$	10.615	10.646 ± 0.019	2/5	0.922	0.925
L_1	$\times$	9.98	7.49 ± 0.11	0/5	0.939	0.913
L_2	$\times$	9.303	10.578 ± 0.023	5/5	0.972	0.923
None	$\surd$	10.377	10.060 ± 0.041	0/5	0.995	0.922
Dropout	$\surd$	9.974	9.951 ± 0.042	0/5	0.863	0.905
L_1	$\surd$	10.28	8.62 ± 0.13	0/5	0.954	0.919
L_2	$\surd$	10.445	9.976 ± 0.043	0/5	0.994	0.923

Table A.5: Clusterability of small CNNs trained on Fashion-MNIST. See start of appendix A for details.

Reg.	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
None	×	8.189	3.819 ± 0.235	0/5	0.958	0.882
Dropout	×	8.844	8.283 ± 0.117	0/5	0.904	0.896
L_1	×	5.294	2.074 ± 0.036	0/5	0.912	0.871
L_2	×	8.548	2.104 ± 0.194	0/5	0.958	0.899
Dropout & L_2	×	8.355	8.246 ± 0.101	1/5	0.900	0.898
Control	√	8.133	3.603 ± 0.213	0/5	0.974	0.909
Dropout	√	6.341	7.977 ± 0.206	5/5	0.905	0.905
L_1	√	4.701	0.741 ± 0.052	0/5	0.930	0.881
L_2	√	8.701	1.730 ± 0.223	0/5	0.974	0.908
Dropout & L_2	√	6.135	7.994 ± 0.184	4/5	0.905	0.902

Table A.6: Clusterability of VGGs trained on CIFAR-10. See start of appendix A for details.

Network	N-cut	Dist. n-cuts	p-value	Top-1 acc.
VGG-16	7.800	8.213 ± 0.019	0.01	0.708
VGG-19	7.330	7.655 ± 0.043	0.01	0.709
ResNet-18	7.90	8.51 ± 0.28	0.01	0.682
ResNet-34	6.318	6.375 ± 0.001	0.01	0.722
ResNet-50	4.125	4.185 ± 0.001	0.01	0.748
Inception-V3	2.906	2.938 ± 0.007	0.02	0.776

Table A.7: Clusterability statistics of networks trained on ImageNet. Top-1 accuracies are taken from https://github.com/qubvel/classification_models.

Dataset	Same/diff	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
MNIST	same	×	10.702	10.679 ± 0.017	0/5	0.999	0.998
MNIST	diff	×	10.548	10.583 ± 0.021	1/5	0.979	0.914
MNIST	same	√	10.347	9.969 ± 0.040	0/5	1.000	0.999
MNIST	diff	√	10.225	10.019 ± 0.039	0/5	0.978	0.902
Fashion	same	×	10.701	10.703 ± 0.017	0/5	0.992	0.958
Fashion	diff	×	10.524	10.567 ± 0.020	3/5	0.860	0.703
Fashion	same	√	10.328	10.109 ± 0.039	0/5	1.000	0.966
Fashion	diff	√	9.953	9.990 ± 0.044	0/5	0.818	0.683

Table A.8: Clusterability of small CNNs trained on stack datasets. See start of appendix A for details.

Dataset	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
MNIST	×	8.718	10.065 ± 0.031	5/5	0.996	0.979
MNIST	√	5.750	9.200 ± 0.038	5/5	1.000	0.983
Fashion	×	8.746	10.022 ± 0.038	5/5	0.938	0.892
Fashion	√	6.265	9.065 ± 0.035	5/5	0.980	0.893

Table A.9: Clusterability of clusterably-initialized MLPs. See start of appendix A for details.

Dataset	Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
MNIST	×	10.528	10.658 ± 0.018	5/5	0.998	0.991
MNIST	√	10.148	10.001 ± 0.039	0/5	1.000	0.993
Fashion	×	10.574	10.662 ± 0.018	5/5	0.982	0.921
Fashion	√	10.357	10.034 ± 0.040	0/5	0.995	0.923

Table A.10: Clusterability of clusterably-initialized small CNNs. See start of appendix A for details.

Prune?	N-cut	Dist. n-cuts	$p < 0.02$	Train acc.	Test acc.
×	8.213	3.905 ± 0.219	0/5	0.957	0.898
√	8.212	3.582 ± 0.229	0/5	0.974	0.902

Table A.11: Clusterability of clusterably-initialized VGGs trained on CIFAR-10. See start of appendix A for details.

Appendix B

Appendix to ‘Importance and Coherence’

Tables B.1 (lesion experiments data) and B.2 (feature visualization experiment data) show the mean Fisher statistics from Table 4.1, their p values, and effect measure results from Table 4.2, but do so in a form that places the mean Fisher statistic, p value, and effect measure data from each network in the same row. All Fisher statistics significant according to the Benjamini-Hochberg method are in **bold**, and all effect measures indicating greater average importance/coherence among true subclusters compared to random subclusters are in **bold**.

Table B.1: Results for lesion-based experiments in networks involving importance as measured through overall accuracy drops and coherence as measured by the class-wise range of accuracy drops. Each row corresponds to a network paired with a partitioning method. Results are calculated as explained in Section 4.3.2—in particular, Fisher statistics are means over 5 runs. Each Fisher statistic which is significant at an $\alpha = 0.05$ level under the Benjamini-Hochberg multiple correction is in **bold**. In this case, significance means $p \leq 0.025$. Effect measures are reported together with their standard errors. For both accuracy drop and classwise range experiments, an effect measure > 1 corresponds to more importance/coherence among true subclusters than random ones. All such effect measures which are further than two standard errors above 1 are in **bold**. Note that C-10 is short for CIFAR-10, and IN is short for ImageNet.

		Importance: Acc. Drop		**Coherence: Class Range**	
Network	**Partitioning**	**Fisher Stat.**	**Effect Meas.**	**Fisher Stat.**	**Effect Meas.**
		(p Value)	**High→Imp.**	(p Value)	**High→Coh.**
MLP, MNIST	Weight/Net.	**2.13** (1×10^{-17})	**1.123 ± 0.058**	0.91 (0.82)	0.701 ± 0.036
	Weight/Layer	**2.05** (5×10^{-27})	1.061 ± 0.048	0.91 (0.91)	0.676 ± 0.029
	Act./Net.	**1.46** (2×10^{-11})	0.883 ± 0.038	1.00 (0.56)	0.646 ± 0.024
	Act./Layer	**1.69** (6×10^{-22})	0.929 ± 0.040	1.03 (0.32)	0.687 ± 0.025
CNN, MNIST	Weight/Net.	**1.29** (2×10^{-5})	0.837 ± 0.048	0.84 (0.98)	0.527 ± 0.031
	Weight/Layer	1.10 (0.063)	0.814 ± 0.046	0.94 (0.82)	0.635 ± 0.033
	Act./Net.	**1.73** (4×10^{-10})	1.078 ± 0.061	0.70 (1.00)	0.543 ± 0.039
	Act./Layer	**1.46** (2×10^{-7})	0.939 ± 0.060	0.92 (0.84)	0.625 ± 0.043
VGG, C-10	Weight/Net.	**1.50** (4×10^{-8})	0.682 ± 0.066	**2.12** (2×10^{-29})	0.407 ± 0.042
	Weight/Layer	**1.15** (0.021)	0.808 ± 0.041	**1.27** (2×10^{-4})	0.692 ± 0.033
	Act./Net.	**1.40** (2×10^{-11})	0.926 ± 0.032	0.97 (0.85)	0.679 ± 0.023
	Act./Layer	**1.56** (2×10^{-23})	0.956 ± 0.030	1.03 (0.27)	0.695 ± 0.021
VGG-16, IN	Weight/Net.	**2.54** (3×10^{-23})	**1.205 ± 0.050**	0.49 (1.00)	0.790 ± 0.010
	Weight/Layer	**2.15** (2×10^{-57})	**1.168 ± 0.019**	0.56 (1.00)	0.825 ± 0.005
	Act./Net.	**1.89** (1×10^{-19})	**1.129 ± 0.026**	0.63 (1.00)	0.859 ± 0.006
	Act./Layer	**1.66** (2×10^{-24})	**1.063 ± 0.021**	0.70 (1.00)	0.876 ± 0.005
ResNet-18, IN	Weight/Net.	**1.42** (3×10^{-4})	0.926 ± 0.045	1.13 (0.13)	0.957 ± 0.011
	Weight/Layer	**1.29** (4×10^{-9})	0.971 ± 0.015	0.99 (0.55)	0.971 ± 0.004
	Act./Net.	**1.30** (6×10^{-7})	0.979 ± 0.017	0.92 (0.92)	0.977 ± 0.004
	Act./Layer	**1.31** (5×10^{-10})	0.971 ± 0.015	0.96 (0.794)	0.967 ± 0.004

Table B.2: Results for feature visualization-based experiments in networks involving coherence as measured by the optimization score of feature visualizations and the entropy of network outputs. Each row corresponds to a network paired with a partitioning method. Results are calculated as explained in Section 4.3.2—in particular, Fisher statistics are means over 5 runs. Each Fisher statistic which is significant at an $\alpha = 0.05$ level after Benjamini-Hochberg multiple correction is in **bold**. In this case, significance means $p \leq 0.025$. Effect measures are reported together with their standard errors. For visualization score experiments, an effect measure > 1 corresponds to more coherence among true subclusters than random ones, while one of < 1 does for softmax entropy experiments. All effect measures which are more than two standard errors away from 1 on the side reflecting greater coherence among true subclusters are in **bold**. Note that C-10 is short for CIFAR-10, and IN is short for ImageNet.

		Coherence: Vis Score		Coherence: Softmax H	
Network	**Partitioning**	**Fisher Stat.**	**Effect Meas.**	**Fisher Stat.**	**Effect Meas.**
		(p Value)	**High→Coh.**	(p Value)	**Low→Coh.**
MLP, MNIST	Weight/Net.	**1.32** (0.001)	1.003 ± 0.004	0.92 (0.71)	1.105 ± 0.021
	Weight/Layer	**1.21** (0.002)	**1.024 ± 0.004**	**1.23** (0.023)	0.931 ± 0.016
	Act./Net.	**1.34** (4×10^{-5})	**1.020 ± 0.003**	**1.15** (0.025)	0.997 ± 0.013
	Act./Layer	**1.36** (9×10^{-7})	**1.026 ± 0.003**	**1.12** (0.013)	1.011 ± 0.012
CNN, MNIST	Weight/Net.	1.10 (0.082)	0.998 ± 0.004	0.93 (0.86)	1.026 ± 0.008
	Weight/Layer	1.02 (0.34)	1.004 ± 0.003	0.99 (0.52)	1.007 ± 0.007
	Act./Net.	1.09 (0.27)	0.933 ± 0.006	0.90 (0.83)	1.025 ± 0.011
	Act./Layer	1.05 (0.22)	0.925 ± 0.005	0.98 (0.67)	**0.970 ± 0.010**
VGG, C-10	Weight/Net.	**1.46** (7×10^{-8})	0.871 ± 0.011	0.99 (0.61)	1.124 ± 0.012
	Weight/Layer	1.00 (0.44)	**1.013 ± 0.006**	0.97 (0.62)	0.992 ± 0.012
	Act./Net.	**2.34** (6×10^{-64})	**1.327 ± 0.005**	1.08 (0.045)	**0.950 ± 0.009**
	Act./Layer	**2.67** (8×10^{-121})	**1.379 ± 0.004**	**1.12** (0.013)	**0.930 ± 0.008**
VGG-16, IN	Weight/Net.	**1.72** (6×10^{-8})	**1.043 ± 0.005**	1.19 (0.052)	0.998 ± 0.001
	Weight/Layer	**1.90** (3×10^{-39})	**1.076 ± 0.003**	1.06 (0.16)	**0.991 ± 0.001**
	Act./Net.	**1.83** (3×10^{-15})	**1.066 ± 0.003**	1.07 (0.21)	1.000 ± 0.001
	Act./Layer	**1.85** (1×10^{-34})	**1.056 ± 0.003**	0.98 (0.63)	1.001 ± 0.001
VGG-19, IN	Weight/Net.	**1.91** (2×10^{-10})	**1.061 ± 0.004**	1.03 (0.40)	1.003 ± 0.001
	Weight/Layer	**2.23** (4×10^{-81})	**1.099 ± 0.003**	1.00 (0.46)	1.001 ± 0.001
	Act./Net.	**1.87** (6×10^{-22})	**1.046 ± 0.003**	1.10 (0.089)	**0.996 ± 0.001**
	Act./Layer	**2.01** (3×10^{-27})	**1.081 ± 0.002**	0.98 (0.65)	1.004 ± 0.001

Appendix C

Appendix to ‘Feature Representation in Reward Models’

C.1 Visualizations

Figures C.1, C.2, and C.3 show de novo visualizations of the 15-epoch CoinRun network.

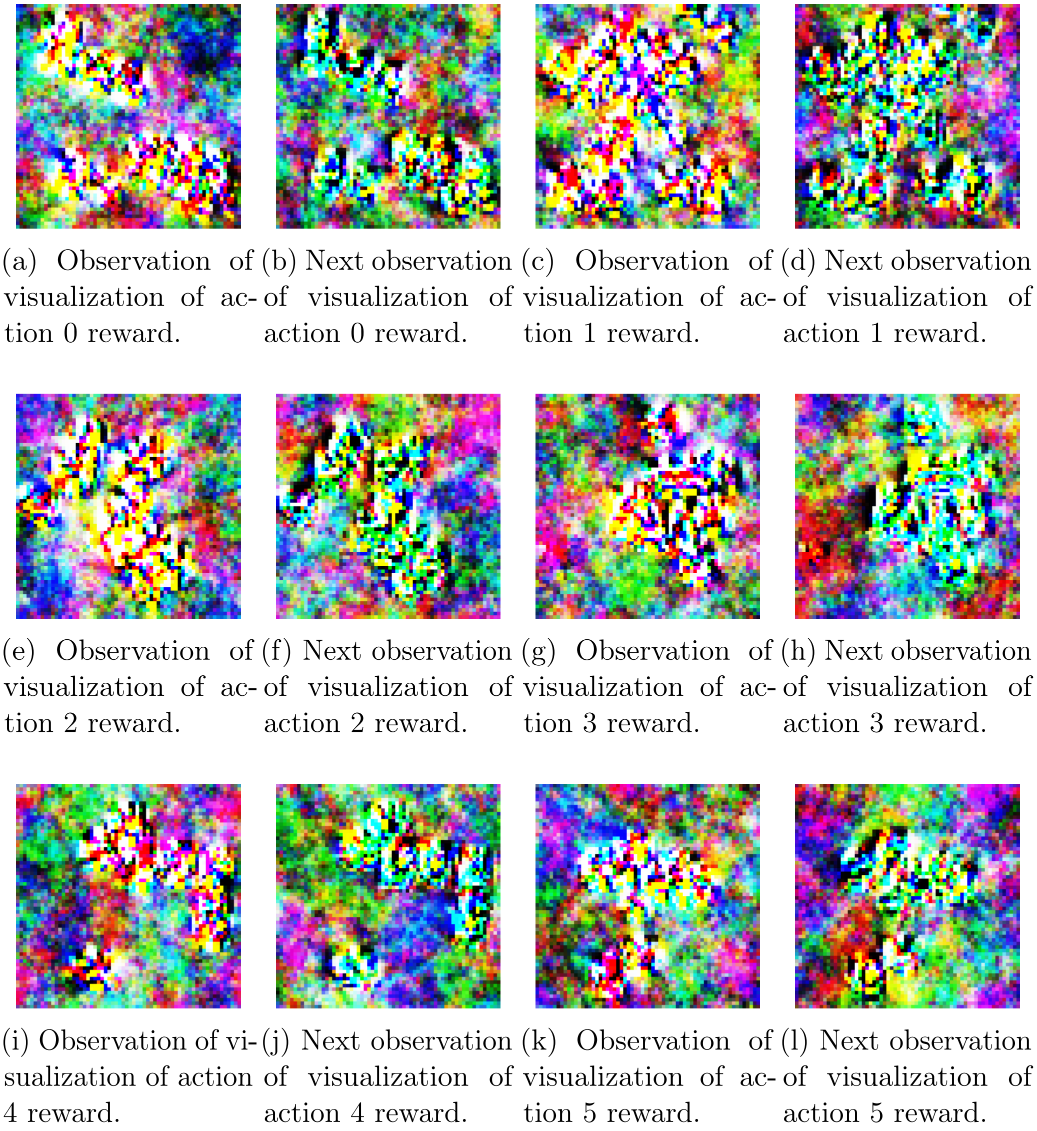
Figures C.4, C.5, C.6, C.7, and C.8 show sample visualizations produced during adversarial training, as described in Section 5.8.

C.2 Manual adversarial examples

Figures C.9 and C.10 show the original transitions used to make manual adversarial examples, as described in Section 5.6. Figures C.11, C.12, and C.13 show transitions of manual adversarial examples where the coin and agent sprite have been translated to the left, and Figures C.14, C.15, C.16 show transitions of manual adversarial examples where the coin has been placed above the agent sprite.

C.2.1 Unused manual adversarial example attempts

Initially, a larger set of manual adversarial example attempts were tested on a smaller network that only took in an observation and an action, allowing more concepts to be tried with less editing. Most were unsuccessful at fooling the network. In Figures C.17, C.18, C.19, C.20, C.21, C.22, C.23, C.24, C.25, C.26, C.27, and C.28, we show the attempts that were not used in the main paper.



(a) Observation of visualization of action 0 reward.

(b) Next observation of visualization of action 0 reward.

(c) Observation of visualization of action 1 reward.

(d) Next observation of visualization of action 1 reward.

(e) Observation of visualization of action 2 reward.

(f) Next observation of visualization of action 2 reward.

(g) Observation of visualization of action 3 reward.

(h) Next observation of visualization of action 3 reward.

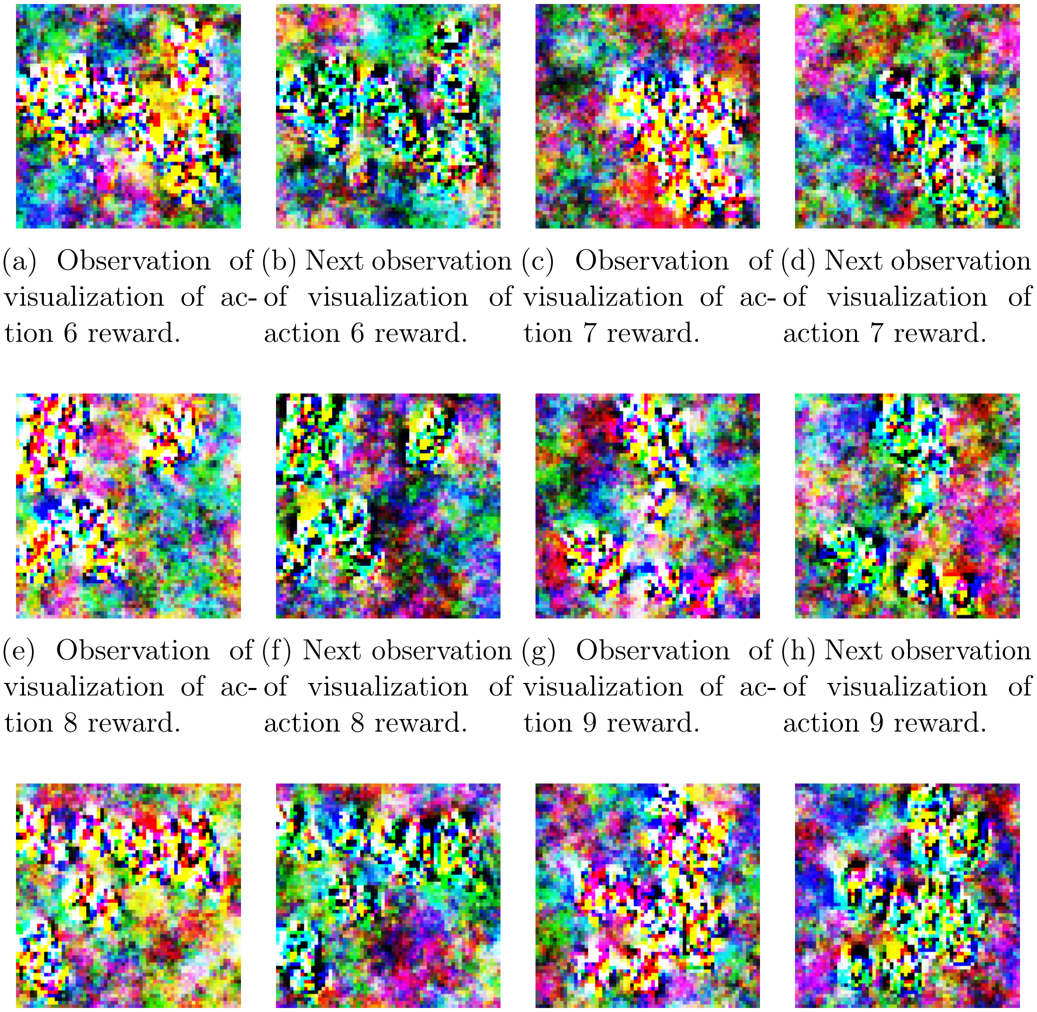
(i) Observation of visualization of action 4 reward.

(j) Next observation of visualization of action 4 reward.

(k) Observation of visualization of action 5 reward.

(l) Next observation of visualization of action 5 reward.

Figure C.1: De novo visualizations of reward predictions of the first 6 actions of the 15-epoch CoinRun reward network. For details, see Section 5.4.



(a) Observation of visualization of action 6 reward.

(b) Next observation of visualization of action 6 reward.

(c) Observation of visualization of action 7 reward.

(d) Next observation of visualization of action 7 reward.

(e) Observation of visualization of action 8 reward.

(f) Next observation of visualization of action 8 reward.

(g) Observation of visualization of action 9 reward.

(h) Next observation of visualization of action 9 reward.

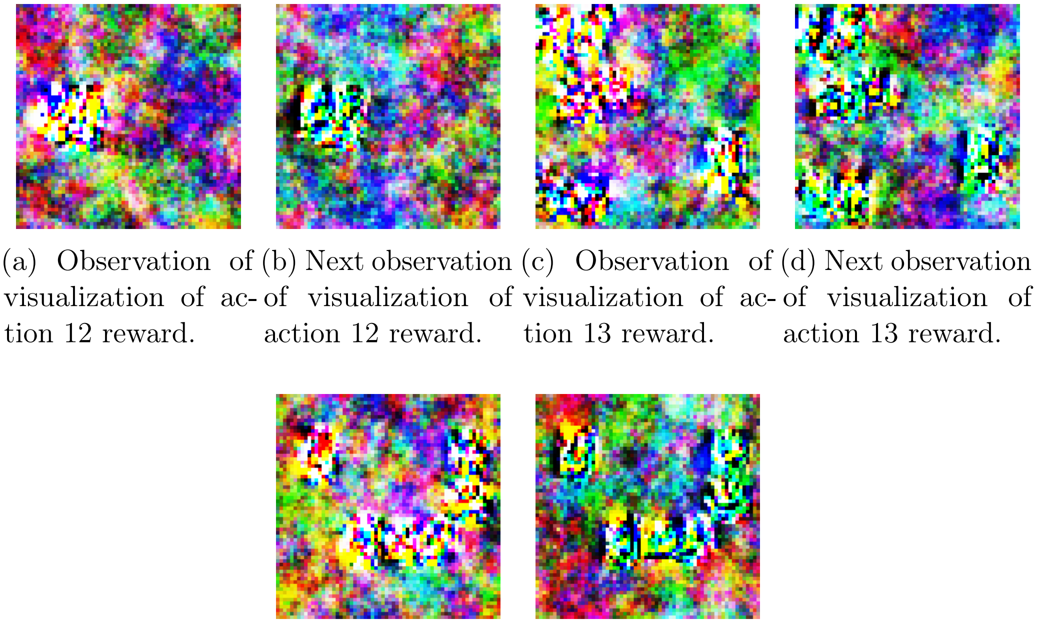
(i) Observation of visualization of action 10 reward.

(j) Next observation of visualization of action 10 reward.

(k) Observation of visualization of action 11 reward.

(l) Next observation of visualization of action 11 reward.

Figure C.2: De novo visualizations of reward predictions of actions 6 through 11 of the 15-epoch CoinRun reward network. For details, see Section 5.4.



(a) Observation of visualization of action 12 reward.

(b) Next observation of visualization of action 12 reward.

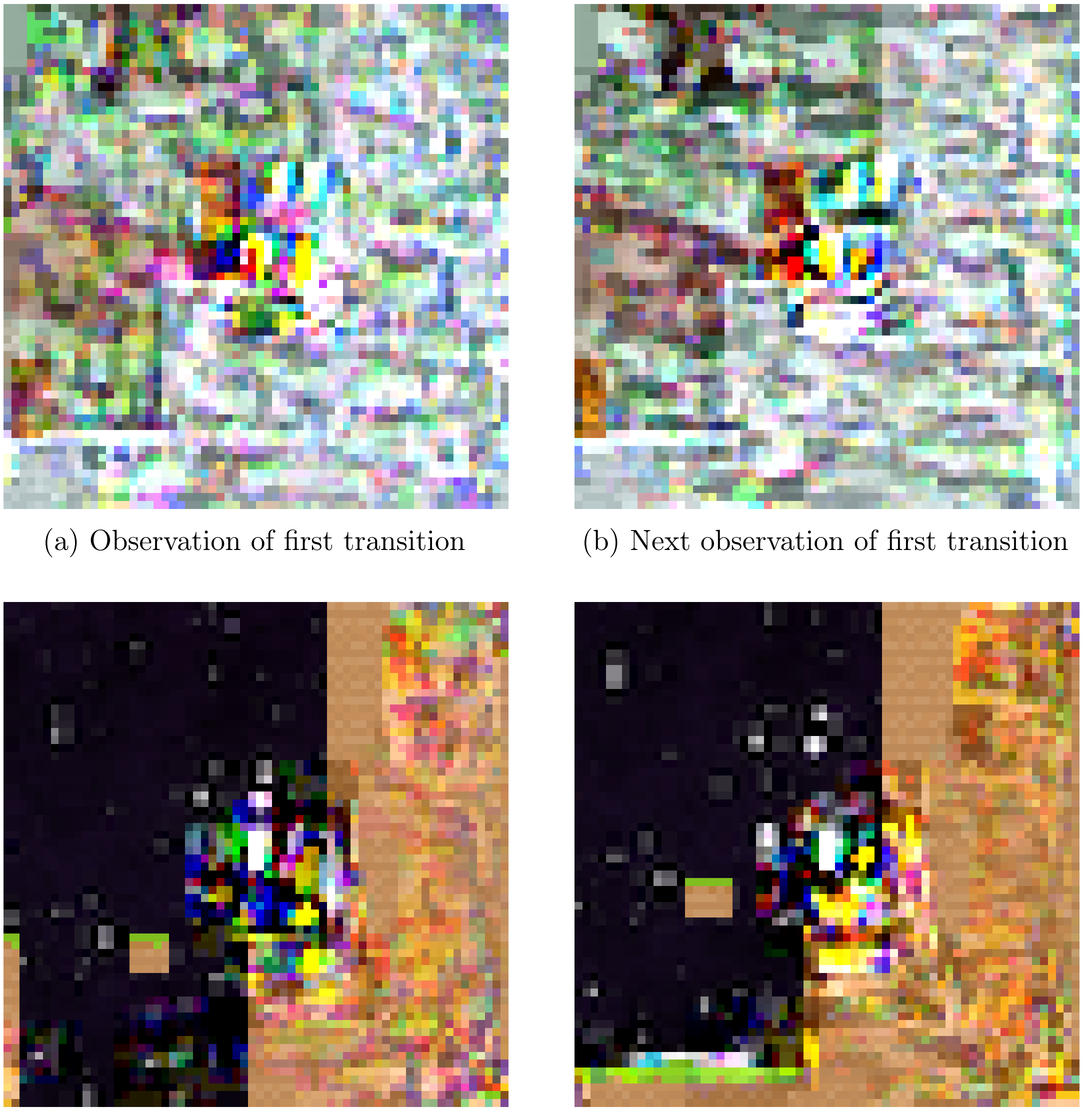
(c) Observation of visualization of action 13 reward.

(d) Next observation of visualization of action 13 reward.

(e) Observation of visualization of action 14 reward.

(f) Next observation of visualization of action 14 reward.

Figure C.3: De novo visualizations of reward predictions of actions 6 through 11 of the 15-epoch CoinRun reward network. For details, see Section 5.4.



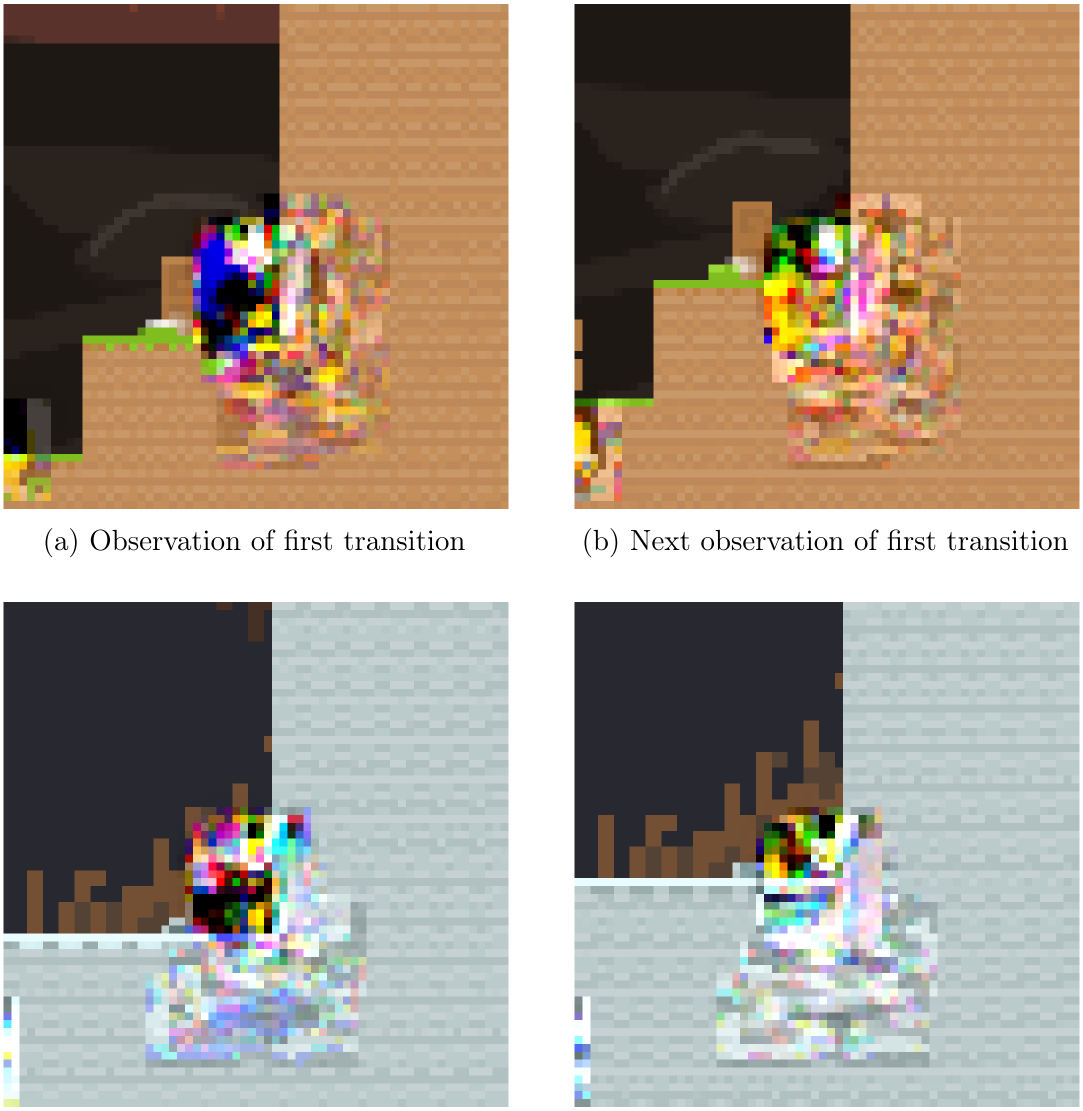
(a) Observation of first transition

(b) Next observation of first transition

(c) Observation of second transition

(d) Next observation of second transition

Figure C.4: Two visualizations produced after epoch 5 of adversarial training.



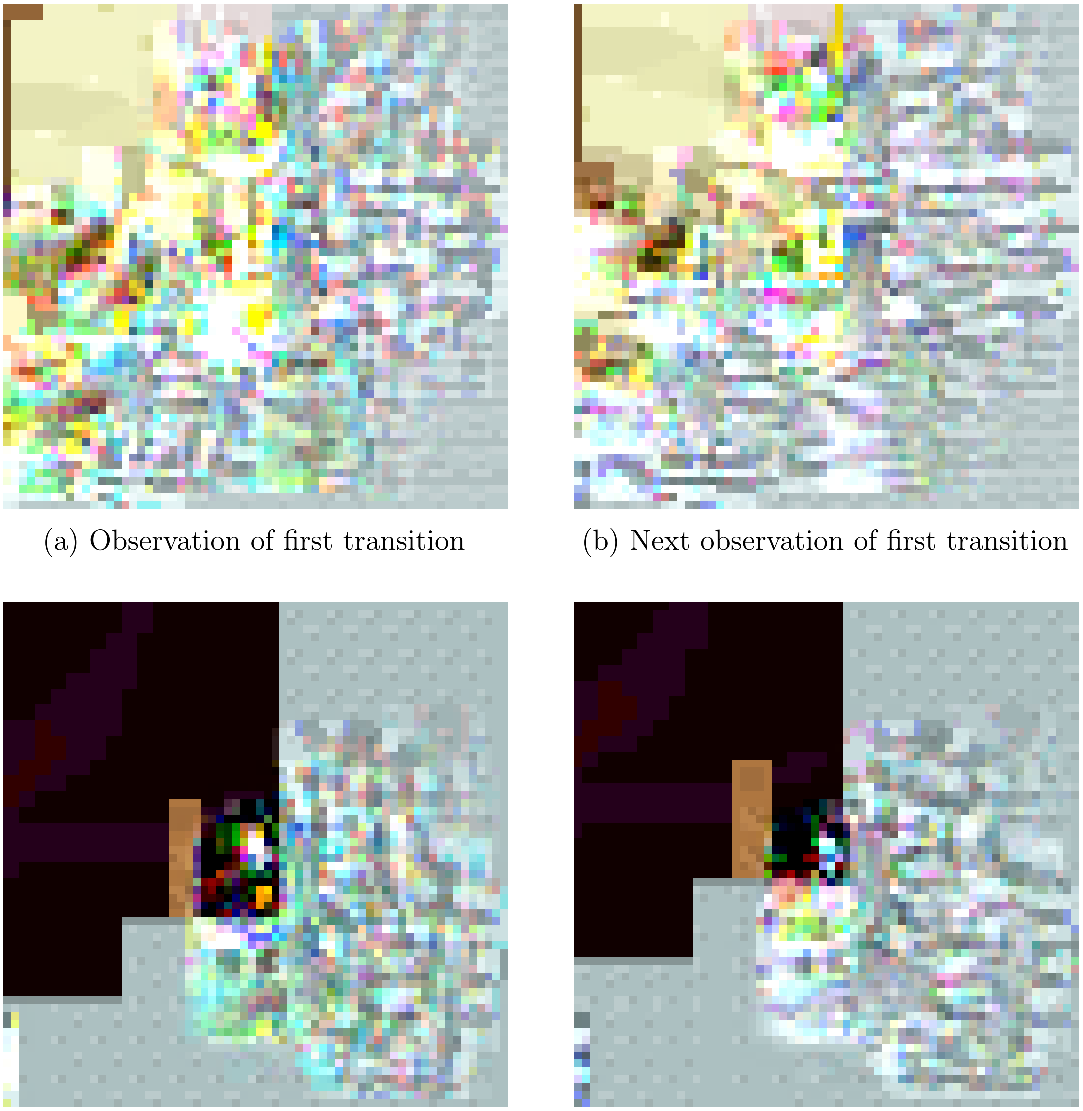
(a) Observation of first transition

(b) Next observation of first transition

(c) Observation of second transition

(d) Next observation of second transition

Figure C.5: Two visualizations produced after epoch 6 of adversarial training.



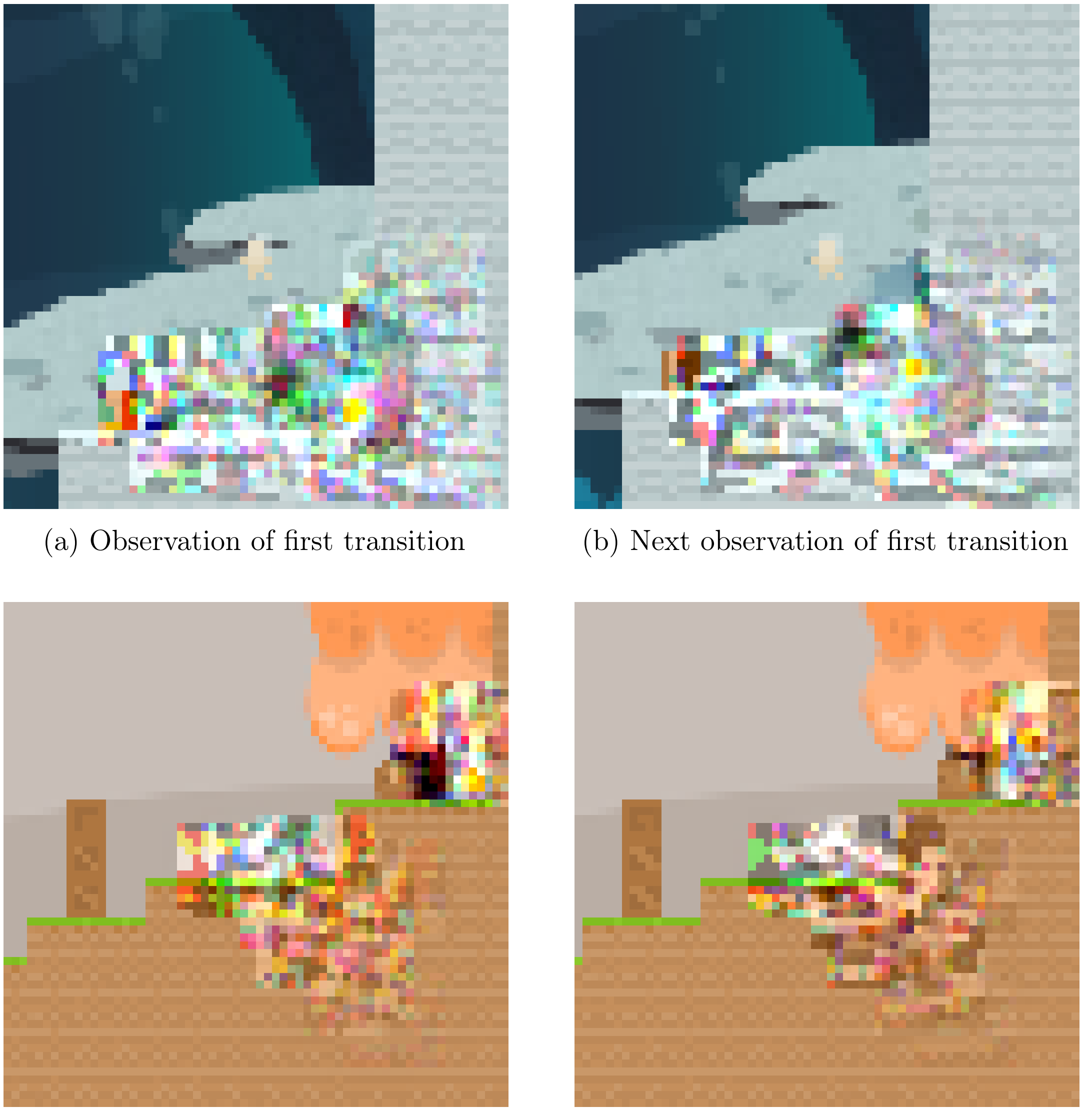
(a) Observation of first transition

(b) Next observation of first transition

(c) Observation of second transition

(d) Next observation of second transition

Figure C.6: Two visualizations produced after epoch 7 of adversarial training.



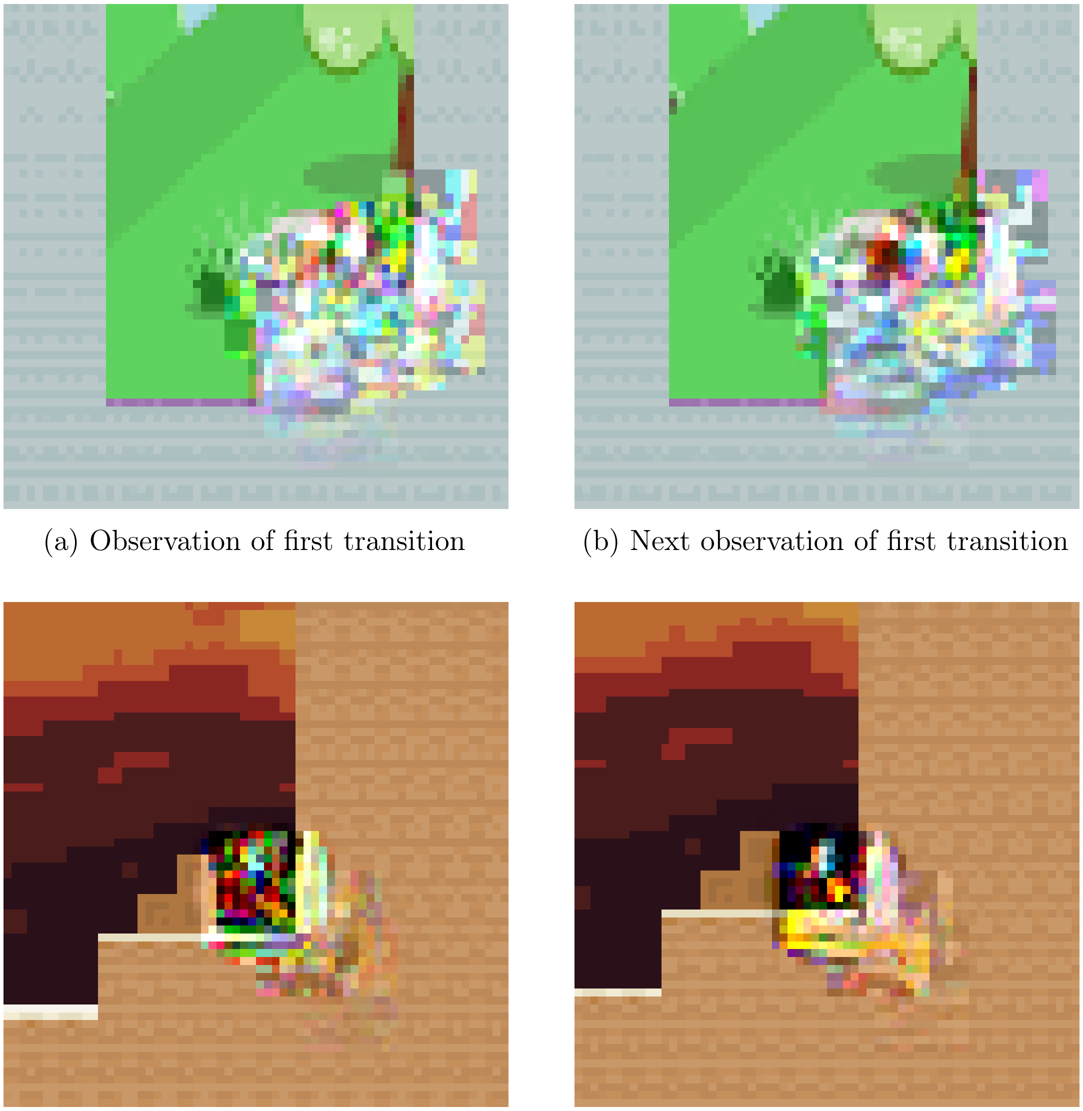
(a) Observation of first transition

(b) Next observation of first transition

(c) Observation of second transition

(d) Next observation of second transition

Figure C.7: Two visualizations produced after epoch 8 of adversarial training.



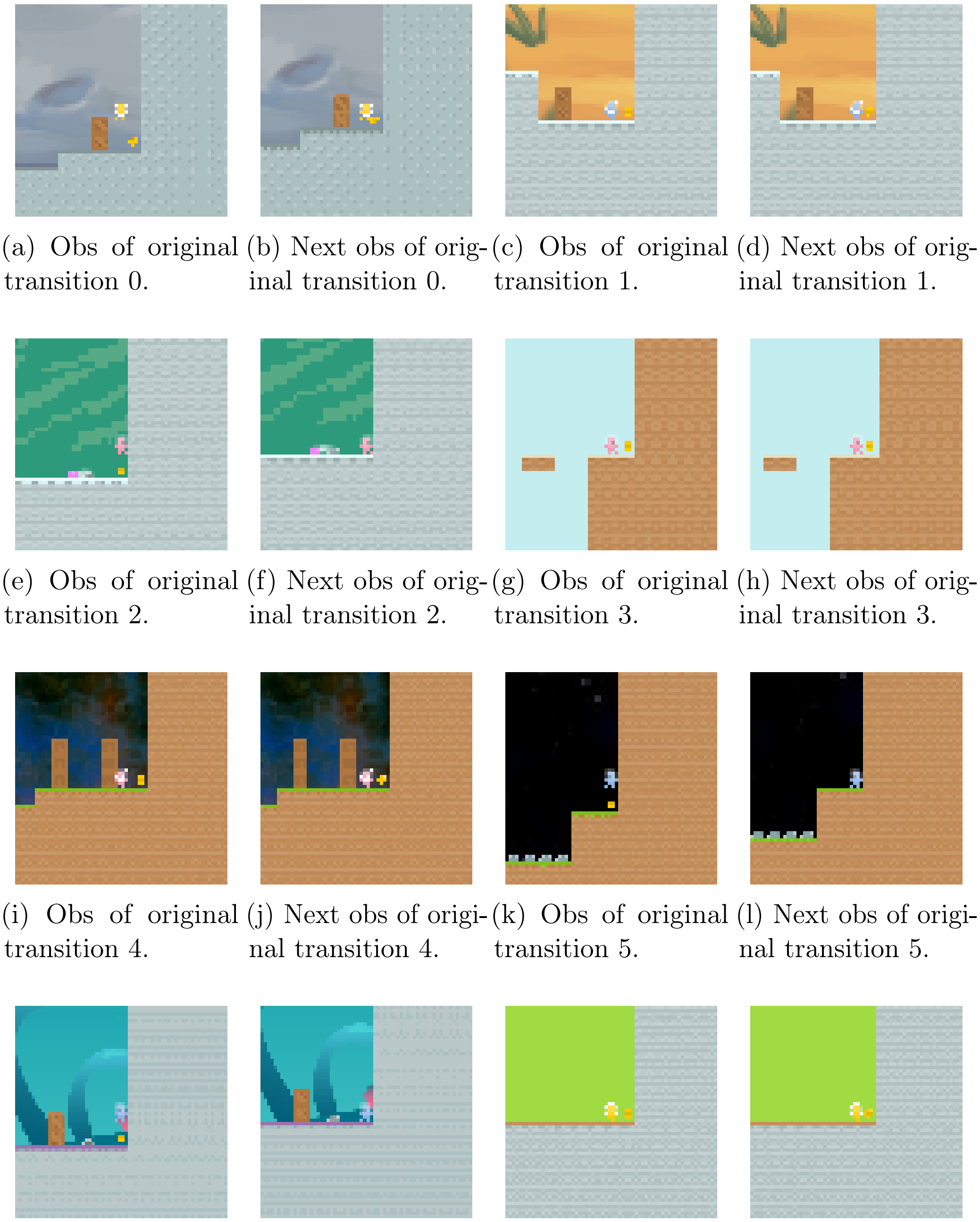
(a) Observation of first transition

(b) Next observation of first transition

(c) Observation of second transition

(d) Next observation of second transition

Figure C.8: Two visualizations produced after epoch 9 of adversarial training.



(a) Obs of original transition 0.

(b) Next obs of original transition 0.

(c) Obs of original transition 1.

(d) Next obs of original transition 1.

(e) Obs of original transition 2.

(f) Next obs of original transition 2.

(g) Obs of original transition 3.

(h) Next obs of original transition 3.

(i) Obs of original transition 4.

(j) Next obs of original transition 4.

(k) Obs of original transition 5.

(l) Next obs of original transition 5.

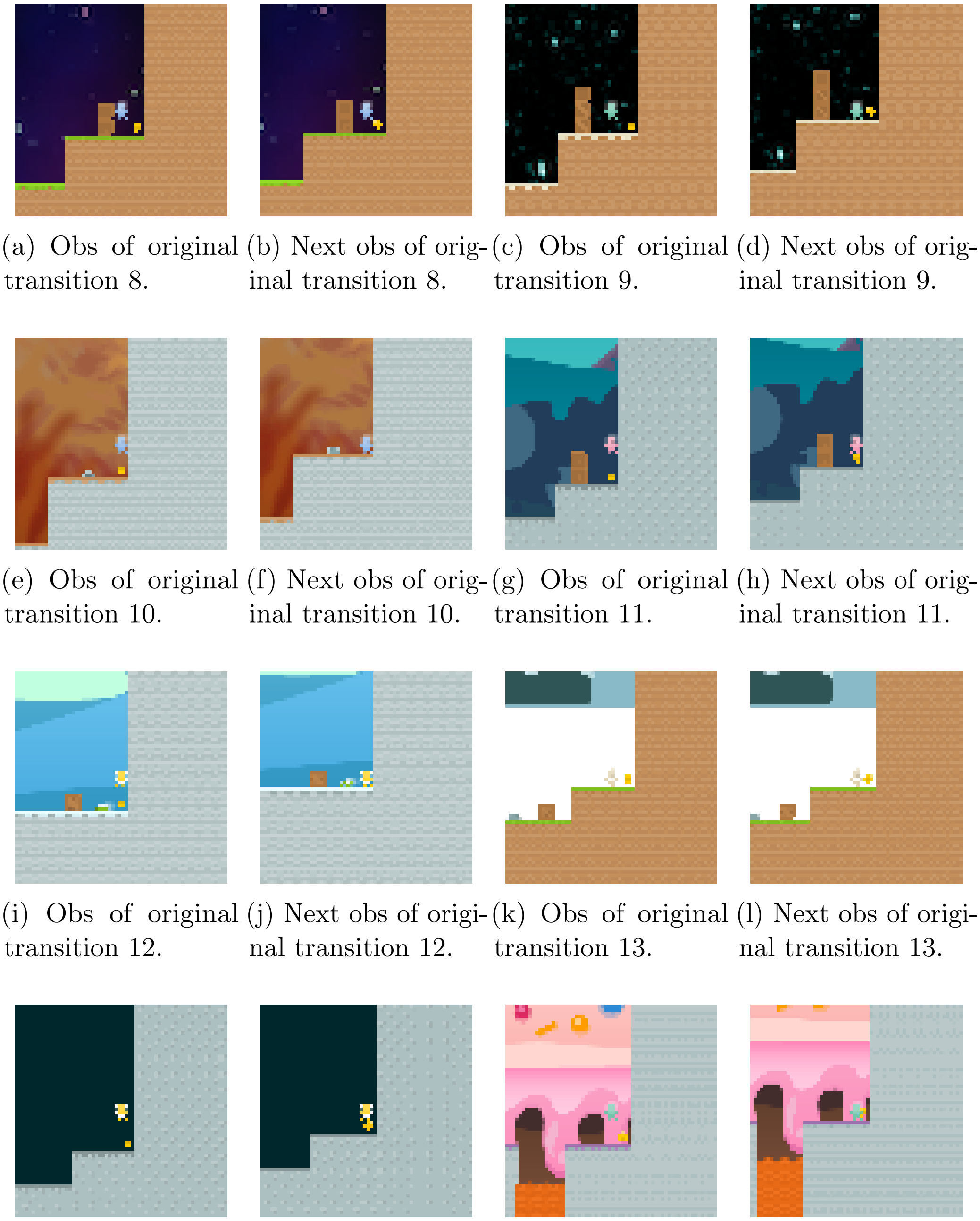
(m) Obs of original transition 6.

(n) Next obs of original transition 6.

(o) Obs of original transition 7.

(p) Next obs of original transition 7.

Figure C.9: Original transitions used to make manual adversarial examples 0 through 7. For more details, see Section 5.6.



(a) Obs of original transition 8.

(b) Next obs of original transition 8.

(c) Obs of original transition 9.

(d) Next obs of original transition 9.

(e) Obs of original transition 10.

(f) Next obs of original transition 10.

(g) Obs of original transition 11.

(h) Next obs of original transition 11.

(i) Obs of original transition 12.

(j) Next obs of original transition 12.

(k) Obs of original transition 13.

(l) Next obs of original transition 13.

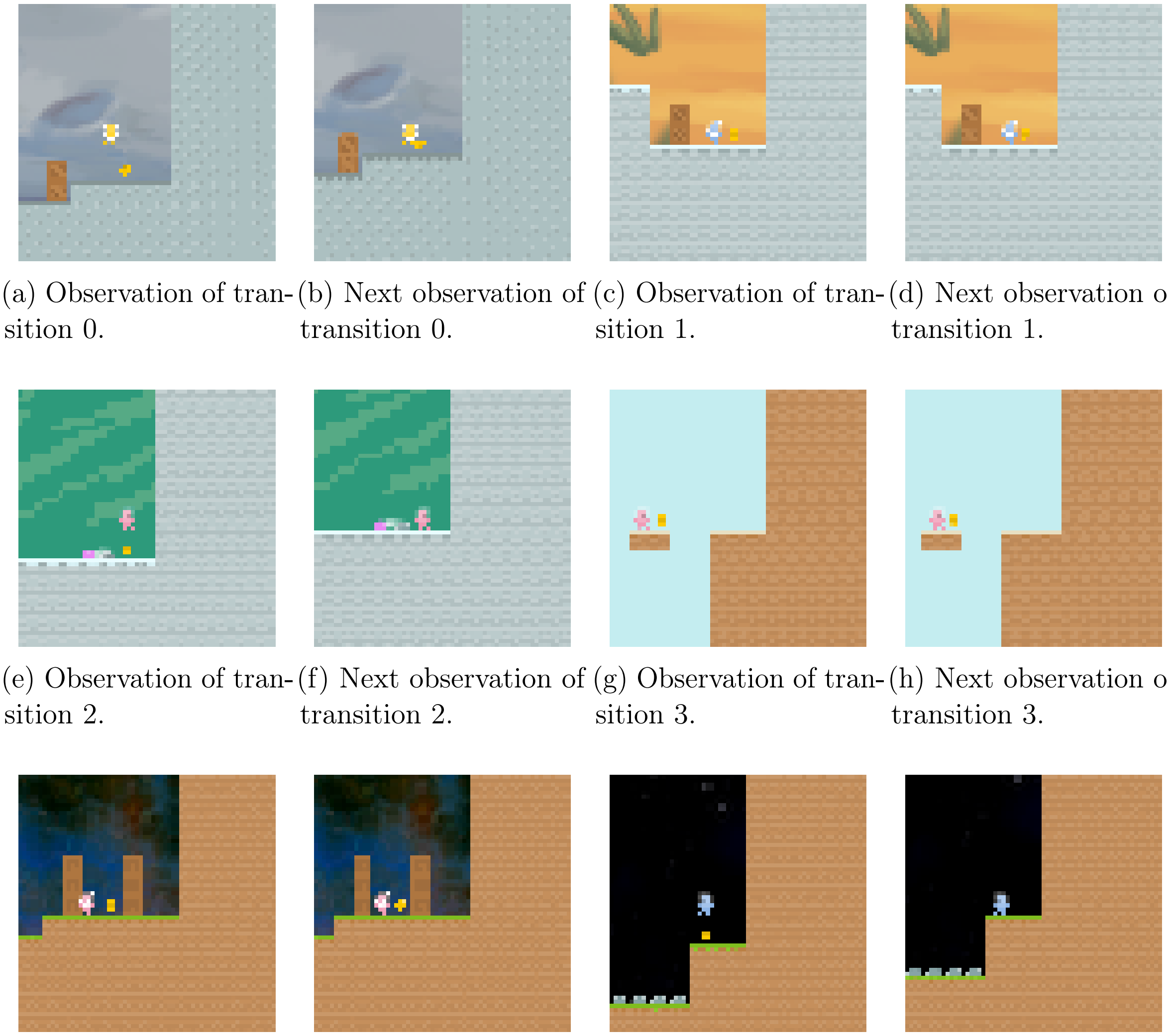
(m) Obs of original transition 14.

(n) Next obs of original transition 14.

(o) Obs of original transition 15.

(p) Next obs of original transition 15.

Figure C.10: Original transitions used to make manual adversarial examples 8 through 15. For more details, see Section 5.6.



(a) Observation of transition 0.

(b) Next observation of transition 0.

(c) Observation of transition 1.

(d) Next observation of transition 1.

(e) Observation of transition 2.

(f) Next observation of transition 2.

(g) Observation of transition 3.

(h) Next observation of transition 3.

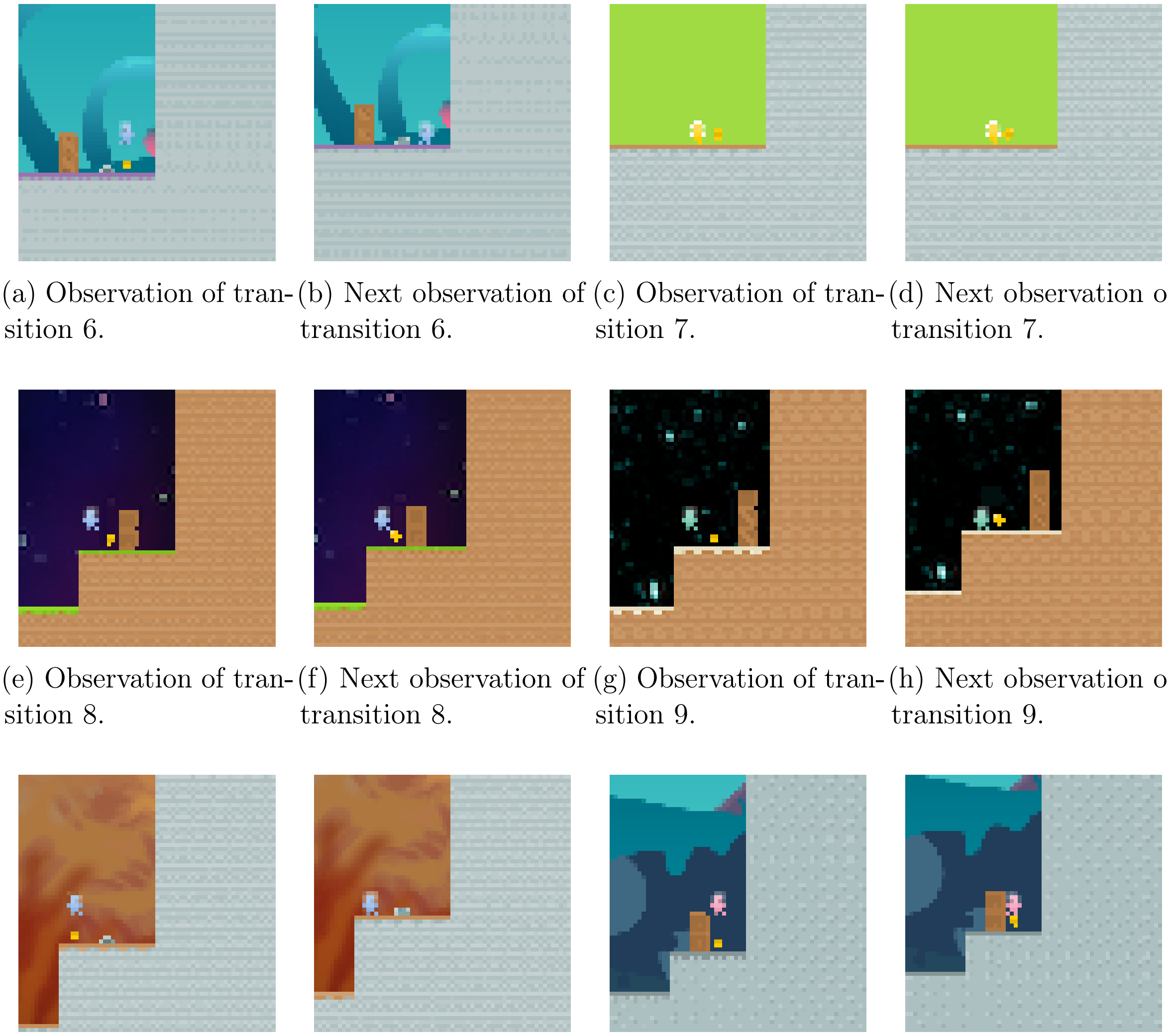
(i) Observation of transition 4.

(j) Next observation of transition 4.

(k) Observation of transition 5.

(l) Next observation of transition 5.

Figure C.11: Manual adversarial examples 0 through 5 where the agent sprite and coin have been manually translated to the left. For more details, see Section 5.6.



(a) Observation of transition 6.

(b) Next observation of transition 6.

(c) Observation of transition 7.

(d) Next observation of transition 7.

(e) Observation of transition 8.

(f) Next observation of transition 8.

(g) Observation of transition 9.

(h) Next observation of transition 9.

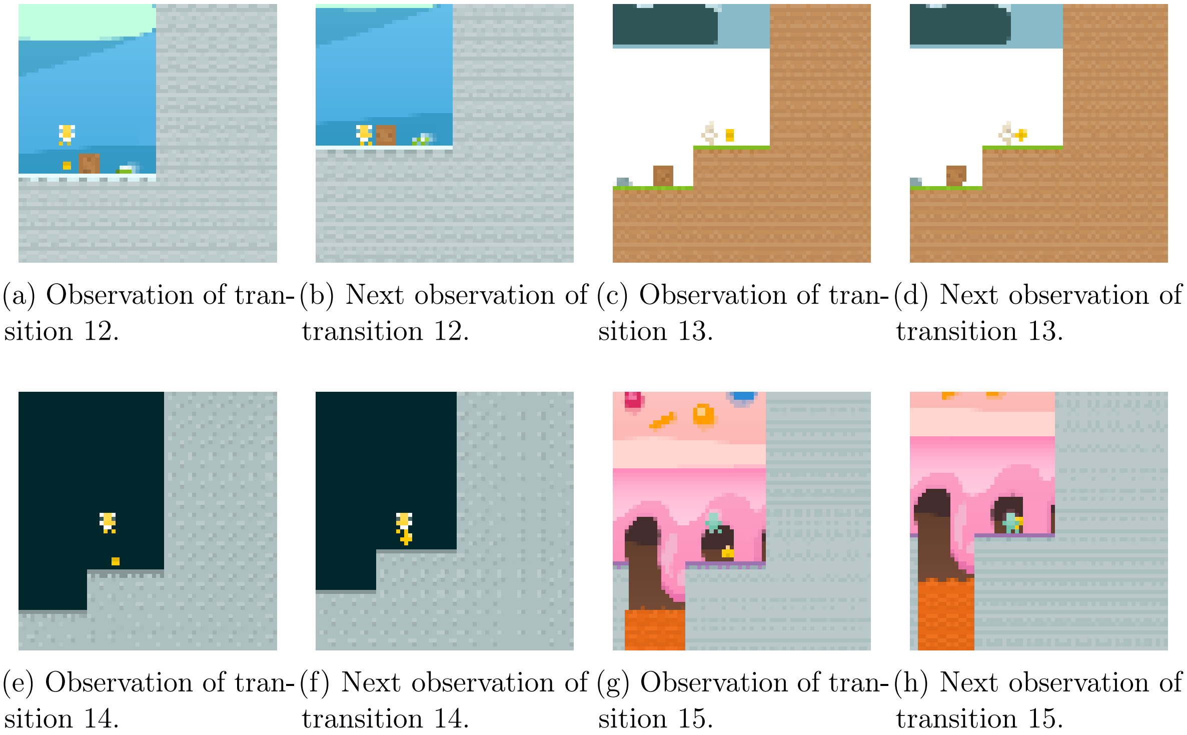
(i) Observation of transition 10.

(j) Next observation of transition 10.

(k) Observation of transition 11.

(l) Next observation of transition 11.

Figure C.12: Manual adversarial examples 6 through 11 where the agent sprite and coin have been manually translated to the left. For more details, see Section 5.6.



(a) Observation of transition 12.

(b) Next observation of transition 12.

(c) Observation of transition 13.

(d) Next observation of transition 13.

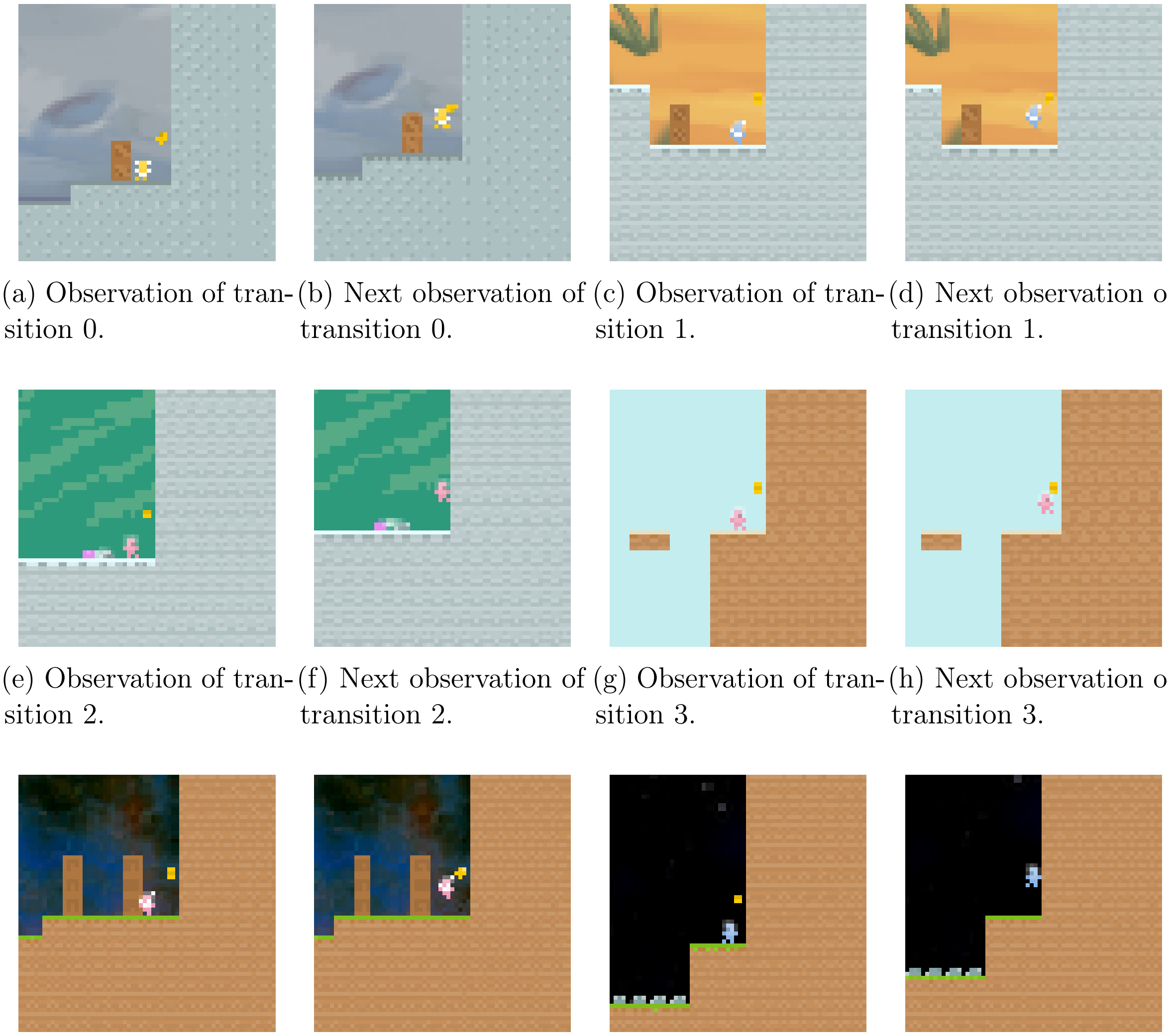
(e) Observation of transition 14.

(f) Next observation of transition 14.

(g) Observation of transition 15.

(h) Next observation of transition 15.

Figure C.13: Manual adversarial examples 12 through 15 where the agent sprite and coin have been manually translated to the left. For more details, see Section 5.6.



(a) Observation of transition 0.

(b) Next observation of transition 0.

(c) Observation of transition 1.

(d) Next observation of transition 1.

(e) Observation of transition 2.

(f) Next observation of transition 2.

(g) Observation of transition 3.

(h) Next observation of transition 3.

(i) Observation of transition 4.

(j) Next observation of transition 4.

(k) Observation of transition 5.

(l) Next observation of transition 5.

Figure C.14: Manual adversarial examples 0 through 5 where the coin has been placed above the agent sprite. For more details, see Section 5.6.

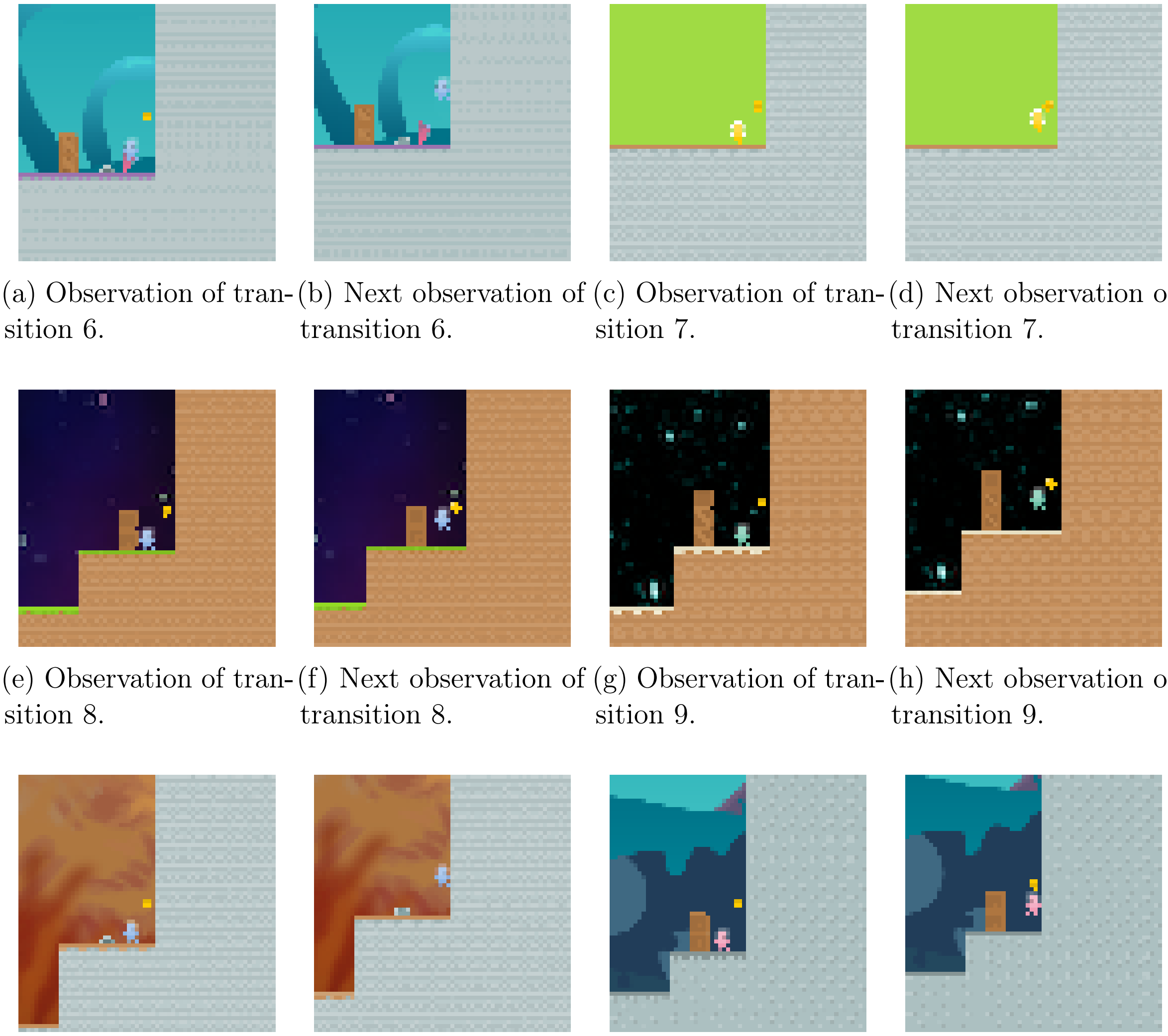
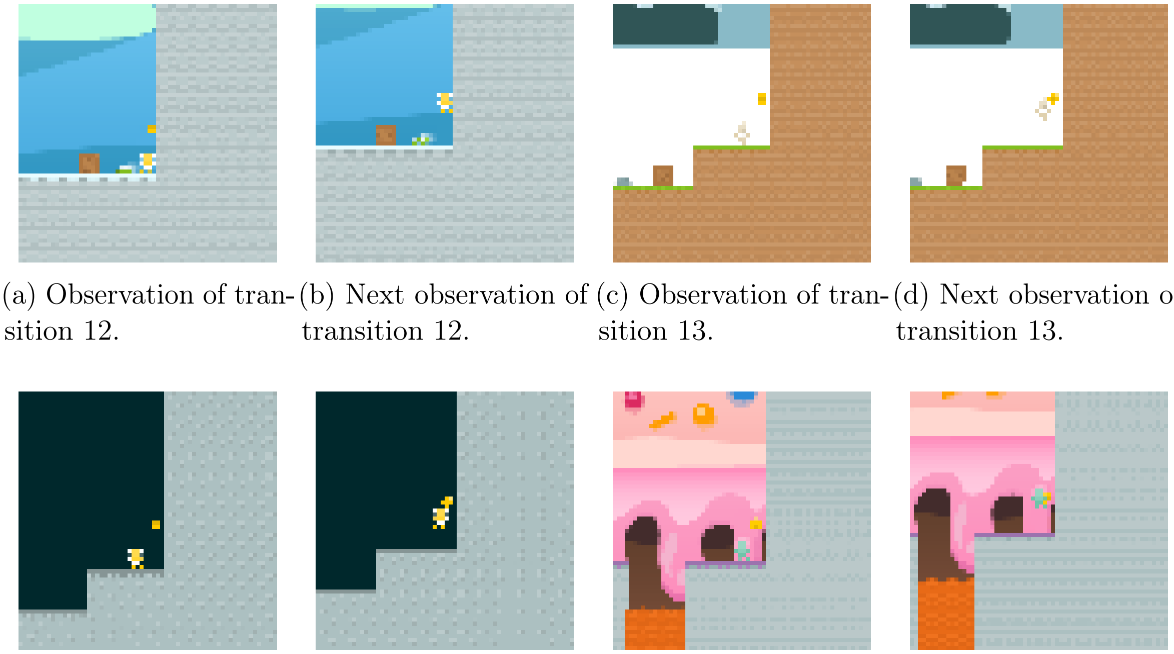

(a) Observation of transition 6.
(b) Next observation of transition 6.
(c) Observation of transition 7.
(d) Next observation of transition 7.
(e) Observation of transition 8.
(f) Next observation of transition 8.
(g) Observation of transition 9.
(h) Next observation of transition 9.
(i) Observation of transition 10.
(j) Next observation of transition 10.
(k) Observation of transition 11.
(l) Next observation of transition 11.

Figure C.15: Manual adversarial examples 6 through 11 where the coin has been placed above the sprite. For more details, see Section 5.6.



(a) Observation of transition 12.

(b) Next observation of transition 12.

(c) Observation of transition 13.

(d) Next observation of transition 13.

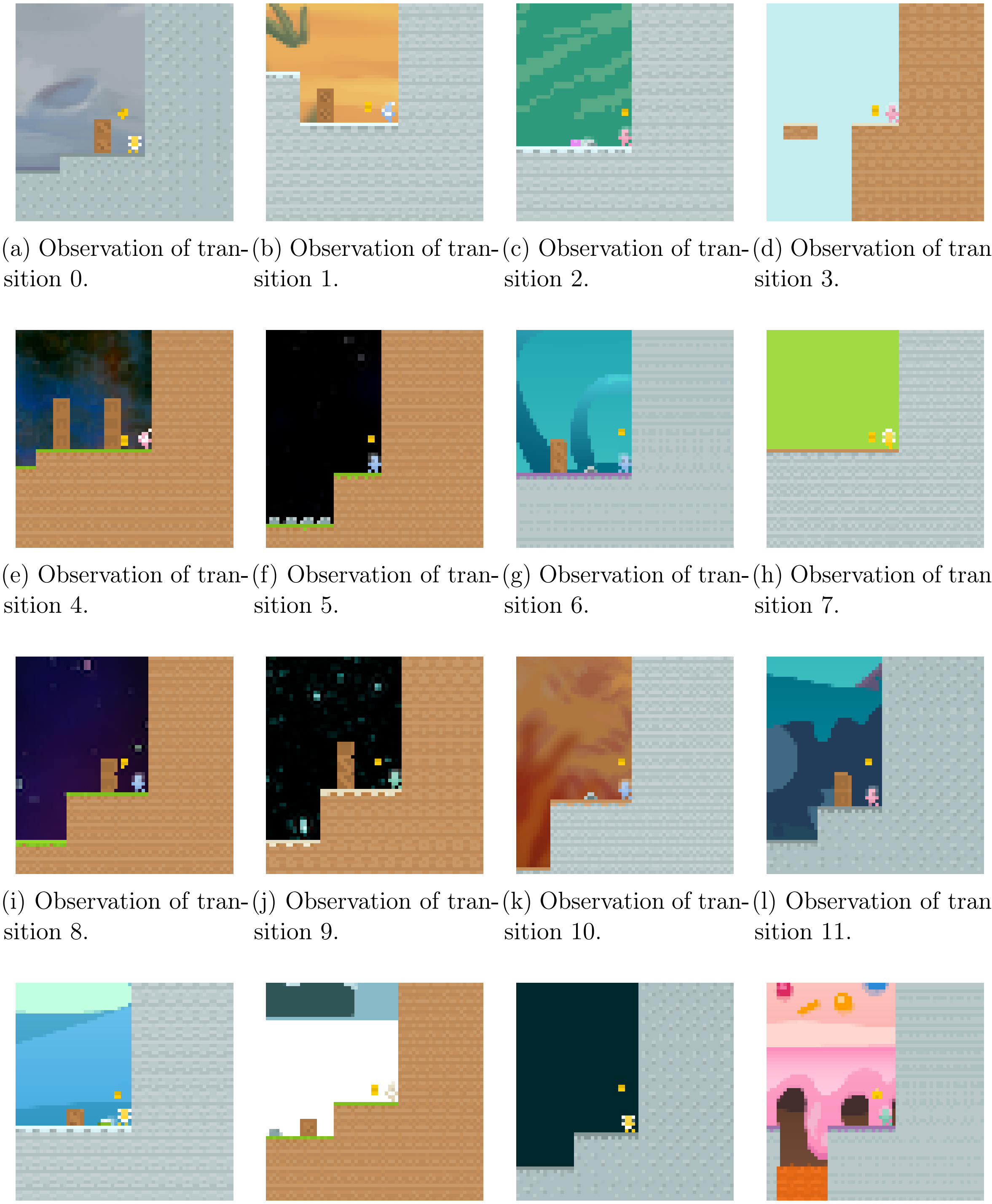
(e) Observation of transition 14.

(f) Next observation of transition 14.

(g) Observation of transition 15.

(h) Next observation of transition 15.

Figure C.16: Manual adversarial examples 6 through 11 where the coin has been placed above the sprite. For more details, see Section 5.6.



(a) Observation of transition 0.

(b) Observation of transition 1.

(c) Observation of transition 2.

(d) Observation of transition 3.

(e) Observation of transition 4.

(f) Observation of transition 5.

(g) Observation of transition 6.

(h) Observation of transition 7.

(i) Observation of transition 8.

(j) Observation of transition 9.

(k) Observation of transition 10.

(l) Observation of transition 11.

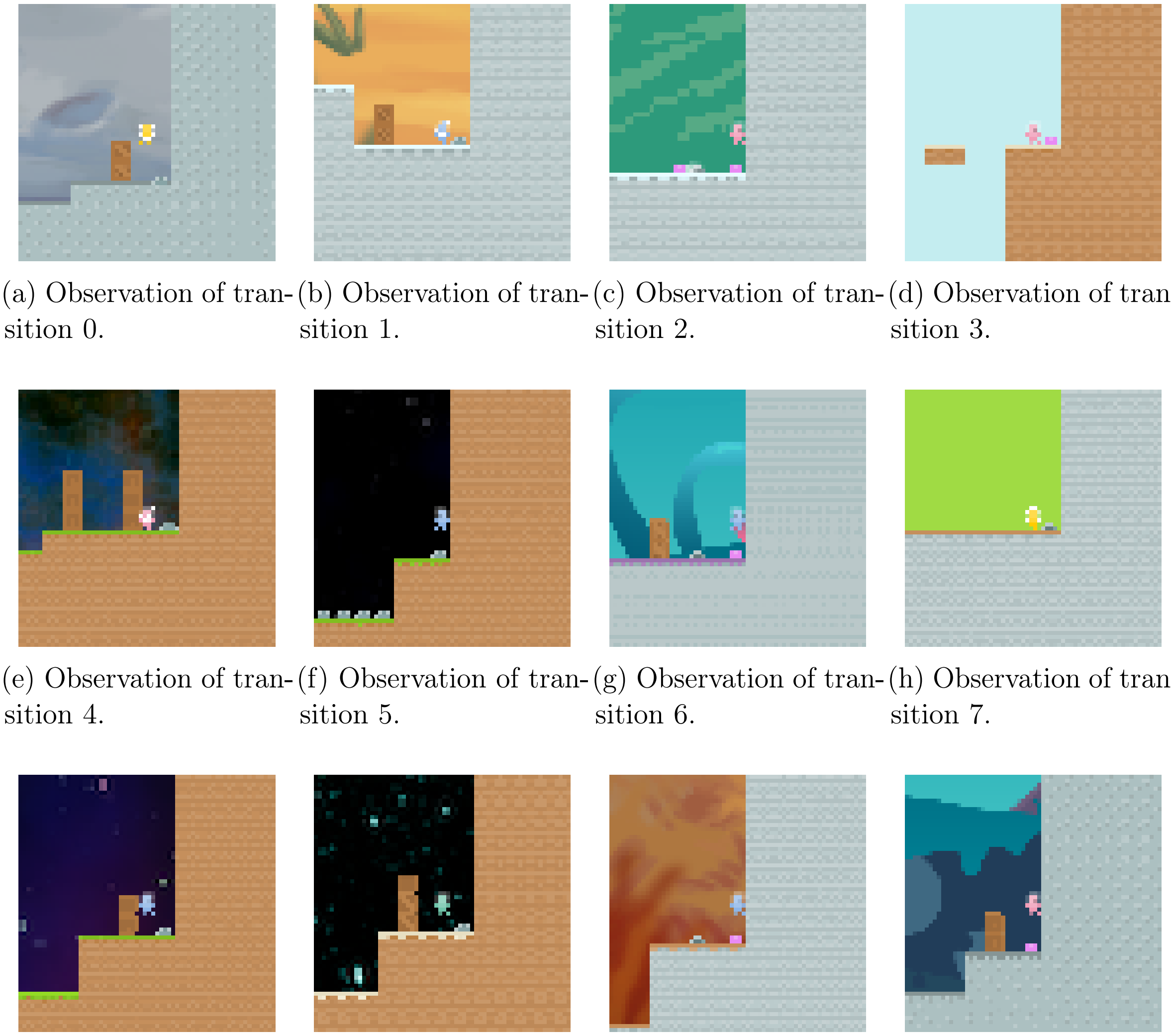
(m) Observation of transition 12.

(n) Observation of transition 13.

(o) Observation of transition 14.

(p) Observation of transition 15.

Figure C.17: Manually edited images where the coin is landing on the agent sprite.



(a) Observation of transition 0.

(b) Observation of transition 1.

(c) Observation of transition 2.

(d) Observation of transition 3.

(e) Observation of transition 4.

(f) Observation of transition 5.

(g) Observation of transition 6.

(h) Observation of transition 7.

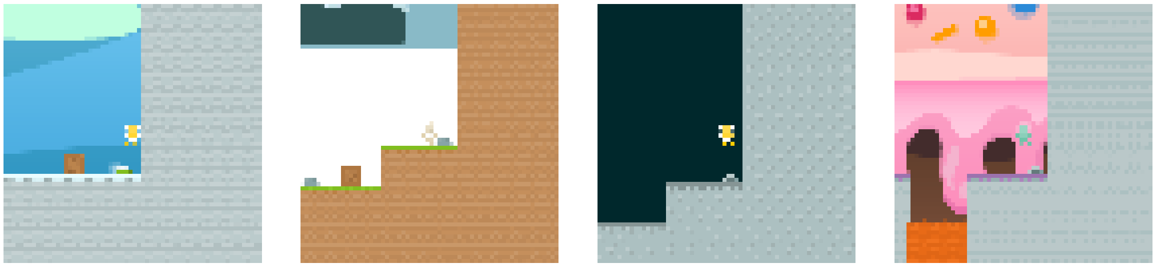
(i) Observation of transition 8.

(j) Observation of transition 9.

(k) Observation of transition 10.

(l) Observation of transition 11.

Figure C.18: Manually edited images 0 through 11 where the coin has been replaced by an enemy, such that if the agent sprite touches the enemy the game will be lost.



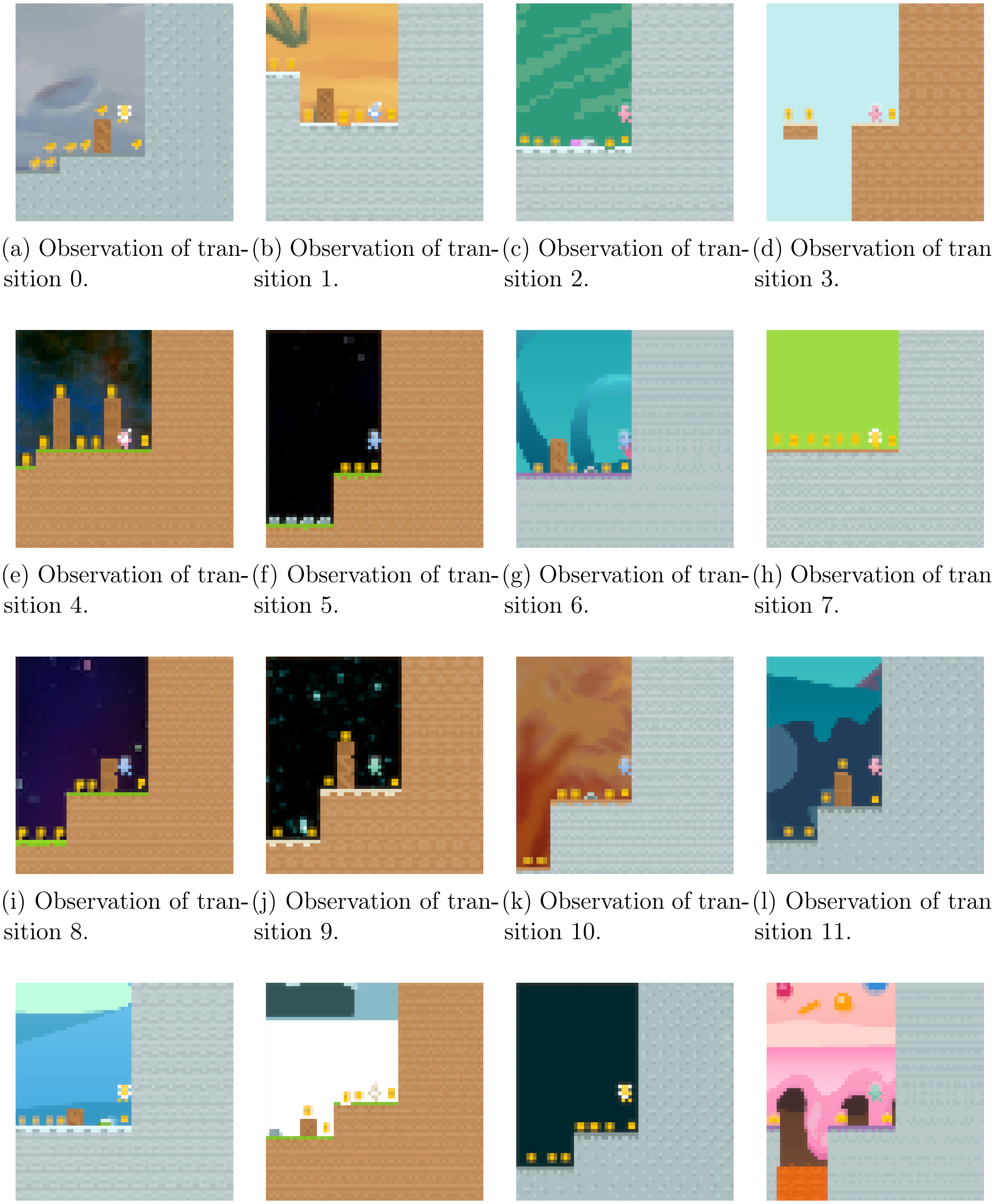
(a) Observation of transition 12.

(b) Observation of transition 13.

(c) Observation of transition 14.

(d) Observation of transition 15.

Figure C.19: Manually edited images 12 through 15 where the coin has been replaced by an enemy, such that if the agent sprite touches the enemy the game will be lost.



(a) Observation of transition 0.

(b) Observation of transition 1.

(c) Observation of transition 2.

(d) Observation of transition 3.

(e) Observation of transition 4.

(f) Observation of transition 5.

(g) Observation of transition 6.

(h) Observation of transition 7.

(i) Observation of transition 8.

(j) Observation of transition 9.

(k) Observation of transition 10.

(l) Observation of transition 11.

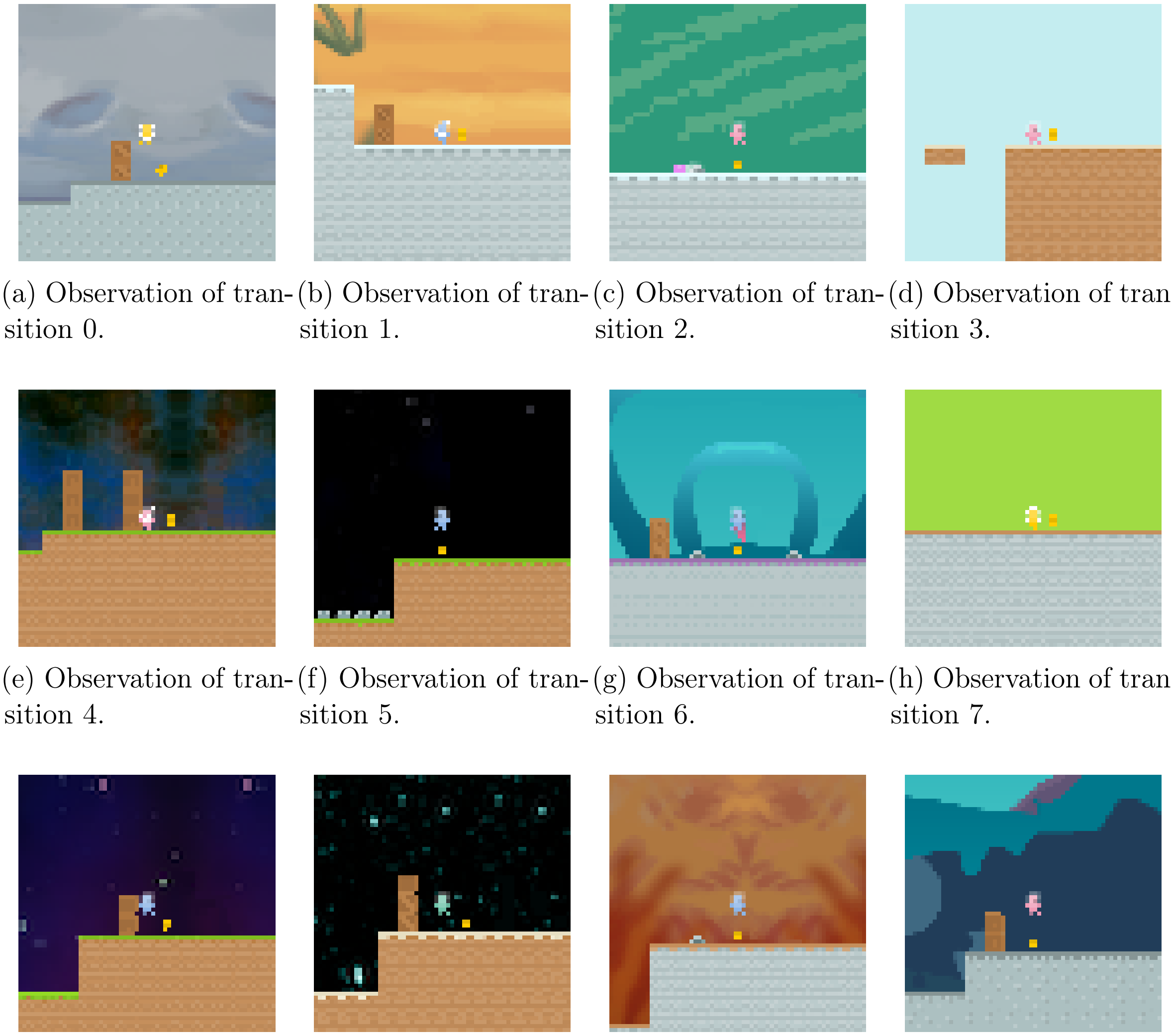
(m) Observation of transition 12.

(n) Observation of transition 13.

(o) Observation of transition 14.

(p) Observation of transition 15.

Figure C.20: Manually edited images where many coins have been placed on screen.



(a) Observation of transition 0.

(b) Observation of transition 1.

(c) Observation of transition 2.

(d) Observation of transition 3.

(e) Observation of transition 4.

(f) Observation of transition 5.

(g) Observation of transition 6.

(h) Observation of transition 7.

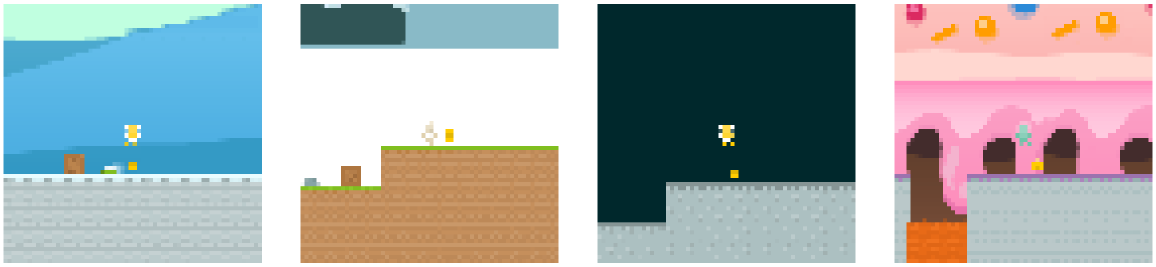
(i) Observation of transition 8.

(j) Observation of transition 9.

(k) Observation of transition 10.

(l) Observation of transition 11.

Figure C.21: Manually edited images 0 through 11 where the platform continues to the right of the coin, rather than ending with a wall.



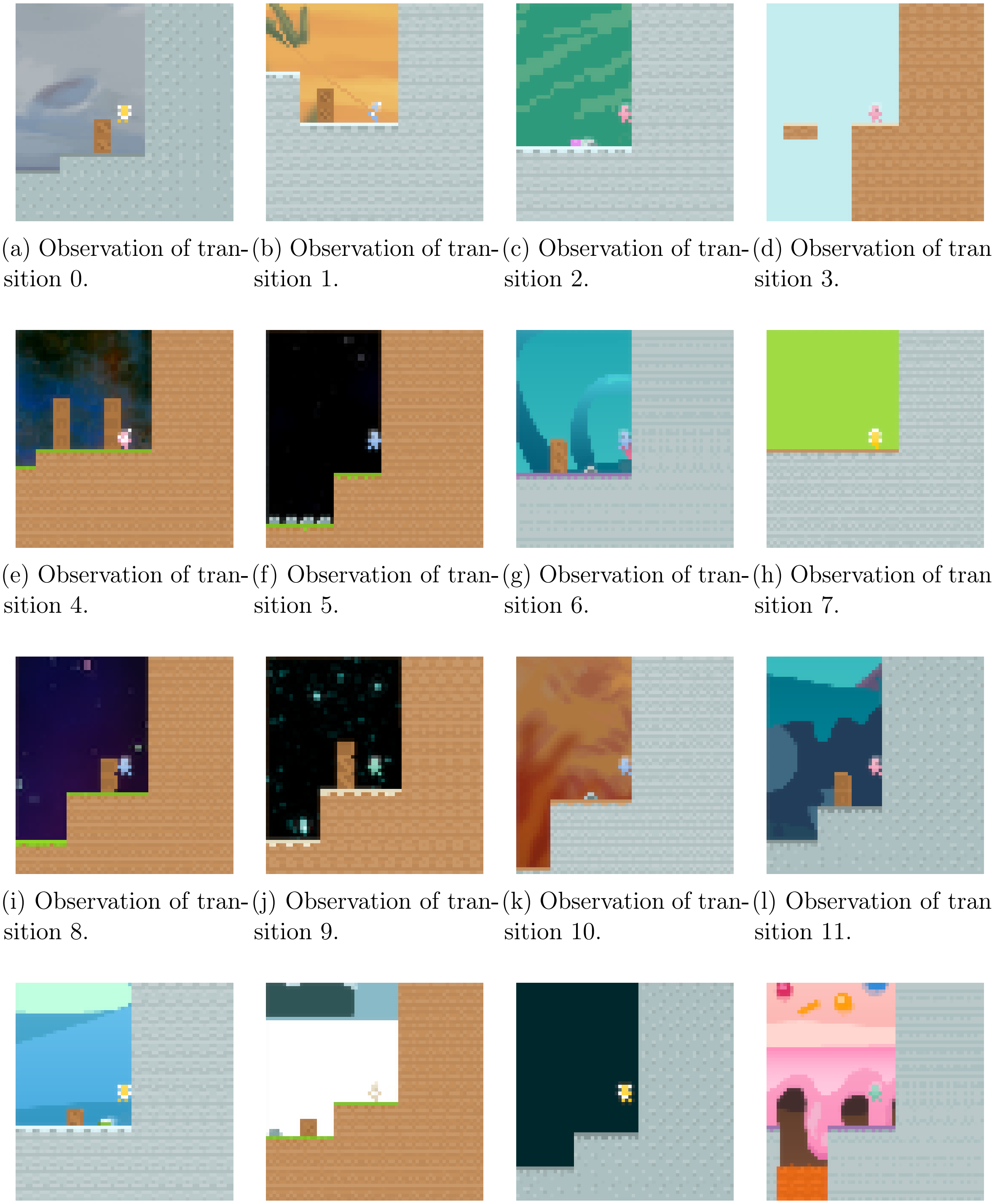
(a) Observation of transition 12.

(b) Observation of transition 13.

(c) Observation of transition 14.

(d) Observation of transition 15.

Figure C.22: Manually edited images 12 through 15 where the platform continues to the right of the coin, rather than ending with a wall.



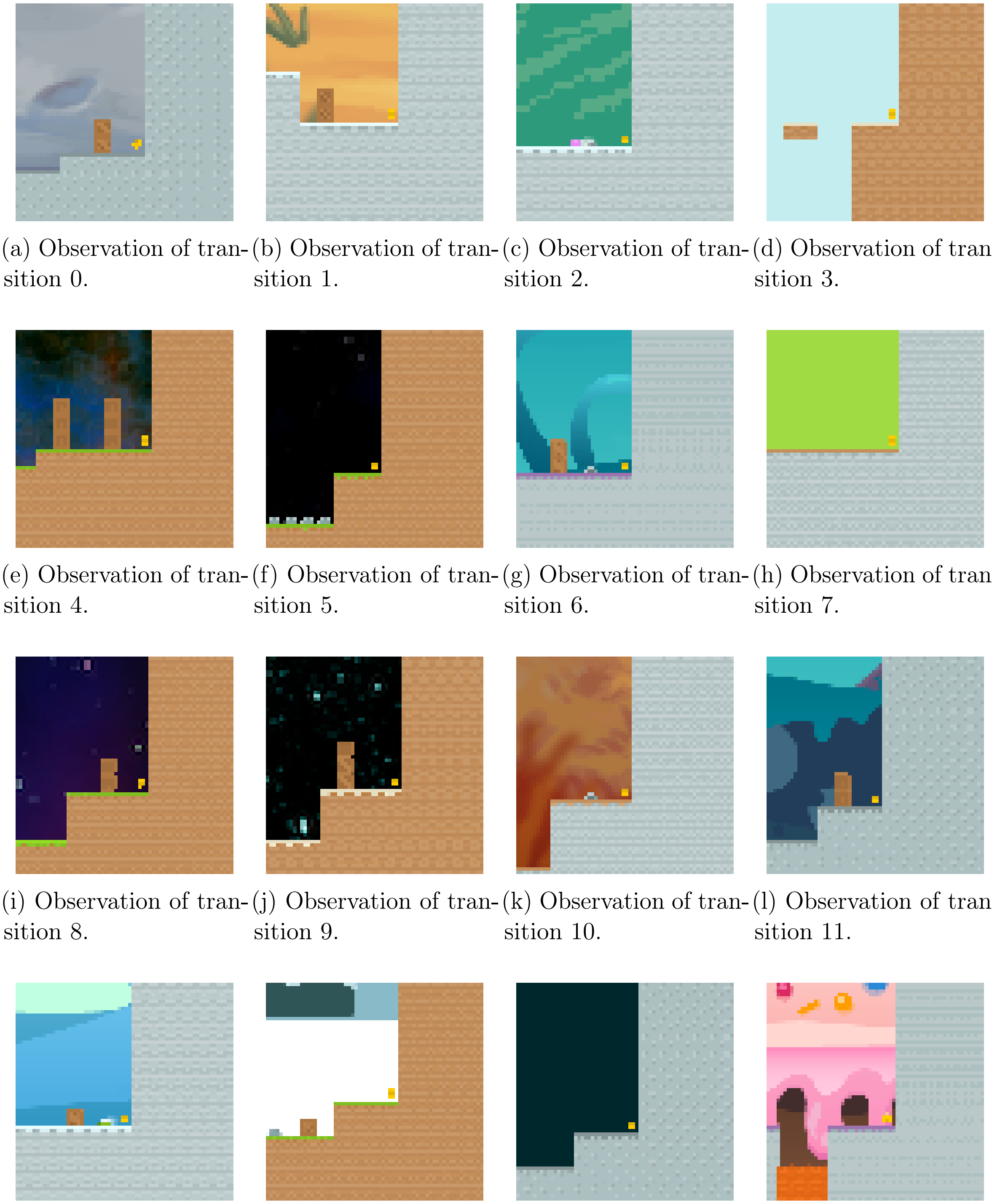
(a) Observation of transition 0.
(b) Observation of transition 1.
(c) Observation of transition 2.
(d) Observation of transition 3.

(e) Observation of transition 4.
(f) Observation of transition 5.
(g) Observation of transition 6.
(h) Observation of transition 7.

(i) Observation of transition 8.
(j) Observation of transition 9.
(k) Observation of transition 10.
(l) Observation of transition 11.

(m) Observation of transition 12.
(n) Observation of transition 13.
(o) Observation of transition 14.
(p) Observation of transition 15.

Figure C.23: Manually edited images where the coin has been removed.



(a) Observation of transition 0.

(b) Observation of transition 1.

(c) Observation of transition 2.

(d) Observation of transition 3.

(e) Observation of transition 4.

(f) Observation of transition 5.

(g) Observation of transition 6.

(h) Observation of transition 7.

(i) Observation of transition 8.

(j) Observation of transition 9.

(k) Observation of transition 10.

(l) Observation of transition 11.

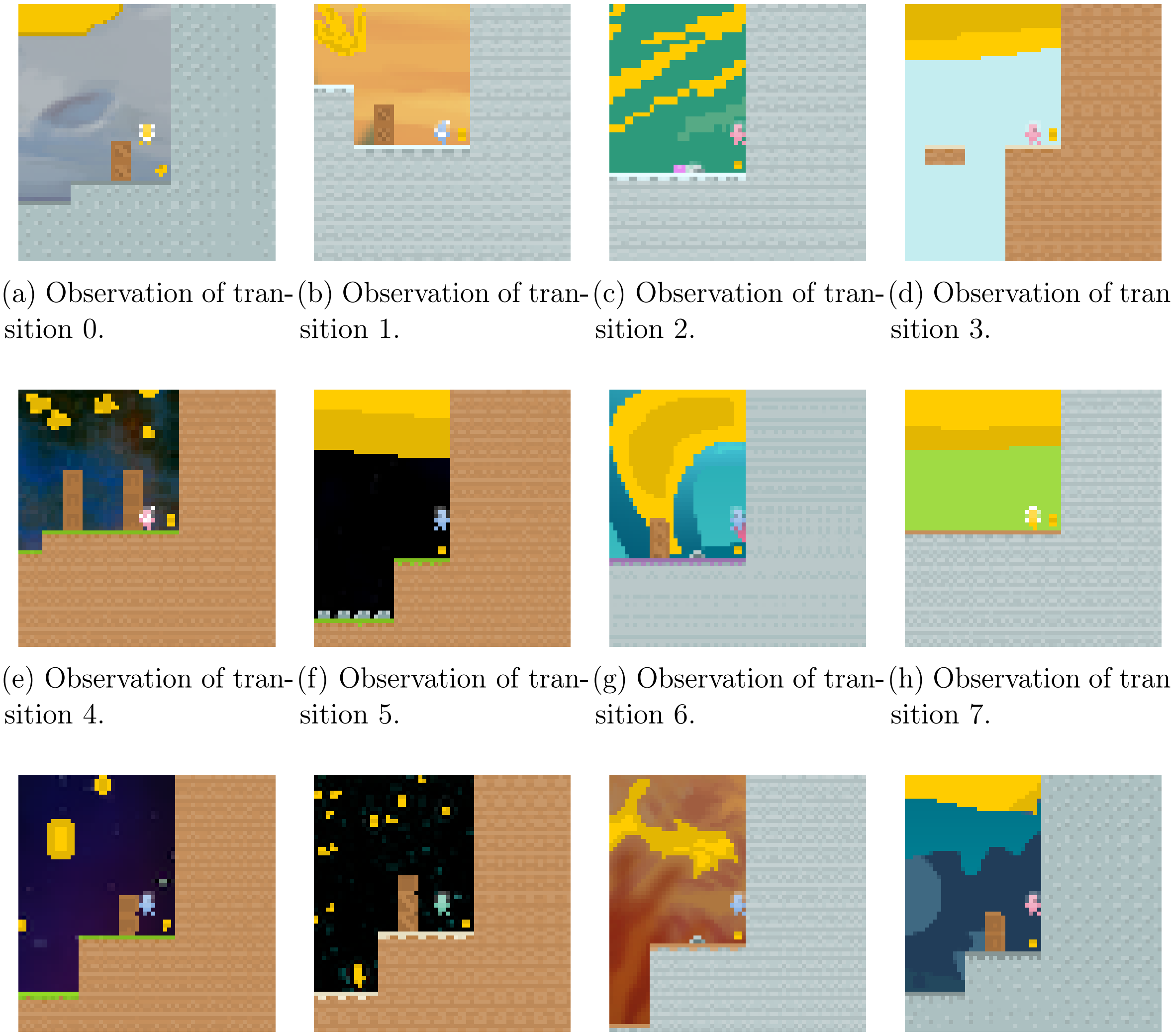
(m) Observation of transition 12.

(n) Observation of transition 13.

(o) Observation of transition 14.

(p) Observation of transition 15.

Figure C.24: Manually edited images where the agent sprite has been removed.



(a) Observation of transition 0.

(b) Observation of transition 1.

(c) Observation of transition 2.

(d) Observation of transition 3.

(e) Observation of transition 4.

(f) Observation of transition 5.

(g) Observation of transition 6.

(h) Observation of transition 7.

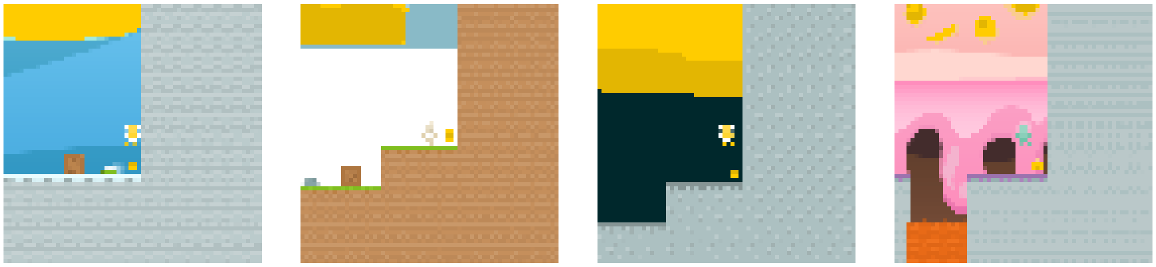
(i) Observation of transition 8.

(j) Observation of transition 9.

(k) Observation of transition 10.

(l) Observation of transition 11.

Figure C.25: Manually edited images 0 through 11 where large parts of the background have been painted in the same colours as the coin.



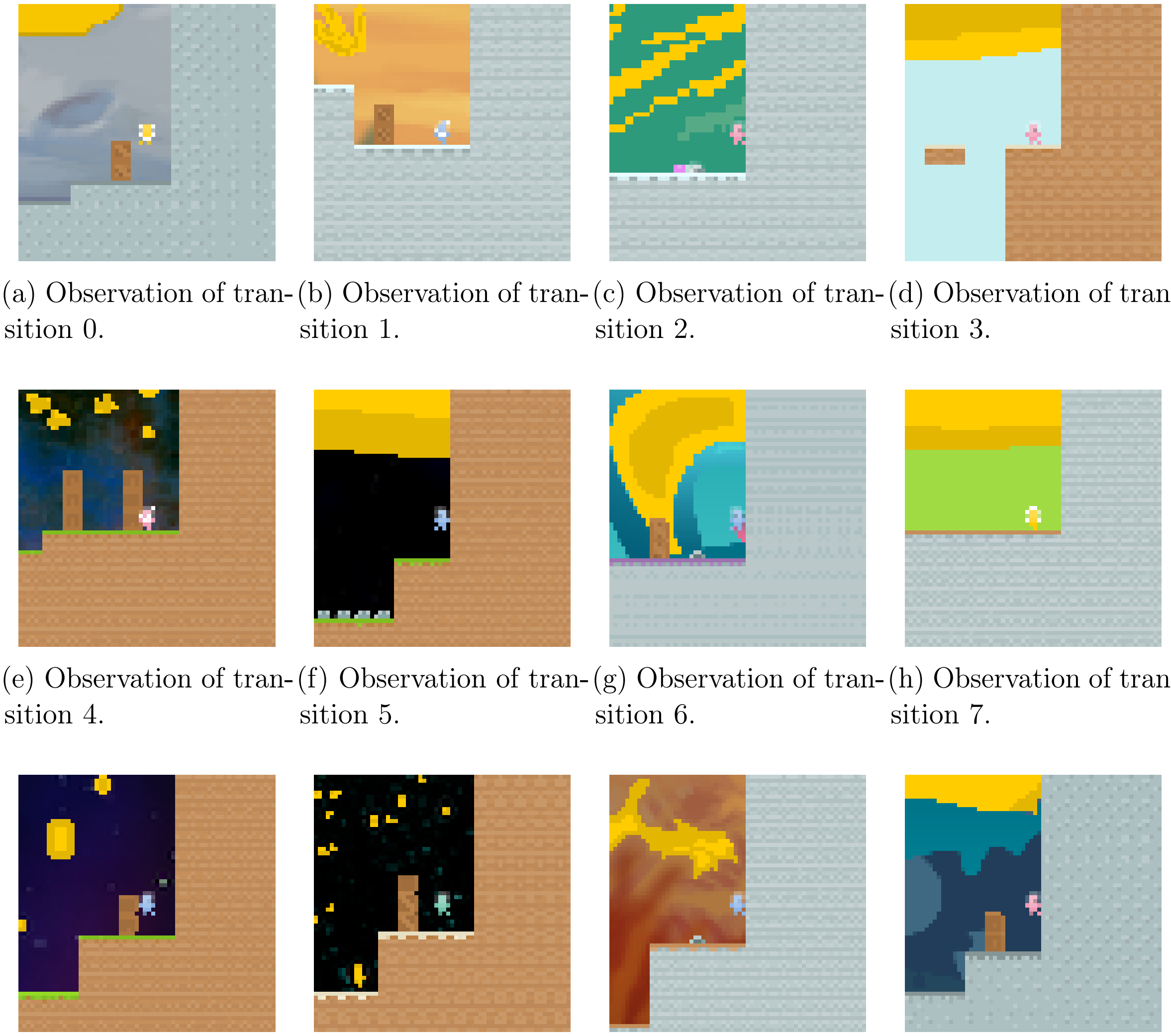
(a) Observation of transition 12.

(b) Observation of transition 13.

(c) Observation of transition 14.

(d) Observation of transition 15.

Figure C.26: Manually edited images 12 through 15 where large parts of the background have been painted in the same colours as the coin.



(a) Observation of transition 0.

(b) Observation of transition 1.

(c) Observation of transition 2.

(d) Observation of transition 3.

(e) Observation of transition 4.

(f) Observation of transition 5.

(g) Observation of transition 6.

(h) Observation of transition 7.

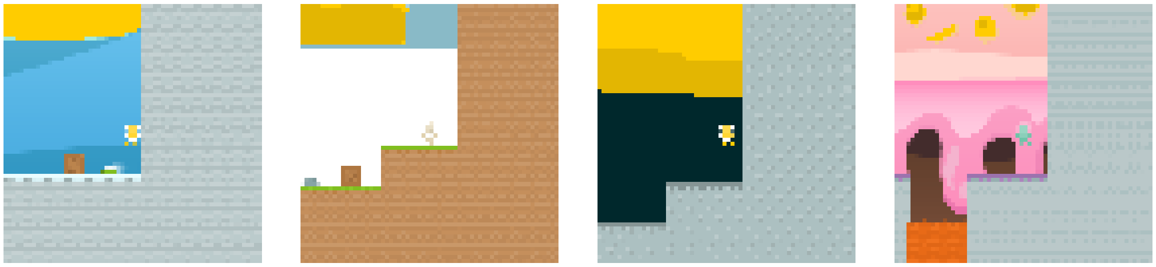
(i) Observation of transition 8.

(j) Observation of transition 9.

(k) Observation of transition 10.

(l) Observation of transition 11.

Figure C.27: Manually edited images 0 through 11 where large parts of the background have been painted in the same colours as the coin, and the coin has been removed.



(a) Observation of transition 12.

(b) Observation of transition 13.

(c) Observation of transition 14.

(d) Observation of transition 15.

Figure C.28: Manually edited images 12 through 15 where large parts of the background have been painted in the same colours as the coin, and the coin has been removed.