Introduction to Vision Interpretability

Séminaire Turing 2023

Contents

Written by Jeanne Salle and Charbel-Raphaël Segerie

1 Introduction

1.1 What is interpretability?

Mechanistic interpretability is the study of reverse-engineering neural networks to get an understanding of their internal functioning. Deep neural networks are often referred to as "black boxes" because their inner mechanisms are complex and difficult to grasp. However, in the last few years there has been a growing interest in developing methods to enhance our comprehension of these models.

1.2 Why is interpretability useful for AI safety ?

Mechanistic interpretability of neural networks is useful for several reasons. Some classic hopes for intepretability include [17] [11]:

1. Intepretability helps **diagnose errors and biases** in models, and identify parts of the network that are not functioning as intended or are learning undesirable patterns. For alignment research, it could be used to **analyse why a model gave a misaligned answer** and understand how to avoid such failure cases. More generally, it helps build **trust and transparency** in neural networks decision-making process. From a governance perspective, the availability of advanced interpretability tools could facilitate cooperation among different actors by enabling mutual understanding and interpretation of each other's systems,

2. Researchers do not actually understand the internal mechanisms of their models, current ML research mainly makes progress by repeated trial-and-error. One could imagine ML research switching towards a paradigm where model understanding is seen as a way to make progress and **build models that are more interpretable *by design***. For example, interpretability techniques could be used as part of the objective during training, to incentivize models to be transparent,

3. It may enable to better **extrapolate from current systems to future systems** and anticipate how they will likely change with scale, or predict discontinuities in capabilities and during training,

4. It offers **empirical evidence that either supports or refutes threat models**. For instance, identifying instances of internal misalignment could potentially make it more persuasive to convince individuals about the existence of existential threats.

However, it is important to note that these expectations are still subject to debate. The adoption of interpretability tools in real-world scenarios is still limited, and assessing their quality and utility remains challenging. Many existing approaches to interpretability are not designed for large-scale use and/or often demand significant human effort and time [22].

1.3 Overview of the course content

- **Feature visualization** introduces a range of techniques for visualizing the internal representations learned by networks. *Atlas activation* expands these tools to make them more human-scale. Using them, *The Circuits thread* explores how networks build up representations of high-level features out of lower-level features.

- **Attribution methods** are a set of methods that provide visual explanations for CNN decisions. In this course, we focus on the Grad-CAM algorithm and *The Building Blocks of Interpretability* which combine different methods to build sophisticated interpretability interfaces. *Understanding RL Vision* then use them to observe RL models learned features and explain in depth their failure cases.

- Finally, we present some **automatic interpretability** techniques, such as *Network dissection* and further works along this line, which depend less on subjective human judgments.

2 Convolutional Neural Networks

In this part, we introduce convolutional neural networks. If you are already familiar with this architecture, you can skip this part. However, I still recommend reading Section 2.3, which clarifies important vocabulary terms that we will use throughout the course.

2.1 Architecture

Convolutional neural networks (CNNs or ConvNets) are a widely used type of deep learning architecture in computer vision applications.

A CNN is composed of stacked layers that progressively transform an input volume (such as an image or batch of images) into an output volume, often representing a probability distribution over image classes for tasks like image classification. The layers situated between the input and output layers are referred to as hidden layers. As the input passes through a the successive layers, it is processed into a set of feature maps. CNNs generally consist of two main types of layers: convolutional layers and pooling layers.

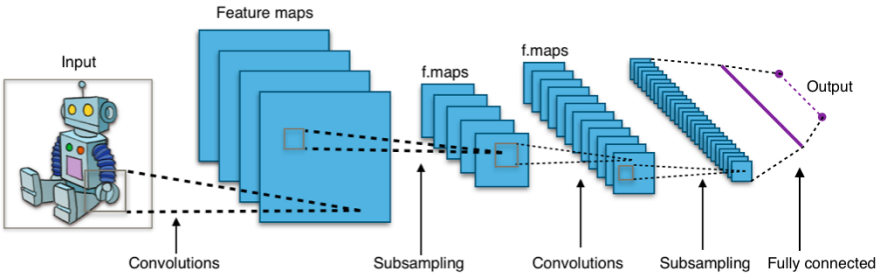


Figure 1: A typical CNN architecture. The input to a CNN is commonly a batch of images, which is represented as a tensor with dimensions of (number of images in the batch, number of channels in each image, height of each image, width of each image). Hidden layers are highlighted in blue. The initial hidden layer consists of 4 feature maps (illustrated as stacked on top of each other), followed by the second hidden layer with 5 feature maps, and so on. Image height and width often tend to diminish along the net, while the number of feature maps increases. The last layer of CNNs is commonly a fully connected layer, which is equivalent to a multilayer perceptron (MLP). It is typically employed to convert the last feature maps into a probability distribution over classes [3].

2.2 Convolutional layer

The convolutional layer is the fundamental building block of a CNN. It consists of a collection of filters, also known as kernels, which convolve over the input and transmit the results to the next layer.

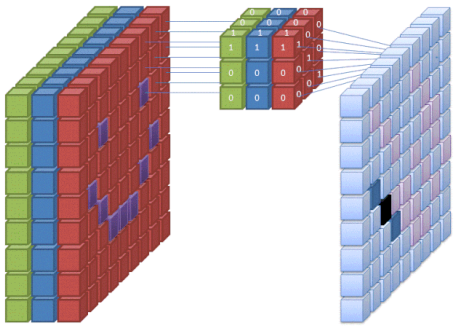


Figure 2: An example of a convolutional layer, consisting of a single kernel. 1. The input volume, which is an image of size (9x9) with three color channels (green, blue, and red), is shown on the left. 2. The middle cube represents the kernel or filter, which has the same number of channels as the input volume. In this case, the chosen kernel size is (3x3). The white numbers represent the learnable weights or parameters of the kernel. 3. On the right is the output feature map. **Note that the number of output feature maps corresponds to the number of kernels**. Each pixel in the output volume is obtained by performing an element-wise product between the input volume and the kernel weights (this operation is known as convolution). The convolutional kernel slides across the spatial dimensions (height and width) of the input volume, generating the entire output feature map, which subsequently becomes the input for the next convolutional layer [8].

The number of kernels and their sizes are fixed before training, while the weights of the kernels are learned parameters of the network.

In computer vision, multilayer perceptrons (MLPs) were previously used, but they only allowed for learning global motifs as all neurons in one layer were connected to all neurons in the next layer. In contrast, convolutional layers in CNNs enable the learning of local motifs by enforcing sparse local connectivity, where each neuron is connected only to a small region of the input volume.

A convolutional layer is defined by four hyperparameters:

- Kernel size (all kernels in a layer are of the same size),
- Number of kernels,
- Stride, which is the spatial distance between successive convolution windows. A stride of S means the kernel is translated S pixels at a time, with a greater stride resulting in smaller overlap between two successive receptive fields,
- Padding size, which is the number of zero pixels added to the border of the image before applying convolution. Padding is often used to preserve the spatial size of the volume.

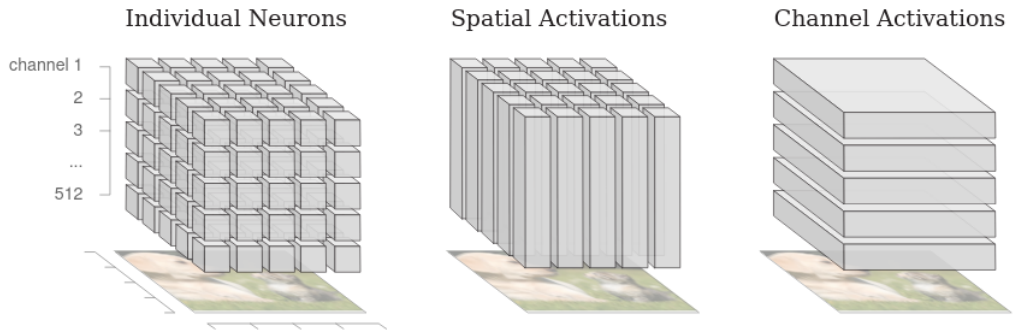
2.3 Vocabulary

Throughout the course, we will interchangeably use the terms **feature map**, **channel**, **activation map**, (and **color**, for the input layer).

A **neuron** will refer to a pixel in one of the feature maps.

A **unit** will refer either to an individual neuron, a feature map, an entire layer or the final class probability in classification.

The activation of a neuron in a specific feature map corresponds to the pixel intensity of that neuron. We will interchangeably use the terms activates and responds to denote the activation of a neuron in response to an input.



The cube of activations that a neural network for computer vision develops at each hidden layer. Different slices of the cube allow us to target the activations of individual neurons, spatial positions, or channels.

Figure 3: A cube of activations refers to a collection of feature maps that belong to the same convolutional layer. It can be thought of as a three-dimensional cube, where each cell is an activation. x- and y- axes indicate the spatial positions in the input image, and the z-axis is the channel. Activations can be considered at different scales. We will see more on this later on [19].

Activations and weights

A common confusion is sometimes made between network activations and kernel weights. Weights are fixed after training and are not modified by the input to the model. On the other hand, activations of layer n are the result of a linear combination of activations from layer $n-1$ and the weights connecting the two layers.

Before reading on, make sure that these terms and distinctions are well understood.

3 Feature Visualization

3.1 Introduction

There are two primary techniques in vision interpretability: **feature visualization** and **pixel attribution**. A fundamental paper in vision, *Feature Visualization* [18], conducted by Chris Olah's team at Google in 2017, focuses on the first approach.

Feature visualization is a technique that aims to understand and interpret the behavior of individual neurons in CNNs. It involves creating or adjusting input images in a way that maximizes the activation of particular neurons. The underlying idea is that patterns that strongly elicit a response from the unit can provide insights into the function (or feature) of this particular unit. By visualizing the images that activate individual neurons, feature visualization helps understand the representations learned by CNNs and uncover their inner workings.

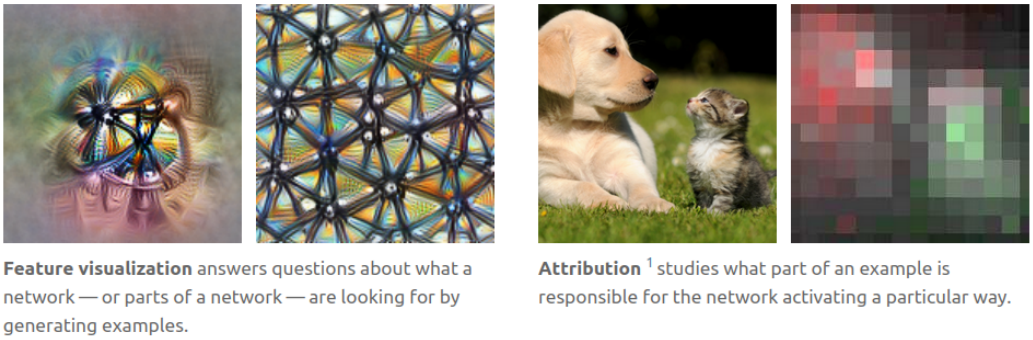


Figure 4: Two main techniques in vision interpretability. We will discuss attribution later on (see section 6) [18].

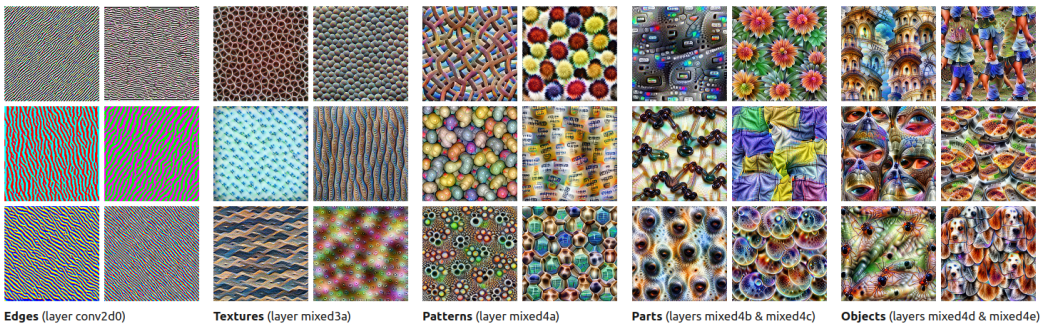


Figure 5: Some examples of feature visualizations on GoogLeNet (also known as InceptionV1) trained on ImageNet. The network progressively builds up its understanding of images over many layers, and each of them responds to a certain type of information such as a particular texture, pattern, object, etc. [18].

3.2 Feature visualization basics

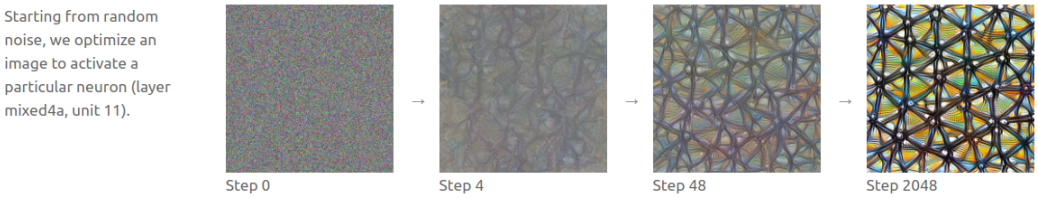


Figure 6: The feature visualization process: starting from a random image (Step 0), and through successive optimization steps, an image which highly activates a given unit in a network can be generated (Step 2048). The optimized image illustrates the representation learned by GoogLeNet at layer mixed4a, channel 11 [18].

Generating feature visualizations can be seen as an optimization problem:

Feature visualization by optimization

Let's denote θ the networks parameters, $h_{ij}(\theta, x)$ the activation of unit i in layer j, producing a feature visualization is equivalent to finding an image x such that (s.t.),

$$x = \arg\max_{x} h_{ij}(\theta, x)$$

A local maximum can be found by gradient ascent in the input space [10].

In the above problem we're looking at the representation encoded in a given neuron i from a given layer j, but different optimization objectives are possible:

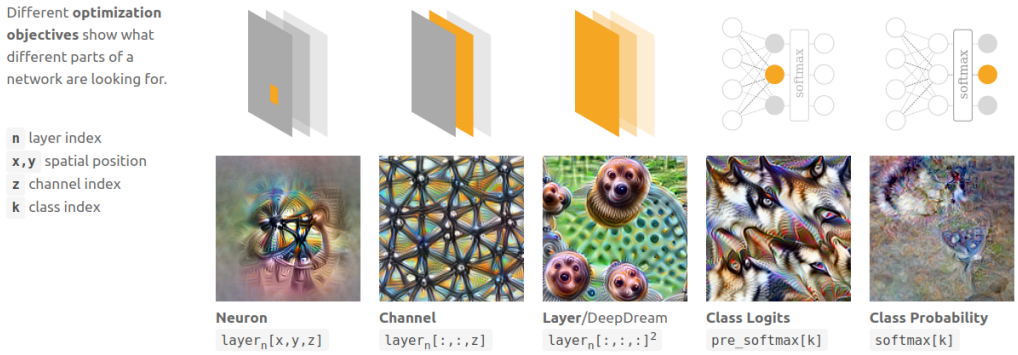


Figure 7: Even though channel optimization is the most common technique to produce visualizations, it's possible to maximize the activations of different types of units [18].

Feature visualization by optimization can be complemented by inspecting the dataset itself. This involves searching for images within the dataset that maximize the activation of a specific unit in the neural network.

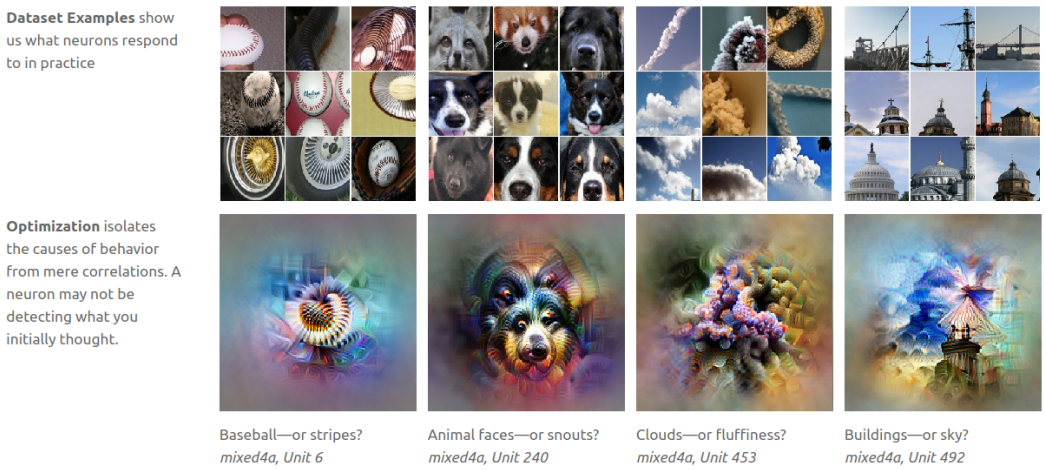


Figure 8: Dataset inspection and optimization are complementary techniques, one can help disambiguate the other [18].

3.3 Challenges to visualizing features with optimization

Unfortunately, feature visualization is not as straightforward as previously explained. In reality, simple optimization techniques often yield visualizations of low quality, characterized by excessive noise.

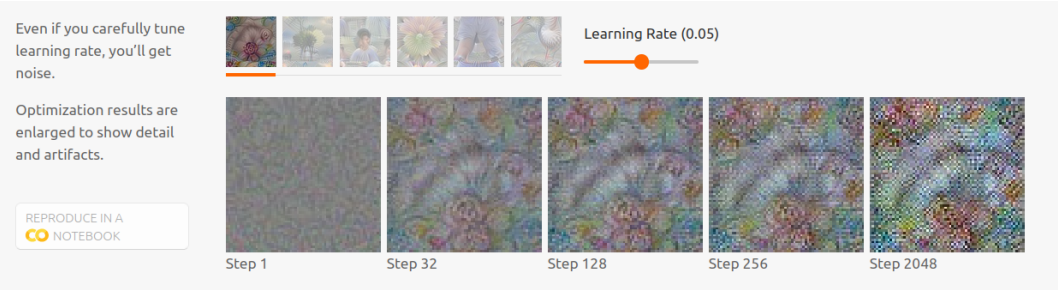


Figure 9: There are significant challenges to getting satisfactory visualizations, naive optimization must be constrained to remove noise and high-frequency patterns. [18].

Two ways we can constrain the naive optimization method to get images of better quality is by adding some kind of **regularization** in the objective or through **preconditioning and parameterization**.

Regularization

Regularization is a technique that involves modifying the loss function, typically by adding a penalty term to the optimization objective.

There are three main types of regularization:

1. **Frequency penalization** directly focuses on high-frequency patterns by penalizing the variance between neighboring pixels,

2. **Transformation robustness** involves stochastically transforming the image before each optimization step. It can be considered a form of data augmentation, where images are subjected to jittering, rotation, or scaling to search for examples that are robust to these transformations,

3. **Learned priors regularization** involves using an image generator and optimizing from its latent space. A prior is a probability distribution that expresses one's initial belief about a quantity. A GAN trained to generate natural images has learned a prior on natural images. This is what enables it to produce plausible images. Optimizing from a GAN latent space instead of from pure noise makes it easier to produce a visualization resembling a natural image.

Preconditioning and parameterization

Preconditioning (or parameterization) involves a change of basis (for exemple from pixel space to Fourier space) in order to simplify the optimization problem. Unlike the aforementioned regularization methods, preconditioning focuses on reducing high frequencies in the gradient rather than directly in the visualization itself. It does not alter the loss function but instead modifies the path of steepest descent during optimization. The minima of the optimization remain unchanged under preconditioning, but the shape of their basins of attraction are affected, changing which minimum the optimization process converges to.

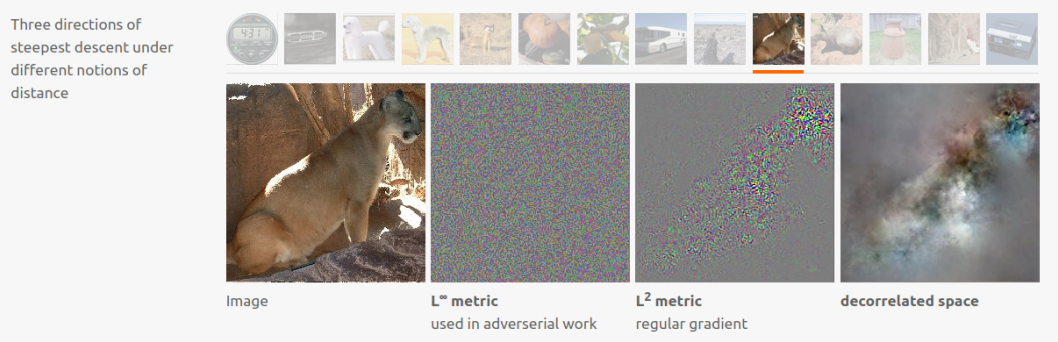


Figure 10: Gradients of a cheetah image under different metrics. The direction of steepest descent can be radically different when changing the metric. In the decorrelated space (color decorelation followed by change to Fourier basis), we get a much smoother steepest descent [18].

3.4 Random directions in the activation space

Feature visualization aims to maximize the activation of specific units, such as individual neurons, channels, or entire layers. Instead of summing the activations of all neurons within a channel or layer, **weighted sums** can be used to optimize a subset of neurons from different channels within a layer. These weights can be considered as a vector in the space of all possible combinations of neuron activations. Individual neuron activations can be seen as the canonical basis vectors of this activation space. Thus, any combination of neuron activations represents a vector or direction within the activation space and can be visualized. This leads to the question: **are the canonical basis vectors more interpretable than random directions in this space?**

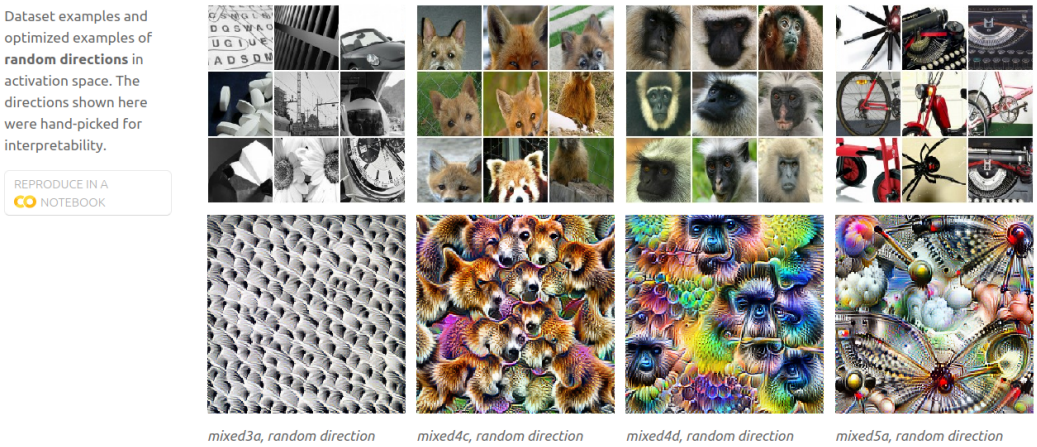


Figure 11: As explained in *Feature visualization*, random directions often seem interpretable, but basis vectors are so more often [18].

3.5 Feature visualization in multimodal models

Follow-up work has delved into the analysis of features learned by CLIP models. CLIP models consist of two interconnected models trained to align pairs of images and text (typically, a vision model such as ResNet and a language model such as a transformer). In an article titled *Multimodal Neurons in Artificial Neural Networks* [12], a wide range of interpretable neuron families were identified within CLIP models. These neurons exhibit responses to various topics, including sensitive subjects like gender, race, religion, sexual orientation, politics, and mental health status, as well as toxic content such as hate speech and sexual content.

The authors suggest that identifying and recognizing these representations could potentially aid in **addressing networks biases**.

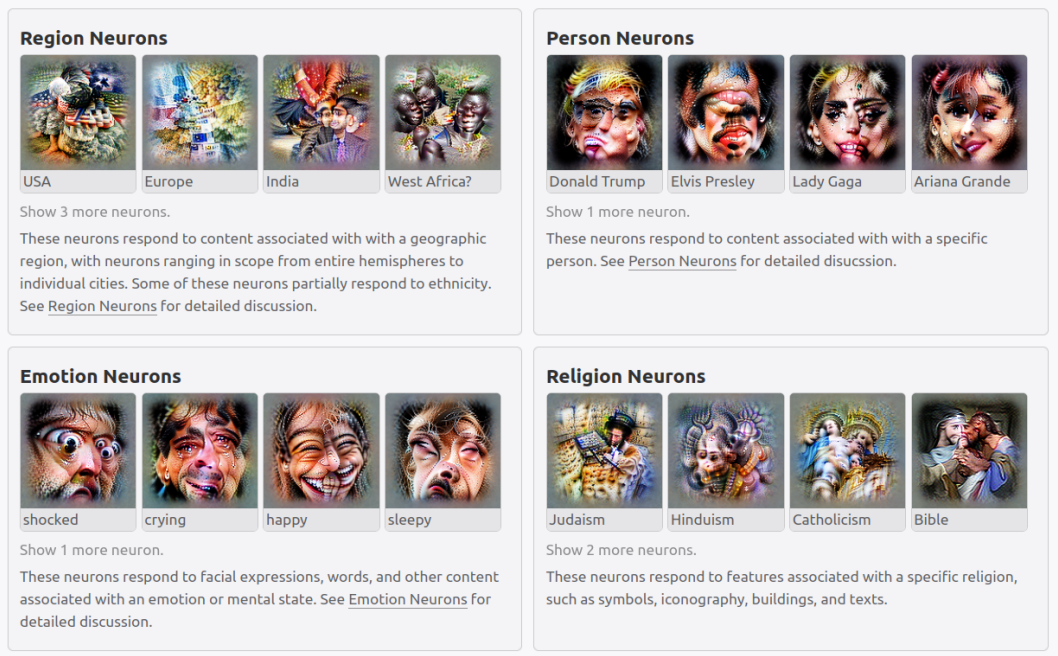


Figure 12: Feature visualization from selected neurons in the final layer of different kinds of CLIP models, organized into families. Families and labels were chosen after looking at dataset examples that maximally activated each neuron [12].

4 Activation Atlas

Feature visualization can be expanded with the concept of **activation grid**.

The **activation grid** of an image for a given layer in a network is a collection of feature visualizations generated for each patch in this image, as illustrated in Figure 13. To construct an activation grid, the input image is divided into small patches, upon each of which a feature visualization is produced. One layer in the network, associated with one patch in the image, defines a vector of activations. These activations are then employed as weights to conduct layer feature visualization (the same way as explained earlier for visualizing random directions in the activation space).

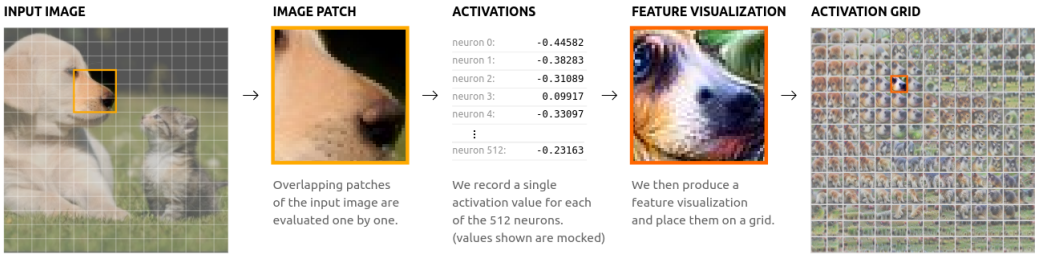


Figure 13: An example of an activation grid and its construction process. It shows how the network interprets different parts of the image [7].

A significant weakness of feature visualization and activation grids is that they're limited to understanding how a network responds to a single input at a time. In 2019, Chris Olah's team introduced the concept of **activation atlases** to illustrate how a network reacts to a whole dataset in a single representation [7].

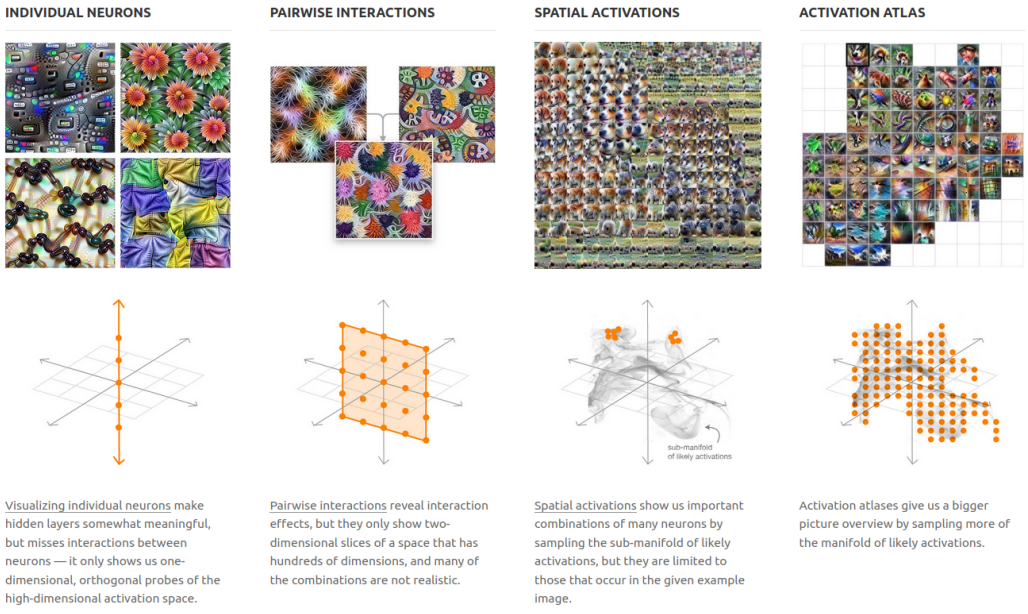


Figure 14: Visualizing individual neurons was already done in *Feature visualization*. Pairwise interactions enable to see how two given neurons interact by jointly optimizing them. Spatial activation helps understand how a layer sees each patch in an image. Activation atlases are an extension of feature visualization used to visualize the response of a layer to several images [7].

The initial stage for building an activation atlas requires gathering a large number N of images and randomly selecting a spatial activation pair (x, y) for each of them. Each spatial activation defines an

activation vector. As a result, N activation vectors, denoted as $h_{x,y}$, are obtained. These vectors are high-dimensional, perhaps with 512 dimensions.

To make sense of this bunch of data, the first step involves projecting it onto a 2D space using dimensionality reduction techniques (such as UMAP or t-SNE). This projection allows for similar activations to be positioned closer to one another in the transformed space. In Figure 16, the activations from images of dogs exhibit closer proximity to each other than to the activation from a shore picture. To reduce the number of points, a grid is applied to the 2D space, and activations are averaged within each grid cell. Feature visualization can then be performed on each averaged vector.

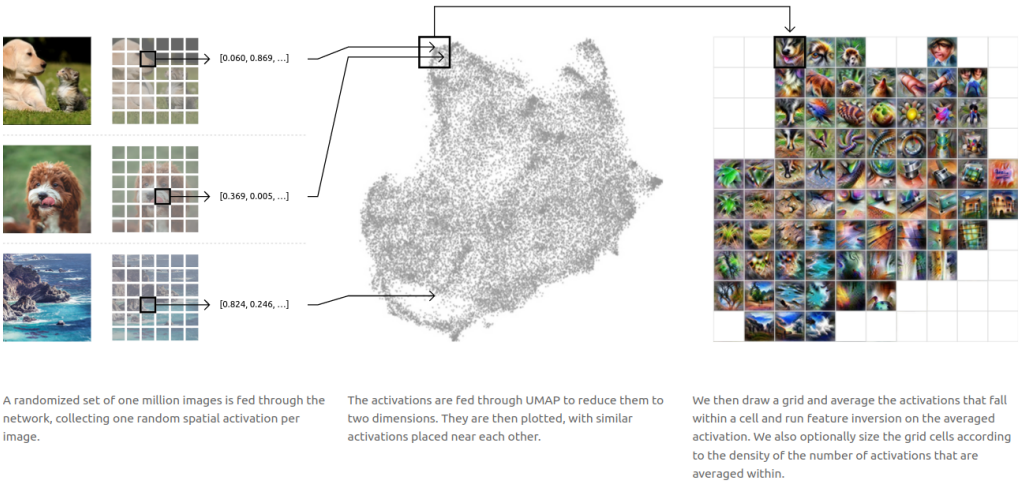


Figure 15: An activation atlas reflects the variety of abstractions and concepts the model has developed in a given layer [7].

With activation atlases, we gain the ability to analyze individual layers as well as compare the learned representations across layers. Generally, the early layers of the model capture abstract concepts, textures, and simple patterns. As we move towards the middle layers, more specific, concrete, and complex concepts begin to emerge, building upon the foundations laid by previous layers. The refinement of the network representations can be observed by comparing activation atlases across layers.

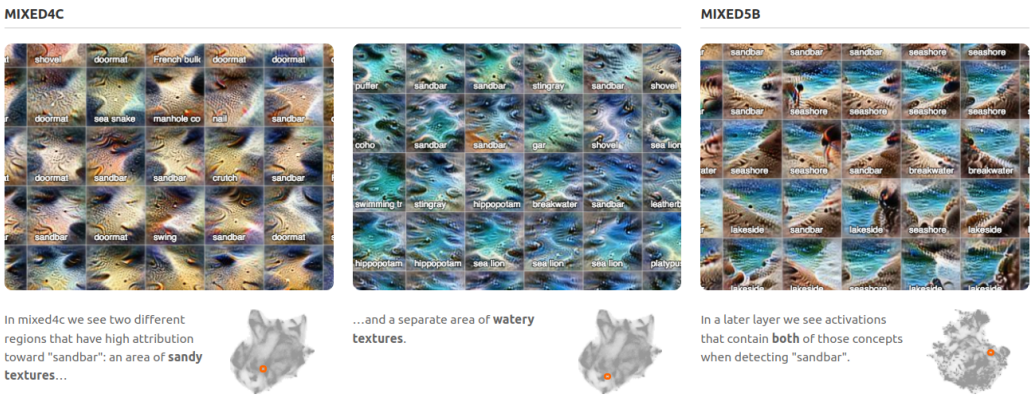


Figure 16: Activation atlases shows how the network can learn to integrate and create new concepts through the merging of existing ones as it progresses through its layers. In mixed4c layer, distinct concepts of sandy and watery textures are represented separately. As we move deeper into the network, these concepts begin to merge in a new representation combining the characteristics of both concepts [7].

5 The Circuits Thread

5.1 Introduction

The *Circuits Thread* [6] is a collection of articles published in 2020 that extensively investigate the inner representations of InceptionV1 trained on ImageNet (all figures from this section were obtained on this network, unless otherwise mentioned). The goal of the circuit project is to gain insight into models by examining each of their neurons individually and understanding the connections between them. The authors refer to this approach as "zooming in," drawing a parallel with neuroscience or cellular biology, where delving into cells and then DNA lead to dramatic shifts in our understanding.

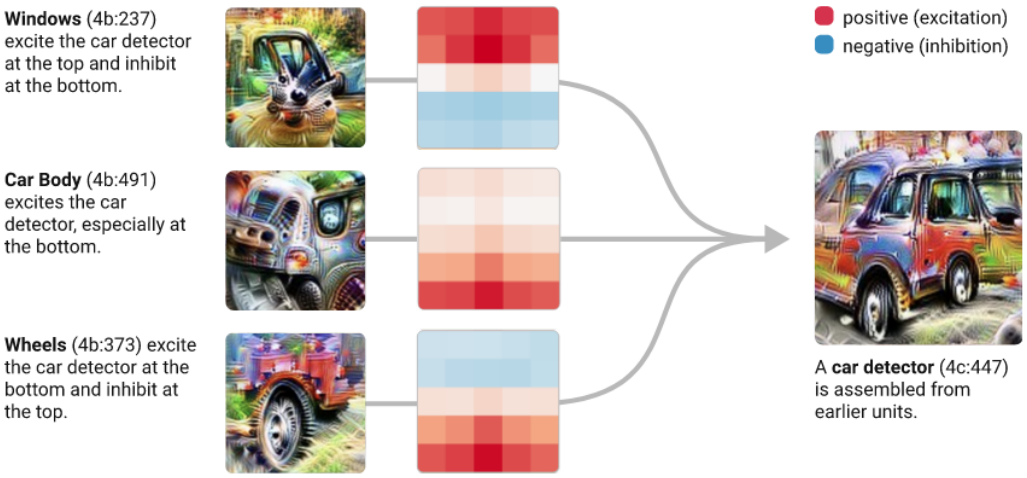


Figure 17: This example of a car circuit discovered in InceptionV1 is one of the most remarkable outcomes of mechanistic interpretability. By utilizing features extracted from the previous layers, the neuron responsible for detecting cars searches for wheels positioned at the bottom of its convolutional window, car body in the middle, and windows at the top. Circuits provide an explanation of how higher-level features are constructed and synthesized from lower-level features learned in earlier layers [20].

The *Circuits Thread* is an ambitious project that has significantly advanced our comprehension of image processing in CNNs. It has yielded a wealth of significant findings and observations, which cannot be fully summarized here.

One important observation is that each layer in a CNN progressively extracts higher and higher-level features of the image. Early layers represent abstract concepts, they almost always learn edges and corners detectors, which evolve in intermediate layers into more complex and specific shapes.

Each layer assemble previous layers into more complex components, forming what the authors call **circuits**. A circuit is a set of features connected by weights. It can be thought of as a subgraph of the original network.

5.2 Early vision

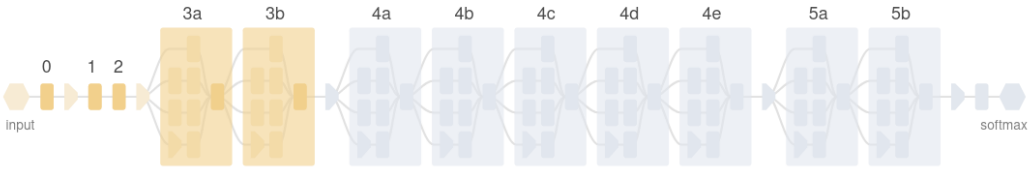


Figure 18: InceptionV1 architecture. Early vision comprises the first five convolutional layers, in orange here [2].

For an exhaustive walkthrough around InceptionV1 early vision, take a look at *An Overview of Early Vision in InceptionV1* [2].

In its initial five layers, InceptionV1 progresses from rudimentary Gabor filters (that also exist in the first layers of the human visual cortex !) and color-contrast detectors to more intricate boundary detectors and basic shape detectors (such as eyes, circles, and triangles detectors), up to sophisticated detectors for small heads. Along the way, various other feature families emerge, including more complex Gabor filters. Interestingly, these "neuron families" recur across different model architectures and training conditions.

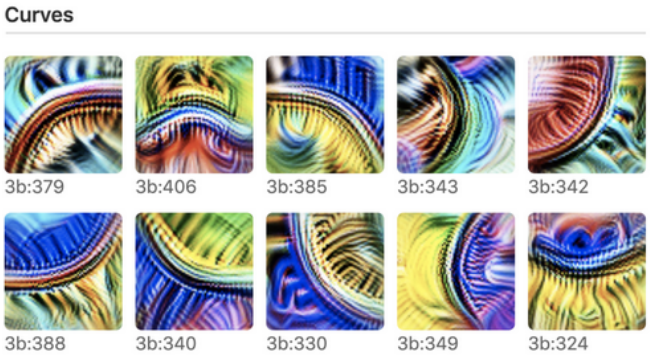


Figure 19: Curve detectors are universally found in early layers of CNNs, they exist in different orientations and colors and collectively span all orientations. Each curve detector responds to a wide variety of curves, preferring curves of a particular orientation and gradually firing less as the orientation changes. For an extensive overview of curve detectors, read the *Curve Detector* article of the circuits thread [5] [20].

CNNs also commonly learn **high-low frequency detectors** in their early layers.

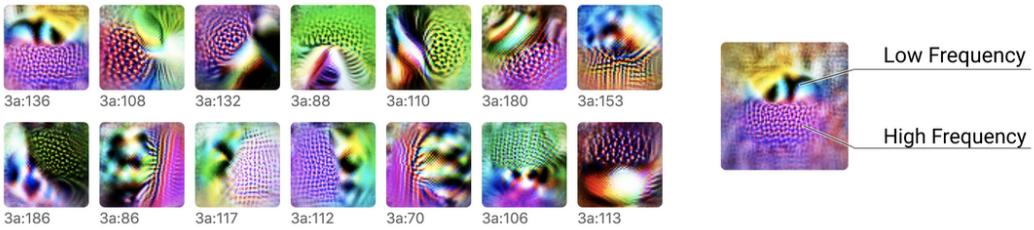


Figure 20: High-low frequency detectors look for low-frequency patterns on one side of their receptive field, and high-frequency patterns on the other side. They exist in different orientations and colors [20].

High-low frequency detectors assemble in deeper layers to form **boundary detectors**.

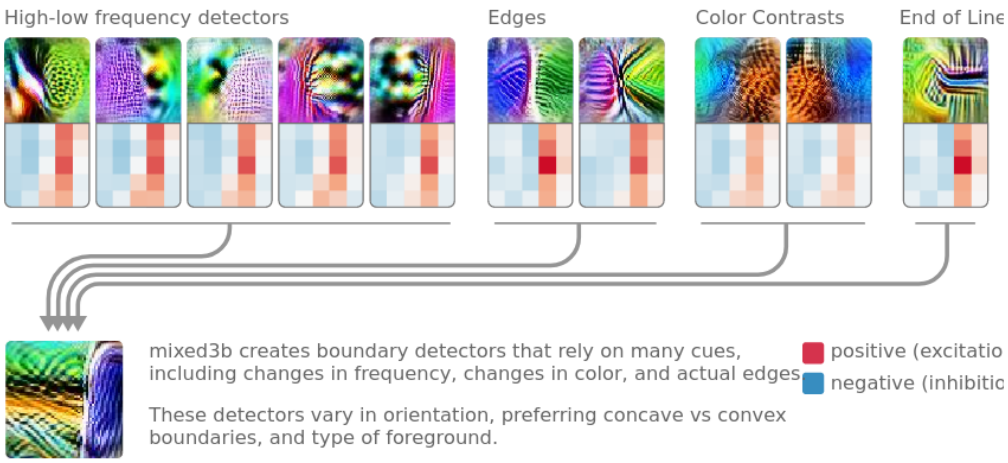


Figure 21: A boundary detector formed in InceptionV1 mixed3b layer, which combines cues from previous layers. [2].

Various methods can be employed to verify whether these units truly detect curves or boundaries. For a curve detector, one can observe that:

1. **Feature visualizations** clearly depict curves,
2. Empirically, curve detectors fire strongly for inputs containing curves. This can be verified by looking at **dataset examples** that maximize detectors activations,
3. Curves detectors also respond strongly to **synthetic curves** images, but not to lines or corners,
4. Curve detector circuits can be observed (and we can read curves off of their **weights**),
5. The **downstream clients** of curve detectors are features that naturally involve curves,
6. The authors tried to manuallly set the weights to reimplement a curve detectors, and showed that these handwritten circuits mimic the original curve detectors.

Note that the last three arguments are circuit-based.

5.3 High-level features and circuits

More complex, high-level features, form in deeper layers. A striking example of a circuit found in the mid-layers of InceptionV1 is the oriented dog-head detector. There are many interesting things to mention about this circuit. Two distinct pathways form in layers 3b and 4a, one recognizing left-oriented dogs and the other right-oriented ones. The two inhibit each other until they finally merge at layer 4b to create an orientation-invariant detector. This pattern of merging two separate detectors to create an invariant one is called **union over cases**. This circuit is an example of how a network can implement sophisticated invariances.

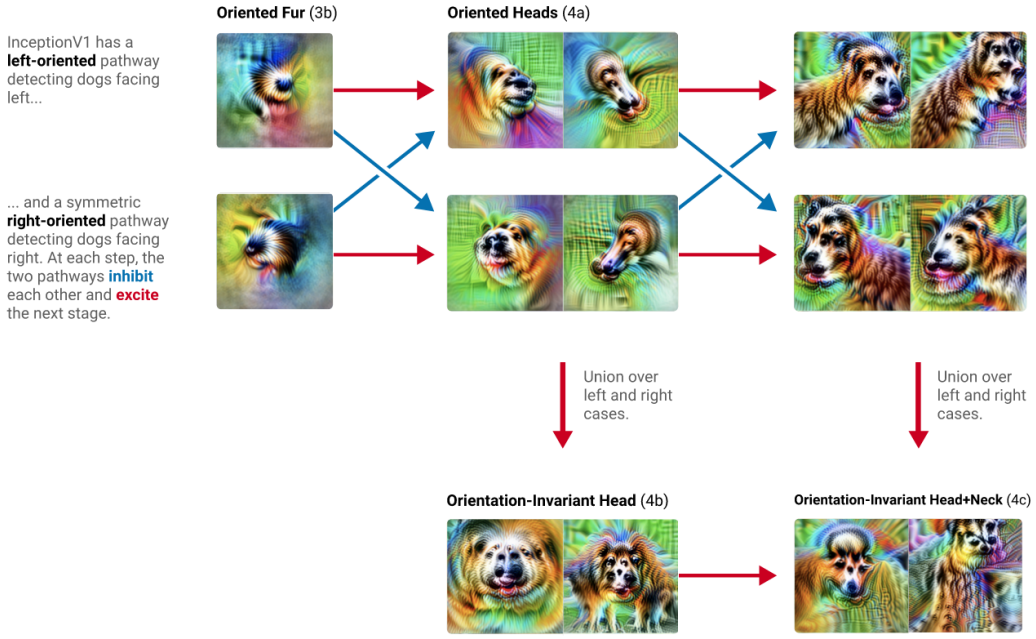


Figure 22: An oriented dog head detector in InceptionV1 trained on ImageNet [20].

5.4 Polysemantic neurons

Previously, we've seen that InceptionV1 implements a car detector at layer mixed4c using cues from the previous layers. An intriguing phenomenon occurs in the subsequent layer. The car feature spreads over multiple neurons, forming what we refer to as **polysemantic neurons**: units that respond to multiple unrelated inputs.

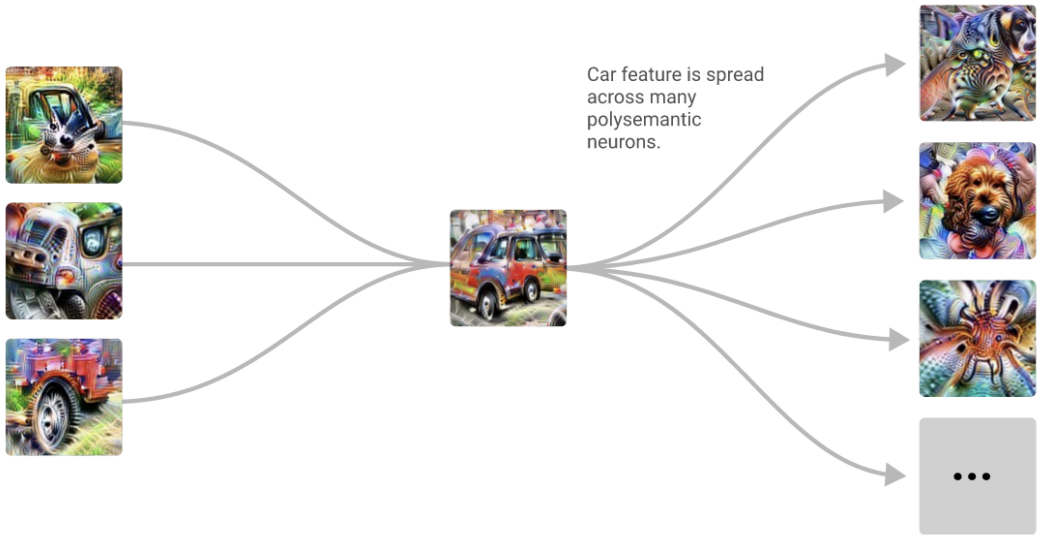


Figure 23: On the left is the car detector circuit from Figure 17. After distinct concepts are formed, they get intertwined into polysemantic neurons, such as a neuron responding to both car and dog images [20].

Networks frequently develop polysemantic neurons, it's a phenomenon also known as **superposition**. The existence of polysemanticity poses a challenge for interpretability research as it makes it even more difficult to achieve a comprehensive understanding of the network's features and circuits. In particular, feature visualization on polysemantic neurons struggles to capture multiple concepts in a

single image and often results in nonsensical or incomplete representations.

The occurrence of superposition is likely a strategy employed by the network to encode a greater number of features within its limited "storage capacity".

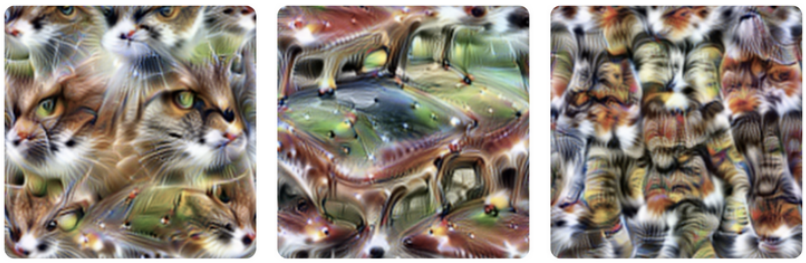


Figure 24: An example of a polysemantic neuron responding strongly to images of cars, as well as cat faces and cat legs. Note that these concepts generally do not co-occur with each other, which explains their entanglement [20].

The literature on representation disentanglement addresses the issue of polysemanticity. Researchers have explored potential solutions to tackle the problem. One approach involves "unfolding" the network, so that each neuron encodes a pure concept. Another strategy is to introduce specific constraints during the learning process to prevent networks from developing polysemantic neurons in the first place.

5.5 Universality hypothesis

The **universality hypothesis**, also known as "convergent learning", proposes that analogous features and circuits are formed across different models and tasks. While we are pretty confident that most CNNs learn Gabor filters in their early layers, it remains uncertain whether the same features are formed and connected in the same way in later layers, regardless of the model architecture and training set. There is evidence both supporting and opposing the universality hypothesis.

The results obtained by the authors suggest that the universality hypothesis is likely true, at least in the context of early vision. It is interesting to note that the human visual cortex also "implements" curve and edge detectors and various low-level features in its very first layers [15]. This observation raises the question of whether the universality hypothesis could also extend to biological neural networks.

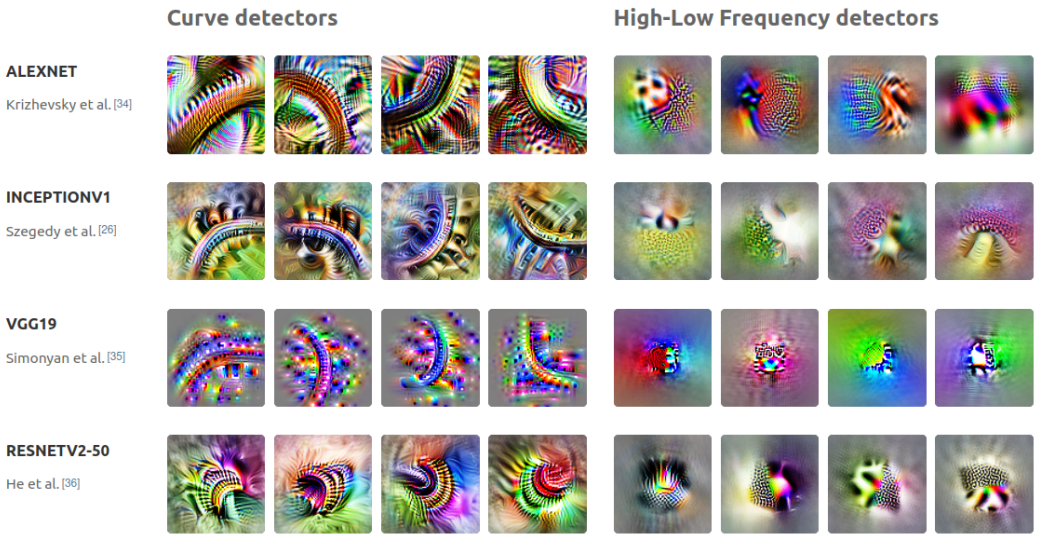


Figure 25: A couple of low-level features seem to form across a variety of model architectures (AlexNet, InceptionV1, VGG19 and ResNetV2-50) and datasets (including at least ImageNet and Places365) [20].

5.6 Caveats

The authors mention several caveats in their work:

- They acknowledge that our understanding of many of these units is of low-confidence. They anticipate that upon further analysis, it's possible we realize at some point that some units and categories have been misunderstood,
- They highlight that many neuron groups serve as catch-all categories or convenient organizational categories, which they do not believe reflect a fundamental structure,
- The boundaries of these concepts can be blurry, and the categorisation of certain neurons require subjective judgement calls.

6 Attribution Methods

6.1 Introduction

Interpretability methods can be broadly categorized into two types: those that investigate the networks internals, and those that aim to explain networks outputs. Methods from the first category - such as feature visualization - don't explain the specific mechanisms by which the network combines its individual units to make decisions and are useful only to those who have access to the model. **Attribution techniques** are a set of methods that attempt to explain classification outputs. They estimate *which part of the input image is the most responsible for the network's decision.* The most common interface for attribution is called a **saliency map**. There are a wide variety of approaches to attribution, among which [25]:

- **gradient-based methods** rely on backpropagating gradients from the output to the input (or a particular layer in the network) to estimate how a change in the input (or layer activations) affects the output. We'll focus here on the **Grad-CAM** algorithm, which is an example of a gradient-based method.
- **relevance-based methods** estimate pixel relevance with reference to a root point in the input space.

6.2 Grad-CAM

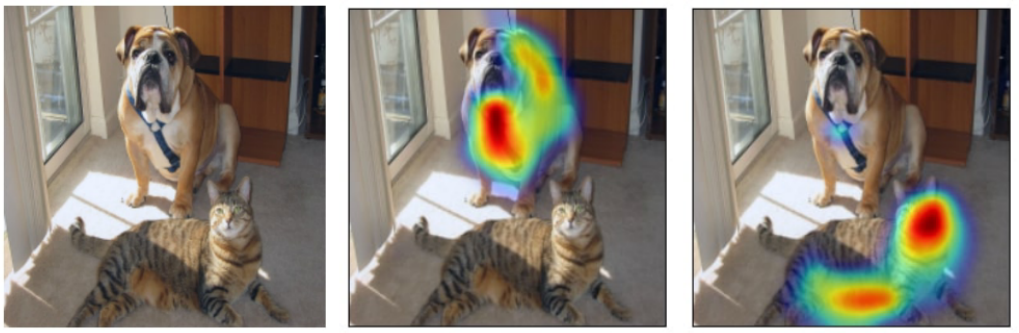


Figure 26: (b) and (c) are two saliency maps corresponding to the original image respectively classified as a dog and as a cat (this is what the authors name counterfactual explanations). (b) shows that the pixels contributing the most to the classification of the image as a dog are those representing the leash and parts of the dog itself. (c) shows that when the image is classified as a cat, the pixels contributing the most are those depicting the cat's head [23].

The Grad-CAM (Gradient-weighted Class Activation Mapping) algorithm [23] creates heatmaps highlighting regions in the input image that contribute the most to the predictions made by a CNN. On the above image, it can be used to verify that when classifying an image, the CNN extracts information from the right regions.

Very briefly, Grad-CAM heatmaps are generated from all feature maps in a chosen layer of the network. Grad-CAM builds a heatmap by summing all feature maps from one layer and weighting them according to how strongly they influence the classification. As previously seen in *Feature Visualization*, feature maps are selective to particular concepts. Intuitively, the weighting coefficients should somewhat represent the similarity between the predicted class and feature maps concepts (for instance, in Figure 26 (b), if the chosen layer contains a dog feature map, it's weight should be very high compared to other feature maps, while in (c) the cat feature map should have a higher weight. This intuition was not verified in the original paper but it's useful to get a high-level understanding of the Grad-CAM algorithm).

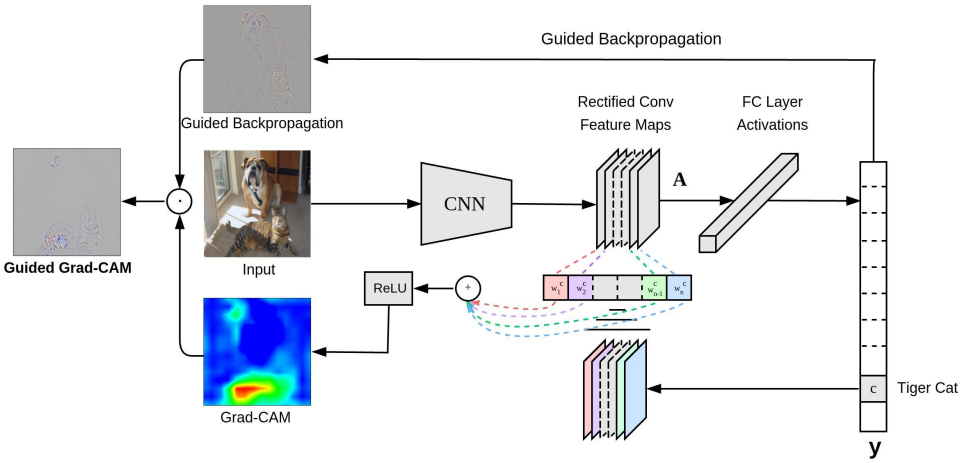


Figure 27: Grad-CAM overview. Given an image, a class of interest and a layer in the CNN, Grad-CAM generates a corresponding saliency map. The Guided Backpropagation path corresponds to the Guided Grad-CAM algorithm, which is a variant of Grad-CAM, but we won't get into the details of it here [23].

A more detailed explanation of the Grad-CAM algorithm is provided in the box below.

Grad-CAM algorithm

The aim of Grad-CAM is to generate a heatmap that highlights the regions where the network focuses its attention. We denote the Grad-CAM heatmap $L^c_{Grad-CAM} \in \mathbb{R}^{u \times v}$, where u and v are the width and height of the map. Given an input image and a network, we denote the score for class c y^c, and the activation of feature map k in a given convolutional layer A^k.

The Grad-CAM map is generated through the following steps:

1. Forward-propagate the input image through the network and store all the feature maps activations (in general, one layer in the CNN is chosen and the following steps are restricted to this layer) and the score y^c for the class of interest (i.e. the activation of the neuron before the softmax layer).

2. Back-propagate the gradient of class c to the chosen convolutional layer. Compute the gradient of the score class with respect to all the different feature map activations : $\frac{\partial y^c}{\partial A^k}$.

3. These gradients are global-average-pooled over the two spatial dimensions (width an height, indexed by i and j) to obtain the neuron importance weights α^c_k:

$$\alpha^c_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A^k_{ij}}$$

   α^c_k captures the 'importance' of feature map k for a target class c.

4. Finally, weight activation maps by the averaged gradients and apply a ReLU:

$$L^c_{Grad-CAM} = ReLU(\sum_k \alpha^c_k A^k)$$

   ReLU is applied to the linear combination of maps because we're only interested in the features that positively influence the classification, i.e. pixels whose intensity should be increased in order to increase the score y^c.

6.3 Difficulties of building saliency maps

Although saliency maps are a popular approach to explaining image classifier outputs, correctly assessing their reliability remains challenging.

Adebayo et al. in *Sanity checks for saliency maps* [1] proposed a series of tests (or *sanity checks*) to evaluate the reliability of an explanation. Surprisingly, the article revealed that certain renowned saliency methods are unrelated to model and data. This means that the explanation they provide are nothing more than mere edge detectors that do not require a trained model with abstract features.

The authors suggest two sanity checks for approving attribution methods:

1. The **model parameter randomization test** checks if saliency methods are independent of the models parameters. A saliency map which is insensitive to the models parameters is of no help for model debugging.

2. The **data randomization test** checks if saliency methods are independent of the data. It consists in comparing the outputs of the saliency method applied to a model trained on a labeled data set with the method applied to the same model architecture but trained on a copy of the data set in which we randomly permuted all labels. The random permutation of labels disrupts the original relationship between images and their labels, which should cause significant difference on the saliency map produced. If the saliency method is insensitive to the permuted labels, it suggests that the method is not reliant on the inherent connections between the input data and the corresponding labels.

Of all the methods tested, only Gradient saliency and Grad-CAM pass the sanity checks, while Guided BackProp and Guided GradCAM are invariant to higher layer parameters and hence fail.

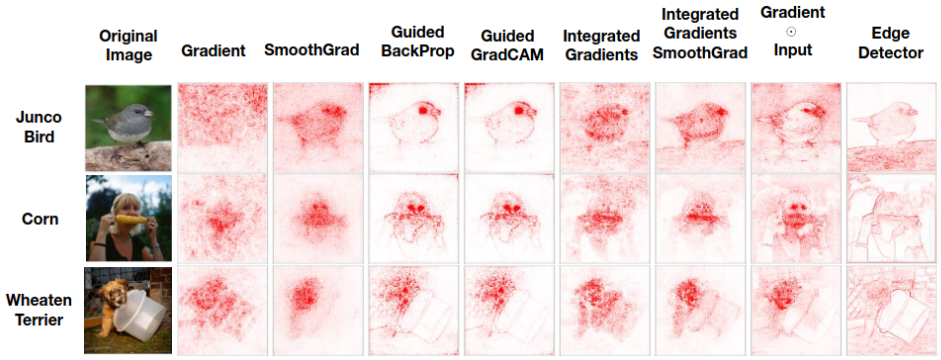


Figure 28: Comparison of the output of standard saliency methods with those of a mere edge detector, which doesn't depend neither on the model, nor on the training data. Surprinsingly, the results it produces are strikingly similar to those of many saliency methods [1].

This observation suggests that relying solely on visual inspection of the explanation is not an effective approach to determine whether it is relevant or not.

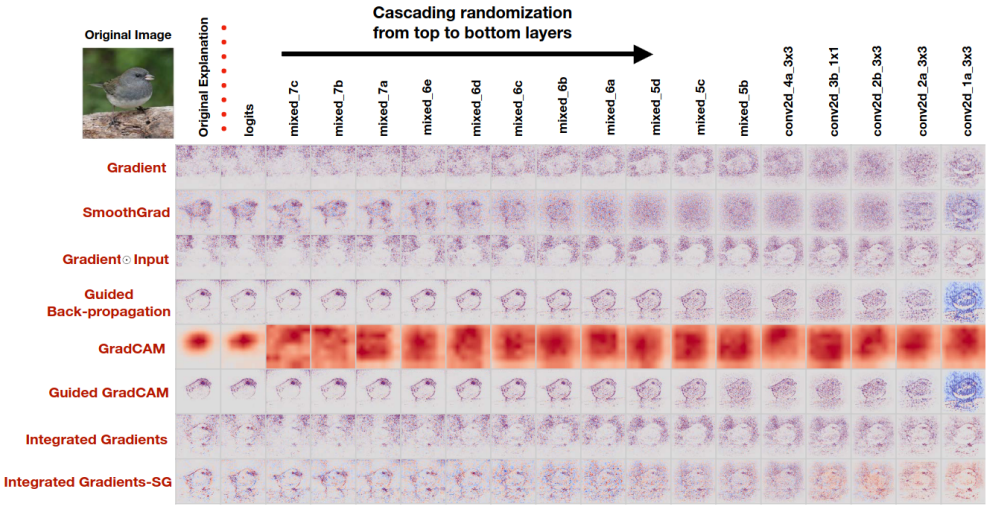


Figure 29: This figure represents the saliency maps produced after applying the cascading randomization technique on an Inception v3 model. It illustrates how the saliency methods highlight the important regions in the input image after the learned weights have been randomized through the cascading process. Starting from a fully trained network, weights are progressively randomized from the last layer to the first one. The saliency maps produced by most attribution techniques depicted on this figure seem to not be affected by weight randomization. This may indicate that the attribution technique does not pass the model parameter randomization test an is thus misleading [1].

Note however that *Sanity Checks for Saliency Metrics* [25] points out that comparative rankings between saliency methods might be misleading. Common metrics used to assess saliency methods can have high variance and are sensitive to implementation details.

7 Combining feature visualization with attribution

Interpretability techniques such as feature visualization, attribution and dimensionality reduction were primarily studied and used in isolation. While feature visualization helps us understand *what* the network sees, attribution explains *how* this affects the output. *The Building Blocks of Interpretability* [19] by Olah et al. proposes to treat them as complementary and composable building blocks and build a unified interface for interpretability.

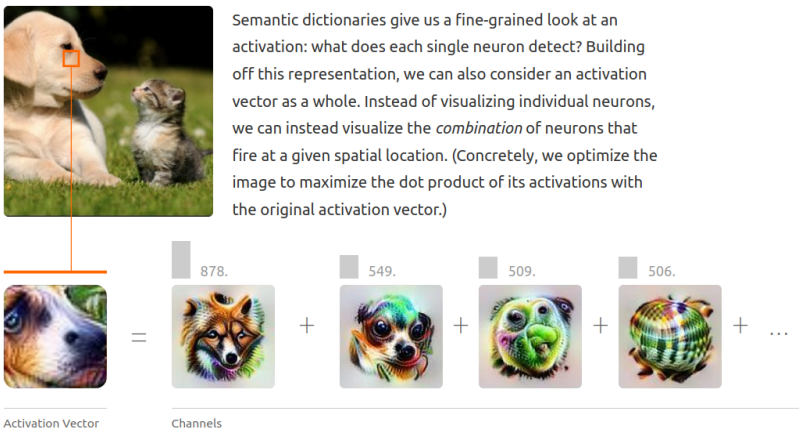


Figure 30: An example of a semantic dictionary, defined for a particular patch on the input image and a particular network. Semantic dictionaries pair each neuron activation with a visualization of that neuron and sort them in order of importance. Here, the semantic dictionary is not formed over single neurons activation but over channels. The image on the bottom left corner is the feature visualization weighted by the activations of each channel, as in *Activation Atlas* (see Figure 13) [19].

Olah et al. introduce the concept of **semantic dictionaries** (see above figure). Choosing a patch in an input image and a layer in a network defines a vector of activation. Each component of this vector is a set of neurons across different features maps. To create a semantic dictionary from this vector, all neurons are sorted by the magnitude of their activation and paired with their feature visualization, as illustrated on Figure 30.

Neuron or channel attribution are meaningful but produce an overwhelming amount of information to explore. One way to make this technique more human-scale is to slice the cube of activations more efficiently. Matrix factorization techniques offer strategies for breaking up matrices. Different information can be prioritized by the choice of splitting: we may want to fully describe the activations to understand what the network detected, or we may want to the factorization to describe the attributions to understand what caused the network decision. A balance of both objectives can also be optimized.

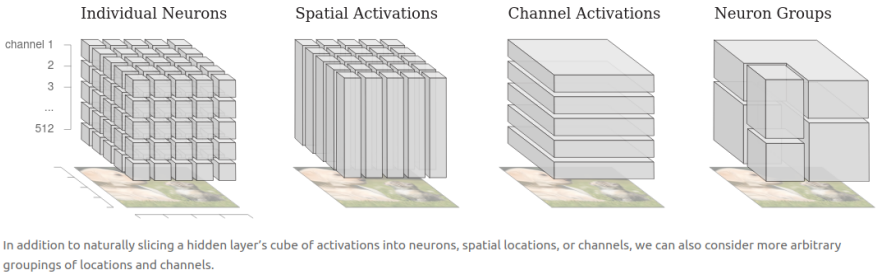


Figure 31: We can group neuron using matrix factorization techniques to make more human scale visualizations [19].

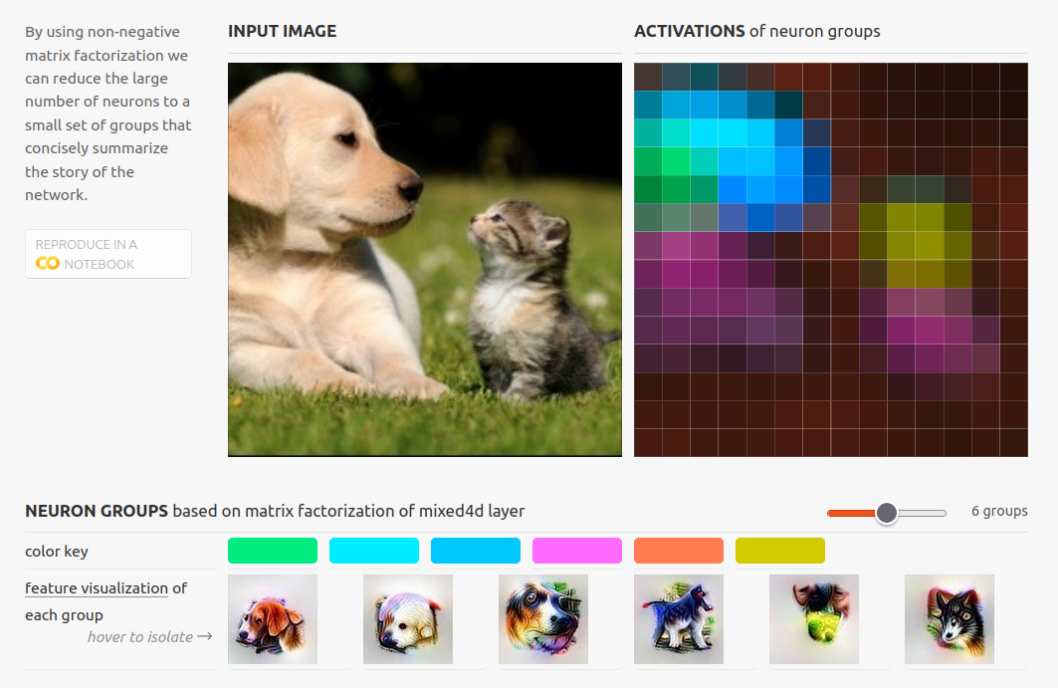


Figure 32: Activations of mixed4d layer were separated into 6 different neuron groups using matrix factorization. The large number of neurons and channels from mixed4d layer was reduced to a small set of groups, making it a lot more convenient to understand the behavior of the model [19].

8 Interpretability in RL vision

8.1 Introduction

In *Understanding RL Vision* [14], Hilton et al. apply interpretability techniques to observe the features learned by an RL agent trained on the video game CoinRun. Using tools from *The Building Blocks of Interpretability*, they uncover the reasons behind failure cases.

To quantitatively validate their analysis, they manually edit the weights of the model to selectively disable certain features and make the agent blind to them. They verified the effects of these modifications by observing which hazards lead the new agent to fail.

Hilton et al. formulate the **diversity hypothesis**, which asserts that the interpretability of a RL model increases as it is exposed to a wider range of environments during training.

8.2 Attribution to identify features predictive of the RL agent success

CoinRun is a side-scrolling platformer game where the agent's objective is to avoid enemies and traps while reaching the end of each level to **collect the coin lying there** (which is the only reward in the environment). Colliding with any obstacle leads to immediate death for the agent. The level concludes when the agent either dies, collects the coin, or reaches the maximum time limit of 1000 steps [21].

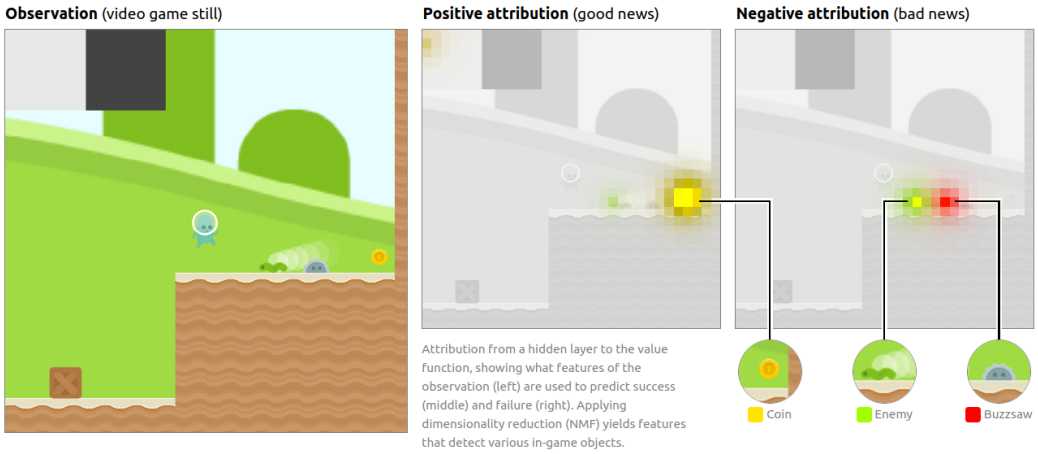


Figure 33: An example of a CoinRun environment on the left. Hilton et al. identified attribution vectors whose components correspond to different types of object (enemies, buzzsaws, the coin) [14].

Hilton et al. trained a convolutional actor-critic model which takes as an input a single downsampled 64x64 image, and outputs two things: a value function (an estimate of the total future time-discounted reward) and a policy (a probability distribution over the actions, from which the next action is sampled).

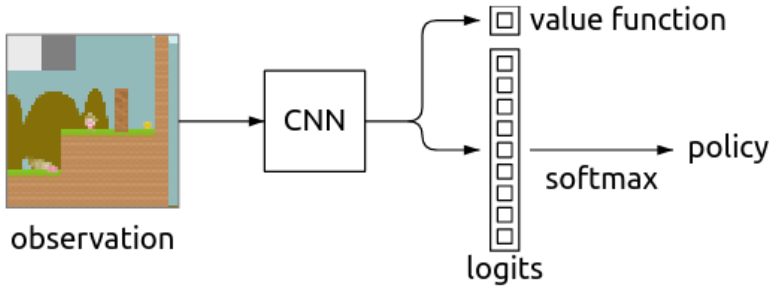


Figure 34: Schematic of the CNN trained on CoinRun images to predict a value function and a policy [14].

8.3 Dissecting failures

Hilton et al. developed an interface to examine trajectories of the agent through the game.

First, they used attribution to identify the hidden neurons explaining the value function and the policy distribution. **Attribution** highlights the elements in the input image explaining the outputs (value function and policy distribution). Using **dimensionality reduction** over attribution heatmaps, they obtain attribution vectors whose components correspond to different types of objects: buzzsaw obstacles, coin, enemies, platforms, steps, etc.

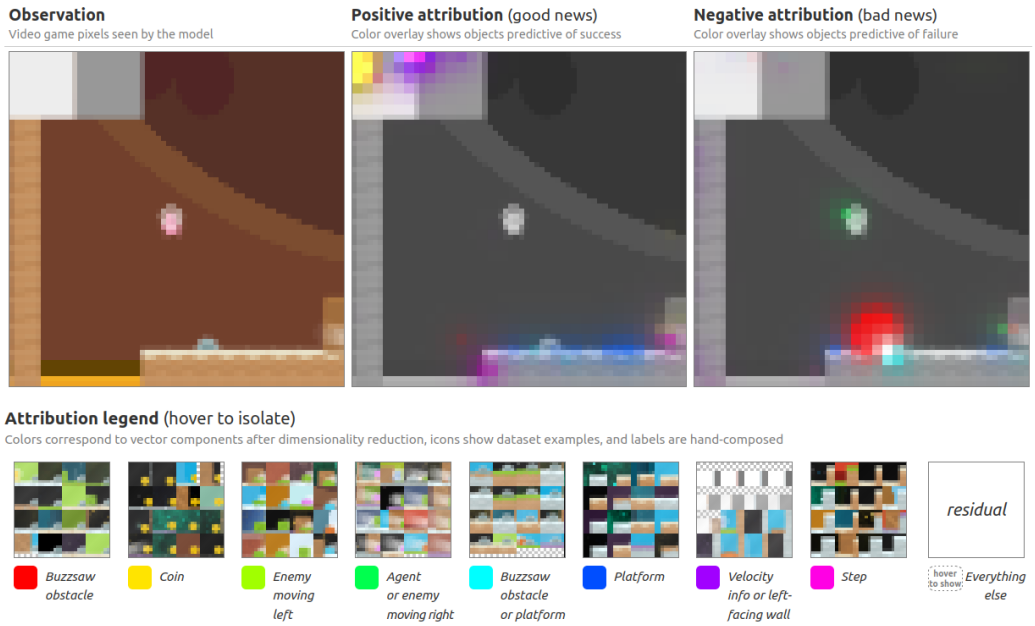


Figure 35: The interface for a typical trajectory. The observation on the left is the input to the CNN. In the middle are the attributions that are predictive of success (i.e. which entail an increase in the value function). On the right are the attributions that are predictive of failure. These visualizations reveal that the model uses obstacles, coins, enemies and more to compute the value function. The presence of the coin in the input image is predictive of success, while a buzzsaw obstacle decreases the value function. [14].

Their interface proved particularly useful to understand why failures occurred. Examples of failures can be analyzed step by step. Failures are often explained by the fact that the model has no memory, and must therefore choose its action based only on the current observation. It is also common for some unlucky sampling of actions from the agent's policy to be partly responsible.

8.4 Model editing

Analyzing the visualizations through an interface is insightful but mostly qualitative. To quantitatively validate their analysis, Hilton et al. hand-edited the model to make the agent blind to certain features they identified through their interface. This process can be thought of as a form of circuit-editing.

Their intuition was that making the agent blind to a certain feature by modifying its circuits could actually influence its performance. For example, making the agent blind to buzzsaw obstacles indeed caused him to collide with them more often.

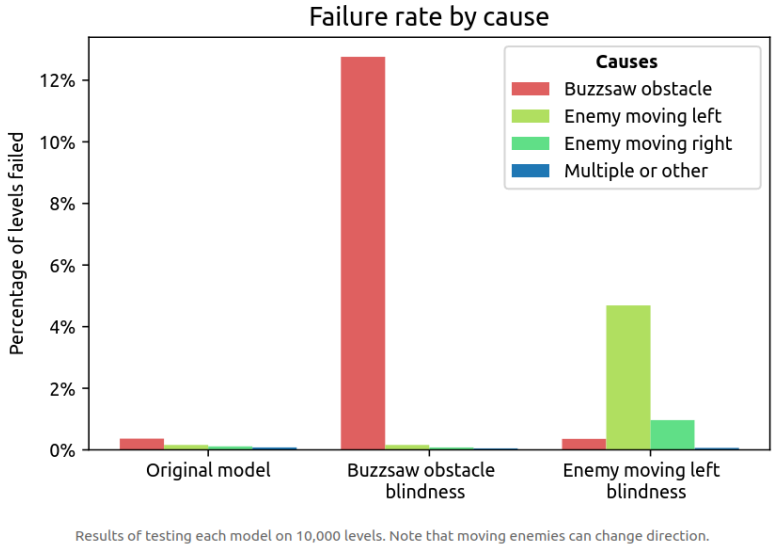


Figure 36: Percentage of levels failed by the agent before and after circuit-editing, categorized by the cause of failure. In the original model, the agent failed due to buzzsaws in only 0.37% of the trials. After the buzzsaw circuit was edited to make the agent blind to them, the agent's failure rate due to buzzsaw dramatically raised to 12.76% (without affecting other failure rates). Another circuit-editing experiment was conducted on the circuit representing enemies moving left. Making the agent blind to them resulted in a significant increase in the percentage of failed levels due to them. Model editing is a useful tool to validate the relevance of circuits identified using attribution and dimensionality reduction techniques [14].

8.5 The diversity hypothesis

CoinRun employs procedural generation, meaning that each level encountered by the agent is randomly generated from scratch. This encourages the model to learn skills that can generalize to unseen levels, as it cannot rely solely on memorizing a limited set of specific paths.

Hilton et al. discovered that increased diversity in training environments is associated with higher interpretability of learned features. Based on this observation, they formulated the **diversity hypothesis**:

> **Interpretable features tend to arise (at a given level of abstraction) *if and only if* the training distribution is diverse enough (at that level of abstraction).**

The right implication ($\Rightarrow$) means that when the training distribution is not diverse enough, models tend to overfit rather than learn interpretable features that generalize.

The left implication ($\Leftarrow$) should not be understood as a strict statement: diversity on its own is not enough to guarantee the development of interpretable features, but in practice the diversity of the training distribution often enables interpretable features to form.

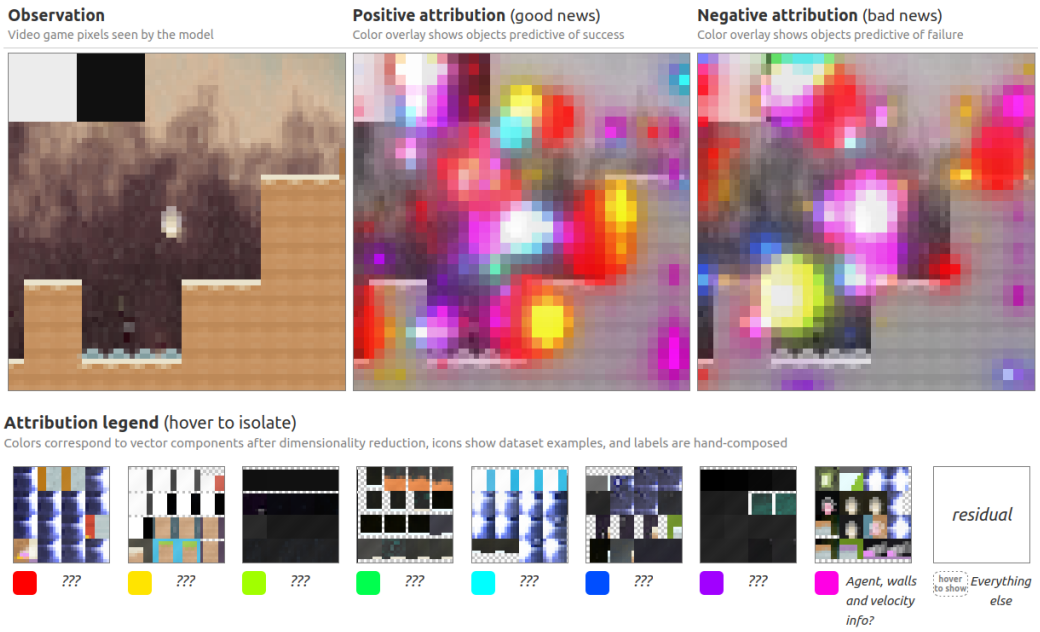


Figure 37: Reducing the diversity in the training environment drastically reduced the interpretability of the model's features. The agent trained on a set of only 100 similar levels doesn't seem to learn any interpretable feature. Here, attribution methods do not manage to provide a meaningful explanation of the value function, compared to Figure 35. It can also be noted that when the number of different training levels is small, the value function predicted by the CNN increases linearly across time, indicating that the model has just memorized the average number of timesteps until the end of the level but does not take into account the objects present on the image [14].

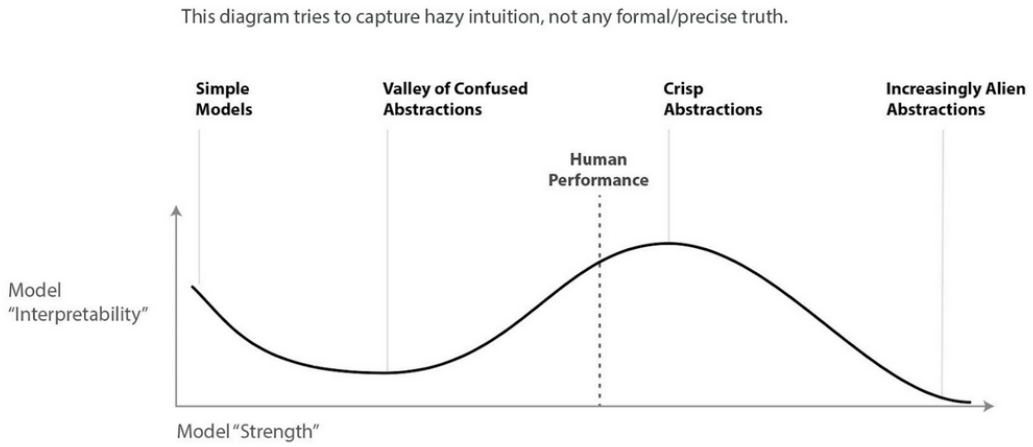


Figure 38: Chris Olah's views on the scaling of interpretability tools. Simple models, such as linear regression, are pretty interpretable, along with models that achieve performance levels similar to humans. Models that fall in between, lacking sufficient power, struggle to represent concepts in a clear and precise manner, resulting in confused representations (for instance, InceptionV1 is considered to be more interpretable than AlexNet). However, as we go beyond human-level performance, interpretability decreases significantly due to the emergence of unfamiliar and alien concepts (say advanced mathematics, ...). The diversity hypothesis seems to support the intuition that interpretability is positively correlated with performance for models close to human performance: as the model is trained on more diverse data it becomes stronger as well as more interpretable [11].

8.6 Feature visualization

Feature visualization techniques [18] did not yield any interpretable result, despite trying many different regularizations and parameterizations. While it works well for classic CNNs trained on ImageNet, it often struggles with RL agents trained on Atari games.

The reason why gradient-based feature visualization methods do not capture anything may be due to the fact that solving CoinRun doesn't fundamentally necessitate advanced visual capabilities. It's possible to solve the game by exploiting straightforward visual heuristics, such as identifying specific small patterns of pixels. These heuristics prove effective within the limited range of images encountered during model training but behave unpredictably in the full space of images in which gradient-based optimization takes place.

9 Automatic interpretability

9.1 Network dissection

The approaches we have encountered thus far involve generating visualizations that have to be **manually interpreted afterwards**. Bau's team from MIT's CSAIL, introduced a method called **network dissection** (also known as NetDissect) that **automates** the labeling of hidden units within a CNN [4]. Compared to the aforementioned methods, **NetDissect does not involve any subjective judgement** from the experimenter. This is why it can be qualified as an automatic interpretability technique.

The first step before performing network dissection is to build a Broadly and Densely Labeled Dataset (Bau et al. call it the Broden dataset). The one they use in the paper comprises 63,305 images with 1197 visual concepts. Each image is associated with one or multiple concept, and a binary mask corresponding to each of them. Six categories of concepts are distinguished: objects, parts, scenes, materials, textures and colors.

Figure 39: Examples from the Broden dataset [4].

The goal of network dissection is to associate each channel of the network being dissected (also called the *probed network*) with one or several of the 1197 concepts from the Broden dataset. At the end of the process, each unit in the probed network is labelled with a list of concepts from the Broden dataset, in order of importance.

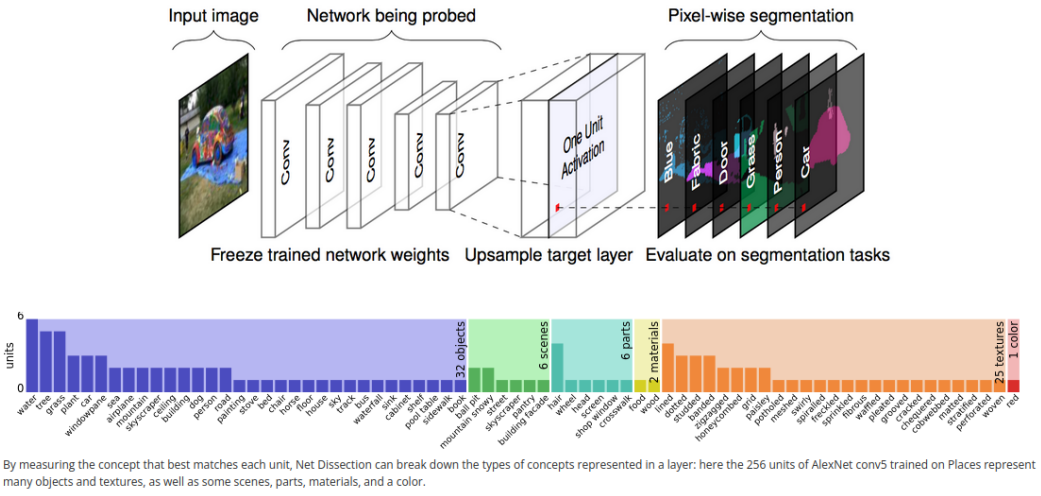


Figure 40: Overview of network dissection for measuring association of concepts and units in a given CNN. Here, we analyse one feature map from the last convolutional layer of a given CNN by evaluating its correspondence to 1197 different concepts from the Broden dataset. This channel is particularly aligned with objects like water, tree, grass, lined textures, etc. [4].

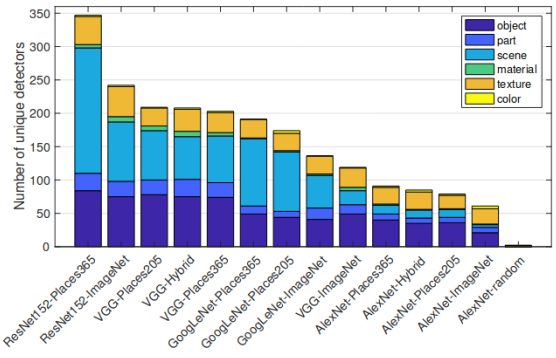
NetDissect works by calculating how each channel in a CNN is able to correctly segment each concept in the Broden dataset. Let's imagine that we're trying to label a channel that is selective for flower images. Intuitively, when an image of a flower from the Broden dataset is passed as input to the network, the activations of the flower channel will likely closely match the segmentation mask of the flower. Thus, the channel provides a high-quality segmentation of the flower, and NetDissect will output a high score for the (channel, concept) pair. For each channel, the scores generated by NetDissect can be sorted, and the highest scores indicate the concepts encoded in each channel.

A more detailed explanation of the NetDissect algorithm can be found in the box below.

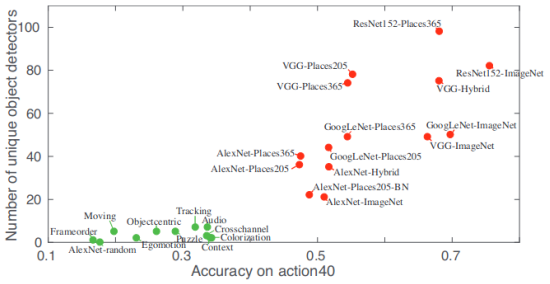
Network Dissection

The alignment between a channel and a concept is quantified using the IoU score (intersection over union). It is computed for each pair (k, c) of channel k and concept c in the Broden dataset. $IoU_{k,c}$ quantifies the accuracy of channel k in detecting concept c. It's calculated through the following steps:

1. For every input image x in the Broden dataset, the activation map $A_k(x)$ of every channel k in the network is collected,
2. We denote by a_k the distribution of activations in channel k,
3. For each channel k, the top quantile level T_k is defined as : $P(a_k > T_k) = 0.005$. From this threshold, a binary segmentation $M_k(x)$ can be drawn: $M_k(x) \equiv A_k(x) \geq T_k$. M_k represents the 0.5% of activation map's spatial locations that are activated the most.
4. The annotation mask for concept c in image x is denoted $L_c(x)$,
5. The score (or accuracy) of unit k in detecting concept c is computed for every (k, c) pair and defined as:
$$IoU_{k,c} = \frac{\sum_x |M_k(x) \bigcap L_c(x)|}{\sum_x |M_k(x) \bigcup L_c(x)|}$$
where $|.|$ is the cardinality of a set. One way to see it is that: $IoU_{k,c}$ =(number of pixels where the binary maps $M_k(x)$ and $L_c(x)$ are both 1) / (total number of unique pixels labelled 1 in both binary maps) [9].
6. If $IoU_{k,c}$ exceeds a certain threshold, unit k is considered as a detector for concept c. The IoU can be thought of as an evaluation score for the quality of the segmentation of a unit.



(a) Bau et al. propose to quantify the interpretability of a network by counting the number of units detecting a single concept (called *unique detectors*). Number of unique detectors across different architectures and training conditions.



(b) The number of unique object detectors in the last convolutioonal layer compared to each representations classification accuracy on the action40 data set. Supervised (red) and unsupervised (green) representations form two clusters.

Figure 41: Some of the results yielded by network dissection [4].

Figure 42 illustrates the interpretability of snapshots of Places205-AlexNet model **at different training iterations**. Object detectors and part detectors begin emerging at about 10,000 iterations. No evidence was found of transitions across different concept categories during training. For example, units in conv5 do not turn into texture or material detectors before becoming object or part detectors.

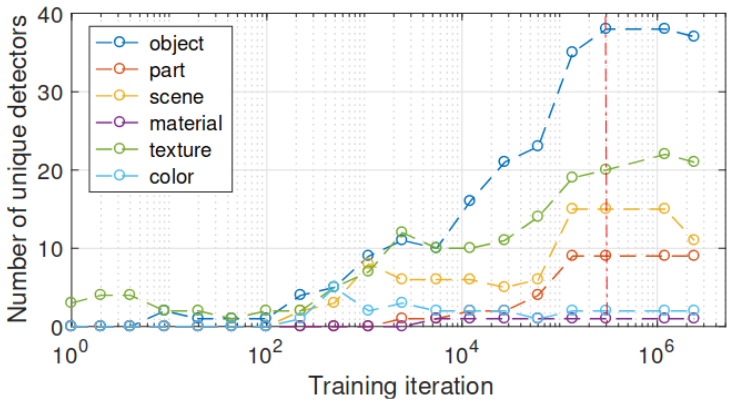


Figure 42: The evolution of detectors in conv5 of Places205-AlexNet accross training iterations. The baseline model is trained to 300,000 iterations (marked at the red line). The number of detectors increases accross training until it drops (probably due to overfitting) [4].

Interpretability over different training conditions was also studied: for a given network, different random initializations converge to to similar levels of interpretability.

9.2 Compositional Explanations of Neurons

Network dissection used concepts from the labeled Broden dataset to explain neurons in a CNN. Bau et al. characterise neurons as detectors of simple concepts, but Mu and Andreas argue in *Compositional Explanations of Neurons* [16] that they may be more accurately explained by **compositional concepts**, such as: `(water OR river) AND NOT blue` (rather than just as a simple concepts). They propose to extend *Network dissection* to generate such compositional concepts automatically.

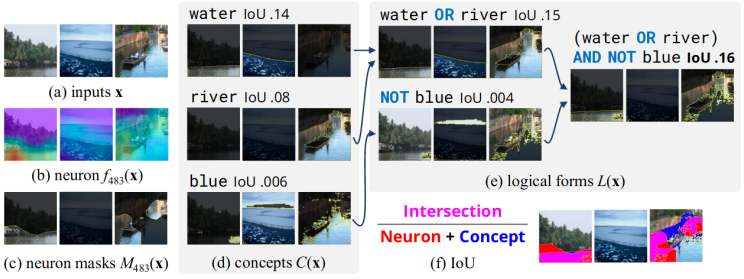


Figure 43: Generation of a compositional explanation for channel 483 in layer 4 of a ResNet-18 model. Starting from a set of input images x (a), and scalar neuron activations for channel 483 (b), converted into binary masks (c), classic network dissection is applied (d) and more complex logical forms are build up from the primitive concepts via beam search (e) [16].

Multiple compositional formulas are tested iteratively, based on the concepts identified via NetDissect. The one that maximize the IoU score is finally kept as the best explanation.

A more detailed explanation of the algorithm can be found in the box below.

Network Dissection extension to generate compositional explanations

Let's denote the probed network f. It maps input images x to vector representations $r \in \mathbb{R}^d$. The activations of an individual channel k in f is denoted $f_k(x)$. To explain this channel, Mu and Andreas generalize *Network dissection* as follows:

Let's denote the set of simple concept (in the Broden dataset) $C \in \mathcal{C}$. Each concept is a function $C : x \mapsto 0, 1$ indicating whether image x contains concept C. C can also be defined over pixels in an image, and thus indicates whether a pixel is contained in an instance of concept C.

Network dissection explains channel k by looking for the concept(s) C that is the most similar to its activations, given some measure of similarity δ:

$$\text{Explain-NetDissect}(n) = \arg\max_{C \in \mathcal{C}} \delta(k, C)$$

The original *Network dissection* method uses the IoU score as a measure of similarity. This procedure can only produce explanations from the fixed concept set $\mathcal{C}$.

The key contribution of *Compositional Explanations of Neurons* is to extend the set of possible explanations by including logical forms $\mathcal{L}(\mathcal{C})$ defined inductively over $\mathcal{C}$ via composition operations such as disjunction (`OR`), conjunction (`AND`), and negation (`NOT`).

Compositional Explanations of Neurons searches for the logical form $L \in \mathcal{L}(\mathcal{C})$ such that:

$$\text{Explain-Comp}(n) = \arg\max_{L \in \mathcal{L}(\mathcal{C})} IoU(k, L)$$

Since it's not possible to exhaustively search through the set of all possible compositional explanations $\mathcal{L}(\mathcal{C})$, in practice, a limit on formulas length N is set. $N = 1$ is equivalent to *Network dissection*. Formulas are iteratively constructed via **beam search**. At each step of beam search, the formulas already present in the beam are composed with new primitives, IoU of these new formulas are measured, and the top one is kept (see 43 (e)).

Using their method, Mu and Andreas found that channels learn compositional concepts that are sometimes semantically coherent, such as `(water OR river) AND NOT blue`, which designate a body of water which is not blue, and sometimes polysemantic (one channel fires for unrelated concepts), like `operation room OR castle OR bathroom`. Manual inspection revealed that 69% of channels learn meaningful combinations of concepts while the remaining 31% are polysemantic.

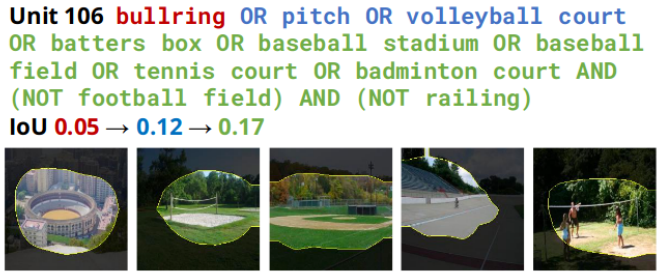


Figure 44: Compositional explanations indeed detect more precise ideas than simple pure concepts. The bullring detector which would have been predicted by *NetDissect* actually detects fields in general, as revealed by the length $N = 3$ (blue) and $N = 10$ (green) explanations [16].

9.3 Overcoming the densely annotated dataset to produce open-ended explanations

Network dissection and *Compositional explanations* both require to pre-define a set of fixed labels from which neuron descriptions are picked. In *Natural Language Description of Deep Visual Features* [13], Hernandez et al. introduce a procedure called **MILAN (Mutual-Information-guided Linguistic Annotation of Neurons) for automatically labeling neurons with compositional and open-ended natural language descriptions**.

An original aspect of the paper is that the labeling of neurons provided by MILAN is used for **model auditing**, as we'll see.

To characterize a specific neuron, MILAN initially examines its activity across thousands of images to identify the regions where the neuron exhibits the highest level of activation. Based on these image patches, it generates a natural language description which is as informative and specific to each neuron as possible.

A more detailed explanation of the MILAN algorithm can be found in the box below.

MILAN procedure

Given a neuron, MILAN generates a description by searching for a natural language string that maximizes pointwise mutual information with the image regions in which the neuron is active (see Figure 45).

Let $f : X \to Y$ be a network, and $f_i(x)$ be the activation of neuron i given an input image x. An **exemplar representation** of neuron i is the set of images that induce a high activation:

$$E_i = \{x \in X : f_i(x) > \eta_i\}$$

for some threshold parameter η_i.

The selected description d for neuron i is the one that maximizes its **pointwise mutual information (pmi) with the exemplar set** E_i:

$$\text{MILAN}(f_i) := \arg\max_d pmi(d; E_i) = \arg\max_d \log p(d|E_i) - \log p(d)$$

In the first term, $p(d|E_i)$ is the probability that a human would describe the image set E_i with d. The second term in the arg max, $-\log p(d)$, can be thought of as a penalization incentivizing the descriptions to be precise. Descriptions that are so general they could apply to a wide range of neurons have a high probability to occur (by definition), so the $-\log p(d)$ term somewhat prevents them from being selected.

To make this procedure work in practice, three prerequisites are needed:

- Constructing a tractable approximation to the exemplar set E_i. E_i can be a very large set, in practive it's restricted to the set of $k = 15$ images that cause the greatest activation in neuron i.
- Modeling the two distributions $p(d|E_i)$ and $p(d)$. Hernandez et al. constructed a dataset of captions that describe regions from images. It is used to train models which approximate the two distributions. $p(d|E_i)$ is approximated with the Show-Attend-Tell image description model [24] trained on MILANNOTATIONS. $p(d)$ is approximated with a two-layer LSTM language model also trained on MILANNOTATIONS.
- Finding a high quality description d in the infinite space of natural language strings.

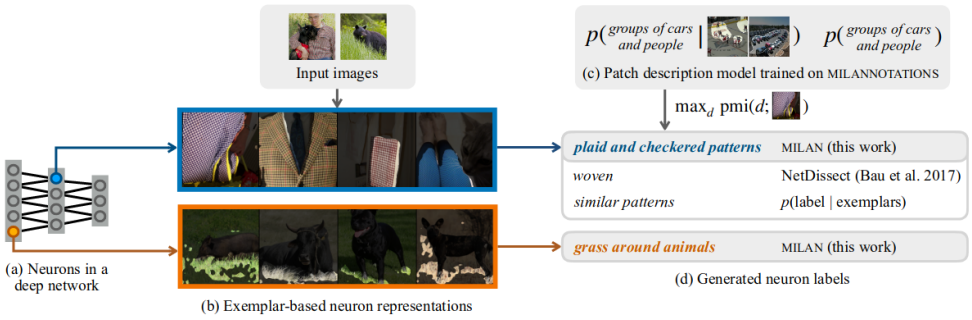


Figure 45: Overview of the MILAN procedure for automatically labeling neurons in a deep network with open-ended descriptions. Two neurons are being labeled, in blue and orange (a). For each of them, an exemplar set of input regions that activate it is collected (b). Previously, a pair of models were trained on MILANNOTATIONS to approximate the distributions $p(d|E_i)$ (a model outputting text from patches of images) and $p(d)$ (a model giving the probability of some text) (c). Using those two models, MILAN searches for a description that has high pointwise mutual information with the exemplar sets. The description generated for the blue neuron is thus "plaid and checkered patterns", while the one for the orange neuron is "grass around animals". The annotations generated are a lot more specific than those from *NetDissect* [13].

Channel labeling can serve as a foundation for model auditing. Hernandez et al. employ labeling to verify that a model trained on a dataset with blurred faces does not acquire any face-selective features. In comparison to the same model trained on unblurred faces, fewer neurons are dedicated to recognizing faces when they are blurred, but blurring does not completely eliminate neurons that are selective to unblurred faces. Importantly, blurring does not prevent models from learning features that are specific to particular demographic categories.

Figure 46: Models trained on blurred data (a) acquire neurons selective for unblurred faces. Hernandez et al. manually inspected neurons for which MILAN descriptions contain the words face, head, nose, eyes, and mouth (using exemplar sets containing unblurred images). They found that several neurons are not just face detectors, but rather exhibit selectivity to specific attributes, such as female faces (b) or Asian faces (c) [13].

References

[1] Julius Adebayo et al. *Sanity Checks for Saliency Maps*. arXiv:1810.03292 [cs, stat]. Nov. 2020. DOI: 10.48550/arXiv.1810.03292. URL: http://arxiv.org/abs/1810.03292 (visited on 06/02/2023).

[2] *An Overview of Early Vision in InceptionV1*. URL: https://distill.pub/2020/circuits/early-vision/ (visited on 06/01/2023).

[3] Aphex34. *English: typical CNN architecture*. Dec. 2015. URL: https://commons.wikimedia.org/wiki/File:Typical_cnn.png (visited on 06/12/2023).

[4] David Bau et al. *Network Dissection: Quantifying Interpretability of Deep Visual Representations*. arXiv:1704.05796 [cs]. Apr. 2017. DOI: 10.48550/arXiv.1704.05796. URL: http://arxiv.org/abs/1704.05796 (visited on 06/05/2023).

[5] Nick Cammarata et al. "Curve Detectors". en. In: *Distill* 5.6 (June 2020), e00024.003. ISSN: 2476-0757. DOI: 10.23915/distill.00024.003. URL: https://distill.pub/2020/circuits/curve-detectors (visited on 06/01/2023).

[6] Nick Cammarata et al. "Thread: Circuits". en. In: *Distill* 5.3 (Mar. 2020), e24. ISSN: 2476-0757. DOI: 10.23915/distill.00024. URL: https://distill.pub/2020/circuits (visited on 06/01/2023).

[7] Shan Carter et al. "Activation Atlas". en. In: *Distill* 4.3 (Mar. 2019), e15. ISSN: 2476-0757. DOI: 10.23915/distill.00015. URL: https://distill.pub/2019/activation-atlas (visited on 05/30/2023).

[8] Cecbur. *English: A filter (=kernel, neuron) in a convolutional artificial neural network. The filter takes a 9 pixel input for each color*. Feb. 2019. URL: https://commons.wikimedia.org/wiki/File:Convolutional_Neural_Network_with_Color_Image_Filter.gif (visited on 06/12/2023).

[9] Nolan Dey. *Breaking down Network Dissection*. en. Feb. 2020. URL: https://medium.com/analytics-vidhya/breaking-down-network-dissection-b4e3a6e9d5f2 (visited on 06/05/2023).

[10] Dumitru Erhan et al. "Visualizing Higher-Layer Features of a Deep Network". In: *Technical Report, Univeristé de Montréal* (Jan. 2009).

[11] evhub. "Chris Olah's views on AGI safety". en. In: (). URL: https://www.lesswrong.com/posts/X2i9dQQK3gETCyqh2/chris-olah-s-views-on-agi-safety (visited on 05/22/2023).

[12] Gabriel Goh et al. "Multimodal Neurons in Artificial Neural Networks". en. In: *Distill* 6.3 (Mar. 2021), e30. ISSN: 2476-0757. DOI: 10.23915/distill.00030. URL: https://distill.pub/2021/multimodal-neurons (visited on 05/30/2023).

[13] Evan Hernandez et al. *Natural Language Descriptions of Deep Visual Features*. arXiv:2201.11114 [cs]. Apr. 2022. DOI: 10.48550/arXiv.2201.11114. URL: http://arxiv.org/abs/2201.11114 (visited on 06/08/2023).

[14] Jacob Hilton et al. "Understanding RL Vision". en. In: *Distill* 5.11 (Nov. 2020), e29. ISSN: 2476-0757. DOI: 10.23915/distill.00029. URL: https://distill.pub/2020/understanding-rl-vision (visited on 06/03/2023).

[15] D. H. Hubel and T. N. Wiesel. "Receptive fields of single neurones in the cat's striate cortex". In: *The Journal of Physiology* 148.3 (Oct. 1959), pp. 574–591. ISSN: 0022-3751. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1363130/ (visited on 06/02/2023).

[16] Jesse Mu and Jacob Andreas. *Compositional Explanations of Neurons*. arXiv:2006.14032 [cs, stat]. Feb. 2021. DOI: 10.48550/arXiv.2006.14032. URL: http://arxiv.org/abs/2006.14032 (visited on 06/05/2023).

[17] Neel Nanda. "A Longlist of Theories of Impact for Interpretability". en. In: (). URL: https://www.lesswrong.com/posts/uK6sQCNMw8WKzJeCQ/a-longlist-of-theories-of-impact-for-interpretability (visited on 05/30/2023).

[18] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization". en. In: *Distill* 2.11 (Nov. 2017), e7. ISSN: 2476-0757. DOI: 10.23915/distill.00007. URL: https://distill.pub/2017/feature-visualization (visited on 05/23/2023).

[19] Chris Olah et al. “The Building Blocks of Interpretability”. en. In: *Distill* 3.3 (Mar. 2018), e10. ISSN: 2476-0757. DOI: 10.23915/distill.00010. URL: https://distill.pub/2018/building-blocks (visited on 05/30/2023).

[20] Chris Olah et al. “Zoom In: An Introduction to Circuits”. en. In: *Distill* 5.3 (Mar. 2020), e00024.001. ISSN: 2476-0757. DOI: 10.23915/distill.00024.001. URL: https://distill.pub/2020/circuits/zoom-in (visited on 06/01/2023).

[21] *Quantifying generalization in reinforcement learning*. en-US. URL: https://openai.com/research/quantifying-generalization-in-reinforcement-learning (visited on 06/03/2023).

[22] scasper. “EIS III: Broad Critiques of Interpretability Research”. en. In: (). URL: https://www.lesswrong.com/posts/gwG9uqw255gafjYN4/eis-iii-broad-critiques-of-interpretability-research (visited on 06/19/2023).

[23] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”. In: *International Journal of Computer Vision* 128.2 (Feb. 2020). arXiv:1610.02391 [cs], pp. 336–359. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://arxiv.org/abs/1610.02391 (visited on 05/27/2023).

[24] *Show, attend and tell | Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. URL: https://dl.acm.org/doi/10.5555/3045118.3045336 (visited on 06/12/2023).

[25] Richard Tomsett et al. “Sanity Checks for Saliency Metrics”. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04 (Apr. 2020). Number: 04, pp. 6021–6029. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i04.6064. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6064 (visited on 06/17/2023).