Interactive Visual Feature Search

Devon Ulrich, Ruth Fong
Princeton University
{dulrich, ruthfong}@princeton.edu

Abstract

Many visualization techniques have been created to help explain the behavior of convolutional neural networks (CNNs), but they largely consist of static diagrams that convey limited information. Interactive visualizations can provide more rich insights and allow users to more easily explore a model's behavior; however, they are typically not easily reusable and are specific to a particular model. We introduce Visual Feature Search, a novel interactive visualization that is generalizable to any model and can easily be incorporated into a researcher's workflow. Our tool allows a user to highlight an image region and search for images from a given dataset with the most similar CNN features. It supports searching through large image datasets with an efficient cache-based search implementation. We demonstrate how our tool elucidates different aspects of model behavior by performing experiments on supervised, self-supervised, and human-edited CNNs. We also release a portable Python library and several IPython notebooks to enable researchers to easily use our tool in their own experiments. Our code can be found at https://github.com/lookingglasslab/VisualFeatureSearch.

1. Introduction

Convolutional neural networks (CNNs) have been at the forefront of computer vision research for the past decade, yet they are difficult to understand due to their size and complexity. This makes it difficult for researchers to improve visual models (i.e. to improve accuracy or mitigate bias). To tackle this problem, many works introduced visualizations that help explain a model's behavior. For instance, Grad-CAM [34] and Network Dissection [6] visualize a CNN's outputs or intermediate features via static diagrams.

However, researchers have recently focused on creating *interactive* visualizations of CNNs, which can visualize more data in an easy-to-use way. Several recent works [26, 7] provide rich graphical interfaces that allow the user to analyze what CNN layers and neurons are looking for.

While these interactive tools are effective at explaining CNN behavior, they are generally designed for a handful of pre-selected models and cannot be used to visualize an arbitrary CNN. For instance, they cannot be applied to recently developed, self-supervised models or models trained for fine-grained classification. As a result, most interactive visualizations have few practical uses for researchers.

Figure 1. **Overview.** The user firsts selects an image and highlights a region within it via an interactive widget (left). Then, our tool searches through an image dataset and returns images (right) with the most similar CNN features to that of the highlighted region.

In this paper, we introduce Visual Feature Search (VFS), a novel interactive visualization that empowers any machine learning researcher to easily search for and explore the visual features of a model. Our tool is designed to be lightweight and flexible, so that users can quickly set up VFS to analyze the intermediate features of any CNN. Our visualization allows a user to highlight a free-form region in an image, and it searches for other images in a dataset that contain similar feature representations to the highlighted region and displays the most similar images (Fig. 1).

Our work most directly builds off of interactive similarity overlays (ISO) [13], which allows a user to hover over fixed-sized patches in an image and overlays a heatmap that highlights how similar other patches are to the currently highlighted patch. While both VFS and ISO are flexible and simple enough to be easily used and adapted for any model, our method differs from ISO in the following ways:

First, ISO only computes similarities between fixed-sized image patches while VFS allows the user to search for a variably-sized region. Second, ISO only shows feature similarity within a given image or a small, select set of images ($N \approx 10$) while VFS performs an efficient, global search within any large dataset ($N = 50k$).

We implement a cache-based search for intermediate CNN features, which can perform large-scale searches for similar features in less than a minute in a single GPU environment. As a result, VFS search results can be computed and displayed in real time, so researchers can quickly experiment with multiple query regions or images and can even compare the representations of multiple CNNs.

We include several experiments to highlight how VFS can be used to explain computer vision models and offer novel insights. Specifically, we compare the representations of supervised and self-supervised CNNs, analyze the behavior of models that are pretrained on the PASS dataset [3], investigate the models produced from editing classifiers [32], and visualize CNN features on out-of-domain images. We also release a Python library and several IPython notebooks to enable anyone to use our tool in their own experiments.

2. Related Work

Static interpretability tools. Most works on understanding CNNs have introduced static visualizations. The two most popular streams of work are attribution heatmaps and feature visualizations. Attribution heatmaps (e.g. Gradient [35] and Grad-CAM [34]) visualize which regions in an input image are important for a model's output decision and can be viewed as an external explanation of a model, in that it does not aim to explain internal model components [4, 14, 28, 30, 34, 35, 38, 45, 46, 47].

In contrast, feature visualization techniques visualize a component of the internal representation of a model, such as the visual patterns that most activate a CNN channel (e.g. Top Activated Patches [45], Net Dissect [6], and Activation Maximization [22, 25, 35]) or that are most similar to an image's activation tensor at a given layer (e.g. Feature Inversion [20, 25]). While our visualization is interactive, it is most similar to static feature visualization methods in that it visualizes which regions from real images yield the most similar activation patterns to those from a query image.

Interactive interpretability tools. While there are a number of interactive visualizations for CNNs [1, 5, 7, 13, 15, 18, 24, 26, 33, 37, 39, 41, 42, 44, 48], most are well-polished demos that cannot easily be extended to novel models and/or datasets. Two examples of large-scale interactive tools are Olah et al.'s Building Blocks [26] and OpenAI's Microscope [33]. Both tools include feature visualizations in their interfaces and allow the user interactively select between different image patches and/or CNN layers to customize the visualizations that are displayed. However, neither project can be easily adapted to visualize a custom CNN: Microscope explicitly states that it only supports a list of built-in models, while significant effort would be required to get Building Blocks to work on custom CNNs.

In comparison, Interactive Similarity Overlays [13] (ISO) is a more lightweight and adaptable tool for visualizing arbitrary CNNs. The tool searches for similar regions within an image by comparing patches of its feature maps, and its user interface allows users to quickly visualize what parts of an image are considered similar by a CNN. Google's TensorBoard [1], Teachable Machines [41], and What If [42] tools are also easily extendable to novel models and/or datasets; however, only TensorBoard visualizes internal components of a model at a basic level (e.g. distributions of feature activations), while the other two only visualize external relationships between model inputs and outputs. Our work is most similar to ISO, as it is an adaptable tool that visualizes internal components of any model.

Image Retrieval Content-based Image Retrieval (CBIR) is a longstanding area of research within computer vision. The goal of CBIR is to find a set of similar images to some query picture; this is typically done by extracting some features from a database of search images, and returning a subset of images from the database with the most similar embedded features to a given query [19]. Recent advancements in CBIR have included using deep learning methods to extract more semantically meaningful features from images [11]. Some image retrieval projects have successfully used feature maps from CNNs to search for similar images [9], which we use as inspiration for our work.

3. Approach

Visual Feature Search (VFS) allows users to perform a "reverse search" on a selected image region using CNN features and retrieves dataset images with the most similar features. We first formalize our search method and then discuss algorithmic details that enable efficient searches.

3.1. Similarity Search

Given a CNN, a layer within it, and a dataset of images to search through (e.g. ImageNet [31]), we provide the user with an interactive widget for selecting a query image and highlighting a free-form region in the image (Fig. 1, left). To search within the dataset for representationally similar image regions, we take the user's selected region and extract its corresponding feature map from a CNN. Similarly, we scan for regions of the same size in our dataset of images and compute their feature maps as well. We then compute cosine similarity scores to compare the features of the query image and search images, and we return the regions that are found to be most similar to the user's query (Fig. 2).

Formally, we accept a query image $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ and a region mask $\mathbf{m} \in \mathbb{R}^{H \times W}$ from the user. Let $f_l(\mathbf{x}) : \mathbb{R}^{H \times W \times 3} \mapsto \mathbb{R}^{H_l \times W_l \times D_l}$ be any function that converts an image into a feature map (i.e. the first l layers of a CNN).

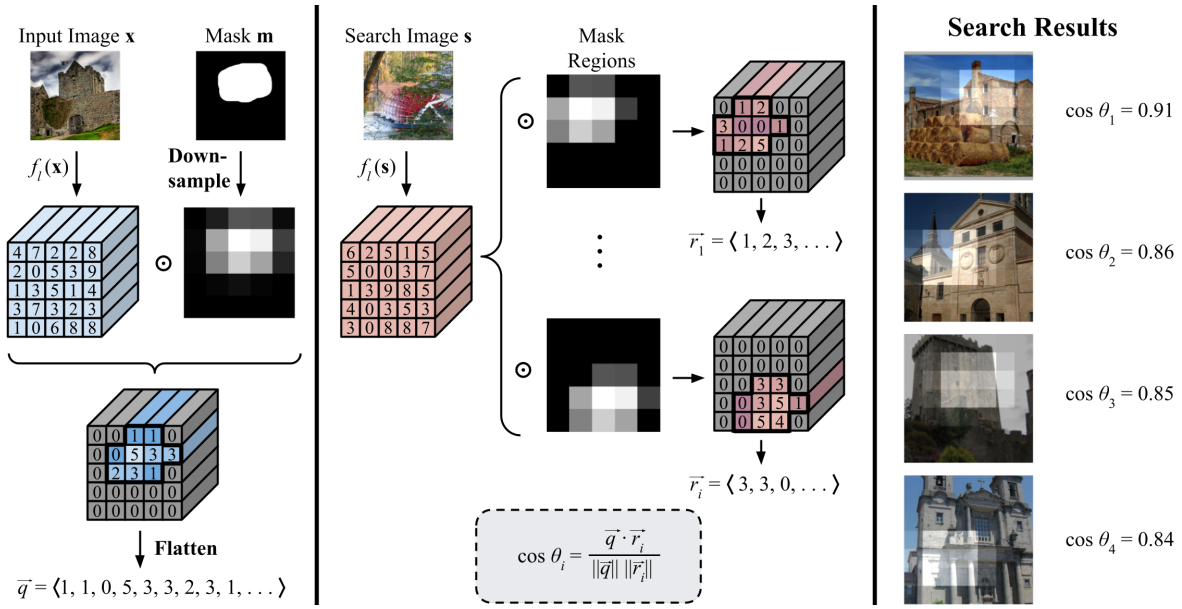

Figure 2. **Approach.** The user first selects an image $\mathbf{x}$ and uses an interactive widget to highlight a region of interest (i.e. mask $\mathbf{m}$). The image is transformed into a feature map by a function $f_l(\mathbf{x})$ (i.e. the first few layers of a CNN), and the feature map is spatially cropped and flattened according to the downsampled mask, resulting in a query vector $\vec{q}$ (left). A similar process is done for multiple images in a search dataset: each image's feature maps are computed and cropped into overlapping regions with the same dimensions as the search query. These regions are likewise flattened into vectors (i.e. $\vec{r_1}$, $\vec{r_2}$, etc.), which are subsequently compared to $\vec{q}$ via the cosine similarity function (middle). Once this is completed for all search images, the most similar regions are returned and displayed to the user (right).

We then downsample $\mathbf{m}$ to create an $H_l \times W_l$ sized mask $\mathbf{m}_l$, with the same width and height as our feature map. We define $\mathbf{z}$ to be the query's feature map after the mask $\mathbf{m}_l$ is applied to it. Specifically,

$$\mathbf{z}_{(i,j,k)} = f_l(\mathbf{x})_{(i,j,k)} \cdot \mathbf{m}_{l(i,j)} \tag{1}$$

Next, we create a flattened query vector $\vec{q}$. First, we create the set of vectors $Q := \{\mathbf{z}_{(i,j,*)} \mid \mathbf{m}_{l(i,j)} > 0\}$. These vectors correspond to every location that the user highlighted on the original image. Then, we concatenate all vectors in Q together:

$$\vec{q} = \text{concat}(Q) \tag{2}$$

This gives us the CNN feature representation of the selected region in the query image (Fig. 2, left).

We perform similar operations to obtain search vectors with which to compare $\vec{q}$ to. Let $\mathbf{s} \in \mathbb{R}^{H \times W \times 3}$ be an image in our search dataset; we can multiply its corresponding feature map with the mask $\mathbf{m}_l$:

$$\mathbf{t}_{(i,j,k)} = f_l(\mathbf{s})_{(i,j,k)} \cdot \mathbf{m}_{l(i-\alpha,j-\beta)} \tag{3}$$

Note that we are offsetting the mask by α rows and β columns: this allows us to scan for similar feature values across multiple locations in the search image (Figure 2, middle). We zero-pad $\mathbf{m}_l$ for this equation, so $\mathbf{m}_{l(u,v)} := 0$ if $u < 0, v < 0, u > H_l$, or $v > W_l$. Additionally, we only use values of $\alpha, \beta \in \mathbb{Z}$ such that all nonzero $\mathbf{m}_l$ values are *within the feature map's bounds*, i.e. if $\mathbf{m}_{l(i-\alpha,j-\beta)} > 0$, then $0 \leq i - \alpha < H_l$ and $0 \leq j - \beta < W_l$.

Thus, $\mathbf{t}$ represents one region within a search image's feature map that we can compare to the query. We construct a similar set of vectors $R := \{\mathbf{t}_{(i,j,*)} \mid \mathbf{m}_{l(i-\alpha,j-\beta)} > 0\}$. This allows us to create a concatenated region vector $\vec{r}$:

$$\vec{r} = \text{concat}(R) \tag{4}$$

Finally, we compute similarity scores between the features of the user's query $\vec{q}$ and that of a particular dataset region $\vec{r}$ using the cosine similarity function:

$$\cos\theta = \frac{\vec{q} \cdot \vec{r}}{\|\vec{q}\| \, \|\vec{r}\|} \tag{5}$$

With this information, we can sort all of the regions from our search dataset in order of similarity to the user's query, and then return and visualize the most similar images and their matched regions (Fig. 2, right).

3.2. Convolution Trick for Region Search

As previously stated, one of our goals is to scan through the search dataset and look for multiple candidate regions within each image. Equation (3) describes how to perform

this region-based scanning across each image; however, if we were to implement this method naïvely, then we would iteratively change α and β in a sliding-window approach, which would be inefficient due to its loops and sequential calculations.

An alternate method is to replace the sliding window with a convolution operation, which is highly parallelizable and can be run quickly on a GPU via PyTorch [27]. To do this, we first take the 3D tensor $\mathbf{z}$ from Equation (1) and apply the mask $\mathbf{m}_l$ on it once more. We also *crop* the tensor, which we define as removing all rows/columns on the exterior of the feature map which only contain zero-valued elements. Let the resulting tensor be $\mathbf{z}'$, where:

$$\mathbf{z}'_{(i,j,k)} = \text{Crop}(\mathbf{z}_{(i,j,k)} \cdot \mathbf{m}_{l(i,j)}) \tag{6}$$

$\mathbf{z}'$ has dimensions $H' \times W' \times D_l$, where $H' \leq H_l$ and $W' \leq W_l$. We use $\mathbf{z}'$ as a 2D convolutional filter; we apply it onto a feature map $f_l(\mathbf{s})$ that we wish to compare our query to.

$$\mathbf{c} := f_l(\mathbf{s}) * \mathbf{z}' \tag{7}$$

Each element $\mathbf{c}_{(a,b)}$ is equivalent to the inner product $\vec{q} \cdot \vec{r}_i$, for a unique region vector $\vec{r}_i$ within the search image's feature map. We can perform a very similar convolution operation to obtain the magnitudes of each $\vec{r}_i$, so we can thus use Equation (5) to compute the cosine similarities for all searchable regions within the image $\mathbf{s}$ without the need of a sliding window.

3.3. Caching Precomputed Feature Maps

Another performance improvement that we use in our implementation of Visual Feature Search (VFS) is the precomputation of searchable feature maps. Computing the feature maps that we search through (i.e. $f_l(\mathbf{s})$) can be computationally demanding since we generally use the convolutional layers of large CNNs as our function f_l. As a result, if we want to use VFS to find the most similar CNN activations across a large dataset of images (i.e. all $50k$ images in the ImageNet validation set [31]), then simply computing all of the required feature maps for comparison can take over 20 minutes in a single-GPU setting like Google Colab.

Additionally, storing all computed feature maps in a computer's memory may become infeasable for large datasets. For instance, say we wanted to search across feature maps with dimensions $14 \times 14 \times 512$ (i.e. VGG16 conv5 features [36]). If we were to compute these feature maps for 50,000 images and store everything directly in memory, we would need about 37 GB of RAM to do so.

To mitigate these limitations, we turn to precomputing feature maps and storing their values on disk. We first select an intermediate layer from a CNN that we wish to study, and then we iteratively compute the search dataset's feature maps at this layer. While we are doing this, we save the computed feature map tensors to disk in a compressed format. We use Zarr [23], a Python library for quickly saving and loading multidimensional tensor data, for this task. After we complete this precomputation, we store all feature maps in a uploaded compressed archive that can be downloaded by others.

When we wish to retrieve these precomputed values, we simply download the archive and load the feature maps to memory as needed. This allows us to both load feature maps without needing to extract them directly from a CNN and also efficiently transfer data from a computer's disk to its memory, which substantially reduces the computational and memory costs for searching across a large dataset. When loading $50k$ feature maps of size $7 \times 7 \times 512$ on Google Colab, we find that computing the maps from scratch takes 20 minutes on average with ResNet50, while loading from a Zarr cache takes just 31 seconds on average. Accessing feature maps in memory is the primary bottleneck of our implementation, and performing feature searches takes approximately the same amount of time with and without caching. When enough RAM is available to store all feature maps in memory at once, VFS takes just three seconds to perform the same search.

3.4. Interactive User Interface

We provide the user with an interactive IPython Notebook widget to carry out the region selection and searching. Our widget implements its interactive UI by injecting HTML and CSS into an IPython cell and is based off of the GANPaint widget [5], which allows a user to highlight a free-form image region that is then edited by a GAN.

With our widget, the user can first select a query image from a gallery of images. The user can then highlight a region of interest in the query image and then perform visual feature search on their selected model, dataset, and query. This interactivity enables users to quickly visualize how models encode different image regions in an intuitive way. Our highlighting widget can be seen in Figure 1 (left).

4. Experiments

In this section, we describe four experiments that showcase how Visual Feature Search (VFS) can be used to understand diverse models and domanis. Our experiments focus on understanding how models process out-of-distribution images from alternate datasets, as well as analyzing how self-supervised learning and classifier editing differ from standard supervised training in ResNet50 [16]. All experiments are implemented with our Python library for VFS, as well as with PyTorch [27].

The experiments we include here were chosen in part because they involve recently published models and/or datasets. Very few of the previously mentioned interactive

visualizations are able to be easily used on novel CNNs or datasets, so they are generally not able to assist researchers in analyzing more recent advancements in computer vision. In contrast, VFS is easily adaptable to new models and image datasets, so it is well-suited to provide novel insights on such works.

Selecting layers to study. Unless otherwise stated, the following experiments use ResNet50 models [16]. In order to analyze these models with VFS, we must choose the specific layer(s) from which to extract feature maps. While it is commonplace to study intermediate features immediately after one of ResNet's five conv blocks, we instead choose to analyze features from the second to last convolutional layer within these blocks. For instance, rather than studying features after conv4, block 6, layer 3, we extract features from conv4, block 6, layer 2. This is because the features after layer 3 is four times larger than the features after layer 2; if we were to extract the features after conv4 for all $50k$ ImageNet validation images, the features would take up approximately 75 GB uncompressed.

We instead opt to make our experiments more portable and available on low-memory environments such as Google Colab, so we choose to use feature data after the 2nd convolution within each residual block. For brevity, we will simply call these features by their "convX" groups; for the remainder of this section, it can be assumed that we take the penultimate feature maps from such ResNet conv layers.

4.1. Out-Of-Domain Images

One potential use case of VFS is to understand how robust a model is when presented with novel images. To demonstrate this, we visualize a ResNet50 model trained on ImageNet [31] using several in-domain and out-of-domain (o.o.d.) queries. We choose to visualize the conv5 features from ResNet50 and use the ILSVRC-12 validation set as our searchable database for this experiment. We then select query images from three different datasets: the in-domain images come from the ILSVRC-12 test set [31], while the o.o.d. images are obtained from ImageNet-A [17] and ImageNet-Sketch [40]. ImageNet-A and ImageNet-Sketch both contain images with the same class labels as ImageNet; however, ImageNet-A consists of natural images that ImageNet-trained CNNs tend to misclassify, while ImageNet-Sketch consists of illustrations of objects.

Several queries and VFS results are displayed in Fig. 3. For each set of queries, we use query images from all three datasets that share the same ground truth label (specifically, we use query images of mosques, unicycles, and bell peppers). Our results show that the in-domain queries are generally most similar to other images of the same class: of the three in-domain queries that we visualize (rows **a**, **d**, **g**), only one nearest neighbor result is from a different ground truth class ('dome' instead of 'mosque' in row **a**). Out-of-domain queries tend to have less related nearest neighbors, which indicates that the model struggles to recognize the subjects of these queries. For instance, the ImageNet-A mosque query in row **b** has conv5 features most similar to those of beacons and towers, as opposed to other mosques.

It can also be observed that the irregular colors and textures in ImageNet-Sketch images tend to confuse the model.

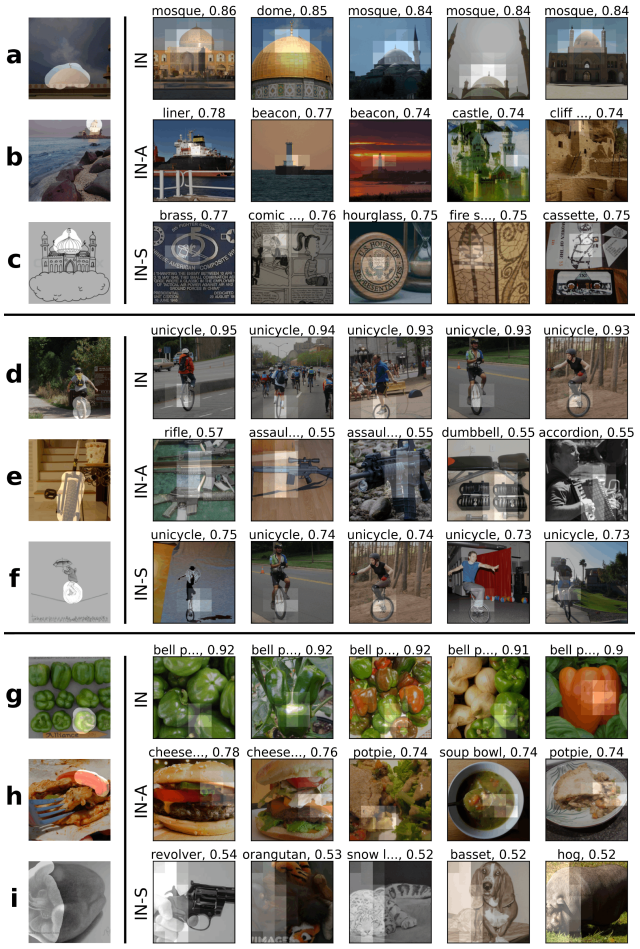

Figure 3. **In-domain vs. out-of-domain (o.o.d.) images.** Three sets of queries and results are shown. Each group contains three rows for queries from ImageNet (in-domain; rows **a**, **d**, **g**), ImageNet-A (o.o.d.; **b**, **e**, **h**) [17], and ImageNet-Sketch (o.o.d.; **c**, **f**, **i**) [40], from top to bottom. The top-5 nearest neighbors using ImageNet-trained ResNet50 features are displayed for each query, and each resulting image includes its ground-truth label and its features' cosine similarity to that of the query region. Rows **a**-**c** show 'mosque' query images, **d**-**f** show 'unicycle' images, and **g**-**i** show 'bell pepper' images. In these examples, the nearest neighbors of the o.o.d. images have lower cosine similarity scores than those of the in-domain images, and they often possess a different ground-truth label than that of the query image.

The mosque sketch (row **c**), for example, is computed to be most similar to other illustrations, despite these images not depicting similar buildings. Similarly, the bell pepper sketch (row **i**) is determined to be most similar to image patches that are black-and-white but do not contain bell peppers or related objects. However, this is not the case for all illustrated images, as the unicycle wheel query has nearest neighbors of other unicycle wheels (row **f**). Overall, our qualitative searches using VFS suggest that the nearest neighbors of o.o.d. images are less similar to one another than those of in-domain images (i.e. they have lower cosine similarity scores and often have different ground truth labels).

4.2. Editing Classifiers

Santurkar and Tsipras, et al. recently introduced a method for editing CNN classifiers to correct for systematic mistakes [32]. For instance, they found that a VGG16 ImageNet classifer consistently misclassifies vehicles that are on a snowy surface. The model generally labels these images as snowmobiles or snowplows; the researchers were able to successfully minimize the mistake by updating the classifier's weights such that model treats snowy terrain as if it were asphalt. We use VFS to explore the original and edited VGG models in the "vehicles on snow" example and how the edit affected the model's features (Fig. 4).

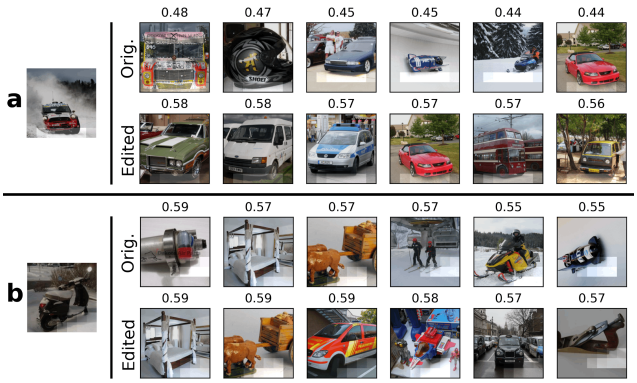

Figure 4. **Before vs. after model editing for "vehicles on snow" error.** For each query image, the top row (orig.) is from an ImageNet-trained VGG16 model that typically misclassifies vehicles on snowy surfaces, and the bottom row (edited) is from the model *after* it has been edited to correct for the "vehicles on snow" error [32]. Both query images were incorrectly classified by the original model but correctly classified by the edited one; cosine similarity scores are shown above each search image.Here, the edited model's nearest neighbors are more likely to contain vehicles on roads and less likely to contain objects on ice/snow surfaces compared to those from the original model, suggesting that the model edit encouraged vehicles on snow to be more similar to other vehicles on roads than to other objects on snow.

Our visualizations suggest that the snow to asphalt edit worked and has a noticeable effect on intermediate features. We highlight the snowy ground in both queries in Fig. 4. For query **a**, the original model does not have a clear understanding that the ground should be treated like an asphalt road; its nearest neighbors include other surfaces such as snow and ice. However, the edited model correctly relates the highlighted region to other roads with cars on them.

Query **b** has a similarly apparent improvement between the original and edited model. The original nearest neighbors all contain white surfaces, some of which are ice and snow. In contrast, the edited model's top-6 nearest neighbors include three regions of asphalt roads and no longer include ice/snow surfaces. Thus, these search results suggest that the model edit successfully changed the feature representation such that vehicles on snow are now more similar to other vehicles on asphalt than other objects on snow.

4.3. Supervised vs. Self-Supervised Models

Self-supervised learning has become an increasingly popular research area, as it promises to leverage the power of extremely large, unlabelled datasets to supersede performance of fully supervised models. SimCLR [8] is a recent self-supervised paradigm that uses contrastive learning to learn a visual representation of images by encouraging visually similar examples to be near each other in feature space and dissimilar ones to be far from one another. In this experiment, we utilize VFS to compare models pretrained via SimCLR self-supervision vs. standard full-supervision.

We use a ResNet50 model trained on ImageNet via SimCLR from PyTorch Lightning Bolts [12]. We also use a supervised ImageNet-trained ResNet50 as a baseline. We then select three layers to study: the SimCLR model's conv5, the supervised model's conv4, and the supervised model's conv5 layers. To use VFS, we pick query regions from images in the ILSVRC test set, and we search for similar features across the ILSVRC validation set [31].

Three sets of VFS search results are shown in Fig. 5. Within each set of results, the three rows show the most similar images using SimCLR conv5 features, supervised conv4 features, and supervised conv5 features, respectively. These results suggest that SimCLR successfully extracts semantic and geometric data from the query images: it finds the bird query most similar to other birds, and also finds the car wheel most similar to other car wheels.

Another observation is that SimCLR features appear to be very similar to supervised conv4 features: their resulting VFS patches look very similar, and they both encode a combination of semantic and geometric information from the query images. In contrast, supervised conv5 encodes primarily semantic data from images, as its VFS results generally include photographs of the same object class, but in different orientations or scenes. We confirmed this trend

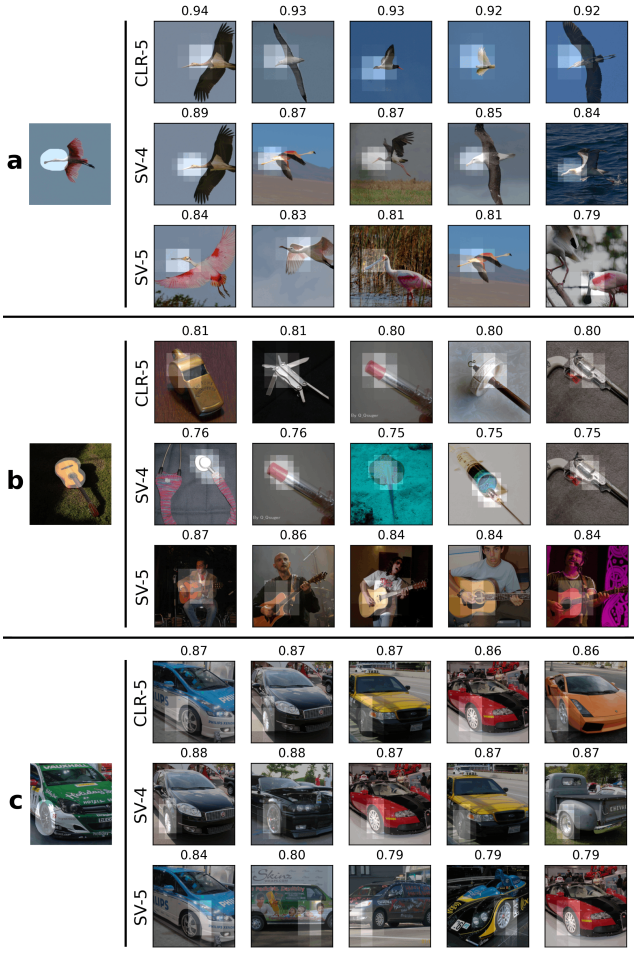

Figure 5. **Self-supervised vs. supervised features.** We include results for three query images (**a** **b**, and **c**). For each query, the top row (CLR-5) shows the most similar regions when using SIMCLR [8] self-supervised ResNet50 conv5 features, the middle row (SV-4) from using supervised ResNet50 conv4 features, and the bottom row (SV-5) from using supervised ResNet50 conv5 features. The SimCLR conv5 nearest neighbors appear to be more similar to the supervised conv4 neighbors than supervised conv5 ones; however, in quantitative experiments, we observe that SimCLR conv5 is more similar to supervised conv5 (Fig. 6).

among additional VFS queries and results (see supp. mat.), and we hypothesized that SimCLR conv5 is more representationally similar to supervised conv4 than conv5.

To test this hypothesis, we run VFS on 100 ImageNet test images and record the top-5 nearest neighbors for each query. We input the entire image into our feature search (i.e. our masks contain all ones). We then analyze results for the supervised conv4 layer, the supervised conv5 layer, and the SimCLR conv5 layer. Two metrics we investigate are the average number of unique ground-truth classes in each layer's nearest neighbors and the average number of

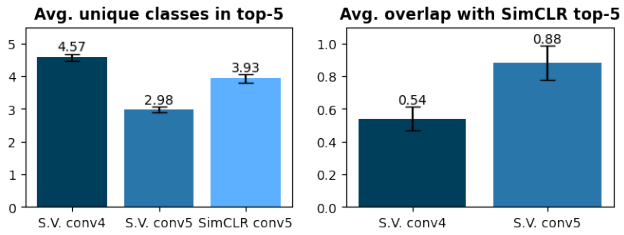

Figure 6. **Self-supervised vs. supervised quantitative comparison.** Left: the average number of unique classes within each layer's top-5 nearest neighbors. Right: the average number of images in common between SimCLR [8] conv5 neighbors and supervised (S.V.) conv4 and conv5 neighbors. This experiment was run on $N = 100$ images; standard error bars are also shown. These results suggest that SimCLR conv5 is more similar to supervised conv5 than supervised conv4.

images in common between the SimCLR conv5 results and the supervised conv4 vs. conv5 results (Fig. 6).

These results show that SimCLR conv5 is more similar to supervised conv5 than anticipated. SimCLR conv5 and supervised conv5's VFS results share 0.88 images on average, in contrast to the average 0.54 images shared between SimCLR conv5 and supervised conv4. Both values are well below the maximum of 5 shared images, so SimCLR is still quite different from either supervised layer.

SimCLR also returns images from 3.93 different classes on average in VFS, which is in between the respective values for supervised conv4 and conv5. This suggests that SimCLR conv5 is representationally between the supervised conv4 and conv5. These somewhat surprising results highlight the importance of substantiating any insights from our qualitative visualization tool with quantitative experiments.

4.4. ImageNet vs. PASS Pretraining

We next study how the dataset choice for model training affects its feature representation. To mitigate privacy concerns of training on images with humans, Asano et al. recently introduced PASS, a large dataset of unlabelled images that does not contain human faces or body parts [3]. PASS is meant be used as an ImageNet replacement for self-supervised learning and has been shown to perform as well as ImageNet-trained models on human-centric tasks (i.e. pose estimation). We compare how two ResNet50 models trained on PASS vs. ImageNet via MoCo-v2 [10] self-supervision process images with human faces.

Our qualitative results from performing VFS on both models are included in Fig. 7. The most notable observation is that the PASS-trained model is able to accurately match face queries to other faces in the dataset, despite never being trained with images of humans. However, the PASS model features appear to be less precise than the corresponding Im-

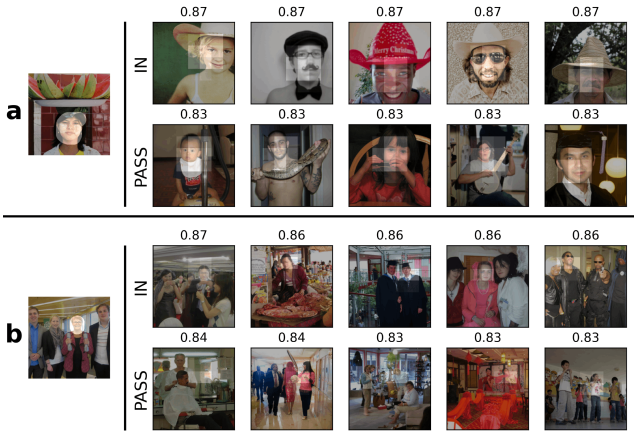

Figure 7. **ImageNet vs. PASS datasets for self-supervised pretraining.** Query images of human faces were visualized in two ResNet50 models trained on PASS [3] vs. ImageNet [31] via MoCo-v2 self-supervision [10] (note: PASS does not contain images with human faces). The top row shows nearest neighbors from the ImageNet model, while the bottom row shows results from the PASS model. Visually, while both models find other faces to be similar to the query face, some PASS results appear less precisely localized on faces (query **b**).

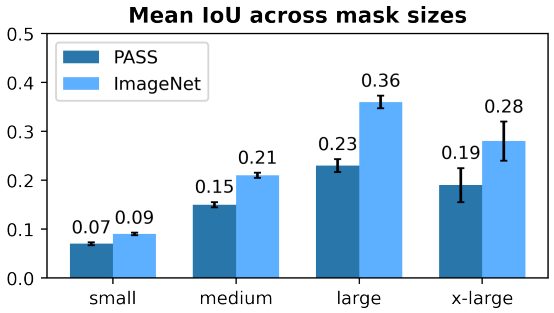

Figure 8. **Face detection using PASS- vs. ImageNet-trained models.** ImageNet query images with faces ($N = 508$ images) were fed into MoCO-v2 models trained on ImageNet [31] and PASS [3]. Using VFS, the nearest neighbor regions were compared to ground-truth face bounding boxes [43]. The results are shown by query face size, and mean IoU and standard error are reported. Both models appear to detect faces, although ImageNet pretraining yields more accurate nearest neighbor results.

ageNet model features for some inputs: for instance, query **b** in Fig. 7 corresponds to nearest neighbor regions from PASS model features do not include any faces.

We found similar trends where smaller face queries led to less faces being present in the PASS model's nearest neighbors, at least compared to the ImageNet model's nearest neighbors (see supp. mat.). To test if this trend is true on a larger scale, we performed VFS with 508 faces from the ILSVRC-12 test set; we used data from Yang et al. [43] to find these faces and use them as query regions.

We then analyze the nearest neighbor regions for each query and create a bounding box around these neighbors' masks; this allows us to compute the IoU between the nearest neighbor bounding boxes and the ground truth faces for each image, which acts as a rough indicator for how accurately each model's features encode faces. Since we are using conv5 features, the resulting masks have a resolution of 7×7: this means that the sizes of faces can affect the resulting IoUs even in a perfect classifier, so we binned IoU results into four groups based on the area of query face size: small ($< 1k$ sq. pix.), medium ($1k$–$5k$ sq. pix.), large ($5k$–$20k$ sq. pix.), and extra large ($> 20k$ sq. pix.).

Fig. 8 reports the mean IoUs and shows that the PASS-trained model is less accurate at finding faces across all four face sizes compared to the ImageNet-trained model. Larger face sizes do have increased IoUs for both models, but this is partially due to the conv5 features' limited resolution. Nonetheless, these results confirm that PASS-trained model features are capable of encoding and detecting faces.

5. Conclusion

In summary, we propose a new interactive tool for understanding the intermediate activations of CNNs. Many existing interactive visualizations can not be easily applied to new models and/or datasets; thus, they are often not utilized by researchers as regular research tool. We demonstrate through our experiments that Visual Feature Search is much more flexible in comparison: it can be used to quickly visualize new models and/or datasets, and we hope that this flexibility allows other researchers to use to our visualization to better understand their own models.

Limitations. The main limitation of our work is its qualitative nature: While our tool is useful for developing intuition about how a model works; any hypotheses should be substantiated with quantitative experiments. Prior work [2] showed that qualitative heatmap visualizations did not accurately explain a specific model's weights [2] or output prediction [21, 29]. Instead, heatmaps were typically used to *confirm* preconceived notions of what should be salient in an image. Similarly, we caution users from using our tool alone to confirm hypotheses and suggest that it be used as an exploratory tool in conjunction with other visualizations and experiments (as done in Secs. 4.3 and 4.4).

Acknowledgements. We are grateful for support from the Open Philanthropy Project (RF), the Princeton Engineering Project X Fund (RF), and the Princeton SEAS IW Funding (DW). We thank David Bau, Sunnie S. Y. Kim, and Indu Panigrahi for helpful discussions and/or feedback on our tool. We also thank the authors of [5], whose widget we based our highlighting widget on. We are grateful to the authors of [**?**] for open-sourcing their code and/or trained models.

References

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[2] Julius Adebayo, Justin Gilmer, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *NeurIPS*, 2018.

[3] Yuki M. Asano, Christian Rupprecht, Andrew Zisserman, and Andrea Vedaldi. Pass: An imagenet replacement for self-supervised pretraining without humans. *NeurIPS Track on Datasets and Benchmarks*, 2021.

[4] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

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

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

[7] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 2019.

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

[9] Wei Chen, Yu Liu, Weiping Wang, Erwin Bakker, Theodoros Georgiou, Paul Fieguth, Li Liu, and Michael S. Lew. Deep learning for instance retrieval: A survey, 2021.

[10] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[11] Shiv Ram Dubey. A decade survey of content based image retrieval using deep learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2021.

[12] William Falcon and Kyunghyun Cho. A framework for contrastive self-supervised learning and designing a new approach. *arXiv*, 2020.

[13] Ruth Fong, Alexander Mordvintsev, Andrea Vedaldi, and Chris Olah. Interactive similarity overlays. In *VISxAI*, 2021.

[14] Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *ICCV*, 2017.

[15] Adam W Harley. An interactive node-link visualization of convolutional neural networks. In *International Symposium on Visual Computing (ISVC)*, 2015.

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

[17] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *CVPR*, 2021.

[18] Fred Hohman, Nathan Hodas, and Duen Horng Chau. Shapeshop: Towards understanding deep learning representations via interactive experimentation. In *CHI Extended Abstracts*, 2017.

[19] Manesh Kokare, B N Chatterji, and P K Biswas. A survey on current content based image retrieval methods. *IETE Journal of Research*, 48(3-4):261–271, 2002.

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

[21] Aravindh Mahendran and Andrea Vedaldi. Salient deconvolutional networks. In *ECCV*, 2016.

[22] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *IJCV*, 2016.

[23] Alistair Miles, John Kirkham, Martin Durant, James Bourbeau, Tarik Onalan, Joe Hamman, Zain Patel, shikharsg, Matthew Rocklin, raphael dussin, Vincent Schut, Elliott Sales de Andrade, Ryan Abernathey, Charles Noyes, sbalmer, pyup.io bot, Tommy Tran, Stephan Saalfeld, Justin Swaney, Josh Moore, Joe Jevnik, Jerome Kelleher, Jan Funke, George Sakkis, Chris Barnes, and Anderson Banihirwe. zarr-developers/zarr-python: v2.4.0, Jan. 2020.

[24] Andrew P Norton and Yanjun Qi. Adversarial-playground: A visualization suite showing how adversarial examples fool deep learning. In *VizSec*, 2017.

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

[26] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary 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. In *NeurIPS*. 2019.

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

[29] Sylvestre-Alvise Rebuffi, Ruth Fong, Xu Ji, and Andrea Vedaldi. There and back again: Revisiting backpropagation saliency methods. In *CVPR*, 2020.

[30] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" explaining the predictions of any classifier. In *KDD*, 2016.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[32] Shibani Santurkar, Dimitris Tsipras, Mahalaxmi Elango, David Bau, Antonio Torralba, and Aleksander Madry. Editing a classifier by rewriting its prediction rules. In *NeurIPS*, 2021.

[33] Ludwig Schubert, Michael Petrov, Shan Carter, Nick Cammarata, Gabriel Goh, and Chris Olah. OpenAI Microscope, 2020.

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

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

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

[37] Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viégas, and Martin Wattenberg. Direct-manipulation visualization of deep networks. *arXiv*, 2017.

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

[39] Antonio Torralba. Drawnet. Online, 2017.

[40] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *NeurIPS*, 2019.

[41] Barron Webster. Now anyone can explore machine learning, no coding required. Online, 2017.

[42] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. The what-if tool: Interactive probing of machine learning models. *TCVG*, 2019.

[43] Kaiyu Yang, Jacqueline Yau, Li Fei-Fei, Jia Deng, and Olga Russakovsky. A study of face obfuscation in imagenet. In *ICML*, 2022.

[44] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *ICML Workshop*, 2015.

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

[46] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *IJCV*, 2018.

[47] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *CVPR*, 2016.

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

Supplementary Material

A. Additional Figures

In this section, we include additional figures (Figs. 9 and 10) from our qualitative experiments in Sections 4.3 and 4.4. We visualize the same model layers, datasets, and training paradigms as we do in Figures 5 and 7 in our paper.

Our additional results for supervised vs. self-supervised features (Fig. 9) display the same trends as our original visuals. Specifically, we observe that the SimCLR-conv5 nearest neighbors and the supervised-conv4 neighbors share a combination of semantic and geometric details, while supervied-conv5 neighbors are primarily related by semantics (i.e. ground truth class labels). For instance, the SimCLR-conv5 and supervised-conv4 neighbors for query **a** include pictures of several bird species while the supervised-conv5 neighbors only contain eagles. Similarly, several dog breeds are included in the SimCLR-conv5 and supervised-conv4 neighbors for query **b**, while only one breed is present in supervised-conv5 results.

The results in Figure 10 further support the trend we observed for ImageNet vs. PASS trained models in our paper. Specifically, the ImageNet-trained model is able to relate feature data of faces to each other more accurately than the PASS-trained model is, especially for smaller queries such as query **c**. However, these results also provide more evidence that the PASS-trained model is able to match faces via visual feature search to a surprising extent, given that it was trained on images that do not contain any human faces or body parts.

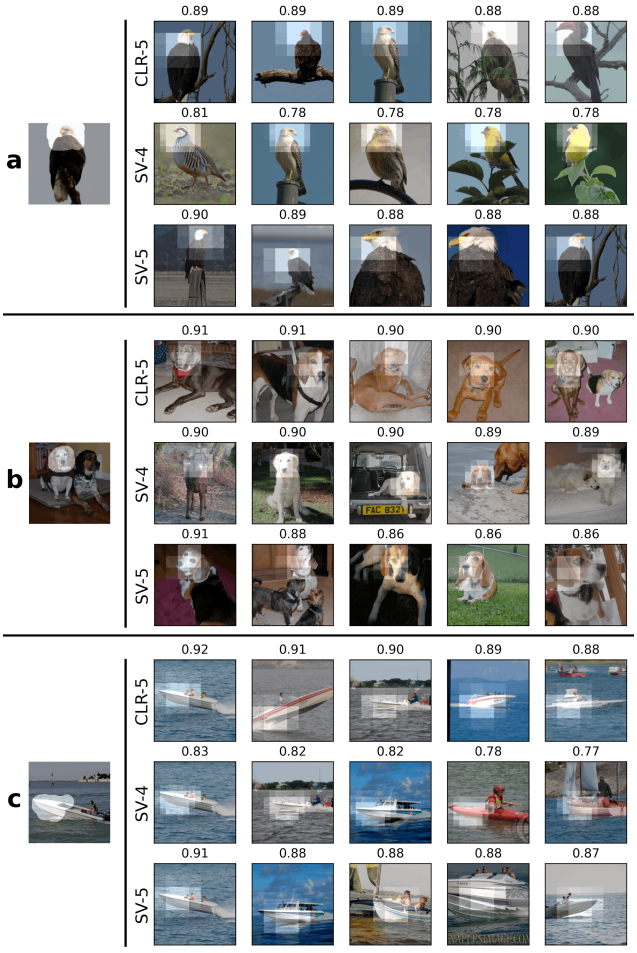

Figure 9. **Additional self-supervised vs. supervised features.** Three additional queries and sets of results are shown. We use models trained via SimCLR and traditional supervision for feature search; for more details, see Section 4.3 and Figure 5 in our paper. These visuals are consistent with our results in Figure 5, since the SimCLR-conv5 nearest neighbors appear to be more similar to the supervised-conv4 neighbors than those of supervised-conv5.

B. Code

B.1. Python Library

A copy of our library is included in the `code/vissearch` directory. We provide five modules for feature caching, data loading/transforming,

Figure 10. **Additional ImageNet vs. PASS pretraining features.** We use three additional queries of human faces for these visualizations, which were discussed in Section 4.4 and Figure 7 in our paper. These results further show that both models find other faces to be similar to the queries, but the PASS-trained model appears to be less accurate on queries with small faces (query **c**).

searching for nearest neighbors, providing utility functions, and displaying interactive widgets.

B.2. Interactive Demo

We provide an interactive IPython notebook for performing visual feature search on a small-scale model and dataset. The notebook is located at `code/InteractiveDemo.ipynb`; we visualize features from a MobileNet_v2 model trained on ImageNet in the notebook, and we use the ImageNet validation set for nearest neighbor searching. We choose to provide a small-scale notebook to be run locally (on a CPU only) as this is the easiest way to share a demo that is (1) anonymous and (2) able to run on any computer.

In order to run the demo, it is first required to download the ImageNet validation set. We use a subset of the validation set (1,000 images) as our searchable database for visual feature search. To get the dataset:

1. Go to image-net.org and login or signup for access.
2. Go to the following URL: `https://image-net.org/challenges/LSVRC/2012/2012-downloads.php`
3. Download "ILSVRC2012_img_val.tar" file from the link titled "Validation images (all tasks)" under the "Images" header (The file size should be 6.3Gb).
4. Extract the .tar file, and set IMAGENET_VAL_DIR variable in the notebook to the directory containing the extracted images.

For running our demo, we recommend using anaconda/miniconda to create a temporary environment. To set everything up, complete the following steps:

1. Run `conda create -n tmp-vfs`
2. Run `conda activate tmp-vfs`, followed by `conda install jupyter`
3. Start a Jupyter notebook server by running `jupyter notebook`. **NOTE:** other notebook environments, such as Jupyter Lab, may not work with our interactive widget. We have only tested our library for Google Colab and `jupyter notebook`.
4. Open the notebook file in Jupyter Notebook, and run all code blocks to install packages and run the search demo.

C. Video

We also include a video recording of our interactive search environment. The video demonstrates how the interactive widget is used to search for regions of interest within images and find nearest neighbors in activation space. For this demo, we use a ResNet50 trained on ImageNet, and we search through all $50,000$ images in the ImageNet validation set to find nearest neighbors. As a result, the resulting nearest neighbors are much more similar to the query than nearest neighbors in our small-scale demo are.