Automatic Discovery of Visual Circuits

Achyuta Rajaram[1,2*] **Neil Chowdhury**[2*]
Antonio Torralba[2] **Jacob Andreas**[2] **Sarah Schwettmann**[2]
[1]Phillips Exeter Academy [2]MIT CSAIL

Abstract

To date, most discoveries of subnetworks that implement human-interpretable computations in deep vision models have involved close study of single units and large amounts of human labor. We explore scalable methods for extracting the subgraph of a vision model's computational graph that underlies a particular capability. In this paper, we formulate capabilities as mappings of human-interpretable visual concepts to intermediate feature representations. We introduce a new method for identifying these subnetworks: specifying a visual concept using a few examples, and then tracing the interdependence of neuron activations across layers, or their *functional connectivity*. We find that our approach extracts circuits that causally affect model output, and that editing these circuits can defend large pretrained models from adversarial attacks.

1 Introduction

Deep neural networks extract features layer by layer, until these features lead to a prediction. In vision models, studying these features at the level of individual neurons has revealed a range of human-interpretable functions that increase in complexity in deeper layers: Gabor filters [6] in the earliest convolutional layers are followed by curve detectors [3], and later, units that activate for specific categories of objects [26, 25, 1, 15, 2, 9]. However, there are many important questions that the study of individual neurons leaves unanswered, for instance, whether one feature is used to compute another, or whether two features share a common backbone. A mechanistic understanding of these kinds of phenomena is useful for understanding how models make decisions, whether a model capability relies on some other capability, and attributing unwanted behavior to learned subcomputations. For example, learned spurious correlations between features and outputs leads to model failures such as misclassification of dermatological images containing rulers as malignant [13], classifying huskies as wolves due to the presence of snow [18], and learning gender and race prejudices from training data and applying them in inference [12]. We want to be able to intervene on the computational subgraph underlying these types of model behaviors and edit the set of features a model is using to make a decision. How can we automatically detect circuits in vision models?

Circuit discovery in vision models. Previously, circuits underlying the detection of specific concepts have been identified in vision models via manual aggregation of model weights [14]. For example, Olah et al. [14] uncover a car circuit in InceptionV1 [22] by creating feature visualizations [15] of individual units, pinpointing a car-detecting neuron in the `mixed4c` layer, and finding that the three neurons in the previous `mixed4b` layer with maximal weight magnitudes also represent car features: wheels, windows, and car bodies. While this technique can indeed recover specific algorithms encoded in the weights of trained networks, scaling circuit extraction to larger models and more complex tasks will require approaches that automatically identify both features of interest and relevant subgraphs.

*Indicates equal contribution. Address correspondence to: achyuta@mit.edu, nchow@mit.edu

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

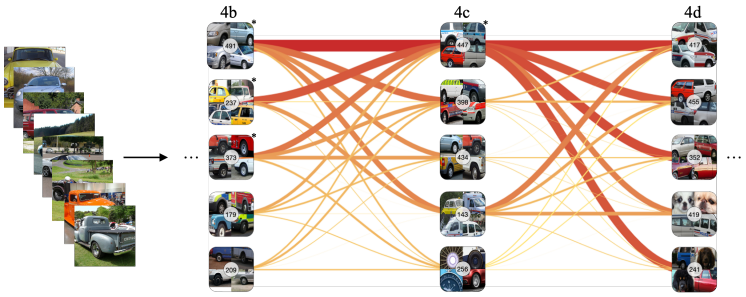

Figure 1: **Automatic discovery of the car circuit inside Inception using CLA.** The * indicates that a unit is present in the weight-based circuit discovered by Olah et al. in [14]. Maximally activating dataset exemplars are shown for each neuron. CLA recovers all units in the car circuit (units 491, 237, 373 in Layer 4b; unit 447 in Layer 4c) from [14], as well as additional car-detecting neurons in all three layers studied. Edge thickness is proportional to cross-layer attribution score.

Automated circuit extraction. Recent work investigating the subcomputations inside neural language models has focused on detection of circuits that execute specific functions, such as indirect object identification (IOI) and identifying numbers greater than an input token (Greater-Than) [24]. Approaches to automating circuit discovery include subnetwork probing, which learns a mask over model components using an objective that combines accuracy and sparsity [4], and Automatic Circuit DisCovery (ACDC), a pruning-based technique that removes edges from a computational subgraph based on their effect on the output distribution [5]. Conmy et al. [5] show that ACDC successfully recovers the entire IOI circuit in GPT2-Small. Motivated by the promise of automated approaches in language models, our work explores the extension of scalable circuit detection to the visual domain.

This paper introduces a method for automatically identifying circuits in vision networks based on *functional connectivity* of neurons between layers, or the interdependence of their activations in response to a particular input distribution. We define a circuit as a computational subgraph of a trained network that derives information from input features to construct an intermediate representation that later affects the output distribution. First, a concept (e.g. "car") is specified in a set of input images containing that concept. Next, we introduce a new algorithm based on Cross-Layer Attribution (CLA) that iteratively refines circuit subgraphs based on attribution scores calculated between units in successive layers (Section 2.1).

To evaluate intermediate concept representation and computation in CLA circuits, we construct CatFish (3.1), a dataset with ground truth visual feature hierarchy. Intervention experiments on an InceptionV1 model finetuned on CatFish find that CLA automatically discovers circuits corresponding to intermediate concepts, and recovers known compositional relationships from CatFish within the model (Section 3). We additionally apply our method to defend CLIP from text-based adversarial attacks using circuit interventions (Section 4).

2 Methods

2.1 Circuit extraction based on Cross-Layer Attribution (CLA)

There are many possible ways to define circuits. A purely structural notion of connectivity focuses on edges (weights) between learned features, such as in [14]. To identify the circuit responsible for performing an arbitrary computation (e.g. detecting a concept) specified in model input, we use an attribution-based notion of functional connectivity. Most existing attribution methods either explain how internal features are derived from inputs [27, 20], or determine which features induce a distribution over outputs [16, 19, 5]. We instead perform attribution *between* internal layers, and iteratively refine a *functional connectivity graph* by computing the set of features in a given layer (e.g. l_i) that maximally affect downstream features (in l_{i+1}). Experiments in Section 3 compare CLA to alternative circuit-discovery methods including the weight-based approach from [14].

Algorithm 1 describes CLA. First, we compute attribution scores between neurons in subsequent layers. The score for a pair of neurons $(m \in l_i, n \in l_{i+1})$ takes into account both relevance and influence, by multiplying together two terms: one corresponding to the magnitude of the activations, and one corresponding to the gradient of activations across layers. By computing

attribution scores across all pairs of neurons, we create an input-distribution-dependent notion of cross-layer connectivity, termed an attribution matrix. Next, we build each circuit layer-by-layer using the attribution matrix, selecting a subset of k neurons in each layer to maximize the sum of their attribution to previous and subsequent layers.

We show that CLA recovers existing manually-identified circuits in the literature, such as the "car detector" in InceptionV1 [14]. To detect the car circuit, we run CLA (for $k = 5$) on 100 car images sampled uniformly from the ImageNet validation classes: `cab`, `minivan`, `pickup`, and `sports car`. Figure 1 shows the expanded car circuit, which not only recovers all neurons in the original, but surfaces additional units that are also all selective for car features.

Algorithm 1 Cross-Layer Attribution (CLA) computes a model subgraph corresponding to an input distribution in two parts. First, we compute Attribution Matrices, which map the functional connectivity between neurons in a set of layers. Then, we use iterative refinement to find a highly connected subgraph for an input distribution.

procedure ATTRIBUTIONMATRIX(model, layer l_i, layer l_{i+1})
  for each input x **do**
    $a_i \leftarrow \text{model}(x).l_i$ ▷ Get l_i activations on image x
    for each neuron $m \in l_i$, $n \in l_{i+1}$ **do**
      $\text{attr}[x, m, n] \leftarrow |a_{i,m}| \cdot \frac{\partial \|l_{i+1,n}(a_i)\|_2}{\partial a_{i,m}}$ ▷ Attribute l_i neurons to l_{i+1} neurons
    end for
  end for
  return $\text{mean}(\text{attr}, dim = 0)$ ▷ Compute mean attribution across images
end procedure

procedure BUILDCIRCUIT(model, layers $\{l_i\}$, sizes $\{k_i\}$)
  for each i in 1 to $L - 1$ **do**
    $\text{attrs}[l_i] \leftarrow$ ATTRIBUTIONMATRIX(model, l_i, l_{i+1})
  end for
  $\text{circuit}[l_L] \leftarrow (\text{top } k_L \text{ argmax}) \text{ attrs}[l_{L-1}, :, n]$ ▷ Initialize circuit with top neurons
  for each i in $L - 1$ to 1 **do**
    $\text{circuit}[l_i] \leftarrow (\text{top } k_i \text{ argmax})_m \text{ sum}(\text{attrs}[l_i, m, n_i], n_i \in \text{circuit}[l_{i+1}])$
  end for
  while circuit $\neq$ last circuit **do** ▷ Iteratively refine circuit
    for i = $L - 1$ to 1 **do**
      $\text{circuit}[l_i] \leftarrow (\text{top } k_i \text{ argmax})_m \text{ sum}(\text{attrs}[l_i, m, n_i], n_i \in \text{circuit}[l_{i+1}])$
    end for
    for i = 1 to L **do**
      $\text{circuit}[l_i] \leftarrow (\text{top } k_i \text{ argmax})_n \text{ sum}(\text{attrs}[l_{i-1}, m_i, n], m_i \in \text{circuit}[l_{i-1}])$
    end for
  end while
  return circuit
end procedure

2.2 Intervention analysis of vision models

We find that models can be steered by specifically intervening on the circuits extracted for particular concepts. To cause the model to fail to represent a specific visual concept, we inhibit the circuit extracted using an input distribution specified by the concept, by *edge pruning*: corrupting (zeroing) all paths between the first and second layers of the circuit (Algorithm 3). To do this, we (i) run the forward pass, saving the ground truth activations in the layers of interest (ii) zero the activations of the top layer of neurons in the circuit (iii) run a model in this corrupted form (iv) overwrite the activations of all neurons in the second layer outside of the circuit with their ground truth from (i), creating new outputs. By doing this, we prevent information from flowing through the circuit, while leaving paths outside unaffected. Additionally, we hypothesize that models form higher-level concepts by composing circuits representing lower-level concepts. To test this, we propose a method to remove a circuit entirely from a model by applying the method in Algorithm 3 to prune edges between all consecutive layers in a circuit (Algorithm 2: *circuit pruning*).

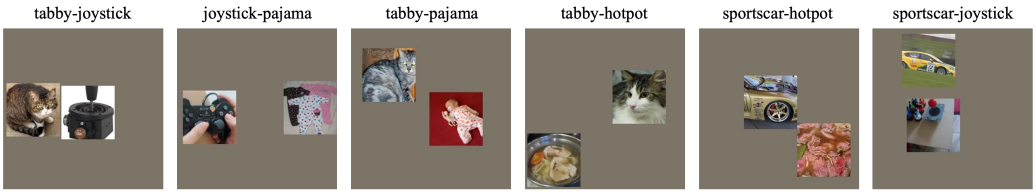

Figure 2: **CatFish dataset examples.** Each CatFish class composes images from two ImageNet classes. CatFish contains 20 composed classes sampled from 10 ImageNet classes.

3 Compositional circuits in Inception-CatFish

3.1 Constructing a dataset with visual feature hierarchy (CatFish)

We seek to test if CLA can recover circuits in models performing composition, where outputs require known operations on intermediate concepts. We construct the CatFish dataset to induce neural circuitry for intermediate concept detection, by sharing intermediate concepts across outputs. Specifically, we use 20 labels, each corresponding to 2 of 10 unique ImageNet classes (such as `tabby` and `pajama`). Composed images (e.g. `tabby-pajama`) are constructed by sampling images from two distinct subclasses and placing them on a neutral background. These subclasses form "intermediate concepts" that are used for final predictions; detecting any of the paired class labels at the output layer would require the detection of subclasses in an intermediate layer. Labeled samples from this dataset are shown in Figure 2. More details on dataset construction are in Appendix C. We fine-tuned InceptionV1 to perform CatFish classification, and use CLA to recover circuits corresponding to subclass categories. Additional details on model training are provided in Appendix B.

3.2 CLA identifies intermediate circuits

To find circuits inside Inception-CatFish which detect intermediate concepts, we run CLA on 25 input images per subclass, generated from pairs of samples of the *same* subclass (e.g. `tabby-tabby`). We focus on the final three layers of Inception-CatFish, where Network Dissection [1] reveals more interpretable features corresponding to specific concepts at the individual neuron level. We construct several baselines to benchmark against: randomly selected neurons in each layer, top neurons per layer by activation magnitude on sample inputs, and top weight magnitudes [14]. We aggregate class prediction accuracy after edge-pruning from all classes into two groups, a "positive" set which contains the subclass, and a "negative" set that does not. Figure 3c illustrates the hypothesized effect of subconcept pruning on positive and negative classes. We use a variety of circuit sizes, denoted by a k parameter corresponding to the number of neurons per layer. We ensure that the same amount of neurons are selected per layer.

As seen in Figure 3d, both random neurons and circuits selected by weight magnitude have negligible effect on classification performance. Pruning maximally activated neurons, and neurons chosen by CLA both have a significant effect on output predictions, with CLA having greater effect. From Figure 3e, we observe small increases in accuracy on negative classes for both CLA and maximally activated neurons, corresponding to the suppression of incorrect predictions in the positive classes. Furthermore, we compare the sets of neurons between CLA and maximally activated neurons with Intersection-over-Union (IoU), plotted in Figure 3b. We see high divergence across these sets of neurons, indicating the presence of neurons with large activations on specific inputs, but low effect on model outputs. Using the gradient term of CLA scores, we directly select against such neurons, choosing those with large impact on downstream model representations.

3.3 Logit inspection reveals global concept ablation

How do we further verify that we are probing and removing a subclass, without any effects on the rest of the model? What does Inception-CatFish "see" when run on an image from a class containing an ablated subclass? To answer such questions, we directly inspect post-ablation model logits across classes. Specifically, for each subclass, we plot model logits before and after edge pruning in Figure 3f. Before pruning, the model correctly identifies the target class. After performing edge pruning of a subclass, the probability mass is split roughly equally between all classes containing the complement of the pruned subclass in the target class. For example, when we prune the `sportscar` circuit

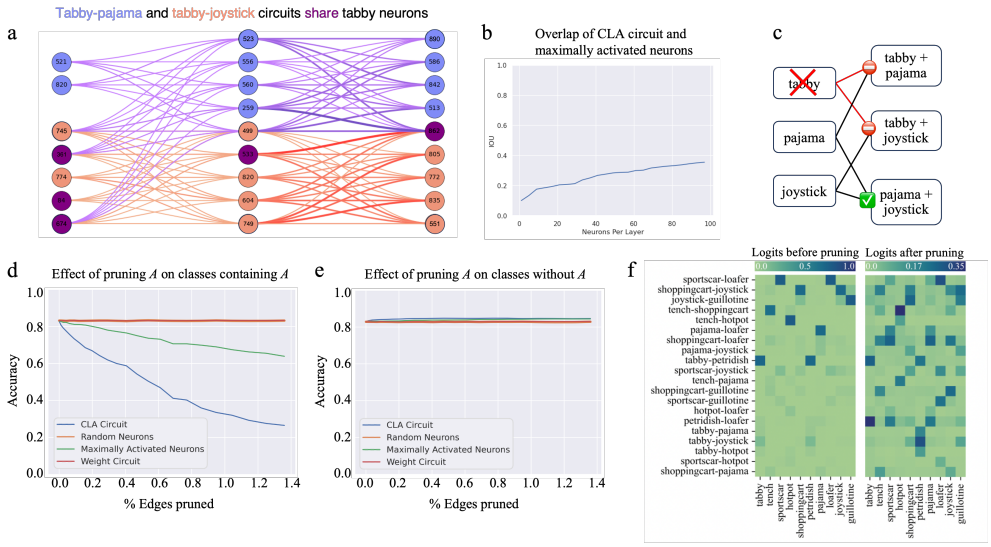

Figure 3: **Intervening on feature composition in Inception-CatFish by circuit pruning. (a)** CLA-generated circuits for the tabby-joystick (red) and tabby-pajama (blue) classes. Neurons in both circuits (purple) correspond to the shared concept of tabby (see Figure A6 for visualization). **(b)** IoU across neurons selected by CLA, and the maximally activated neurons for a given subclass. **(c)** Predicted impact of subclass knockout. Eliminating a subclass through edge pruning will only affect class outputs containing that subclass, with no effect on other subclasses. **(d)** Model accuracy on positive classes (those containing the target subclass) for various baselines, across various circuit sizes. **(e)** Model accuracy on negative classes. **(f)** Inception-CatFish prediction logits before and after pruning each subclass, on images sampled from a class containing the subclass.

and input `sportscar-loafer` images, Inception-CatFish is roughly equally likely to predict any loafer-containing class (e.g. `sportscar-loafer, pajama-loafer, shoppingcart-loafer`).

3.4 Paired concept circuits

CLA sheds light on the composition of circuits, through identifying circuits for output classes. We can construct circuits corresponding to CatFish output classes (e.g. `tabby-pajama`) that contain neurons from two sub-classes (see Figure 3a, figure A6 for visualizations). Using IoU [10] to compare neurons, we show that the shared concept circuits from CLA are built from individual concept circuits in Figure A4. The high overlap between the subclass circuits and their corresponding shared concept circuits (e.g. [`tabby` neurons $\cup$ `pajama` neurons] $\cap$ `tabby-pajama` neurons) suggests that subclass circuits compose to form paired class circuits.

4 Subcircuit pruning defends CLIP from text-based adversarial attacks

A potential use case of model editing is the defense of models from spurious correlations and unwanted features in data. Previous work has found that large multimodal models such as CLIP are vulnerable to adversarial attacks from natural images containing text conflicting with image content [7]. For example, an apple with a paper saying "iPod" taped onto it might be misclassified as an iPod. To prevent such adversarial attacks, we propose performing interventions based on the underlying decision-making pipeline of the model, as discovered by circuit analysis. Previous approaches to defending such attacks include MILAN [9], wherein all units that appear to recognize text in ImageNet validation images are ablated. Our more efficient method only requires inference on a small sample of 50 images and performs small interventions that are minimally destructive to model performance.

4.1 Traffic light dataset for benchmarking textual defense

To benchmark the performance of circuit interventions, we propose an example scenario: the case of using CLIP to label traffic lights based on their color (red or green), in the presence of potential

text-based adversarial attacks. We construct a dataset of 50 training images for circuit identification and 100 testing images for circuit validation for three classes of images: (i) red/green traffic lights (found using CLIP retrieval of "red traffic light" or "green traffic light" on LAION-5B) (ii) ImageNet validation images with instances of "red traffic light" or "green traffic light" overlaid with random font size, color, and position, and (iii) holdout adversarial images with red/green traffic lights with overlaid text indicating the opposite color. Figure 4 shows several example images from the dataset.

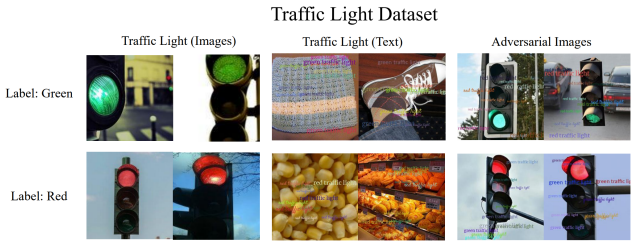

Figure 4: Samples from the Traffic Light dataset, including real traffic light images, ImageNet with overlaid traffic light text, and adversarial text-attacked images.

4.2 Model intervention protects CLIP from adversarial attacks

Fundamentally, the main "traffic light" classification circuit in CLIP is composed of at least two subcircuits: an image detector that classifies real traffic lights, and a text detector that detects and classifies text. We find the text detector automatically using CLA, and then use circuit pruning (Algorithm 2) to remove the text detector from the model. We perform a sweep of CLIP layer (2, 3, or 4) and the width of the text circuit (k); results are shown in Figure 5b. On the full test set, the accuracy of CLIP on adversarial images improves from 3% to 87% while pruning only 6% of edges in layer 3. Thus, the intervention successfully defends against text-based adversarial attacks. Examples of neurons for each circuit are shown in Figure 5c.

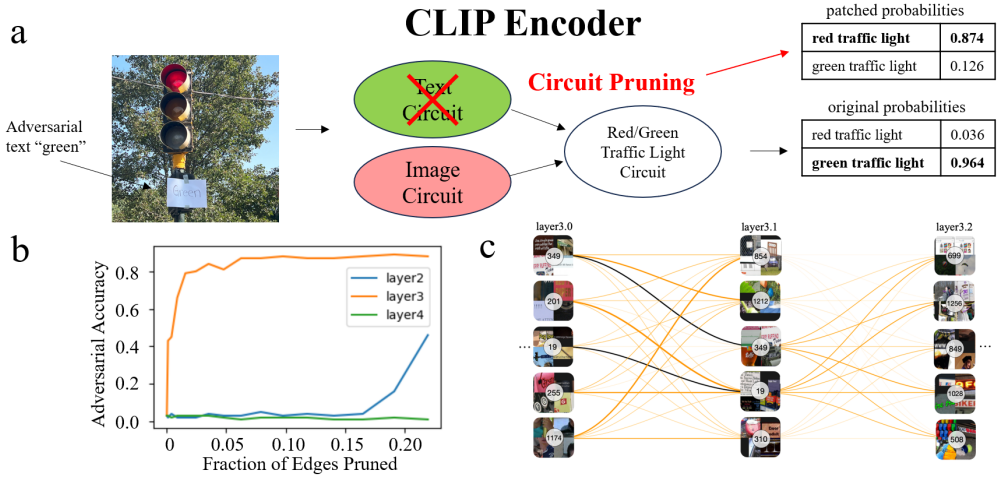

Figure 5: **Intervening on CLIP to prevent text-based adversarial attacks. (a)** Schematic of the intervention showing that pruning the text circuit defends CLIP from a real-world adversarial attack. **(b)** Adversarial accuracy as a function of circuit width when intervening on three CLIP layers **(c)** CLA text circuit for layer 3. Residual connections between equivalent channels are shown in black.

5 Discussion

We introduce a method for automatically discovering circuits in vision networks. We perform experiments on Inception-CatFish, a model with induced visual feature hierarchy, and CLIP, a multimodal foundation model. CLA identifies subgraphs for computation over intermediate concepts. The primary limitation of CLA is its strict topological restriction on circuit shape. Future work should allow for automatic selection of different numbers of neurons per layer.

6 Acknowledgements

We are grateful for the support of the MIT-IBM Watson AI Lab, and ARL grant W911NF-18-2-0218. We thank David Bau, Tamar Rott Shaham, and Tazo Chowdhury for their useful input and insightful discussions.

References

[1] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

[2] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 2020.

[3] Nick Cammarata, Gabriel Goh, Shan Carter, Ludwig Schubert, Michael Petrov, and Chris Olah. Curve detectors. *Distill*, 2020. https://distill.pub/2020/circuits/curve-detectors.

[4] Steven Cao, Victor Sanh, and Alexander M Rush. Low-complexity probing via finding subnetworks. *arXiv preprint arXiv:2104.03514*, 2021.

[5] Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability, 2023.

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

[7] Gabriel Goh, Nick Cammarata †, Chelsea Voss †, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 2021. https://distill.pub/2021/multimodal-neurons.

[8] Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. Localizing model behavior with path patching, 2023.

[9] Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. Natural language descriptions of deep visual features. In *International Conference on Learning Representations*, 2022.

[10] Paul Jaccard. The distribution of the flora in the alpine zone. *The New Phytologist*, 11(2):37–50, 1912.

[11] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[12] Cristina Manresa-Yee and Silvia Ramis. Assessing gender bias in predictive algorithms using explainable ai, 2022.

[13] Akhila Narla, Brett Kuprel, Kavita Sarin, Roberto Novoa, and Justin Ko. Automated classification of skin lesions: From pixels to practice. *Journal of Investigative Dermatology*, 138(10):2108–2110, 2018.

[14] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. https://distill.pub/2020/circuits/zoom-in.

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

[16] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[18] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[19] Sarah Schwettmann, Neil Chowdhury, Samuel Klein, David Bau, and Antonio Torralba. Multimodal neurons in pretrained text-only transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2862–2867, 2023.

[20] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[21] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[24] Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small, 2022.

[25] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[26] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

[27] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

Appendix

A Automatic circuit discovery in pretrained InceptionV1

We test CLA-based circuit discovery in an InceptionV1 model [23] trained on ImageNet, and show evidence of in-the-wild ablation of concepts. We use input images sampled across four car-containing ImageNet classes: "cab," "minivan," "pickup," and "sports car." We compute CLA across layers 4b,4c, and 4d of the model, to test the ability of an attribution-based approach to automatically recover the circuits manually found in [14]. We perform Path Patching as outlined in [8] through every path that passes through the circuit, replacing the uncorrupted input activations with those from a randomly chosen image (drawn from the "banana" class). We use KL divergence [11] as a measure of effect on model output, taken against the ground truth outputs, over a set of images drawn from the "sports car" ImageNet class. We compare our approach against circuits derived from weight magnitudes, output attribution [16] to select neurons relevant to the car-containing output classes, and randomly chosen neurons. Results, visualized in Figure A1, show evidence of concept deletion, and greater effect on model output compared to other methods.

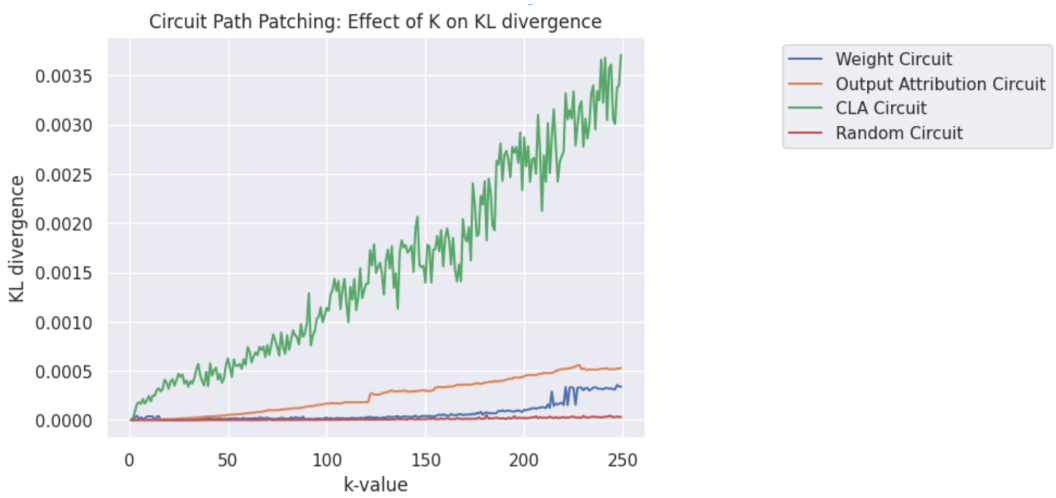

Figure A1: **CLA circuit maximally affects model performance.** CLA discovers a unified "car circuit," with large effects on model output on sports-car images.

B Inception-CatFish training details

We train the Inception-Catfish model on the CatFish dataset to form classification with a cross-entropy objective. We finetune a model with the InceptionV1 architecture [23] pretrained to perform the ImageNet Classification task. Finetuning was performed in PyTorch using data-parallelism across 6 RTX 3090s [17]. For finetuning, a hyperparameter search was performed, over choices of learning rate, batch size, and LR decay. We use an SGD optimizer with learning rate 0.01, and momentum set to 0.9 [21]. We add a fixed learning rate schedule, specifically a LR decay of 0.5% per epoch, and train until model convergence. A batch size of 512 was chosen, in order to maximize training speed. We preprocess images from CatFish through normalization to match the mean and standard deviations of ImageNet.

C CatFish dataset

We create 20 pairs from 10 classes that were hand selected in order to ensure semantic differences. From each pair of classes, we generate images using the following methodology: first, select an image from each class. Next, rescale each image into 100x100 "patches." Finally, place both patches on a 300x300 background, colored with the mean color of ImageNet images, ensuring there there is no overlap. Figure A4 contains example images drawn from the CatFish dataset. With each of our 20

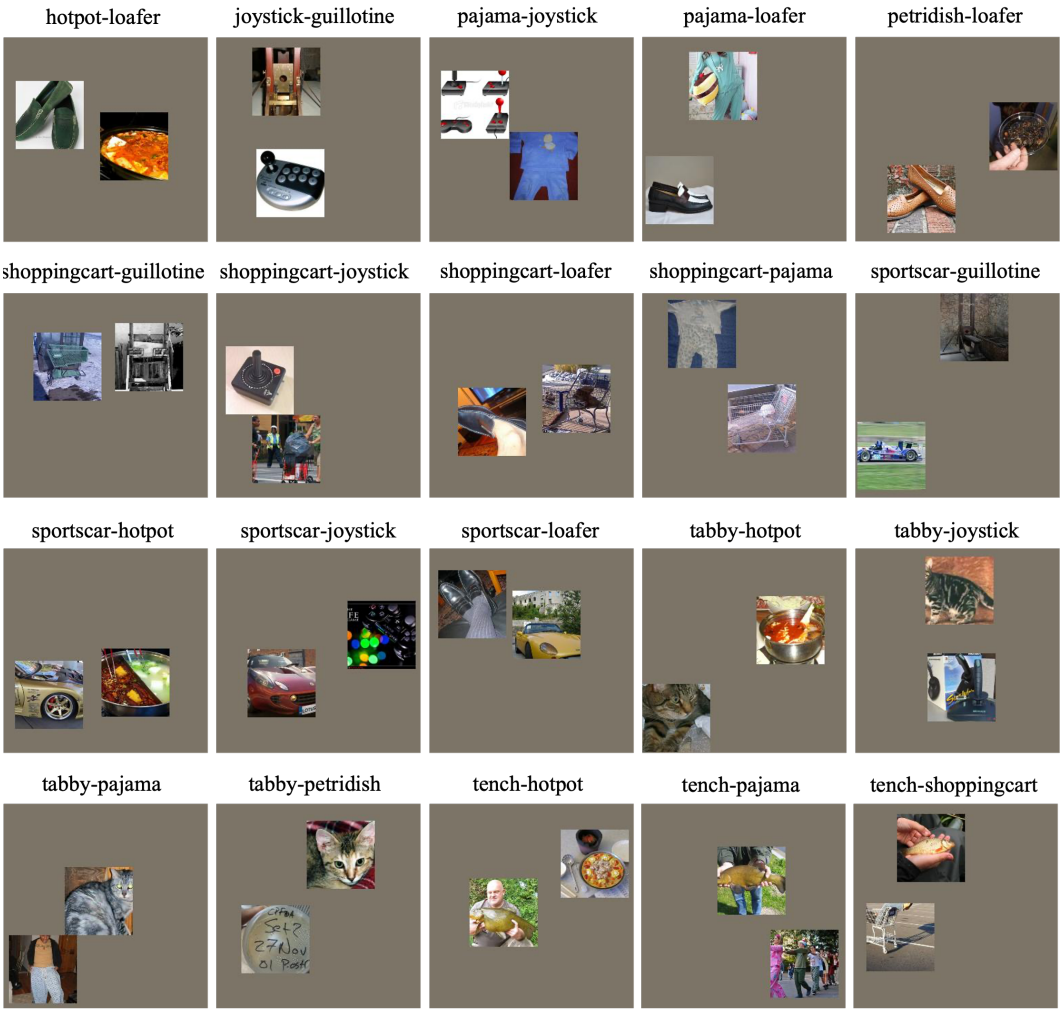

Figure A2: **Catfish dataset.** Each Catfish class composes images from two ImageNet classes. Catfish contains 20 composed classes sampled from 10 ImageNet classes.

classes, we generate 3000 images per class, for a total of 60,000 training images. For validation and testing, we generate 150 unique images per class, for 3000 validation and test set images. We ensure that these datasets use different sets of source images, in order to prevent data leakage.

D CLA circuits are stable across values of k

Throughout this work, we generate circuits using different choices of k, a parameter denoting the number of neurons per layer to be used in iterative refinement, which produces an ordered list of k neurons for each layer. We show that this process of regenerating circuits is equivalent to setting a high k value and choosing the top N neurons per layer. We compute the IOU between the sets of neurons derived from consecutive circuits across all sizes. Figure A3 shows that circuits derived from CLA are "stable", and do not depend much on the value of k, except for very low values. In all cases, IOU of consecutive circuits never drops below 0.85, indicating very high overlap.

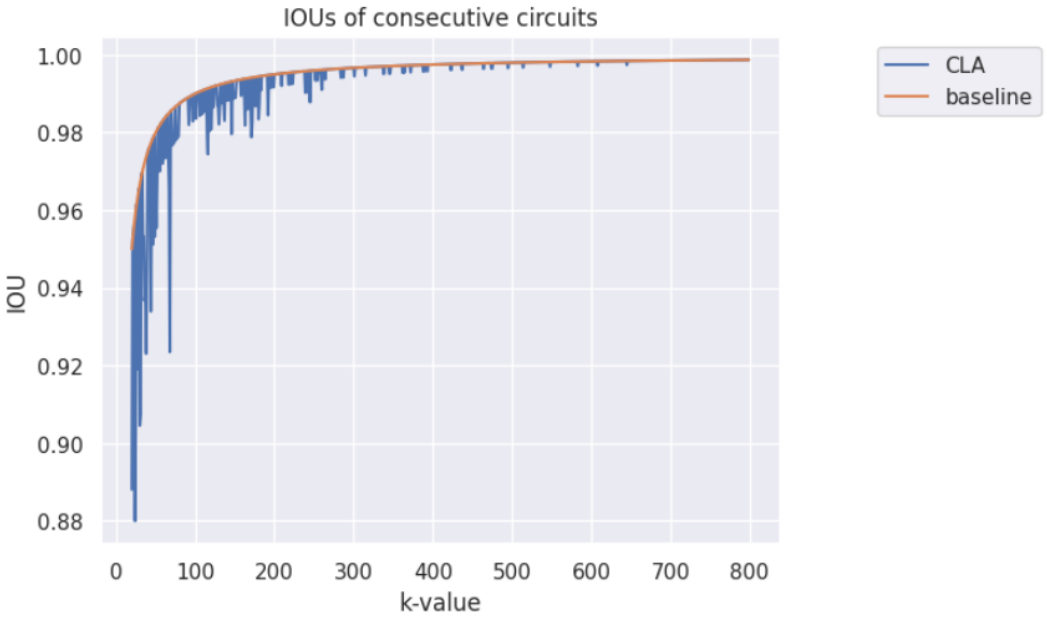

Figure A3: **Consecutive circuit overlap.** CLA derives circuits that contain similar neurons, independent of the choice of k.

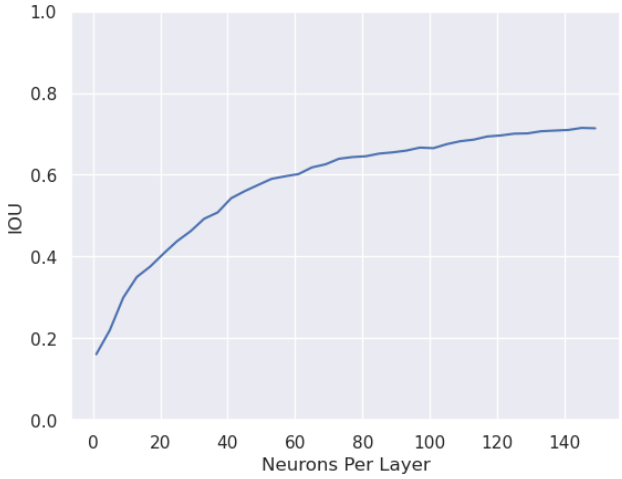

Figure A4: **Overlap between output class and intermediate subclass circuits:** Aggregated across all classes, the IoU between a class circuit with k neurons per layer, and the union of the corresponding two subclass circuits shows high set overlap.

E Partial concept removal

We use logit inspection in order to illustrate concept removal. We find with ablation of smaller circuits, we see partial concept ablation, creating "interpolation" between the unmodified model, and one without knowledge of the ablation. Results of this as a function of circuit size, represented by the k value denoting neurons per layer, is shown in Figure A5. We prune the `tabby` circuit and inspect model output logits on all CatFish classes containing `tabby`.

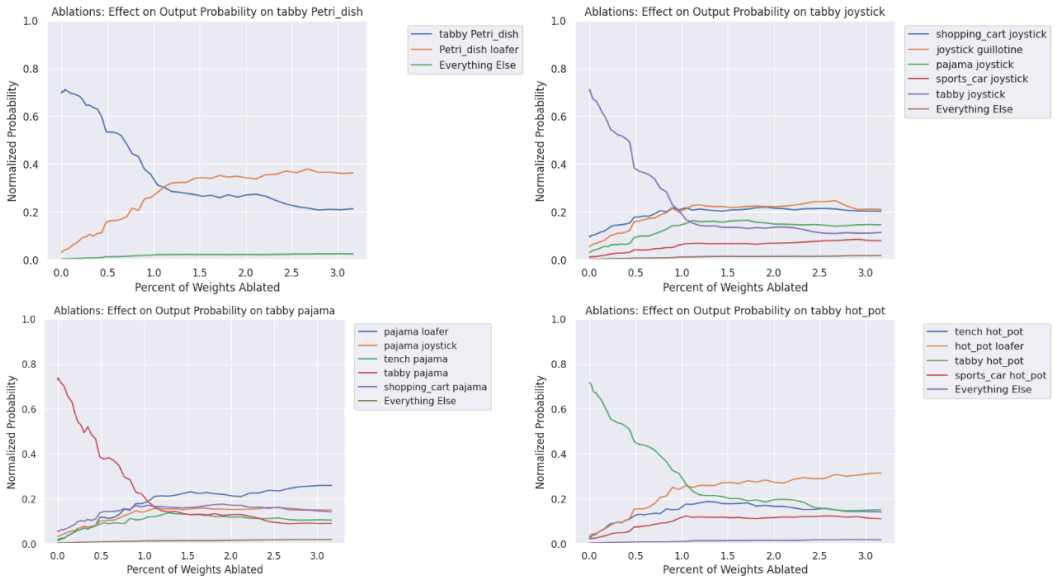

Figure A5: **Partial concept ablation.** Varying circuit size, we show that partial intermediate concept ablation is achievable.

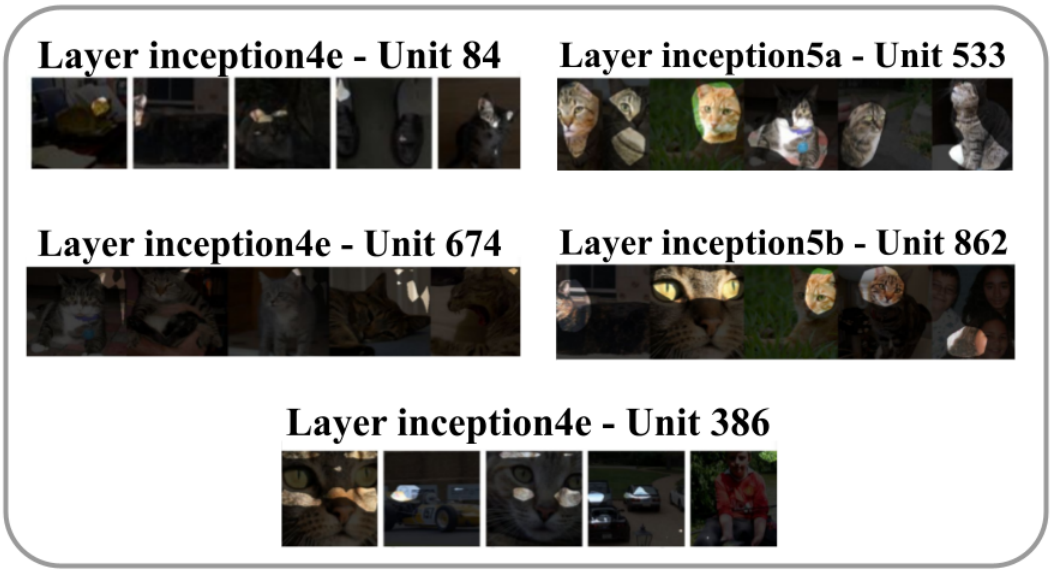

Figure A6: **Shared tabby neurons.** Computing the overlap between the tabby-pajama and tabby-joystick circuits for k=5 yields 5 neurons across 3 layers. Visualizing the top 5 network dissection examplars for each neurons shows that they correspond to the shared "tabby" concept.

Algorithm 2 Circuit Pruning removes all neurons in a circuit, cutting it off from the rest of the model. This is achieved by repeating the edge ablation process across all layers, and all edges.

procedure CIRCUITPRUNE(model, input, circuit, layers $\{l_i\}$)
    clean_activations ← model(image) ▷ Get clean activations from model
     return model(input) with intervention for all l:
    for layer l_i in $\{l_i\}$ **do**
        activations = model.l_i(activations[l_{i-1}])
        for each channel c in layer l_i **do**
            if channel $c \in$ circuit **then**
                activations[l_i, c] = 0 ▷ Ablate neurons in circuit
            else if channel $c \notin$ circuit **then**
                activations[l_i, c] = clean_activations[l_i, c] ▷ Preserve neurons outside circuit
            end if
        end for
        model.l_i = activations ▷ Set newly patched activations
    end for
end procedure

Algorithm 3 Edge Pruning inhibits a circuit by pruning the edge connections between the first and second layers of the circuit. This prevents information flow through the circuit.

procedure EDGEPRUNE(model, input, circuit, layers $\{l_1, l_2\}$)
    clean_activations ← model(image) ▷ Get clean activations from model
     return model(input) with intervention for all l:
    activations[l_1] = model.l_1(input)
    for each channel c in layer l_1 **do**
        if channel $c \in$ circuit **then**
            activations[l_1, c] = 0 ▷ Ablate neurons in circuit
        end if
    end for
    activations[l_2] = model.l_2(activations)
    for each channel c in layer l_2 **do**
        if channel $c \notin$ circuit **then**
            activations[l_2, c] = clean_activations[l_2, c] ▷ Preserve neurons not in circuit
        end if
    end for
     return model.output(activations) ▷ Proceed from layer l_2
end procedure