Understanding and comparing learned features in CLIP models via Sparse Autoencoders

Franklin Zhu
Stanford University
fzhu22@stanford.edu

Logan Graves
Stanford University
graves1@stanford.edu

Abstract

Vision models learn complex semantic relationships within their training data, but it is not well-understood what concepts those models contain, nor how architectural differences influence the concepts a model can learn. In this work, we train Sparse Autoencoders (SAEs) on a ResNet and a Vision Transformer which have been pretrained similarly and experiment with training methods, in service of a comparative analysis of human-interpretable concepts contained in similar visual networks with different architectures (CLIP-ResNet50 and CLIP-ViT-L/14). We experiment extensively and use quantitative and qualitative metrics to compare the two autoencoders; ultimately, despite poor performance in our ResNet SAE when we train it similarly to the ViT, we nonetheless find some nontrivial instances of feature alignment, indicating positive potential for future research in the area.

1. Introduction and Related Work

Deep neural networks exhibit exceptional performance across many domains, but their internal representations (e.g. activations in latent layers) are in many cases opaque. To explain this phenomenon, previous work such as [9] has introduced the so-called *"superposition hypothesis"*, in brief that models want to learn more features than they have dimensions to represent, so they pack these features in "almost-orthogonal" directions in their high-dimensional latents. Thus, while individual dimensions in the latent spaces of a deep learning model may not be interpretable, with the right techniques this 'superposition' of features can be undone and understood.

To achieve this comprehension, it has been long understood that sparsity aids in finding basis functions, enabling the decomposition of the network into fundamental functions or concepts [2]. In the modern day, one prominent approach to achieve this decomposition has been the use of Sparse Autoencoders (SAEs). Sparse Autoencoders were initially developed for use in language models [5] and in part popularized by research done for language model interpretability by the Anthropic team in [12], [4]; in the past two years they have continued to receive significant attention. SAEs are autoencoders trained to reconstruct the latents of another model, but with a much larger ("overcomplete") latent space of their own and a strong sparsity penalty. The input to the SAE is the residual stream or part of the residual stream of another model, usually either the CLS token in a ViT, or the output of a pooling layer in the ResNet; the SAE expands it to the overcomplete dimensions, and then attempts to reconstruct its inputs; thus, the outputs are an attempt to reconstruct the other model's residual activations. Intuitively, by incentivizing most dimensions in the model to stay inactive, SAEs are more likely to learn to represent each superimposed feature in its own dimension, as opposed to spreading them between dimensions, thus undoing the superposition.

Though initially popularized in the context of language model interpretability, SAEs have emerged as a powerful tool for uncovering interpretable and efficient feature encodings in high-dimensional image data. As noted by [7], which trained a nine-layer sparse autoencoder on a dataset of 10 million 200×200-pixel images, even without labels, the network was able to discover high-level concepts, such as face and cat categories, from raw image data. This work highlights how sparse autoencoders can extract semantically meaningful features from very high dimensional visual inputs. However, sparse autoencoders are particularly useful to use with vision transformers. Though there is relatively little research in this area, with the growth of large models with high dimensional image embeddings like OpenAI's CLIP ViT models, this interpretability has become increasingly important. In [6], it was shown that sparse autoencoders trained on the vision transformer submodel in OpenAI's CLIP[3] produce robust, highly human-interpretable features. This research was extended in [1], which scaled the approach and used it to develop steering mechanisms. Here we take a similar approach but extend it towards the end of comparison, training SAEs both

on the vision transformer and the ResNet versions of the CLIP model, and undertaking a comparative analysis of their learned features. In this paper, we take a similar approach, exploring the use of SAEs to produce sparse, semantically meaningful representations of the model's residual stream.

2. Methods

Our aim with this project was to train two working SAEs, one trained on CLIP-ResNet50 and one on CLIP-ViT-Large14, and then to conduct a comparative analysis of the features learned from both.

2.1. SAE Architecture

In line with previous work[6], we set up an overcomplete linear SAE with input and output dimensions d (for the ViT CLIP, $d = 768$, and for ResNet50 CLIP $d = 1024$). Our overcomplete latent layer expansion factor of 16 yields a latent layer of dimension $16d$.

The SAE architecture is very simple; much of the difficulty in training them is finding the right parameters and implementing techniques to improve sparsity. Let $\mathbf{x} \in \mathbb{R}^d$ be the activations of the residual stream in the CLIP model. The SAE first applies an encoder, which maps $\mathbf{x}$ to a latent dimension:

$$\mathbf{h} = ReLU(W_e\mathbf{x} + \mathbf{b})$$

with $W_e \in \mathbb{R}^{16d \times d}$ and $\mathbf{b} \in \mathbb{R}^{16d}$. Next the SAE applies the decoder and attempts to reconstruct $\mathbf{x}$:

$$\hat{\mathbf{x}} = W_d\mathbf{h} + \mathbf{c}$$

with $\mathbf{c} \in \mathbb{R}^d$. This formalism maps onto the intuition given above: the sparse autoencoder maps into a latent space which expands out any superimposed features into their own dimension, then the autoencoder reconstructs the input to ensure that it is indeed generating faithful 'expansions' of the activations that are fed to it.

We choose 16 for the expansion factor as opposed to 64 or 32 for multiple reasons. Previous work [6][12] discovered so-called "ultra-low-density clusters" of SAE features that fire extremely rarely, yet which make up a significant proportion of model features. The author of [6] described achieving relatively similar performance in an SAE with an expansion factor of 16 as opposed to 64, indicating to us that 16 is empirically viable as an expansion factor, reducing training costs significantly without sacrificing significant performance.

For loss we use MSE loss with strong L1 regularization, and train using the Adam optimizer. Mathematically, we have

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m} \left\| \mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)} \right\|_2^2 \; + \; \lambda \left\| \theta \right\|_1$$

with m the batch size, θ the parameters (including W, $\mathbf{b}$, and $\mathbf{c}$). Empirically, effective parameters tend to hover around a learning rate of 0.0004 and an L1 strength of 0.5; these parameters were effective for both ViTs and SAEs.

2.2. Hook layers

In order to train SAEs, one must choose a specific part of the model to 'hook', i.e. choose which activations, at which layer, to feed into the SAEs. The choice of hook layers is not straightforward, and involves multiple factors, among them computational cost and susceptibility of representations to SAE learning. In general, it is desirable to choose layers that are later in the model, because in order to learn abstract representations, the SAE needs to take in the activations of a relatively abstract layer.

We chose the CLS token after the `ln_post` layer at the end of the CLIP model (768 dimensions). This is a sufficiently abstract layer to have useful representations. In the ResNet, we hook the output of the `attn_pool` (1024 dimensions); though it is of a slightly larger dimensionality than the hooked layer in the ViT, they are close enough in magnitude that it remains a natural comparison.

2.3. SAE Evaluation metrics

Once we trained our SAEs using Adam, we used a number of metrics to evaluate them. We of course care initially about loss; if our model cannot competently reconstruct its inputs, then its latent representations are not faithful, and thus the features it learns do not matter to us.

In general, relatively sparse latent representations in the SAE with few dead units tend to result in interpretable neurons. Thus, assuming we have low loss, we evaluate SAE performance using a number of metrics:

1. *Population sparsity.* Population sparsity is given by the average number of active neurons for any given input, which expressed mathematically is

$$\frac{1}{n}\sum_{i=1}^{n} 1[a_i(x) > \epsilon]$$

where $a_i(x)$ is the activation of a neuron on an input x and ϵ is the threshold for the neuron to be considered active. We aim for a range between 5-25% sparsity; sparsity is an indicator, but not a perfect correlate with interpretable features. (We easily trained models with very low sparsity, but typically trying to push sparsity too low would simply result in a large fraction of 'dead' neurons.) We visualize population sparsity with two different kinds of diagrams; see the results section for examples. Also see section 4 (experiments) for more information about how we induce sparsity, covering techniques such as neuron reinitialization, as

well as more mundane methods like choosing the right hyperparameters.

2. *Identifying dead units.* Sparse learning with L1 and ReLU can cause some neurons in the latent layer to 'die', i.e. fire very rarely or never; this is undesirable, because it reduces the effective capacity of the SAE since these neurons once 'dead' will no longer update their respective weights. We track dead units using a histogram of activations, and quantify it using a threshold value (e.g. if the total activation across the batch is less than some threshold ϵ, in this work $1e-6$, we consider it to be dead; we can examine the histogram of neuron sparsity to see the number of dead neurons.)

3. *Qualitative human interpretability.* Lastly, in addition to quantitative metrics we use qualitative observations to confirm that our sparse features are indeed corresponding to some important concept. To evaluate qualitative interpretability, we choose neurons randomly from the latent layer, run a number of forward passes, and visualize the images that have the strongest activation for that neuron. A well-trained SAE will have similar images in this visualization, e.g. 'images of cats' or 'images of people working out' or 'images of animals eating other animals.' It is encouraging for the SAE to find features that directly correspond to a class in the dataset – and indeed, those that we trained frequently did – but far more interesting is emergent understanding of 'concepts' not accounted for by the class outputs. As a simple example, our best-performing ViT has an identifiable 'weights' feature that triggers on images of barbells *and* dumbbells, when there is no general weights category in the training data, only separate classes for barbells and dumbbells.

2.4. Feature Comparisons

To measure the similarity of features learned in the ViT SAE and the ResNet SAE, we use a number of metrics:

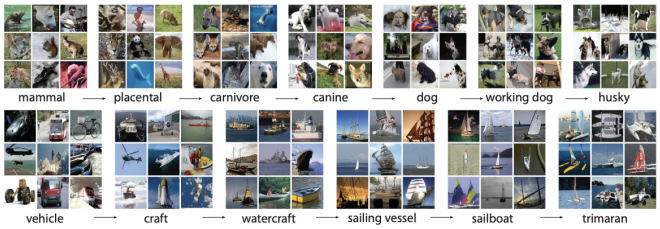
1. *Average top-k image similarity.* For each feature in the SAEs, save the indices of the top k activating images for those features. Then for each ResNet SAE feature, compute the average overlap each SAE feature has with its most similar feature in the ViT. Since some or many features may be dead, we may limit the number of features across which we compute the average.

2. *Pairing.* By pairing each ViT feature with a corresponding ResNet feature based on the similarity of its top-k images, we can create a list of possible 'same feature' relationships; then we can compute the percent overlap between them.

3. *Qualitative similarity.* By looking at visualizations of quantitatively similar features we can confirm or falsify the notion that these features are indeed similar in a human-interpretable way.

3. Dataset and Features

To train the SAE models, we used the ImageNet-1k dataset from Hugging Face Hub, which has a training split of around 1.28 million images. There are 1000 distinct classes in this dataset and about 1300 images per class. These classes cover a wide variety of natural and man-made objects, like animals, vehicles, electronics, household items, sporting equipment, etc. Due to computational and some storage constraints, we decided to train our models on up to four of the five total parts of the training split, with each part containing around 256 thousand images, resulting in a total size of about 1.024 million images. Then, for evaluative metrics, we used the validation split, consisting of 50 thousand images. In the ImageNet-1k dataset, there is no fixed image resolution and images come in a variety of sizes and aspect ratios. Some examples of images and their classes are shown below:



3.1. Preprocessing

To process this data for the OpenAI CLIP models, we used the preprocess function given with the CLIP model. This function resizes images to be 224x224 by first using the bicubic interpolation method, which is a resizing technique that produces smoother results than simpler methods, to reduce the shorter side to 224 pixels, and then center cropping the other dimension to create a 224x224 image. Afterwards, the image is converted into a PyTorch tensor of shape (3, 224, 224) and normalized, allowing for a stable distribution. Some examples of preprocessed images are shown below:

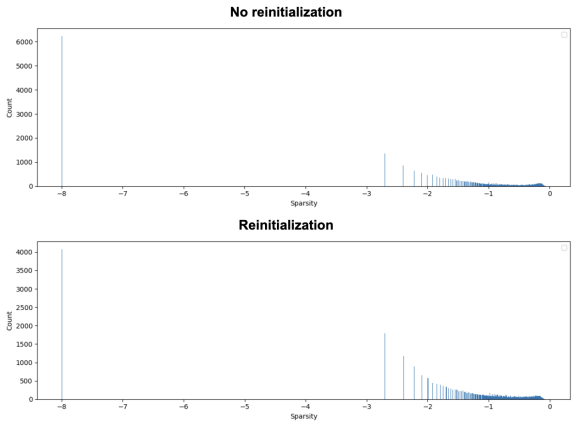
To save time during training, we decided to first preprocess all of the data and push it back to Hugging Face Hub, that way it could be directly streamed and used for model training, instead of having to be preprocessed every single time. For each split, we loaded the data into a Hugging Face Dataset object and preprocessed them with the OpenAI CLIP preprocess function. We flattened the tensors into raw bytes, allowing them to be stored in the .arrow file format. These files store one shard of the dataset, containing a batch of examples, which may include one or more tensors as column values. Another option for data storage would have been using Parquet files for more compression, but we streamed the data later on, so we used Arrow IPC files, which are larger, but more efficient for streaming.

4. Experiment

4.1. Auxiliary Experiments

In addition to our primary training runs, we conducted a number of auxiliary experiments in order to better understand training dynamics:

1. *Neuron reinitialization.* We wanted to determine whether our neuron reinitialization strategy was functioning, so we trained two models with identical parameters except one had reinitialization and the other didn't. Results were significant: the two different models had a 0.1% difference in sparsity, while the model trained with weight reinitialization had approximately 33% fewer features. Further training continued to yield stronger results, ultimately reducing the dead neuron count to 1200, an almost 80% reduction.



2. *Toy experiments with relating learning rate and L1.* Experiments with learning rate and regularization parameters revealed that both have crucial roles in ensuring sparsity, and they impact one another. A low learning rate can cause the model to fail to develop sparse representations, even with a very strong L1 regularization; we believe that this is due to the fact that, without a strong enough learning rate, the L1 regularization does not have enough pressure to properly zero out neurons, leaving them simply small rather than 0. Reducing L1 regularization has a similar effect on training, as would be expected; with too low an L1 rate, the model will fail to learn sparse representations.

3. *Diversity in training data.* We hypothesized that, between two datasets of the same size, one more diverse than the other, that the more diverse dataset would cause the SAE to use its capacity more and learn sparse representations, and, on the other hand, that a dataset without much diversity would not be engaging enough of the model's capacity to successfully learn sparse representations, and would stay dense. Indeed, this was empirically the case; switching from the same number of images in CIFAR-10 to ImageNet-1k led to superior qualitative results in interpretability.

4.2. Evaluating the ViT SAE

At the start of training, we began with similar hyperparameters to [6] to try to replicate their results. We explain how we came up with each of our hyperparameters below.

1. *Expansion Factor.* From the results of [6], which used an expansion factor of 64, we saw that 64 was more than needed to capture important features for all images in the ImageNet-1k dataset. Since ImageNet-1k is much smaller than the dataset used to train the OpenAI CLIP models, [6] notes that an expansion factor of 16 is sufficient to capture all the features and the

expressive capacity of the original model. We did test larger expansion factors, like 32, but this led to more dead neurons and did not improve results.

2. *Learning Rate.* We started with the learning rate from [6], which was 0.0004. However, with this learning rate the training was significantly overcorrecting in response to noise and visually spiky, so we reduced the learning rate down to 0.0001.

3. *L1 Strength.* One metric that we focused on while training was sparsity. Ideally, we wanted to lower our average population sparsity as much as possible while not having many dead neurons. We started with an L1 strength on the order of 1e-5, but quickly needed to increase it significantly because it did not have enough of an impact on the loss. We settled on about 0.4 to 0.6, which is notably significantly higher than the previous work we referenced, but which was nonetheless the most effective range.

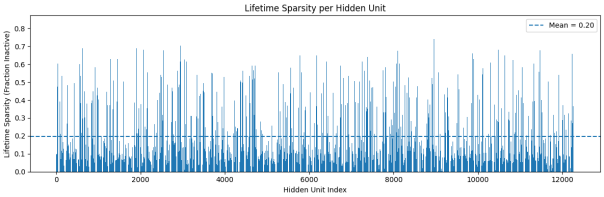
4. *Batch Size.* One other factor that we noticed with the model was that it performed better with larger batch sizes because a particular unit might not fire for certain inputs and batch sizes that are too small lead to lots of variability and noise in ρ_j. However, we were unable to go beyond a batch size of 256 because doing so led the HTTP streaming from Hugging Face to timeout sometimes, interrupting the training. Thus, after doing another hyperparameter sweep, we ended up with a batch size of 256.

5. *Epochs.* Since our training data was so large, we were unable to optimize our hyperparameters for all of the training data beyond one epoch due to time constraints. As such, we trained our SAE for only one epoch. With more time, training more epochs could decrease the sparsity and improve performance overall.

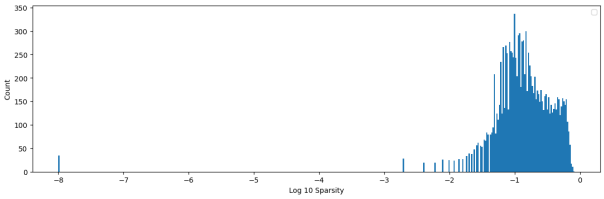
4.3. ViT SAE Results

The metrics we used to evaluate the ViT SAE results include population sparsity, plots of lifetime sparsity, and qualitative analysis, with emphasis. We believe that our model did not overfit on the training set, because our feature visualizations are done with the validation set and our models continue to provide semantically meaningful features.

Our most interpretable ViT SAE had a population sparsity of 0.2, though other models reached as low as 0.002. Although 0.2 was a bit higher than the expected ideal value, later qualitative analysis showed that the model reconstructed embeddings of the ViT well and tended to have more meaningful than some more-sparse models we had trained. A plot of the lifetime sparsities per hidden unit and histogram of lifetime sparsities is shown below.

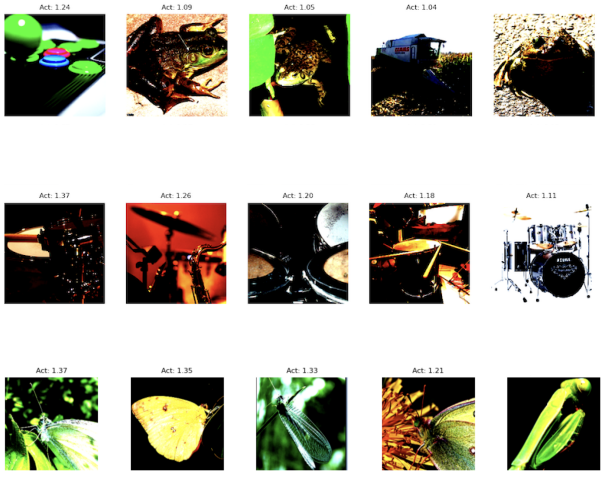


The lifetime sparsity per hidden unit does not have many gaps, indicating that there are not many dead neurons and neurons are actively firing on different inputs.



The histogram further confirms the low number of dead neurons, as the majority of neurons are clustered around a sparsity of 0.1, with only a small count of dead neurons. (Since this is log 10 sparsity, we added a small ϵ to prevent dead neurons from having values of $-\infty$, thus why dead neurons are clustered at exactly $1e-8$.) These results indicate that the neurons of the SAE are learning features well, as they are somewhat sparse and almost all neurons are actively contributing on certain inputs.

As such, we then displayed the highest activation images for random neurons, showing the features that the SAE learns. From the selected neurons, we can see different categories that they learn. These images are from the validation split and each feature is shown below.



The neuron in the first row clearly learned a feature related to the shape and color of frogs. Many of the images in the row are frogs, but one of them is a video game controller,

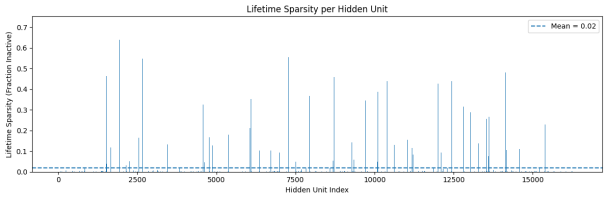
due to how the controller is shaped like a frog and has many similar features, like its green buttons, which look like body parts of the frog. In the next row, the neuron learned a feature related to drums, leading to all of the displayed images being drums or drumsets. The last category that is displayed is different types of insects. There are various kinds of flying insects like butterflies, but also ones without wings, like the praying mantis. This elucidates some kind of common feature between these images, which could be related to the green background, insect body shape, etc. Other than these specific categories, there are many more features that were learned by other neurons as well, highlighting how well the ViT SAE was able to capture different high level features in single neurons. By analyzing these categories, more correlations between certain types of images can be discovered and more understanding of the features the ViT actually learns can be gained.

4.4. ResNet SAE

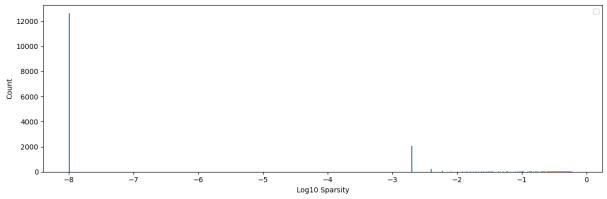
Since we wanted to compare the features learned by the ViT SAE and ResNet SAE, we decided to use the same hyperparameters to ensure that they were trained in the same way.

4.5. ResNet SAE Results

The ResNet SAE had a population sparsity of 0.02. Although this seems low, the lifetime sparsity plot explains why this is not accurate. The lifetime activations for each neuron are shown below.

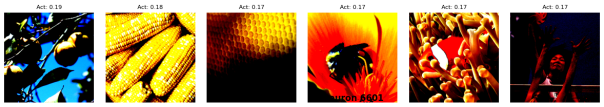


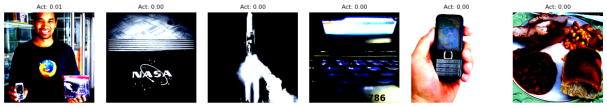
The lifetime sparsity per hidden unit looks similarly distributed to the distribution of the ViT SAE, but with more blank areas. This indicates that some hidden units are not firing at all, leading to a very low sparsity because many neurons are dead.



A histogram of the log10 sparsity further confirms the problem with dead neurons, as the majority of the neurons are not firing at all. From these plots, it is clear that the ResNet SAE suffered a lot more from dead neurons than the ViT model using the same hyperparameters. This could suggest the ResNet embeddings are easier to reconstruct and less neurons are actually needed for each input, meaning that most neurons are unnecessary and end up dead due to the L1 penalty. Other factors like the different input dimensions and therefore different hidden layer dimensions may have also played a role in the ResNet suffering more from dead neurons.

Afterwards, we also displayed the highest activation images for random neurons, showing the features that the SAE learns. From the selected neurons, we can see different categories that they learn. These images are from the validation split and each feature is shown below.





The neuron in the first row clearly learned a feature related to yellow, circular objects, demonstrated by the yellow fruits, corn, coral, and volleyball. Another category is shown by the next row, in which the neuron clearly learned a feature related to male human faces, as all of the images are portraits of men. Furthermore, it seems like the neuron below learned a feature related to engineering and technology, as the images show a rocket, phones, computer, and NASA, albeit one extraneous image that has nothing to do with technology. With more reduction in dead neurons, more categories could be formed, but these results demonstrate how the firing neurons are working to represent the features that the ResNet is learning. We did notice that some neurons fired on images that did not form categories, but this is due to how many neurons are dead, leading to their highest activation scores to not be very meaningful. However, the amount of neurons that represent key abstract features still demonstrates that the ResNet SAE captured many key concepts from the later layers of the ResNet.

4.6. Comparison Results

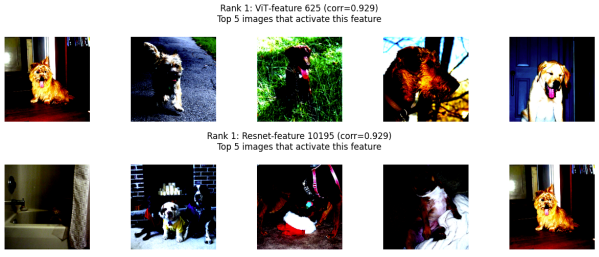
To compare the SAEs for the ViT and ResNet models, we used a Pearson correlation score to find the most correlated neurons for a particular set of images, creating a one-to-one neuron mapping between the ViT SAE and the ResNet SAE. Pearson correlation is defined by the formula

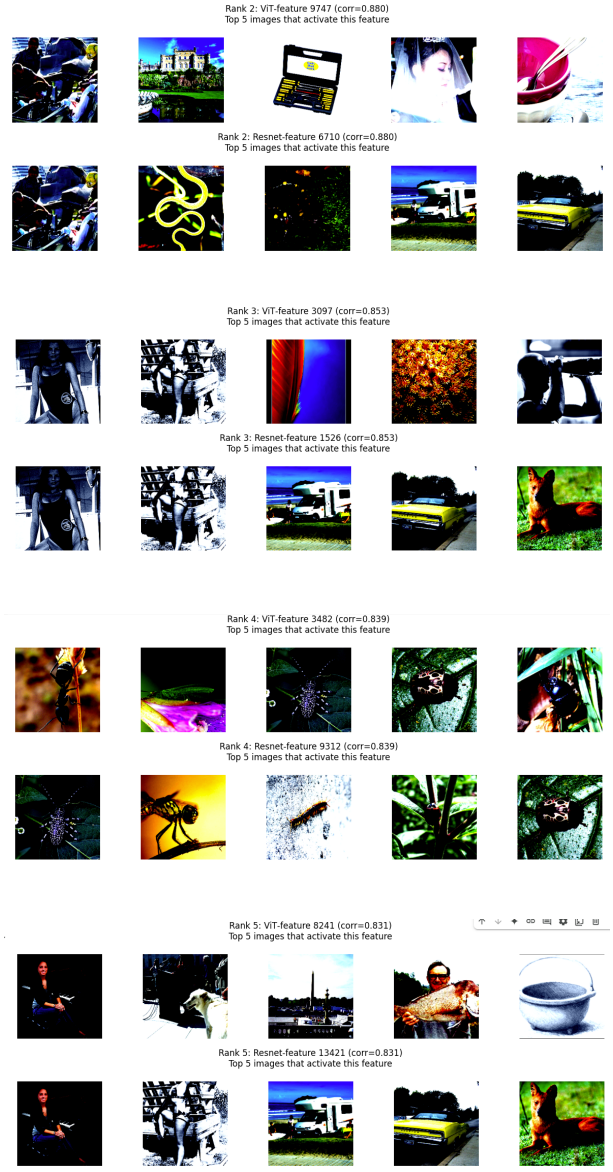
$$r_{X,Y} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})\,(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\,\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

, which in this context, measures how similarly two neurons fire on a set of images. We calculated this coefficient by first normalizing the activations from each of the SAE models on a validation set of images and then doing matrix multiplication to obtain the Pearson correlation coefficient matrix. Using this matrix, we filter out dead neurons (which would have a correlation of 1, since they always do not fire), and then search for the neurons that have the highest correlation. Below are some of the mappings and correlations between the mapped neurons, where max_diff is simply a measure of the largest difference in the activations of the neuron in the ViT SAE and the neuron in the ResNet SAE.

```
Top-10 non-trivial matches (j, k, corr, max_diff):
  ViT-feat   625 ↔ ResNet-feat 10195, corr=0.9286, max_diff=7.2847e+00
  ViT-feat  9747 ↔ ResNet-feat  6710, corr=0.8803, max_diff=1.0777e+00
  ViT-feat  3097 ↔ ResNet-feat  1526, corr=0.8527, max_diff=8.8750e-01
  ViT-feat  3482 ↔ ResNet-feat  9312, corr=0.8391, max_diff=5.9245e+00
  ViT-feat  8241 ↔ ResNet-feat 13421, corr=0.8308, max_diff=9.2760e-01
  ViT-feat  3174 ↔ ResNet-feat  3003, corr=0.8308, max_diff=2.7847e+00
  ViT-feat  5190 ↔ ResNet-feat  8073, corr=0.8256, max_diff=1.5728e+00
  ViT-feat  9000 ↔ ResNet-feat 13372, corr=0.8248, max_diff=1.2976e+00
  ViT-feat  9975 ↔ ResNet-feat 10008, corr=0.8217, max_diff=1.2931e+00
  ViT-feat  1127 ↔ ResNet-feat  5568, corr=0.8199, max_diff=5.8715e+00
```

Using these mappings, we displayed the images that each of the neuron pairs had the highest activations on and noticed that both the ViT and ResNet SAEs were learning similar features in many of their neurons.





From these images that each mapped ViT and ResNet SAE neuron had the highest activation scores on, we noticed that the mapped neurons were firing on similar or even the same images. For the most correlated neurons, they both had the highest activation scores for dog images and the highest activating image for the ViT neuron was the fifth highest activating image for the ResNet neuron. A similar phenomenon is shown by the fourth most correlated neurons, as they share many images of bugs and insects. The other most correlated neurons do have some overlapping images but do not appear to have any clear feature that they have learned, which is largely due to how the ResNet model had many dead neurons, leading to some neurons that did not actually capture a specific category. However, these results indicate that the ViT and ResNet SAEs are learning very similar features, as they have matching neurons for

images in the same category. Thus, this confirms our hypothesis that the later layers of the OpenAI CLIP ViT and ResNet do share many common learned features.

5. Conclusion

In summary, our hypothesis that we would find similar features across the ViT and the ResNet was weakly validated: we did often find non-trivially, semantically-aligned features across the ResNet and the ViT, such as the insects feature in the previous figure. This occurs *even with* a fairly poor-performing ResNet SAE, offering a positive direction for future results. An additional point of interest is that sparse autoencoders are clearly capable of undoing superposition in both the ViT and the ResNet, though parameter scales vary wildly; matching parameter values for the ViT SAE led to a poor-performing ResNet SAE, indicating that training dynamics are different in this aspect. This causes us to suspect that ResNet features *may* be meaningfully different in some important sense we could not identify, and future work might explore the cause of differences in training dynamics between the ViT and the ResNet SAEs.

Future work could also explore more advanced SAE training techniques, such as neuron resampling (we implemented neuron reinitialization, which was somewhat but not tremendously effective) in order to achieve even stronger and consistently interpretable results.

6. Contributions and Acknowledgments

In terms of data, F.Z. preprocessed the ImageNet-1k data and set up the HuggingFace Hub repository. He also implemented the different sparsity metrics, as well as the correlation scores and top images between matching pairs of neurons between the ViT SAE and ResNet SAE. Other than that, he trained the ResNet SAE using the same training loop as the ViT SAE.

L.G. set up the SAE and the hooks for each specific model. He wrote the code for streaming the data from HuggingFace Hub, as well as the training loop for the ViT SAE, and trained it by performing many hyperparameter sweeps. He also plotted the histograms for sparsity and the top activating images for specified neurons.

As a group, we both analyzed our qualitative results and wrote this paper together.

We thank the developers of PyTorch [10] and the OpenAI team who pretrained the CLIP model [11], and the creators of the python Datasets library. [8]

References

[1] Steering clip's vision transformer with sparse autoencoders.

[2] Emergence of simple-cell receptive field properties by learning a sparse code for natural images, 1996.

[3] Learning transferable visual models from natural language supervision, 2021.

[4] Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet, 2024.

[5] Sparse autoencoders find highly interpretable features in language models, 2024.

[6] Towards multimodal interpretability: Learning sparse interpretable features in vision transformers, 2024.

[7] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning, 2012.

[8] Q. Lhoest, A. Villanova del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. Šaško, G. Chhablani, B. Malik, S. Brandeis, T. Le Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. Delangue, T. Matussière, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. Lagunas, A. M. Rush, and T. Wolf. Datasets: A community library for natural language processing, 2021.

[9] C. O. N. S. T. H. S. K. Z. H.-D. R. L. D. D. C. C. R. G. S. M. J. K. D. A. M. W. C. O. Nelson Elhage, Tristan Hume. Toy models of superposition, 2022.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[11] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, Jul 2021.

[12] J. B. B. C. A. J. T. C. N. L. T. C. A. C. D. A. A. R. L. Y. W. S. K. N. S. T. M. N. J. A. T. K. N. B. M. J. E. B. T. H. S. C. T. H. C. O. Trenton Bricken*, Adly Templeton*. Towards monosemanticity: Decomposing language models with dictionary learning, 2023.