Interactive Visual Feature Search

Devon Ulrich
Jane Street*
dulrich@alumni.princeton.edu

Ruth Fong
Princeton University
ruthfong@cs.princeton.edu

Abstract

Many visualization techniques have been created to explain the behavior of computer vision models, but they largely consist of static diagrams that convey limited information. Interactive visualizations allow users to more easily interpret a model's behavior, but most are not reusable for new models. We introduce Visual Feature Search, a novel interactive visualization that is adaptable to most modern vision models 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 model features. We demonstrate how our tool elucidates different aspects of model behavior by performing experiments on a range of applications, such as in medical imaging and wildlife classification. Our tool is open source and can be used by others to interpret their own models.[1]

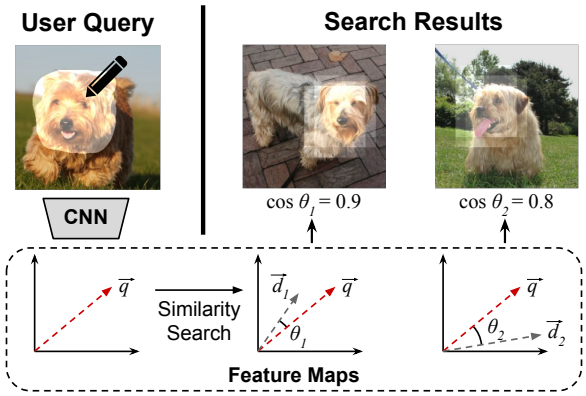

Figure 1. **Overview.** The user highlights a patch in the query image via our interactive tool (left); then, our tool computes a model's intermediate features for the selected query ($\vec{q}$) and searches through an image dataset to find patches with similar features ($\vec{d}_i$) via cosine similarity. Patches with the highest similarity scores to $\vec{q}$ in feature space are displayed (right) to visualize the model's intermediate features for the query. See Section 2 for more details.

1. Introduction

Computer vision models such as convolutional neural networks (CNNs) and vision transformers are notoriously hard to interpret due to their size and complexity. Various techniques have been proposed to help visualize and "explain" these models with static figures; for instance, attribution heatmaps [3, 13, 31, 32, 39] such as Grad-CAM [30] visualize which input image regions are important for a model's output decision, and feature visualization techniques help explain internal aspects of models (e.g. what visual stimuli most activates a given neuron) [4, 21, 24, 31, 37].

However, researchers have recently focused on creating *interactive* visualizations of CNNs, which can present more data in an easy-to-use way. Several works [5, 8, 15, 20, 23, 25, 29] provide graphical interfaces that allow the user to interact with CNNs and produce rich visualizations. While these tools are effective at explaining CNN behavior, they are generally only designed for a handful of pre-selected models. A key criterion for the adoption of interpretability techniques is how easy they are to incorporate into a researcher's workflow; significant effort is required to utilize these interactive tools in new experiments, so they are unfortunately not widely used in practice.

Some works, such as Teachable Machines [35], What If [36], TensorBoard [1], and Interactive Similarity Overlays (ISO) [14] are more lightweight and easy to integrate with new models, but only the latter two can visualize internal feature data. TensorBoard only supports basic feature visualizations (i.e. plotting distributions of activations) while ISO enables users to qualitatively compare spatial CNN features, but only for a handful of images at a time.

In this paper, we introduce Visual Feature Search (VFS), a novel interactive visualization that empowers machine learning researchers to easily explore the visual features of almost any computer vision model. Our tool is designed to be lightweight and flexible so that users can quickly set up VFS to analyze the intermediate features of arbitrary CNNs

*Work done as a student at Princeton University
[1]Source code: https://github.com/lookingglasslab/VisualFeatureSearch

and vision transformers; the only requirement is that the model has intermediate spatial features (i.e. a 3D tensor of shape $H \times W \times C$). 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 results (Fig. 1). In essence, this provides a visual explanation to answer the question, "what does the model consider to be most similar to this image region?" To fully showcase the interactivity of VFS, we provide several interactive Jupyter notebooks in addition to this paper (see supp. mat.).

Our work is similar to CNN-based approaches that tackle content-based instance retrieval (CBIR), which aims to find visually similar images to a query image [9, 12, 38]. However, our work differs from CBIR methods in two ways: First, our goal is to understand a CNN (i.e. interpretability) by investigating what visual patterns are similar in *feature space*, whereas instance retrieval aims to retrieve *visually similar* images (and often uses CNN features to do so). Second, we explicitly focus on leveraging *interactivity* to allow users to quickly and iteratively gain insights on their selected models (e.g. by experimenting with multiple ideas in quick succession).

In the remainder of this paper, we summarize our implementation of VFS and include several experiments to highlight how it can be used to better understand computer vision models. While our method is similar to those commonly used in prior instance retrieval works (i.e. nearest neighbor search via cosine similarities) [9], VFS implements these search techniques in a novel, easy-to-use Python library that is designed specifically for interpreting arbitrary computer vision models. Our tool works best with PyTorch, but it can be used with models and datasets in other frameworks as well. Our source code and interactive Jupyter notebooks are available on GitHub;[2] our goal is to enable other researchers and practitioners to use VFS as a new method for interactively interpreting their models.

2. Approach

To use VFS, the user first selects a model and a layer within it to study, as well as a dataset of images to search across. We provide the user with an interactive widget for selecting a query image and highlighting a free-form region in it to use as the search query. To perform a feature search across the dataset, our tool computes the feature maps of images immediately after the user's selected layer and compares the highlighted regions within these maps.

Formally, let $f_l(\mathbf{q}) \in \mathbb{R}^{H \times W \times C}$ be the l-th layer's feature map for the query image $\mathbf{q}$. If we down-sample the user's selected region of $\mathbf{q}$ into a mask $\mathbf{m}_l \in [0,1]^{H \times W}$, we can apply the mask to obtain a 3D tensor $\mathbf{z} \in \mathbb{R}^{H \times W \times C}$ s.t. $\mathbf{z}_{(i,j,k)} := f_l(\mathbf{q})_{(i,j,k)} \cdot \mathbf{m}_{l(i,j)}$. To convert this into a query vector $\vec{q}$ for similarity search, we crop $\mathbf{z}$ to remove any zero padding and flatten the resulting data into a vector. We use a similar process to convert the feature maps of all images in the search dataset into vectors. We apply the mask $\mathbf{m}_l$ as a sliding window over each image in the search dataset to create region vectors $\vec{d_i}$ with the same dimensions as $\vec{q}$. This allows us to compare the query vector to each $\vec{d_i}$ via cosine similarities; we sort all search regions by their similarity scores and display the most similar image regions to the user (Fig. 1), thereby visualizing images with the most similar intermediate features to the user's selected region.

In order to use this algorithm for large-scale, real-time searches, we precompute the features $f_l(\mathbf{d}_i)$ by performing a forward pass for all dataset images $\mathbf{d}_i$, and we store the resulting data in a compressed cache file via the Zarr Python library [22]. The cache file allows VFS experiments to be easily shared and reproduced between multiple users, such as by downloading the file and running VFS on Google Colab environments. Furthermore, if the user wishes to search across a large dataset with features that cannot be stored in-memory, then VFS can load features from the cache file in batches to compute search results. Additionally, VFS is implemented with several GPU optimizations in PyTorch [26] in order to compute results in real time (see supp. mat. for more details).

VFS is most efficient when the cache can be loaded entirely into a GPU's VRAM: for instance, when testing on a Google Colab instance with an NVIDIA T4 GPU, VFS can search through the ResNet50 conv5 features of 50,000 images in 0.26 seconds on average, well within an acceptable time range for providing real-time visualizations. However, when the cache's size exceeds the available memory on the user's GPU, then VFS must resort to loading features from RAM or disk in batches, with data bandwidth becoming the primary performance bottleneck for the tool (see supp. mat.). As a result, we expect that VFS will be most useful for interpreting a model's features on validation sets with around 100,000 images (i.e. approximately 9.3GB in ResNet50 conv5 features), as well as for analyzing the performance of models on downstream tasks with smaller datasets.

3. Experiments and Demonstrations

Domain Generalization. One application of VFS is to understand how robust a model is when presented with novel images. To demonstrate this, we visualize ResNet50 [16] conv5 features of in- and out-of-domain (o.o.d.) images in two sets of experiments; our goal is to investigate whether a model's internal feature representations of in-domain images are similar to those of o.o.d. images.

[2] https://github.com/lookingglasslab/VisualFeatureSearch

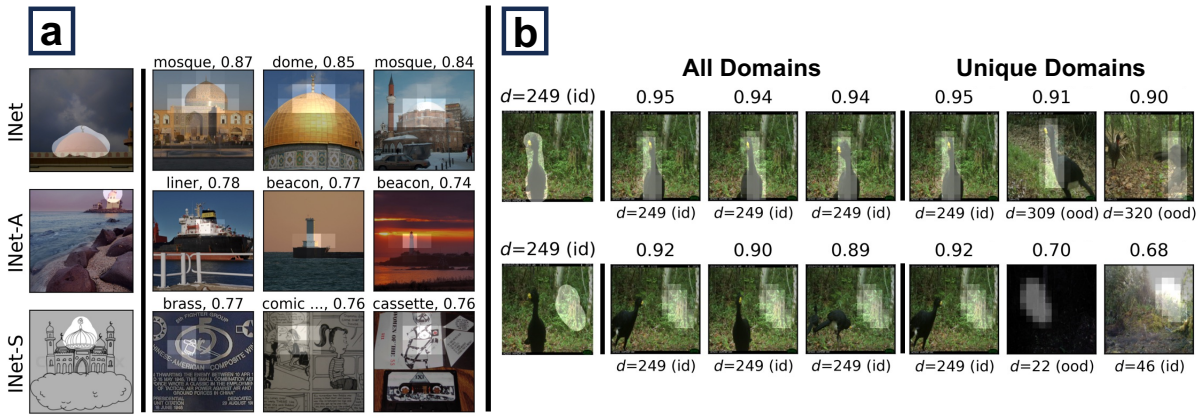

Figure 2. **Domain Generalization.** Each row contains one query image and a highlighted patch on the left, followed by multiple VFS results and similarity scores. **a**: Three queries of mosque images from ImageNet, ImageNet-A [17], and ImageNet-Sketch [33] on an ImageNet-trained model. **b**: queries and results from the iWildCam dataset [6] on a pretrained classifier. The iWildCam dataset contains images from multiple domains (i.e. camera trap locations); results include the top-3 overall results and the top-3 results for unique domains, with at most one image per camera location. The ImageNet model features for out-of-domain (ood) mosque images appear to be most similar to images with similar textures (rather than those containing mosques), while the iWildCam model produces animal features that generalize to ood settings.

The first experiments search for the most similar images in the ImageNet validation set [27] when similar query regions are selected from images in the ImageNet test set (in-domain), ImageNet-A dataset [17], and ImageNet-Sketch dataset [33] (both o.o.d.). One example is shown in Figure 2a, where images of mosques are selected as query images from all three datasets. The nearest neighbor results for all three queries show that the model can accurately extract semantic data from the in-domain query image, but it fails to encode the other two queries as mosques due to their out-of-distribution scale and texture; additionally, the cosine similarities are much higher for the in-domain query as opposed to the two o.o.d. queries. Additional queries corroborate these trends (see supp. mat.).

The second set of experiments uses the iWildCam dataset [6], which consists of images of wildlife from various trap camera locations. Some locations are included in the training subset while some are withheld and are thus out-of-domain. We investigate the conv5 features from a pretrained model that was trained to detect the presence of animals and classify their species [19]; representative VFS queries and results are shown in Figure 2b. These results support the finding in [19] that the model is able to generalize fairly well, as the feature representations for animals have high similarity scores (e.g. 0.9) across different domains; in contrast, when a background patch of an image is queried, only images from the same domain have similarity scores above 0.7.

Chest X-ray Classifiers. Another use of VFS is to understand why a model made a particular decision by finding image regions that correlate with certain classification labels. To demonstrate this, we turn to the domain of chest X-ray classification: we study the last feature layer of a pretrained DenseNet-121 [11, 18] that classifies pathologies in the ChestXray-14 dataset [34].

We specifically investigate an X-ray image of a patient with cardiomegaly (i.e. a condition where the heart is enlarged). The model is able to correctly classify this patient as having cardiomegaly, and we use the same X-ray image as a query in VFS to provide interpretable visualizations of the model features. Our results are shown in Figure 3: we find that highlighting the heart of the patient returns nearest neighbor regions of other hearts from patients with cardiomegaly; in contrast, searching for an unrelated region of the same patient's X-ray (e.g. right lung) yields nearest neighbors with mixed diagnoses. Empirically, when the heart is highlighted, images of patients with cardiomegaly tend to have higher similarity scores than images of patients without the condition; in contrast, when the lung is selected as the search query, the distribution of similarity scores is virtually identical for X-rays with and without cardiomegaly. This suggests that the classifier's prediction of cardiomegaly is correlated specifically with the heart region of the X-ray.

Editing Classifiers. Recently, a method to correct for systematic CNN mistakes was introduced [28]. For instance, [28] found that a VGG16 ImageNet classifier consistently misclassifies vehicles that are on a snowy surface and typically predicts these images as snowmobiles or snowplows. Their CNN-editing method mitigated this mistake by updat-

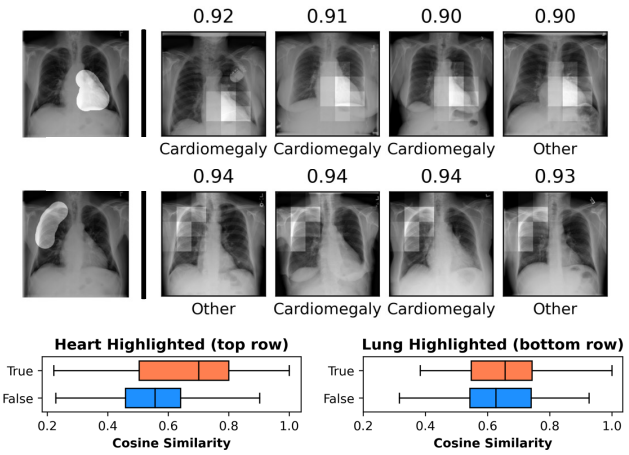

Figure 3. **Chest X-ray Classification.** Two queries and results of a patient X-ray with cardiomegaly (a condition where the heart is enlarged) from the ChestXray-14 dataset [34]. The same image is used for both queries, but the heart is selected in the top row while the lung is selected in the bottom row. Box plots display the distribution of cosine similarities between the query regions and all dataset images. While the qualitative results between the two queries only differ slightly, the box plots show that the similarity scores of images containing cardiomegaly tend to be higher than those without when the query region contains the heart.

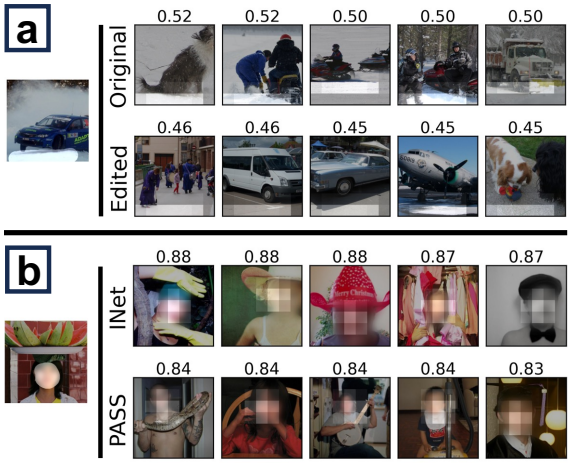

Figure 4. **Editing a Classifier and Training on PASS. a**: Edit a classifier using [28] to improve accuracy for vehicles on snow. The top row shows that for the original model, the most similar images to the query are those that contain other objects on snowy or icy surfaces. In contrast, the bottom row shows that the edited model's most similar images contain vehicles on asphalt, which suggests that the classifier edit was successful. **b**: Facial feature data for an ImageNet- vs. PASS-trained model.[3] Despite not being trained on images containing humans, the PASS model encodes features that can match the query face to other faces in the dataset.

ing the model's weights such that the 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 visualize how the edit affected the model's features by analyzing nearest neighbor search results from the ImageNet validation set.

Our visualizations suggest that the snow-to-asphalt edit works and has a noticeable effect on intermediate features. We highlight an example in Figure 4a. Our search results show that the original model does not have a clear understanding that the ground should be treated like an asphalt road; the nearest neighbors include other objects on snowy surfaces, such as snowmobiles and snowplows. In contrast, the search results from the edited model include asphalt surfaces and several cars (as opposed to snowmobiles and snowplows in the original search results), which indicates that the model edit successfully changed the feature representation of snowy roads to achieve the desired result.

ImageNet vs. PASS. We next study how the choice of training dataset affects a model's feature representation. To mitigate privacy concerns of training on images with humans, the authors of [2] introduced PASS, a dataset of unlabeled images that do not contain human faces or body parts. PASS is meant to serve as an ImageNet replacement for self-supervised learning and has been shown to perform as well as ImageNet-trained models on human-centric tasks (e.g. pose estimation). We compare the feature representations from two ResNet50 models, one trained on ImageNet and the other on PASS, that were trained via MoCo-v2 [10] self-supervision. Our results when using VFS on both models are included in Figure 4b. The most notable observation is that the PASS-trained model can accurately match face queries to other faces in the dataset, despite never being trained with images of humans. However, the similarity scores for the ImageNet model results are generally greater than those for the PASS model.

4. 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 others as regular research tools. We demonstrate through our experiments that our tool is much more flexible in comparison: it can be used to quickly visualize new models and datasets, and we hope that this flexibility allows other researchers to use it to better understand their own models. Lastly, we emphasize that our tool is qualitative and should be paired with quantitative analysis to fully corroborate findings.

[3] Faces in this figure are blurred to preserve the privacy of the photographed individuals.

Acknowledgments

We are grateful for support from Open Philanthropy (RF), the Princeton Engineering Project X Fund (RF), and the Princeton SEAS IW Funding (DU). 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 [2, 7, 10, 11, 19, 22, 26, 28] 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. 1

[2] 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, 5

[3] 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. 1

[4] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *CVPR*, 2017. 1

[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. 1, 5

[6] Sara Beery, Elijah Cole, and Arvi Gjoka. The iwildcam 2020 competition dataset. *arXiv preprint arXiv:2004.10340*, 2020. 3

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 5

[8] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 2019. 1

[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. 2

[10] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020. 4, 5

[11] Joseph Paul Cohen, Joseph D. Viviano, Paul Bertin, Paul Morrison, Parsa Torabian, Matteo Guarrera, Matthew P Lungren, Akshay Chaudhari, Rupert Brooks, Mohammad Hashir, and Hadrien Bertrand. TorchXRayVision: A library of chest X-ray datasets and models. In *Medical Imaging with Deep Learning*, 2022. 3, 5

[12] Shiv Ram Dubey. A decade survey of content based image retrieval using deep learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2021. 2

[13] Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *ICCV*, 2017. 1

[14] Ruth Fong, Alexander Mordvintsev, Andrea Vedaldi, and Chris Olah. Interactive similarity overlays. In *VISxAI*, 2021. 1

[15] Adam W Harley. An interactive node-link visualization of convolutional neural networks. In *International Symposium on Visual Computing (ISVC)*, 2015. 1

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2

[17] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *CVPR*, 2021. 3, 1

[18] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018. 3

[19] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton A. Earnshaw, Imran S. Haque, Sara Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. Wilds: A benchmark of in-the-wild distribution shifts, 2021. 3, 5

[20] Seongmin Lee, Zijie J. Wang, Judy Hoffman, and Duen Horng (Polo) Chau. Viscuit: Visual auditor for bias in cnn image classifier. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21475–21483, 2022. 1

[21] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *IJCV*, 2016. 1

[22] 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, 2020. 2, 5

[23] Andrew P Norton and Yanjun Qi. Adversarial-playground: A visualization suite showing how adversarial examples fool deep learning. In *VizSec*, 2017. 1

[24] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017. 1

[25] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018. 1

[26] 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. 2, 5

[27] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 3

[28] Shibani Santurkar, Dimitris Tsipras, Mahalaxmi Elango, David Bau, Antonio Torralba, and Aleksander Madry. Editing a classifier by rewriting its prediction rules. In *NeurIPS*, 2021. 3, 4, 5

[29] Ludwig Schubert, Michael Petrov, Shan Carter, Nick Cammarata, Gabriel Goh, and Chris Olah. OpenAI Microscope, 2020. 1

[30] 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. 1

[31] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR workshop*, 2014. 1

[32] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv*, 2017. 1

[33] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *NeurIPS*, 2019. 3, 1

[34] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M. Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3, 4

[35] Barron Webster. Now anyone can explore machine learning, no coding required. Online, 2017. 1

[36] 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. 1

[37] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 1

[38] Liang Zheng, Yi Yang, and Qi Tian. Sift meets cnn: A decade survey of instance retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 40(5):1224–1244, 2017. 2

[39] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *CVPR*, 2016. 1

Interactive Visual Feature Search

Supplementary Material

Code Our source code and sample Jupyter Notebooks that showcase VFS are available at https://github.com/lookingglasslab/VisualFeatureSearch. The notebooks are designed to run in Google Colab, and the repository includes additional instructions for running VFS locally with Jupyter Notebooks.

Additional Figures In this section, we provide additional figures for the domain generalization, editing classifiers, and ImageNet vs. PASS experiments.

Figure 5 includes additional queries and results for the ImageNet (in-domain), ImageNet-A, and ImageNet-Sketch (o.o.d.) experiments. In Fig. 5a, all queries are pictures of unicycles; when the unicycle wheels are highlighted in each query, both the ImageNet and ImageNet-Sketch queries are successfully matched to other unicycle wheels, while the ImageNet-A query is matched with various unrelated nearest neighbors. Similarly, in Fig. 5b, all three queries are images of bell peppers; however, only the in-domain query yields nearest neighbors that are also bell peppers. For both the unicycle and the bell pepper queries, the resulting similarity scores are highest for the in-domain queries (i.e. > 0.9) when compared to the scores for the o.o.d. queries (i.e. < 0.8).

Similarly, Figure 6 includes additional domain generalization visuals for the iWildCam dataset. Two sets of queries and search results are shown: one is of a cow during the day, while the other is of a deer at night. The results for the deer query are similar to those in Figure 2, as the features appear to be highly generalizable and have nearest neighbors of other deer across a variety of domains. However, the encoded features for the cow are less generalizable and simultaneously less accurate, as the nearest neighbors in other domains have lower similarity scores (0.94, 0.93) than those from the same domain (0.96), and the nearest neighbors from other domains contain horses, not cows. This particular query image is misclassified by the model as containing a horse, so the search results help visualize the features associated with this misclassification.

Figure 7 contains an additional example of the Editing Classifiers visualization, as well as an additional visual for the ImageNet vs. PASS experiment. Fig. 7a includes a query of a scooter on snow-covered ground; when the ground is highlighted, the original model's VFS results contain no other images of scooters or cars. However, when the edited model is used, four of the top-5 results contain cars, and the instance of a bobsled on ice from the original results is no longer included. Thus, this example provides further evidence that the model edit was successful.

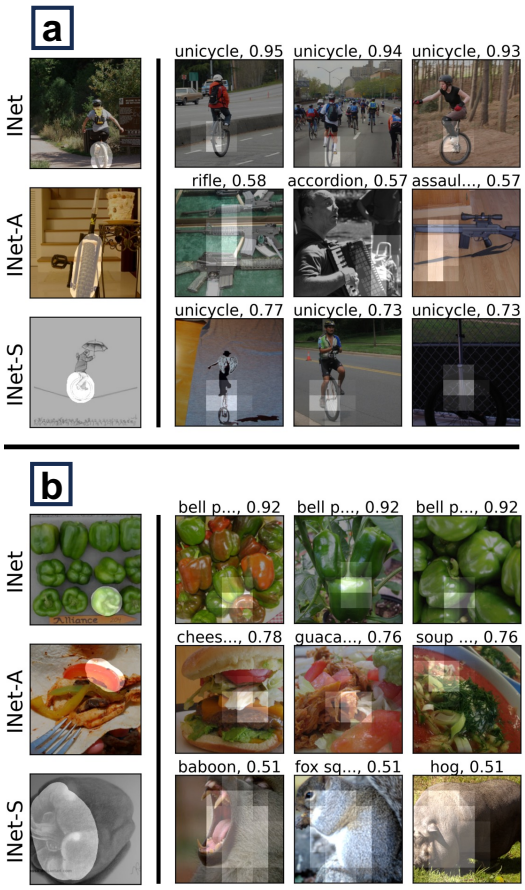

Figure 5. **Additional ImageNet Generalization.** Three queries of images from ImageNet, ImageNet-A [17], and ImageNet-Sketch [33] on an ImageNet model. **a**: All queries are of unicycle wheels. The ImageNet-Sketch unicycle is matched with other unicycle wheels in the dataset, but the similarity scores are lower than those from the in-domain query. **b**: All queries are of bell peppers, but only the in-domain search contains bell peppers in the results. Such visuals may provide insights into why a particular model misclassified a challenging example.

Figure 7b shows an additional example of a query containing a face with two sets of results for the ImageNet and PASS models, respectively. Although several of the highlighted regions in the PASS results contain no faces, two such results contain faces elsewhere in the image. Thus, the PASS-trained model is again able to successfully encode human faces and retrieve other images containing faces via VFS.

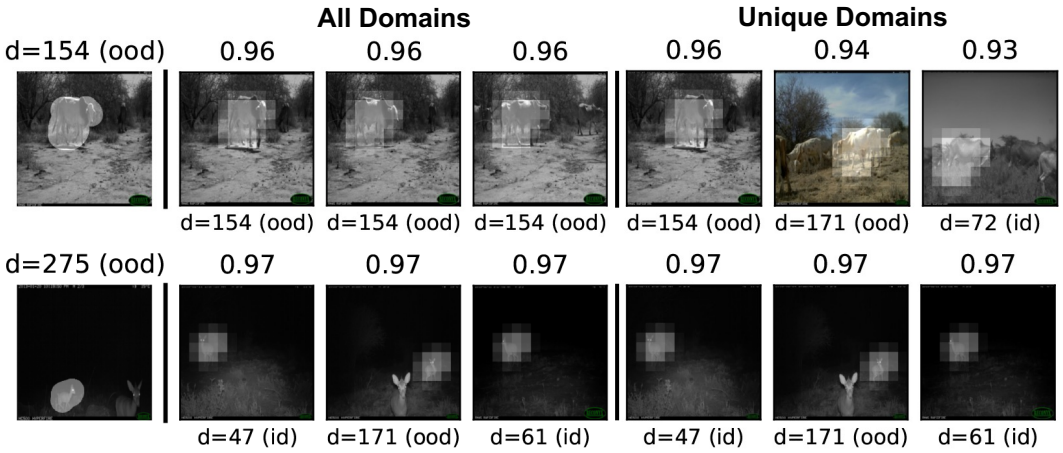

Figure 6. **Additional iWildCam Generalization.** Two sets of queries and results for a cow (top) and a deer (bottom). While the deer features appear to be highly generalizable with all top-3 results originating from different camera locations, the cow has a comparatively worse feature representation since its nearest neighbors from other domains have slightly lower similarity scores (0.94, 0.93) than inner-domain results (0.96); additionally, its nearest neighbors from other domains are images of horses, not cattle.

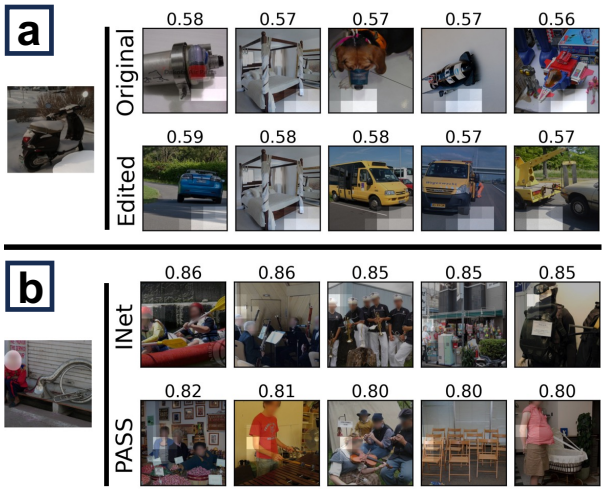

Figure 7. **Additional Results for Edited Classifier and PASS Training. a**: An additional visualization of the "vehicles on snow" classifier edit. The original model's VFS results consist entirely of unrelated objects such as ice, floors, and a tabletop, whereas the edited model's results contain several cars on asphalt. **b**: An additional search query and results for ImageNet- vs. PASS-trained models. While the localization of nearest neighbors is relatively poor for PASS (i.e. the highlighted regions in some search results do not contain faces), the PASS model is able to successfully able to match the query to other images that contain faces.

Implementation Details In Section 2, we described how to perform the region-based similarity search for VFS. However, in order to perform the search efficiently, we use a modified version of this algorithm that substitutes the sliding window approach with a convolution operation. This allows us to compute the searches on a GPU via PyTorch, which allows for extensive parallelization and enables each search to be much faster than if it were run on a CPU.

To implement the convolution-based searches, we first take the 3D tensor $\mathbf{z} \in \mathbb{R}^{H \times W \times C}$ from Section 2 and apply the mask $\mathbf{m}_l$ on it once more. We then crop the tensor, which we define as removing all rows/columns on the exterior of the feature map which only contains zero-valued elements. Let the resulting tensor be $\mathbf{z}'$, where:

$$\mathbf{z}'_{(i',j',k')} = \text{Crop}(\mathbf{z}_{(i,j,k)} \cdot \mathbf{m}_{l(i,j)}) \tag{1}$$

$\mathbf{z}'$ has dimensions $H' \times W' \times C$, where $H' \leq H$ and $W' \leq W$. We use $\mathbf{z}'$ as a 2D convolutional filter and apply it to a feature map $f_l(\mathbf{s})$ from the search database.

$$\mathbf{c} := f_l(\mathbf{s}) * \mathbf{z}' \tag{2}$$

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

Runtime Performance We measured the runtime performance of our VFS implementation by running multiple sets of searches over varying dataset sizes. We used subsets of the ImageNet dataset with n = 25,000, 50,000, 75,000, and 100,000 images; the similarity searches were computed for a ResNet50 model's conv5 features with dimensions $7 \times 7 \times 512$. In order to understand the impact of memory bandwidth, we performed two sets of experiments: the first set used a cache that was pre-loaded directly into a

GPU's VRAM, while the second set loaded the feature data in batches from a cache in regular RAM. For each dataset size and cache location, we computed search results for 20 hand-drawn queries; the mean runtimes and standard errors are included in Table 1. All searches were computed on a Google Colab VM with a 16 GB NVIDIA T4 GPU.

n	Cache Size	VRAM Time (s)	RAM Time (s)
$25k$	2.34 GB	0.139 ± 0.004	0.820 ± 0.009
$50k$	4.67 GB	0.263 ± 0.006	1.668 ± 0.021
$75k$	7.01 GB	0.420 ± 0.010	2.542 ± 0.018
$100k$	9.35 GB	0.564 ± 0.008	3.435 ± 0.023

Table 1. **Runtime Performance.** The mean runtime (and SEM) of VFS was measured over a set of 20 query regions, with a search dataset of size $n = 25,000$ through $100,000$. Searches were computed for ResNet50 conv5 features with dimensions $7 \times 7 \times 512$ and 32-bit precision; the resulting cache sizes for all feature data are included for reference. Results are reported for two cache locations: one set of experiments pre-loaded the cache entirely onto a GPU's VRAM, while the other set loaded features from a cache in regular RAM in batches. The results indicate that VFS is most efficient when the feature cache is able to be loaded directly into VRAM; loading from RAM may be necessary for especially large datasets, but this increases runtime significantly.

The results in Table 1 show that VFS is most efficient when the dataset's feature cache can be loaded entirely in a GPU's VRAM. The mean runtime for searching through a set of 50,000 images is 0.263 seconds, which enables VFS to be a fast and highly interactive tool. Computing search results with a cache in regular RAM is much slower, mainly due to the bandwidth required to move each batch of feature data to the GPU for similarity search. However, if a user's dataset features are too large to fit in their GPU VRAM, then loading from batches in RAM is a viable (albeit slower) method when using VFS.