A Modern Convolutional Neural Network Interpretability Workflow

Thalis Goldschmidt, Blanka Visy and Paul Wollenhaupt

thalisg@uio.no, blankav@uio.no, paul.wollenhaupt@uio.no

29.11.2024

Abstract

Deep learning models have shown remarkable success in computer vision tasks over the past decade. However, these models are typically uninterpretable. Making neural networks interpretable is an active field of research that promises to increase trust and transparency, enable error analysis, bias correction, and more. This work focuses on proposing an efficient workflow that can be used to understand the internal mechanisms of a model by disentangling features in superpositions into more interpretable features and providing visualizations of these simplified features. Systematic quantitative and qualitative ablations of the workflow applied to a modified ResNet18 on Imagenet are included.

Contents

1 Introduction

Since the breakthrough of AlexNet in 2012, computer vision has experienced rapid advances driven by deep learning. AlexNet [Krizhevsky et al., 2012], a deep convolutional neural network (CNN), revolutionized image classification by significantly reducing error rates on benchmark datasets. Following this success, architectures such as [Simonyan & Zisserman, 2015], GoogLeNet [Szegedy et al., 2014], and ResNet [K. He et al., 2015] have further refined CNNs, achieving remarkable performance across various tasks. The development of techniques like transfer learning, data augmentation, and more efficient models (e.g., MobileNet [Howard et al., 2017], EfficientNet [Tan & Le, 2019]) enabled real-time applications on mobile and embedded devices. In recent years, transformers and attention-based models such as Vision Transformers (ViTs) [Dosovitskiy et al., 2020] have gained prominence, driving state-of-the-art performance in both image and video analysis.

The progress in computer vision has led to its increasing adoption across a wide range of industries. Increased model performance and the availability of large labeled datasets have facilitated the integration of computer vision into applications such as autonomous vehicles, medical imaging, retail analytics, security, and entertainment. Advances in hardware, particularly GPUs and specialized accelerators, have further enabled real-time processing, making computer vision accessible in mobile devices and edge computing environments. These developments have expanded the scope of computer vision, allowing for more accurate and efficient systems in complex, real-world scenarios, and enabling innovations like facial recognition, augmented reality, and smart manufacturing.

2 Motivation

The increasing reliance on deep learning models in computer vision has highlighted the need for interpretability to ensure their trustworthiness, transparency, and ethical use. As these models become more complex, they often function as "black boxes", making it difficult to understand how they arrive at certain decisions. This lack of transparency can lead to unintentional biases, errors, and a lack of accountability, especially in high-stakes domains like healthcare, autonomous driving, and law enforcement. Interpretability in computer vision not only helps to diagnose and mitigate such issues but also enables users to gain insights into model behavior, build confidence in their predictions, and ensure that models comply with ethical and regulatory standards. Moreover, it facilitates the improvement and refinement of models by providing a deeper understanding of the learned features and decision-making processes.

3 Background

3.1 Interpretability in Computer Vision

Interpretability in computer vision has evolved alongside advancements in deep learning models, paticularly as these models have become increasingly complex and deployed in critical applications. Early efforts in computer vision focused on simpler models, such as traditional machine learning algorithms and hand-crafted features (e.g., SIFT, HOG), where interpretability was more manageable due to the simplicity of these methods.

With the rise of deep learning, particularly convolutional neural networks, interpretability became a growing concern. Initially, the focus was on improving model performance, often at the expense of understanding model behavior. However, as deep models began to achieve state-of-the-art results, the need for interpretability emerged as a crucial challenge.

The first significant strides in interpretability came through visualization techniques, such as activation maximization [Erhan et al., 2009], saliency maps [Ismail et al., 2021], and Class Activation Mapping (CAM) first introduced by Zhou et al. [2015], which aimed to highlight regions in images that contributed most to the model's decision. Techniques like Grad-CAM, introduced in 2016 [Selvaraju et al., 2016], provided a more refined way to visualize the impact of specific layers in the network, helping researchers understand which parts of an image influenced predictions. Examples of an application of Grad-CAM are shown on Figure 1.

Figure 1: Examples of Grad-Cam. Figure source: [Lin & Jhang, 2021]

In parallel, researchers also explored post hoc interpretability methods, such as model-agnostic tools like LIME [Guo et al., 2017] and SHAP [Lundberg & Lee, 2017], which could explain the predictions of any black-box model by approximating it with simpler, interpretable models. As interpretability gained more attention, more advanced methods like feature attribution and counterfactual analysis were developed to provide deeper insights into how models learn and make decisions.

3.2 Mechanistic Interpretability

A recent and promising direction in the pursuit of interpretability is mechanistic interpretability (MechInterp) [Olah, Cammarata, Schubert, et al., 2020b], which aims to understand the internal mechanisms and structures within neural networks. Unlike traditional post-hoc interpretability techniques, which explain what a model does after training, mechanistic interpretability seeks to analyze and reverse-engineer how models function at a deeper level - identifying the specific roles of neurons, layers, and connections in producing a model's behavior. This approach draws on tools from neuroscience and cognitive science to elucidate how networks process and represent information. By uncovering the underlying mechanisms of model behavior, MechInterp aims to provide more fundamental insights into model decision

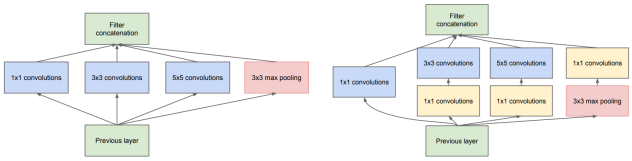
making, potentially leading to better control, efficiency, and generalization in neural networks.

Although this field is still in its early stages, significant progress has already been made, especially in the field of natural language processing. Our work aims to combine MechInterp tools, which have been successfully applied to large language models with standard computer vision interpretation tools to improve the workflow of interpreting computer vision models.

4 Related Work

4.1 MechInterp applied to InceptionV1

One of the first applications of what would become MechInterp [Olah, Cammarata, Schubert, et al., 2020b] was the interpretation of large parts of the well-known InceptionV1 model depicted on Figure 2. The first few layers were grouped into a taxonomy of different neurons, like Gabor filters, constrast filters, frequency and (multi-) color, lines, and more complicated patterns [Olah, Cammarata, Schubert, et al., 2020a]. Combinations of these simple neurons allow for the detection of curves. A mechanism, that was closely studied [Cammarata et al., 2020; Schubert et al., 2021]. Combining curve detectors in different rotations allows the model to do computations invariant and equivariant under rotations of the input image [Olah, Cammarata, Voss, et al., 2020]. A surprising result was that some later neurons seemed to activate when both high and low frequency patterns were present simultaneously [Schubert et al., 2021]. These studies were performed by manually inspecting real and synthetic images that maximally activate given neurons [Cammarata et al., 2021].



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception modules. Figure source: [Szegedy et al., 2014]

4.2 MechInterp in Language Models

MechInterp in NLP follows a similar trajectory to its use in computer vision, focusing on understanding inner workings of models, especially deep neural networks like the successful transformer architecture, which have become the backbone of many state-of-the-art NLP systems. For models in NLP, unlike MechInterp in CV, it is typically not easy to generate synthetic inputs that maximally excite a given neuron. This is because the input typically consists of discrete tokens (i.e., word fragments) and optimizing over discrete and combinatorial spaces is generally more computationally expensive compared to gradient based optimization in the continuous setting. However manual inspection of activations and attention patterns, which describe how information is dynamically routed in the model, on real and randomly generated data has led to the description of many neurons and two circuit types, called copy heads and induction heads.

A troubling finding however was, that LLMs seemed to represent multiple unrelated or anticorrelated features in single neurons, which made MechInterp research significantly harder. This phenomenon was later called polysemanticity and the mechanisms of its occurrence were further studied. Sparse autoencoders (SAE) were proposed as a promising solution for disentangling polysemantic neurons into their monosemantic (i.e., activating on only one semantic feature) constituents. They work by projecting activations of a given layer into a sparse representation and then mapping it back with a negligible reconstruction error. Interpreting the individual dimensions/neurons of this sparse representation has had remarkable success, thus finding better ways to achieve such representations has become an important research direction.

The successful application of SAEs to LLMs raises the interesting research direction of applying SAEs to computer vision models. This is exactly what was done in the study by Gorton [2024], which re-explores the first few layers of InceptionV1 with sparse autoencoders and finds curve detectors that resisted the searches by the previously described efforts. This work closely follows the procedure of Gorton [2024]. The main difference between their work lies in focusing on the last layer instead of the first one, as well as exploring various variations, such as different filtering functions, different SAE architectures, and comparing the features of standard feature maps with those of SAE-generated features.

Further related work is discussed in the sections of the individual parts of the workflow.

5 Methodology and Experiments

The workflow closely follows Gorton [2024]. It consists of four steps: First training the desired CNN model, second generating an activations dataset, third training a SAE on the activations, fourth applying a neuron visualization method to the hidden neurons of the SAE.

5.1 Training the Image Model

As a CNN we choose ResNet18 for its simplicity. However we modify it by replacing max-pooling by average pooling and ReLU activations by LeakyReLU to allow a better gradient flow through the model, since the gradient of max-pooling is zero for all non-maximum values in their window and the gradient of ReLU is zero for all negative values. The model is finetuned with heavy data augmentation until it reaches comparable accuracy to the original model. It should however be noted that a high classification accuracy was not an important objective. Further details can be found in Section 6.

5.2 The Sparse Autoencoder Dataset

The activations dataset is generated by sampling feature activations from the last convolutional layer of the model from the validation dataset. Different strategies of sampling were considered, including setting the probabilities proportional to the magnitude, the maximum, the number of neurons active above some threshold and results of the grad-cam localization method. The rationale behind this sampling according to Gorton [2024]

is to avoid features of background patches, which are easy to compress and easy to reconstruct, and therefore do not provide a challenging learning signal to the SAE during its training. Further details can be found in Section 7.

5.3 Training and visualizing Sparse Autoencoders

On these datasets SAEs with different hyperparameters and different architectures were trained, as further described in Section8. After a good set of hyperparameters was identified neuron visualization methods were applied to the hidden neurons of the CNN with SAE, as well as CNN without for comparability. These techniques include creating synthetic images, that maximize the activation, of a given neuron and filtering ImageNet for highly activating example images. Further details can be found in Section 9.

6 Model training

6.1 Related work

Deep learning has been an active area of research in recent years, with significant focus on computer vision, particularly image classification.

Image classification models Lecun's seminal work in 1998 introduced LeNet, a foundational architecture for convolutional neural networks, designed for handwritten digit recognition [Lecun et al., 1998]. This groundbreaking model laid the groundwork for many subsequent advancements, inspiring a variety of approaches aimed at the same objective.

Research on deep CNNs includes AlexNet [Krizhevsky et al., 2012], which introduced ReLU activations and dropout to combat overfitting. ZFNet [Zeiler & Fergus, 2013] improved on AlexNet by using smaller kernels and shorter strides, enhancing performance and visualizing feature maps. GoogLeNet increased network depth with the Inception structure [Szegedy et al., 2014], VGG focused on increasing network depth [Simonyan & Zisserman, 2015] as other explored feature channel separation [Hu et al., 2017].

Region-Based Convolutional Neural Networks (R-CNNs), introduced by Girshick et al. [2013], identify regions of interest in an image and classify them independently using a CNN, improving classification in complex scenes. Subsequent advancements reduced computational costs and incorporated techniques like masking to enhance performance [Girshick, 2015; Ren et al., 2015; K. He et al., 2017].

Residual Networks (ResNets), introduced by K. He et al. [2015], address the challenge of training deep neural networks by introducing skip connections, which enable the network to learn residual mappings instead of direct mappings. This innovation mitigates vanishing gradient issues and significantly enhances training stability and convergence, allowing networks to achieve unprecedented depths. ResNets demonstrated state-of-the-art performance on image classification benchmarks, also on ImageNet, by enabling deeper architectures to generalize effectively. Several newer models have built on the foundational principles of ResNet, extending its ideas or addressing its limitations to achieve improved performance and efficiency. Without the need of completeness, some of them are: Resnet In Resnet model by Targ et al. [2016], the ResNeXt model by Xie et al. [2016], the Wide ResNet Zagoruyko & Komodakis [2016], the Res2Net by S. Gao et al. [2019] as well as the ConvNet by Liu et al. [2022].

Research on lightweight networks focuses on reducing size and computational cost while maintaining accuracy. ShuffleNet [Zhang et al., 2017] introduces pointwise group convolution and channel shuffle, making it suitable for mobile devices. Other notable lightweight networks include MobileNet [Howard et al., 2017], DenseNet [Huang et al., 2016], FBNet [Dai et al., 2021], and EfficientNet [Tan & Le, 2019].

The Transformer model, introduced for sequence transduction [Vaswani et al., 2017], was adapted for image classification by Dosovitskiy et al. [2020] with the Vision Transformer (ViT), which relies entirely on attention mechanisms. Further advancements, like Swin Transformers [Liu et al., 2021], refine transformer architectures for image pixel domains using hierarchical models with shifting windows.

To sum up, there have been many different approaches in image classification, which also have been evolving and continuously changing. A comparison of the different models performances based on the work of K. Xu et al. [2023] can be seen in Table 1. Model performance is reported on the ImageNet-1K dataset Deng et al. [2009].

Model	**Top-1 Accuracy (%)**	**Top-5 Accuracy (%)**
AlexNet	63.3	84.7
GoogLeNet	-	92.68
ResNet-50	77.15	93.29
ResNet-101	78.25	93.95
ResNeXt-101	78.8	94.4
ShuffleNet 3.4 M	71.5	-
DenseNet-169 14M	76.2	-
EfficientNet-B0 5.3M	77.1	-
FBNetV3-A 5.3 M	79.1	-
VIT-L/16	76.5	-
Swin-B Transformer	83.5	-

Table 1: Comparison of Model Performance in Top-1 and Top-5 Accuracy.

Our focus in this project was not to aim for the highest classification accuray, but to develop an interpretability pipeline. Therefore, we decided to choose one the Resnet18 model [K. He et al., 2015], due to its wide applications and simple architechture.

Modifications However, there are other crucial parts of neural networks than their architecture, such as activation functions. Recent studies indicate that incorporating a nonzero slope for the negative part of rectified activation functions can consistently enhance the performance of CNNs [B. Xu et al., 2015; Khalid et al., 2020; Varshney & Singh, 2021]. More advanced activation functions, such as the fully parametric rectified linear unit (FReLU) proposed by Varshney & Singh [2021], have demonstrated even higher accuracy in image classification. However, these methods often involve more complex optimiza-

tion processes. For the sake of simplicity, we opted to use LeakyReLU in our project as a balanced alternative.

Another key component in the design of CNNs is the pooling block. Pooling blocks in CNNs reduce spatial resolution for better semantic extraction and computational efficiency [Bieder et al., 2021]. Max-pooling, introduced by Weng et al. [1992], is widely used, while average pooling was first used in LeNet [LeCun et al., 1998]. While both methods often yield similar performance [Nurjannah et al., 2022; Bieder et al., 2021], max-pooling can sometimes outperform average pooling Zhao & Zhang [2024]. However, since our primary goal was not to maximize classification accuracy, we ultimately chose average pooling. Max-pooling can introduce extreme outliers by selecting only maximum values and passes gradient signals to only a subset of pixels. Given the need for robustness and smooth feature representation, we opted for average pooling over max-pooling.

6.2 Methodology

We considered two approaches for our model: fine-tuning a modified, pre-trained ResNet-18 model or building our own custom CNN. The architectures for both models are shown in Figure 3.

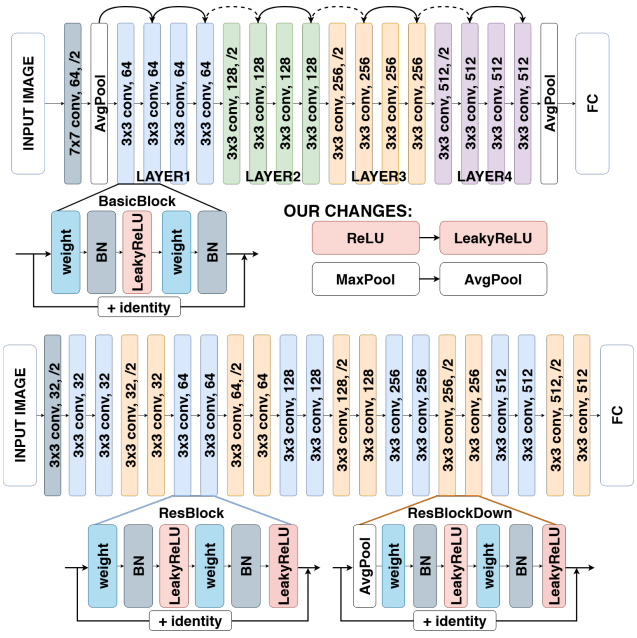


Figure 3: Model architectures used. *Top*: ResNet18 architecture with the applied modifications. *Bottom*: our own CNN architecture.

Initially, we trained both models on the original, shuffled training set. For both models, we observed severe overfitting. To address this, we implemented two strategies:

- **Image augmentations**: on the training dataset, we applied different random augmentations. We employed random resized cropping, random horizontal flips, color jitter, and random erasing. The combination of these data augmentation techniques enhances the diversity of the training data, making the model more robust to variations in object position, orientation, lighting, and occlusions. Together, these augmentations improve the model's ability to recognize objects under diverse and challenging conditions and help to achieve better validation accuracy.
- **Learning rate scheduler**: we also applied a learning rate scheduler. Finding the appropriate learning rate curve is crucial in the training of deep neural networks. We decided to employ cosine annealing learning rates initially proposed by Loshchilov & Hutter [2017]. This learning rate has empirically showed in multiple studies that it resulted in lower final loss than not using a dynamic learning rate ceteris paribus [Johny & Madhusoodanan, 2021; L. He et al., 2024].

6.3 Experimental setup

We used two datasets for our experiments. The first was the Tiny ImageNet dataset [Hugging Face, 2023], which consists of 100,000 images across 200 classes, with each class containing 500 training images, 50 validation images, and 50 test images. All images are 64×64 pixels and in color.

Subsequently, we applied the same training pipeline to the full-sized ImageNet dataset [Deng et al., 2009]. This dataset includes 1,281,167 training images and 50,000 validation images, covering 1,000 object classes. We cropped the images to have a uniform 224*224 resolution.

In both cases, the data was split into training and validation sets, with augmentations applied only to the training set. The models were trained using a batch size of 32 and a cosine annealing learning rate schedule starting at 4e-5. To ensure reproducibility, the augmentations were configured with an optional seed value.

On TinyImagenet, we ran both models described on Figure 3 for 100 epochs.

On ImageNet, we fine-tuned the pre-trained Resnet18 model for 10 epochs. Due to the size of the data, more epochs would have been too computationally expensive.

6.4 Results

TinyImageNet We ran the two models shown on on TinyImagenet. First, we had problems with the model overfitting. With the use of augmentations and the learning rate scheduler, we had a better generalization. Figure 4 it shows the training of the two models on the augmented TinyImagenet dataset, pointing out that fine-tuning the pre-trained ResNet model is better both in validation performance. Therefore we choose to take that model for the larger dataset.

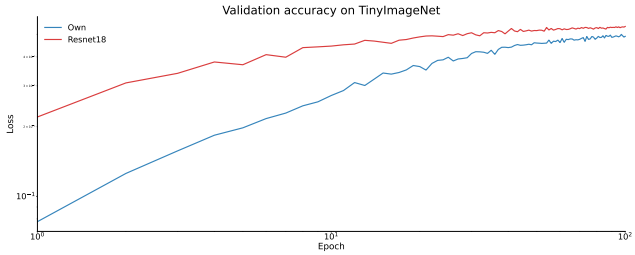


Figure 4: Validation accuracy during training on TinyImagenet

ImageNet On Figure 5 we can see that the model already starts with a relatively good performance that increases even more. No over-fitting can be seen on the plot. The validation accuracy is already surpassing the training accuracy at the start, which can be attributed to the use of significant data augmentation on the training dataset. Our final validation performance is evaluated in Table 2.

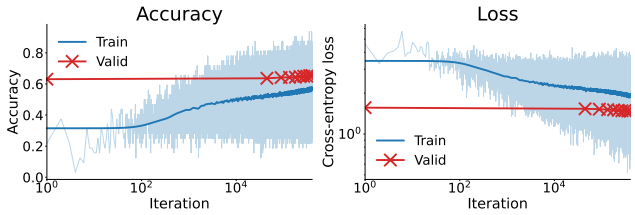


Figure 5: Accuracy and cross-entropy loss during the fine-tuning of the modified Resnet18 model

Model	Top-1 Accuracy	Top-5 Accuracy	Loss
Our model	65.45%	86.32%	1.47
Pre-trained Resnet18	69.76%	89.08%	*NA*

Table 2: Model Performance Comparison. Pre-trained model performance is by [PyTorch, 2024]

.

Next, we performed error analysis taking the work of [Wu et al., 2017] as a basis. The results can be seen on Figure 6.

Figure 6: Error analysis of our final model on the ImageNet dataset. *Left*: labels with least error. *Middle*: labels with most error. *Right*: mostly mistaken classes.

6.5 Discussion

Our model on ImageNet achieved slightly worse results compared to the pre-trained weights-one. Additionally, when compared to newer, more complex models that dominate classification leaderboards also summarized in Table 1, its performance is inferior. However, our choice of model was deliberate: we opted for a widely used network with a relatively simple structure, which we modified to improve gradient flow. Considering this, the performance is comparable to the pre-trained model and competitive with other models of similar parameter size and release date. Thus, we are satisfied with the results.

Our results regarding error analysis align with the work of Wu et al. [2017]. The plot demonstrates that, similarly to their findings, the class with the highest accuracy primarily consists of similar elements that fit into a coherent color scheme (green as the background) and feature highly distinguishable characteristics (such as the nose). This could be particularly interesting for feature visualizations—specifically, whether there are neurons that specialize in detecting these noses. On the other hand, the class with the lowest accuracy contains more varied items, both spatially and in terms of color, with multiple objects often present in a single image. Classes that are frequently misclassified also tend to be difficult to distinguish even for humans, as they often involve very similar dog breeds, which likely extend beyond purely visual characteristics.

7 Activation filtering

In Convolutional Neural Networks used for computer vision tasks, a feature map—also known as an activation map—represents a two-dimensional grid of values produced by applying convolutional filters (or kernels) to an input image or the feature map from a previous layer. Feature maps are integral to the CNN architecture, as they capture various visual features or patterns in the input data [Azam et al., 2023].

In our project, we use these feature maps as a training basis for Sparse Autoencoders (SAEs) with the goal of enhancing explainability, particularly for neurons with polysemantic behavior. This points are further evaluated in Section 8.

7.1 Related work

The concept of filtering activations in this work was inspired by Gorton [2024]. As already summarized in Section 4. In their study, they applied SAEs to the early vision layers of InceptionV1, a well-known CNN. During dataset creation, they oversampled activations based on the magnitude of the activation maps. Building on their work, we introduce several key differences: our focus shifts to ResNet18, specifically targeting its later layers, while also exploring and comparing different activation filtering functions. Additionally, we incorporate visual confirmation to validate these hypotheses, providing a more comprehensive analysis.

There has been prior research investigating feature maps used for training SAEs. For example, Azam et al. [2023] analyzed feature maps generated by convolutional layers, quantifying their diversity through statistical methods. Their study includes various analyses, such as comparing feature maps across layers, examining changes over iterations, and analyzing interclass variances. They demonstrate how feature maps evolve during training, and arenot only different across different images but also for a single image across multiple layers. Additionally, they quantify the number of black feature maps that can occur. While their work aligns with ours in focusing on feature map analysis, our study places greater emphasis on SAE training and specifically on comparison of feature maps from a single layer.

We want to filter out unimporant activations: class activation mapping (CAMs) is also doing a similar things, as is a widely adopted technique that allows to visualize the predicted class scores on any given image, highlighting the discriminative object parts detected by the CNN Zhou et al. [2015]. Grad-CAM Selvaraju et al. [2016], a variant of this approach, enhances the methodology by leveraging gradients to assess the impor-

tance of activation maps, thus making it applicable to a wider range of CNN architectures. By visualizing input regions with high-resolution details that contribute significantly to predictions, Grad-CAM has increased the transparency of CNN-based models. Numerous subsequent studies have built upon this technique to further advance its capabilities [Chattopadhyay et al., 2017; Omeiza et al., 2019; Desai & Ramaswamy, 2020]. The main difference of these works and our project is that we focus not on explainability, but on using saliency maps to enhance SAE training.

7.2 Methodology

To extract feature maps and build a training dataset, we utilized a fine-tuned model introduced in Section 6 with fixed weights and the validation dataset. We processed the validation dataset by iterating over a number of images to obtain the outputs from a selected layer. We refer to the output of a layer as activation maps and to the vectors of each patch as activations.

Gorton [2024] points out that most image positions result in small activations (e.g., in background areas). To reduce computational cost by avoiding the modeling of these small activations, they applied an oversampling technique, selecting activation positions in the image based on the magnitude of their activations.

Based on this idea, we tested different methods to filter out important activations.

Definition 7.1 (Activation filtering functions)**.** An activation filtering function f is a function that maps $\mathbb{R}^{(n,x,x)} \to \mathbb{R}^{(x,x)}$, where the first dimension n represents the activations of the input, and the two x dimensions correspond to the image patches. These functions reduce the activations within each patch to a single value. Since they are used for sampling purposes, the output values are non-negative.

7.2.1 Magnitude

We initially used the absolute magnitude of the activations, as recommended by Gorton [2024]. Given an activation in an activation map $\mathbf{A}$, the function is defined as:

$$f_{magnitude}(\mathbf{A}) = \sum_{i=1}^{n} |\mathbf{A}_i|$$

where $\mathbf{A}_i$ represents the elements along the summation dimension, $|\cdot|$ denotes the absolute value, and n is the size of the summation dimension.

7.2.2 Number of Activated Neurons

Based on recent research on activation-based pruning techniques [Ganguli & Chong, 2024], we adopted a filtering method that counts neurons in the activations whose values exceed a threshold ϵ.

$$f_{neurons}(\mathbf{A}, \epsilon) = \mathbf{B}, \quad \text{where} \quad \mathbf{B}_j = \sum_i \mathbb{I}(|\mathbf{A}_{i,j}| > \epsilon)$$

$\mathbf{A}$ is the input activation. The threshold ϵ determines whether a neuron output is considered active.

7.2.3 Median

We also incorporated a distribution-based metric using the median. A higher median indicates that the top 50% of activations in a patch are greater compared to others, signifying stronger overall activation.

$$f_{median}(\mathbf{A}) = \max\left(\text{median}(\mathbf{A}, \text{dim} = 0), 0\right)$$

Where $\mathbf{A}$ is the input activation vector. The result of the median computation is then clamped such that all negative values are set to zero, using the operation $\max(\cdot, 0)$ element-wise. This is so that the in the sampling process, only non-negative values remain.

7.2.4 Maximum

To account for significant outliers, we included a maximum-based filtering approach. This method emphasizes patches where even a single neuron has a notably high activation.

$$f_{max}(\mathbf{A}) = \max\left(\max(\mathbf{A}, \text{dim} = 0), 0\right)$$

Here $\mathbf{A}$ is again the input activation vector, represented as a tensor. The result of the maximum computation is also clamped to zero to ensure non-negativity.

7.2.5 Grad-CAM

: Finally, we incorporated Grad-CAM Selvaraju et al. [2016] in our comparison. Grad-CAM generates saliency maps by combining the activation map with gradients relative to the true class of the image, offering an interpretability-focused perspective. For the implementation, we leveraged the work of Gildenblat & contributors [2021] as a foundation, alongside the original paper [Selvaraju et al., 2016]. First, we compute the importance weights for the target class of a given image as follows:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{1}$$

Where:

- $L^c_{\text{Grad-CAM}}$: Class-discriminative localization map for class c, with dimensions $u \times v$ (width u, height v).
- y^c: Score for class c before softmax.
- A^k: Activations of the k-th channel of the convolutional layer.
- $\frac{\partial y^c}{\partial A^k}$: Gradient of the class score y^c with respect to the activations A^k.
- α_k^c: Neuron importance weight for the k-th channel, obtained by global average pooling over the width and height.

This equation represents the mean of the gradients obtained via backpropagation. Next, we perform a weighted sum of the forward activation maps, followed by a ReLU operation to derive the saliency map:

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right) \tag{2}$$

Finally, we interpolate the resulting map to a 7×7 grid and normalize the values to the range $[0, 1]$ for subsequent sampling.

7.2.6 Sampling pipeline

After defining the filtering functions, we followed the pipeline illustrated in Figure 7. We applied the filtering functions to an activation map, generating a grid of individual values from each of the activations. We then checked whether the sum of these values was zero or if all of them were NAs (a situation that can occur with for example Grad-CAM). In such cases, we performed random uniform sampling and tracked the total number of these occurrences. Next, we replaced the NA values with zeros to prevent them from being sampled. Subsequently, we normalized the probability values to the range [0, 1] by dividing them by their sum, and then performed sampling from the activation vectors based on these probabilities, with replacement.

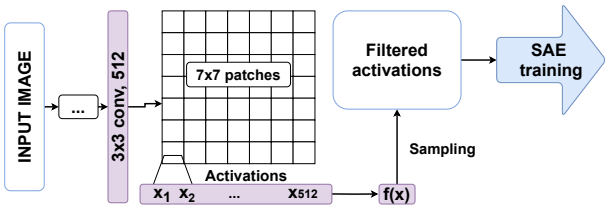


Figure 7: Pipeline of activation sampling

7.3 Experimental setup

We evaluated the activation maps using the filtering functions on `layer4` of our model. We chose this layer because it provides the deepest output before the final fully connected layers, making it a suitable candidate for evaluation. The output of `layer4` for each image was a $512 \times 7 \times 7$ activation map. After applying the filtering functions on the activations, we obtained a single value for each 7×7 grid element.

We performed the evaluation on the validation dataset, as it was not augmented and allowed us to assess the interpretability of unseen features. We visualized and saved 5-5 images showcasing the filtering values and ranks for qualitative evaluation.

Next, we applied the sampling process with all five filtering functions to a total of 20,000 elements from the validation dataset, as a higher number would have been computationally expensive. For each image, we sampled $7 \times 7 = 49$ activation vectors to preserve the original size of the image, ensuring comparability. Finally, we saved both the 7×7 filtered maps and the sampled activation maps for further analysis. We used a threshold of 0.1 for the number of activated neurons, as both lower and higher thresholds often resulted in identical values for many patches.

For SAE training, we modified the training data using different sampling models while keeping other conditions constant. The results were validated on magnitude-sampled datasets using 10% of the data. Each dataset was reduced to 90% of its original size to ensure consistent training data sizes.

7.4 Results

First we evaluated the results visually. Figure 8 shows an example of the validation dataset, showcasing the original image and the filtering function values on the patches of the differing filterings.

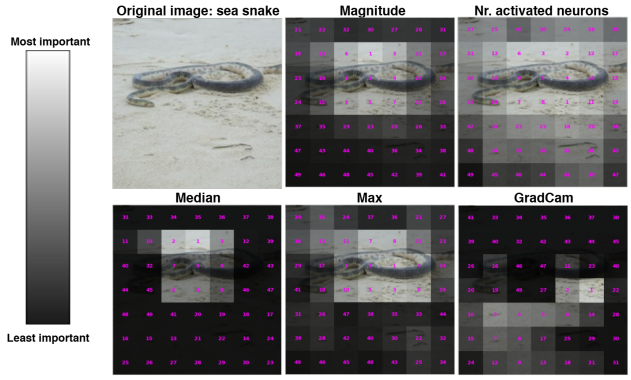


Figure 8: Values of the filtering function applied to different patches of a single image in the validation dataset. Transparency is represented on a continuous scale, ranging from the least important to the most important. The numbers indicate the rank of the patches.

On this example as well as the other one can see the differences in the different filtering functions. Almost all of them most often capture the patches including the target object well, except for the GradCam. The GradCam often fails to do so, giving important patches low weight and background patches high weights.

Regarding all-zero cases, neither the magnitude, the number of activated neurons, nor the maximum filtering functions resulted in all-zero weights. However, for the median, 0.46% of the cases had negative median values for all patches, which were clamped to zero. In these instances, random sampling was performed. For GradCAM, this occurred in 71.8% of the cases. As GradCAM was also the most computationally expensive filtering method relying on gradients as well and of this high fracton of random sampling we decided to not further evaluate GradCAM.

Next, we compared the similarities of the weights produced by the filtering functions. To do this, we calculated the cosine similarity and Spearman Rank Correlation Coefficient ["Spearman Rank Correlation Coefficient", 2008] of the weights for each image and then averaged them. The left side of Figure 9 shows the cosine similarities between the different weights. The results indicate that all features exhibit positive correlations, meaning that the weight vectors generally point in the same direction. The magnitude-based weights are similar to most other vectors, aligning almost perfectly with the weights based on the number of activated neurons but showing less similarity to the median weights. The closest pair is the number of activated neurons and the maximum value, while the most distant pairs are median-magnitude and neurons-median. This suggests that the median function likely produces the most distinct results among these filtering functions. The rank correlations on the right side of Figure 9 show similar patterns. These two combined shows that not only are the values close to each other, their ranks are also often similar.

Finally, we compared the training performance of the Sparse Autoencoder (SAE) on the differently sampled datasets. Figure 10 shows the training error, where the lowest error is achieved with the original dataset. The other filtering functions perform similarly, with sampled datasets converging faster. Among the sam-

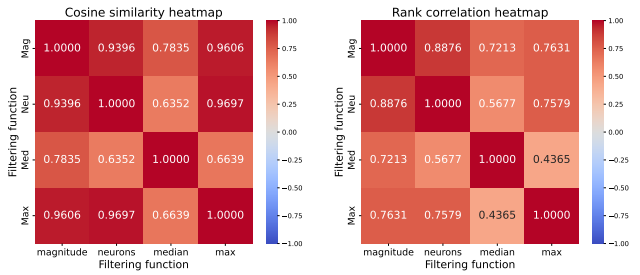


Figure 9: *Left*: Cosine similarities of the weights resulting of the filtering functions. *Right*: Rank correlation of the weights resulting of the filtering functions.

pled datasets, the median and the number of activated neurons achieve the lowest training error in the end.

However, training error is not necessarily a good measure for our project. Since the goal is to create a more challenging dataset, higher error could simply reflect this increased difficulty. Therefore, as described in Section 7.3, we validated all models on the same validation dataset of magnitude sampling. The smoothed validation loss of the different datasets can be seen on 11.

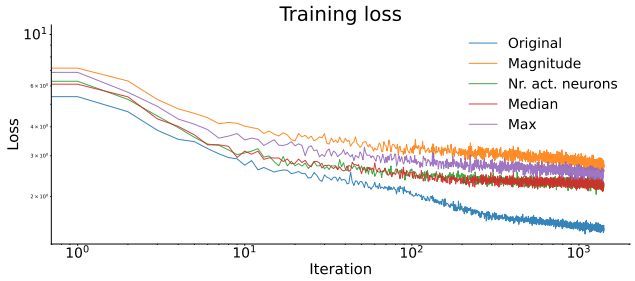


Figure 10: Training error of the SAE.

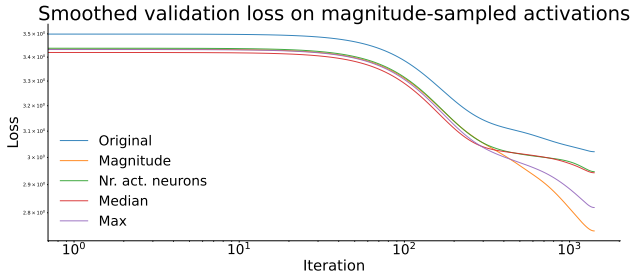


Figure 11: Validation error of the SAE models, validated on part of the magnitude sampled dataset. We used 1d Gaussian smoothing to make the results more comparable.

7.5 Discussion

The approach described in Gorton [2024] involved sampling background patches using only magnitude-based filtering. Our visual results confirm that magnitude-selected patches primarily focus on regions containing the target element, while background patches often exhibit lower magnitude values. However, the similarity across different filtering functions suggests that magnitude may not be the only way to go.

Applying GradCAM on a patchwise basis did not yield satisfactory results. The issue of generating only zero outputs stems from the ReLU layer in the algorithm: when the model fails to identify any patches contributing to the true underlying label, all values become negative and are subsequently mapped to zero. This highlights that a GradCAM-based method may not justify the additional computational cost, particularly when the validation error is relatively high.

The training errors of the SAEs depicted on Figure 10 further support the objective of Gorton [2024], which aimed to create a more challenging dataset for the SAE. As expected, the original dataset achieves lower errors due to the background patches often containing low-amplitude noise, which reduces the overall error. However, a comparison of the learning curves suggests that SAEs trained on the sampled datasets exhibit a slightly faster convergence rate compared to those trained on the original dataset.

Examining the validation losses in Figure 11, the previously described hypothesis is confirmed. For more complex datasets, the SAE trained on the original images with more background patches struggles to achieve good performance. In contrast, SAEs trained on oversampled datasets outperform the original model from the initial iterations and exhibit slightly steeper loss curves. Differences between the filtering techniques are minimal, though magnitude and maximum filtering yield the best results, likely due to their similarity in training data.

To summarize, our experiments demonstrate that applying filtering techniques to the activation datasets of SAEs significantly impacts performance. Training on filtered datasets has larger training loss due to the less background patches. When validated on more challenging datasets, models trained on filtered data outperform those trained on unfiltered data. Thus, we conclude that filtering mechanisms enhance SAE training, achieving better validation performance in fewer iterations. Additionally, our results suggest that the filtering functions used are often similar in their weights, leading to comparable influences on SAE training.

8 Sparse Autoencodrs

Autoencoders are neural networks designed to learn compressed representations of input data by training to reconstruct the input at the output layer. Typically consisting of an encoder that compresses the input into a latent space representation and a decoder that reconstructs the input from this representation, autoencoders are widely used in unsupervised learning tasks, such as feature extraction, dimensionality reduction, and anomaly detection [Rumelhart et al., 1986; Bank et al., 2021].

A Sparse Autoencoder (SAE) is a variation of the basic autoencoder that incorporates a sparsity constraint on the latent space representation. This constraint encourages the network to learn representations where only a small number of neurons are active for any given input. In this way, sparse autoencoders promote interpretability, improve generalisation, and facilitate the learning of meaningful and disentangled features, particularly in high-dimensional datasets [Cunningham et al., 2023].

Sparsity in a SAEs is typically enforced by one or both of the following two mechanisms. The first mechanism is to add regularisation terms to the gradient based training of the SAE.

Adding a sparsity penalty to the loss function can encourage some neurons in the hidden layer to remain inactive. The other mechanism works by explicitly restricting activations. For example, by applying a specific activation in the latent space of the SAE [Rajamanoharan, Conmy, et al., 2024; Rajamanoharan, Lieberum, et al., 2024; L. Gao et al., 2024]. An illustration of the SAE architecture is given in Figure 12.

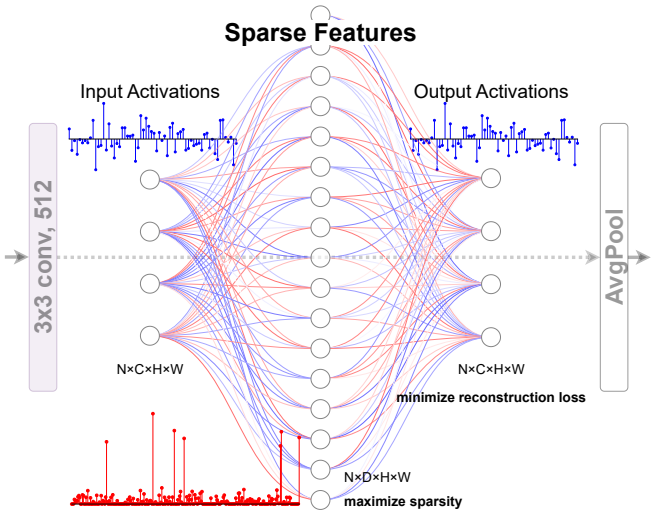


Figure 12: An overview of the sparse autoencoder. The inputs with layout NCHW, where N is the batch, C is the number of channels and HW is the height and width, are expanded by a linear neural network along the C-axis. The previously dense activations become positive and sparse and are then mapped back to resemble the input as closely as possible [Gorton, 2024].

8.1 The Importance of Sparsity

Mechanistic interpretability aims to understand how neural networks process and represent information at a detailed mechanistic level, uncovering how features are encoded in network activations and how these features contribute to decision making [Olah, Cammarata, Schubert, et al., 2020b]. Sparse autoencoders are particularly well suited to this task because of their empirically demonstrated ability to learn interpretable and disentangled representations [Cunningham et al., 2023]. Neural networks often encode features in high-dimensional spaces with dense activations, making it difficult to trace specific features back to a small and simple set of contributions. Dense activations also lead to redundant or highly distributed representations, which complicate efforts to assign unambiguous meanings to individual neurons or features [Anthropic, 2024]. Furthermore, dense and possibly overlapping feature representations often obscure the relationships between inputs, learned features and outputs, making the network's decision-making process opaque [Anthropic, 2024].

Sparse autoencoders address these challenges by constraining activations, ensuring that only a small subset of neurons respond to a given input. This sparsity facilitates mechanistic interpretability by reducing redundancy, promoting discrete feature representations, and making it easier to isolate the most relevant features driving a network's behaviour. Sparse representations are also consistent with observations from biological systems, such as the selective responses seen in sensory cortices, and provide a biologically inspired framework for understanding efficient feature encoding [Olah et al., 2017; Anthropic, 2024].Mechanistic interpretability aims to understand how neural networks process and represent information at a detailed mechanistic level, uncovering how features are encoded in network activations and how these features contribute to decision making.

8.1.1 The Linear Representation Hypothesis

The linear representation hypothesis proposes that the internal representations learned by neural networks can often be interpreted as linear combinations of meaningful features. According to this hypothesis, the hidden layers of a network encode information in a way that preserves semantic structure, allowing features such as edges, textures, or high-level concepts to be linearly separable within the activation space. This suggests that tasks such as classification, translation or image synthesis can be achieved by relatively simple operations on these representations [Park et al., 2024].

The hypothesis has important implications for the understanding and interpretation of neural networks. It provides a basis for analysing internal activations by assuming that complex transformations performed by the network can be approximated or explained in terms of linear relationships. For example, in natural language processing models such as transformers, the hypothesis underpins techniques such as probing classifiers, which test whether a particular feature (e.g. syntactic structure or sentiment) is linearly encoded in the model's representation space. Similarly, in vision models, the hypothesis is consistent with findings that convolutional networks learn filters corresponding to interpretable patterns such as edges and shapes [Park et al., 2024].

Empirical evidence supports the linear representation hypothesis in many cases, particularly for features that match the network's training goals. However, deviations from linearity can occur in networks that rely heavily on non-linear activations or encode highly entangled representations [Park et al., 2024].

8.1.2 Polysemanticity

Polysemanticity refers to the phenomenon where a single neuron or feature in a neural network represents multiple, often unrelated, concepts or functions. This is a common feature of large, over-parameterised models, where neurons are often reused across different tasks or contexts. While this property allows for efficient use of network capacity, it complicates efforts to interpret the role of individual neurons or functions, as their activations cannot be easily mapped to a single concept [Anthropic, 2024].

Polysemanticity poses a significant challenge to mechanistic interpretability. When a neuron responds to several different patterns or encodes several features, it becomes difficult to disentangle its contributions to specific network behaviours. For example, in models of vision, a polysemantic neuron might activate in response to both geometric patterns, such as circles, and unrelated semantic categories, such as animals. Similarly, in language models, a single feature might contribute to syn-

tax parsing in one context and sentiment analysis in another [Anthropic, 2024; Gorton, 2024].

8.1.3 The Residual Stream and the Priveleged Basis

In transformer architectures, or more generally in neural networks with residual connections, the residual stream is a critical component that facilitates the flow of information through the network by accumulating and combining the outputs of successive layers. It serves as a central repository for the model's intermediate representations, with each layer operating by reading from and writing to this shared stream. The structure of the residual stream is additive: the output of each layer is added to the existing residual, allowing the model to propagate information efficiently while allowing each layer to contribute incremental updates [Anthropic, 2024].

When the linear representation hypothesis is adopted, a basis of the vector space of activations is chosen. This basis is called the privileged basis, because when the activation is rotated into this basis, each entry can correspond to a specific semantic feature. However, this basis does not have to correspond to the standard basis in which the neural network is implemented. Especially in the transformer residual stream, there is little reason to assume that the standard basis is privileged. The false assumption that the standard basis is privileged leads trivially to polysemy. However, this insight is not sufficient to fully motivate the use of sparse autoencoders, since in this case it would be sufficient to find the priveleged basis and then align to it.

8.1.4 Superposition

However, finding monosemantic features is often not as simple as finding the privileged basis, because neural networks can exploit a far less intuitive phenomenon, superposition [Anthropic, 2024; Cunningham et al., 2023].

Superposition in mechanistic interpretability refers to the phenomenon whereby neural networks encode multiple unrelated features in overlapping dimensions of their internal representation spaces. Unlike human-designed systems, which often devote distinct dimensions to specific features, neural networks optimise for efficiency during training, resulting in representations that use the same dimensions to encode multiple features simultaneously. This phenomenon occurs because the number of features a model needs to represent often exceeds the dimensionality of its layers, leading to a form of feature "compression" [Anthropic, 2024].

This phenomenon is closely related to the geometry of the representation space. Features encoded in superposition typically occupy subspaces rather than individual dimensions, allowing multiple features to coexist in overlapping but distinct and nearly orthogonal directions. Techniques such as projection onto feature-specific subspaces or sparse representations can help to analyse and disentangle these features. Superposition along with interference, the linear representation hypothesis, the privileged basis and polysemanticity is illustrated by a 2D example in Figure 13

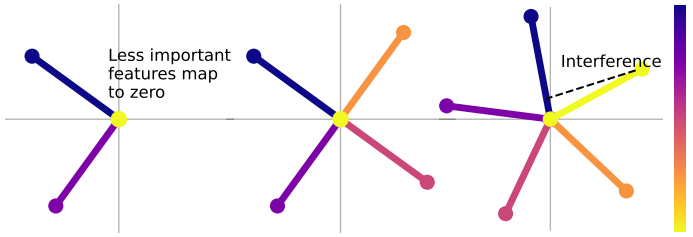


Figure 13: This figure visualises a different number of features with increasing importance, plotted linearly. The first plot on the right shows two features in a two-dimensional space. The privileged basis given by the features does not coincide with the standard basis. The second plot shows how sparsity can be used to represent twice as many features as there are neurons. Here the antipodal features cannot be active at the same time. However, this is not a problem if they are sufficiently anti-correlated. The third plot shows how even more features can be fitted into this two-dimensional space using superposition. Here, however, if a feature is active, it can positively interfere with neighbouring features because they are only almost orthogonal. This effectively adds some noise to the calculation, which can lead to false positive activations. Models trade off the capacity enabled by superposition and the noise introduced by stronger superposition [Anthropic, 2024].

8.2 Related Work

The concept of sparsity in neural networks has its roots in early work on sparse coding and dictionary learning. In particular, Olshausen & Field [1997] introduced the notion of sparse coding in the context of natural image processing, showing that sparse representations could capture important features such as edges and textures. This concept laid the groundwork for later developments in sparse autoencoders, where the sparsity of hidden layer activations is encouraged to discover such features more explicitly.

8.2.1 Problem Statement

The typical sparse autocoding problem can be formalised as follows. Let $(x_1, \dots x_n) \overset{\text{i.i.d.}}{\sim} \mathrm{P}$ be independent and identically distributed random variables in the surrounding space $\mathbb{R}^c$ with the number of features, also called channels $c \in \mathbb{N}$. Then the goal is to find an encoder function $f_{\text{enc}} : \mathbb{R}^c \to \mathbb{R}^d$ and a decoder function $f_{\text{dec}} : \mathbb{R}^d \to \mathbb{R}^c$, with latent space dimensionality $d \in \mathbb{N}$, which optimises the following two goals

$$\mathcal{L}_{\text{fidelity}} := \sum_{i=1}^{n} p((f_{\text{dec}} \circ f_{\text{enc}})(x_i), x_i) \tag{3}$$

$$\mathcal{L}_{\text{sparsity}} := \sum_{i=1}^{n} \|f_{\text{enc}}(x_i)\|_0 \tag{4}$$

$$\mathcal{L}_{\text{combined}} := \mathcal{L}_{\text{fidelity}} + \lambda \cdot \mathcal{L}_{\text{sparsity}}\,, \tag{5}$$

$p : \mathbb{R}^c \times \mathbb{R}^c \to \mathbb{R}_{\geq 0}$ is a function that increases with the distance of its two inputs, and $\|\cdot\|_0$ counts the number of non-zero entries in a given vector, and $\lambda \in \mathbb{R}^+$ is a parameter that balances reconstruction fidelity with sparsity. For large λ sparsity is prioritised and for small λ reconstruction fidelity is increased at the cost of more non-zero activations.

For SAE, the number of input channels is typically much smaller than the dimensionality of the latent space $c \ll d$. This is also known as the "overcomplete case" [L. Gao et al., 2024]. The distance p is typically chosen as the squared Euclidean metric because it offers a simple probabilistic interpretation of maximising the data log likelihood with Gaussian likelihoods of data points. However, this problem is NP-hard in the general case, so approximations are needed [Tillmann, 2015].

8.2.2 Offline Sparse Dictionary Learning

Early approaches to the above problem focused on applying matrix factorization algorithms to find an overcomplete dictionary $D \in \mathbb{R}^{c \times d}$, such that each data point can be represented as a sparse linear combination of the columns of D. Two examples are the method of optimal directions [Engan et al., 1999], which combines sparse coding with analytical dictionary updating, and K-SVD, which is a generalisation of the popular K-means algorithm formalised in the language of matrix factorisation [Aharon et al., 2006].

These methods are called offline sparse dictionary learning because they typically require the entire data set and large intermediate results to be stored in memory. This makes them unsuitable for interpretability in the era of deep learning, as these datasets are typically in the gigabyte to petabyte range and have dimensionalities in the hundreds to thousands, with dictionary sizes reaching millions in some cases.

8.2.3 L1 Activation Penalty

The following method was one of the first to address the online sparse dictionary learning problem for MechInterp. The main insight to translate the offline problem into an online one was to apply mini-batch gradient descent to a neural network based autoencoder with a differentiable sparsity penalty applied to the latent space [Cunningham et al., 2023]. The sparsity penalty takes the form of the L_1 norm of the activations.

$$\mathcal{L}_{\text{sparsity}} := \sum_{i=1}^{n} \|f_{\text{enc}}(x_i)\|_1 \,. \tag{6}$$

This trick is well known from the Lasso method, where it is applied to the weights of a linear regression model to shrink its coefficients to zero.

The architecture of the encoder and decoder can in principle be any neural network that maps $\mathbb{R}^c \to \mathbb{R}^d \to \mathbb{R}^c$, but simple linear layers with activations are usually chosen.

$$f_{\text{enc}}(x_i) := \max(W_{\text{enc}}(x_i + b_{\text{enc}}) + b_{\text{latent}}, 0) \tag{7}$$
$$f_{\text{dec}}(z_i) := W_{\text{dec}} z_i + b_{\text{dec}} \,. \tag{8}$$

The ReLU activation applied to the latent space forces all activations to be positive and gives the model the ability to bias weights to zero by choosing large negative values in b_{latent}.

In addition, the weights can be shared between the encoder and decoder, i.e.

$$W_{\text{enc}} = W_{\text{dec}}^{\mathsf{T}} \tag{9}$$
$$b_{\text{enc}} = -b_{\text{dec}} \,. \tag{10}$$

This mitigates the problem that the model can effectively minimise latent activations by multiplying the encoder weights by a small constant and dividing the decoder weights by the same constant.

8.2.4 TopK Activation Function

TopK SAE also uses simple neural networks to parameterise the encoder and decoder. However, it does not add a penalty term to the loss function to increase sparsity. Instead, it uses the TopK activation function to enforce sparsity in the latent activation by construction [L. Gao et al., 2024].

The TopK activation function sets all activations except the largest k activations to zero.

$$\sigma_{\text{TopK}}(\vec{x}) := \begin{cases} x_i & x_i \geq x_{(k)} \\ 0 & \text{else} \end{cases} \,,$$

where $x_{(k)}$ is the kth highest value in the vector $\vec{x}$. The activation function is further illustrated in Figure 14 This means that there can never be more than k active neurons. This has the disadvantage that the degree sparsity is essentially the same for all data samples, even though simple data points can be compressed into fewer active neurons, while more complicated data points may require more active neurons.

Another drawback of this method is that some latent neurons may not be activated for any realistic input, wasting computational resources. It has been found that simultaneously training the TopK-SAE with a small and a large value of k and initialising the model with shared weights reduces this phenomenon and makes the TopK-SAE outperform the L1-SAE in many scenarios.

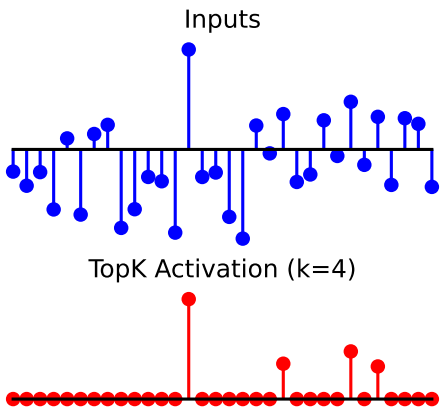


Figure 14: The TopK activation function. The input is shown as a stemplot above. The output of the TopK activation function with $k = 4$. All values that are not in the four highest values are set to zero in the output below.

8.2.5 JumpReLU Sparse Autoencoder

The JumpRelU SAE attempts to make the sparsity penalty $\|\cdot\|_0$ directly differentiable by using an estimate for the partial derivatives of the JumpReLU activation function based on kernel density estimation and straight through estimation Rajamanoharan,

Lieberum, et al. [2024]. The JumpReLU activation function is defined as

$$\sigma_{\text{JumpRelU}}(x, \theta) := \begin{cases} x & x - \theta \geq 0 \\ 0 & \text{else} \end{cases}.$$

A plot of this activation function is given in Figure 15. By assuming that the samples x_i come from a mixture defined by their kernel density estimates, the expected value sparsity becomes differentiable. This also avoids the so-called shrinkage effect of the L_1 penalty, where the active neurons have systematically lower values than optimal. For the sake of brevity, details of the derivatives are not given here.

JumpReLU SAEs can be seen as an update to gated SAEs, which also solve this problem, but have become redundant since the introduction of JumpReLU SAEs. [Rajamanoharan, Conmy, et al., 2024].

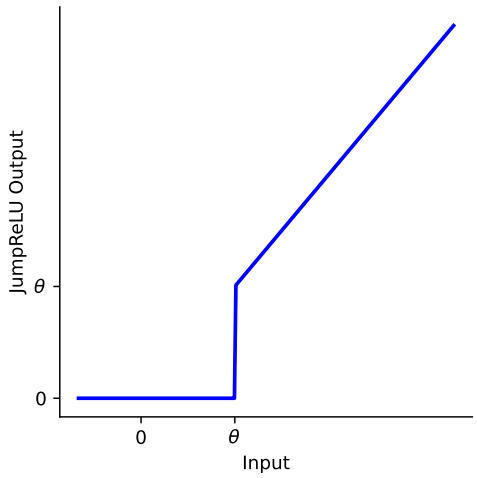


Figure 15: The JumpReLU activation function. It is similar to the well-known ReLU activation function, but stays at zero for longer and has a jump at θ.

8.3 Methodology

Activations of the ImageNet validation dataset without augmentations are sampled from the activations of the last convolutional layer of the modified ResNet18 architecture. Any filtering technique can be used here. The activations are further split 90%-10% into a training and validation set for the sparse autoencoder.

The SAE models are then trained on shuffled batches of size 128 using the Adam optimiser with a learning rate of $3 \cdot 10^{-4}$ for 10 epochs. For the L1 SAE, activations larger than a small constant 0.1 are clipped to zero after checking that this does not significantly increase the reconstruction loss. A simplified L1 SAE without bias terms is also trained.

The number of channels of the convolutional layer considered is 512, which is expanded by the SAE by a factor of $\times 8$ to 4096. The models loss and sparsity are evaluated on training and validation data batches throughout the training.

8.4 Experiments

8.4.1 Architecture Comparison

The different SAE models are trained with different sparsity settings using the described methodology. This allows a Pareto frontier of sparsity versus reconstruction fidelity to be found. This is shown in Figure 16. The TopK and JumpReLU SAEs systematically outperform the L1 SAEs by a small but not insignificant margin.

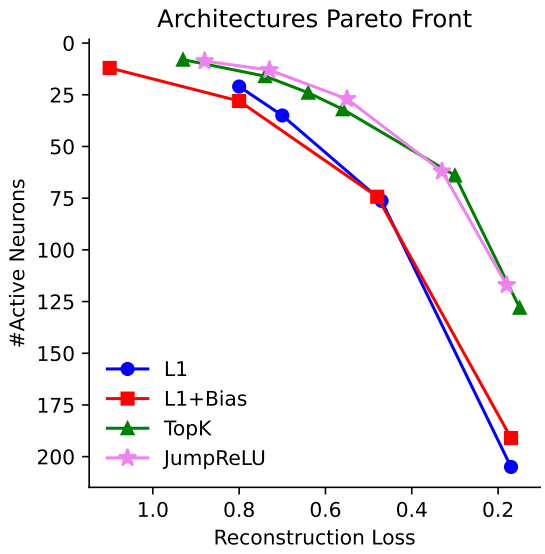


Figure 16: The Pareto frontiers of the different SAE architectures with different sparsity settings

8.4.2 Ablation Studies

Further ablation studies on latent space dimension, clipping value and training duration were also carried out for the L1 SAE. In general, a larger latent space improves the results, while the results for different clipping values remain stable in the range between 0.025 and 0.1. This is shown more closely in Figure 17

8.4.3 Convergence Behaviour

Overfitting is not an issue even when training for hundreds of epochs, while become almost insignificant after 25 epochs. The number of activated neurons decreases rapidly, then increases and stabilises.

8.5 Discussion

The high level of sparsity achieved strongly suggests that CNN activations inherently consist of a few activated features. Extracting these is likely to be very useful for further MechInterp research, as it minimises the number of moving parts that need to be considered in explaining a given model decision.

The ablation results of this study can be used in future work on the interpretation of CNN models. If a suitable implementation of one of the advanced sparse autoencoder methods is not available, implementing the L1 SAE with shared weights and biases is sufficient to achieve good sparsity results and save time for interpreting the SAE features, since the other methods require

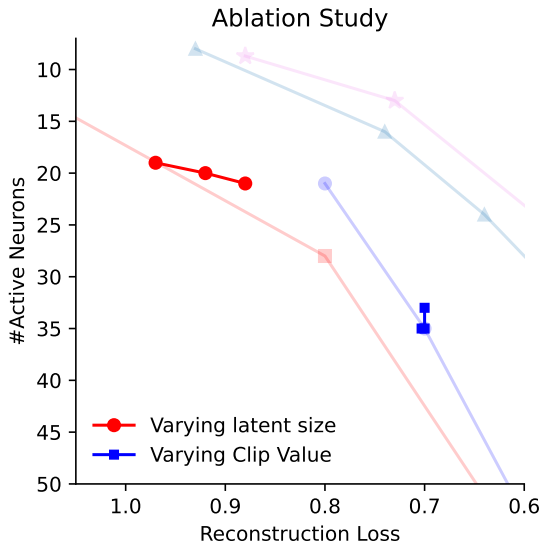


Figure 17: The influence of varying the latent dimension between 512 and 4096 and the clip value between 0.025 and 0.1.

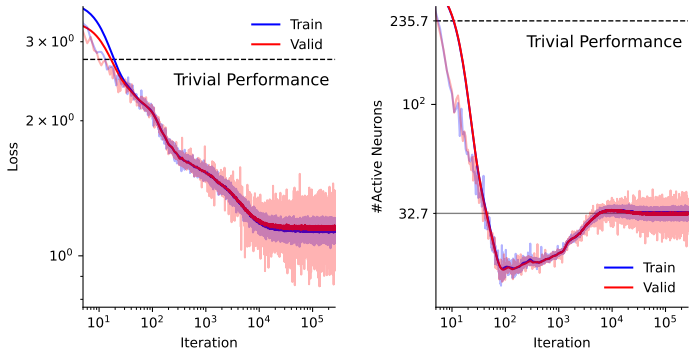


Figure 18: The convergence behaviour of the L1 SAE training. The loss and sparsity converge about 25 epochs.

more hyperparameters to be tuned and do not provide improvements that would justify the added complexity of dealing with unused codes in the TopK case or implementing custom gradient estimators for the JumpReLU case. Another practical drawback of the TopK SAE is that training is slower by a factor of about 4 for the same number of epochs. This could be mitigated by using a parallel algorithm to find the kth largest element on the GPU. However, PyTorch does not parallelize this optimally.

8.6 Limitations and Future Directions

It is not clear how the results of this research generalize to earlier layers in the CNN or to newer architectures such as ViTs. However, the early layers of CNNs have already received much attention in the literature, and SAEs have been widely applied to transformers in language models. Nevertheless, we would recommend that research teams with more computational resources than those available during this work apply the given methods to newer and larger models.

A technical difference between the transformers to which SAEs are typically applied and the ResNet considered in this study is that the element-wise activation function is applied directly to the residual stream. This did not result in any perceived qualitative differences, but should be made explicit as it makes some assumptions about the unprivileged nature of the standard base incorrect.

The reconstructions of the SAE could be further validated by confirming that there is no drop in accuracy after inserting the SAE into the trained classifier model. This would be very compatible with confirming the purposes of different neurons through causal interventions, i.e. deleting certain neurons and checking how the model outputs change for given inputs compared to what the theory of neuron purpose would predict.

8.7 Implementation Details

In order to simplify the use of the implemented SAE models, they have been implemented as an abstract base class that takes into account the latent dimensionality, the sparsity weight/setting and the axis along which the SAE is to be applied, which is the first one for the BCHW activation layout used by convolutional neural networks in PyTorch.

9 Feature Visualization

Feature visualization is a critical tool for understanding how convolutional neural networks process information and represent learned features. It involves optimization, where the objective is to maximize activations for a chosen target. The target can be customized to represent different levels of abstraction within the network, such as an individual neuron, a channel, or an entire layer [Olah et al., 2017].

Visualizing these activations reveals the hierarchical representations within a network, from simple edges in shallow layers to complex patterns in deeper layers. This hierarchical understanding is fundamental for improving model interpretability [Olah et al., 2018].

In this project, neuron-level feature visualization served as a necessary step to tackle challenges like polysemanticity — where a single neuron activates for multiple unrelated features [Scherlis et al., 2023]—and to evaluate neuron-specific behavior in both the customized ResNet-18 model (referred to as the CNN model) and its sparsity-enforcing counterpart, the ResNet-18 with Sparse Autoencoders (referred to as the CNN+SAE model).

My contribution centered on developing a practical methodology for generating clear and interpretable visualizations, enabling meaningful comparisons of these architectures with respect to sparsity and interpretability.

9.1 Related work

Feature visualization methods have advanced significantly to address challenges in understanding and interpreting neural networks. These methods are broadly categorized into two main approaches: code inversion and activation maximization.

Code inversion involves synthesizing an image that produces an activation vector at a specific layer, matching the activation

vector of a given real image. This approach, pioneered by Mahendran & Vedaldi [2014] and further developed by Dosovitskiy & Brox [2016], provides insight into how specific image features are encoded at different layers of a neural network. It allows researchers to reconstruct the visual information that contributes to specific internal representations.

In contrast, the more widely adopted approach, activation maximization, focuses on generating an image that maximally activates a specific target, such as a neuron in our case. Introduced by Erhan et al. [2009], this technique uncovers the features a neuron has learned to detect. While effective in principle, early implementations often produced noisy and high-frequency visualizations that were difficult to interpret [Nguyen et al., 2019]. Olah et al. [2017] proposed that these artifacts arise from intrinsic properties of neural networks, such as strided convolutions and pooling operations, which introduce high-frequency patterns in gradients. The authors state that without constraints, optimization-based methods tend to produce adversarial phenomena, creating images that activate neurons strongly but fail to resemble real-world features. To address these issues, a variety of regularization techniques have been developed over time, which where categorized into three families:

1. **Frequency Penalization**: Targets high-frequency noise to produce smoother and more coherent visualizations.
2. **Transformation Robustness**: Ensures visualizations remain consistent under slight input transformations such as scaling, rotation, or jittering.
3. **Learned Priors**: Uses data-driven constraints to generate realistic outputs but may obscure the origin of the observed features.

For our use case, we prioritized techniques from the first two families due to their effectiveness in addressing noise and robustness challenges in neuron visualization without relying on computationally expensive learned priors.

9.2 Methodology

9.2.1 Naive Approach

To establish a baseline for neuron visualization, we implemented a straightforward technique based on gradient ascent. The goal of this approach was to optimize a random input image to maximize the activation of a specific neuron. Starting with a random noise image, the input was passed through the model to compute activations for the target neuron. The loss function was defined as the activation value of the neuron, and an Adam optimizer [Kingma & Ba, 2017] was used to iteratively adjust the input image, applying gradient ascent to maximize the loss. As anticipated, initial experiments revealed that the resulting images were dominated by noisy, high-frequency artifacts, significantly limiting their interpretability.

9.2.2 Improved Approach

To address the limitations of the naive approach, we incorporated several regularization techniques aimed at reducing high-frequency noise and improving robustness. High frequencies were penalized by combining the Total Variation Loss with the activation of the neuron. This application of Total Variation Loss in the context of feature visualization was first explored by Mahendran & Vedaldi [2014], effectively reducing variations between neighboring pixels and enforcing smoother transitions in the generated images. Additionally, Gaussian blur was applied after each optimization step, further suppressing high-frequency artifacts as suggested by Yosinski et al. [2015].

Transformation robustness, as outlined by Olah et al. [2017], aims to ensure that visualizations represent features that activate the target neuron reliably, even under slight input modifications. In our implementation, we expanded on this concept by introducing random affine transformations (scaling, rotation, and translation) alongside horizontal flipping. These augmentations diversify the input during optimization, preventing overfitting to specific patterns while encouraging the visualization to generalize to a broader range of input configurations, aligning with the principles described in Mordvintsev et al. [2015].

Finally, we applied RGB channel decorrelation via Cholesky matrix decomposition to reduce dependencies between color channels, ensuring that each channel represented distinct features more effectively. Initially, this technique was planned for use within a Fourier transform approach to suppress high-frequency artifacts. However, its standalone application resulted in visualizations with vibrant colors and balanced contrasts, further enhancing interpretability. A sketch of the improved approach is provided in Figure 19.

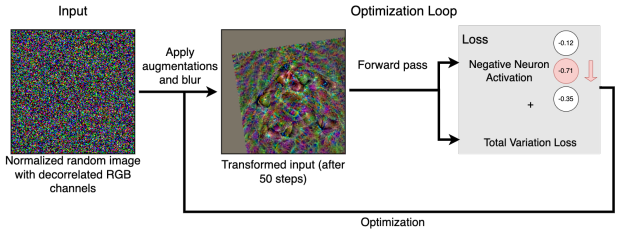


Figure 19: Workflow of the improved visualization approach.

The improvements achieved by the improved approach compared to the naive method are illustrated in Figure 20. The naive approach produces noisy, unstructured visualizations, while the improved approach generates smoother, more interpretable images that are much more likely to represent features.

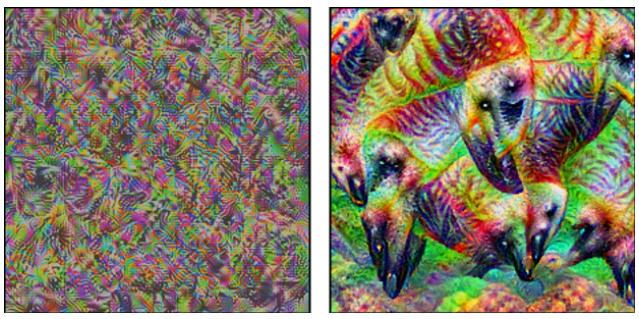


Figure 20: Visualizations for Layer 4, Neuron 12 of the custom CNN. Naive approach (*left*) shows noisy artifacts, while the improved approach (*right*) more interpretable features.

9.2.3 Hyperparameters

The process of generating optimized neuron visualizations required careful selection of hyperparameters to balance computational efficiency and visual clarity. Evaluating the quality of visualizations is inherently subjective, as no universally accepted criteria exist to define what constitutes a "better" visualization. Our primary objective was to achieve representational fidelity—ensuring that the generated image accurately reflects the underlying feature of a neuron. However, this goal remains open to interpretation due to the absence of definitive metrics.

The stochastic nature of random initialization and transformations further compounded this subjectivity. Each run produced unique outputs shaped by the initial noise image and the random transformations applied, making direct comparisons across configurations challenging. Consequently, hyperparameter tuning relied on iterative experimentation and practical adjustments informed by visual inspection and experience.

During this process, we observed that increasing the number of optimization steps generally improved visualization quality. However, to ensure computational feasibility, we adopted a balanced configuration that provided satisfactory results within reasonable time constraints. The final hyperparameters and transformation settings for generating all subsequent images were:

- **Steps:** 200
- **Learning Rate (lr):** 1.0
- **Total Variation Loss Weight (tv_weight):** 1×10^{-3}
- **Blur Strength (blur_strength):** 0.3
- **Transformations:**
  - **Scaling:** Random affine transformations with scaling in the range $[0.8, 1.2]$.
  - **Rotation:** Random affine transformations with angles in the range $-15°$ to $+15°$.
  - **Translation (Jitter):** Horizontal and vertical translations with offsets up to 20% of the image dimensions.
  - **Horizontal Flip:** Random horizontal flips applied with a probability of 50%.

9.2.4 Experimental Setup

To compare the interpretability of the customized CNN and CNN+SAE architectures, we generated neuron visualizations, focusing on Layer 4, the final layer of the network. This layer was chosen as it is expected to capture the most high-level and intricate features.

The neurons in Layer 4 were ranked based on their activation magnitudes in response to a husky input image (see Figure 21). The activation magnitude for each neuron was calculated as the sum of absolute activation values across all spatial positions in the feature map.

From the highest-ranking neurons, we handpicked five for detailed evaluation and visualization. These neurons were selected for their distinct and interpretable features, aiding in a qualitative comparison of the architectures.

Figure 21: Husky input image used for model comparison.

To further enrich the interpretability analysis, we retrieved the top three activating images from the ImageNet validation set for each selected neuron. These real-world examples provided valuable context, helping to identify what the visualized features represented in reality, such as fur, ears, or snouts.

For the least activated neurons, we randomly selected five for comparison and analyzed their visualizations to observe if differences exist between the two models.

9.3 Results

The neuron visualizations generated for both the simple CNN and CNN+SAE models exhibit intricate and colorful patterns, as shown in Figures 22 and 23. These visualizations capture diverse features, ranging from textures to high-level shapes.

Figure 22: Highly activated neurons with top three activating images from ImageNet for the CNN model.

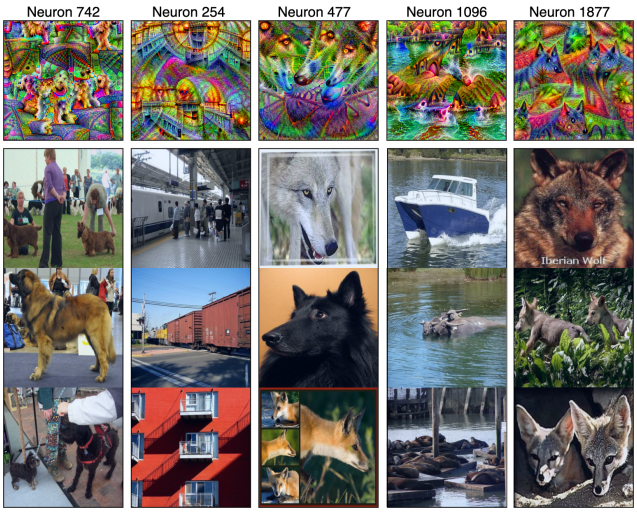


Figure 23: Highly activated neurons with top three activating images from ImageNet for the CNN+SAE model.

9.3.1 Top Activating Neurons

The top activating neurons from the simple CNN model captured dog-related attributes. Neuron 9 clearly depicted dog legs, while Neuron 271 visualized fur texture and furry ears. Neuron 163 also responded to fur-related features; however, the highly activating images from the ImageNet dataset for this neuron included not only a dog but also a hyena and a bear.

The CNN+SAE model was also able to capture dog-related features. Neuron 477 distinctly visualized snouts, Neuron 742 represented fur, and Neuron 1877 depicted pointy ears. Notably, while not evident from the plots, 8 out of the top 20 neurons represented dog-specific attributes.

9.3.2 Least Activating Neurons

In the simple CNN model, even the least active neurons produced bright, colorful, and contrast-rich visualizations, with every neuron showing some level of activation (see top row of Figure 24).

In contrast, the CNN+SAE model displayed darker and lower-contrast visualizations for the least active neurons (see bottom row of Figure 24). These neurons consistently had activation magnitudes of zero, reflecting the enforced sparsity constraint.

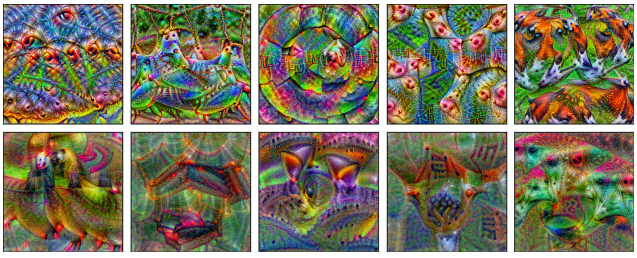


Figure 24: Visualizations of least activated layer for the CNN model *(top)* and CNN+SAE *(bottom)*.

We also validated our hypothesis that improved gradient flow in the modified CNN leads to faster generation of meaningful visualizations as shown in Figure 25.

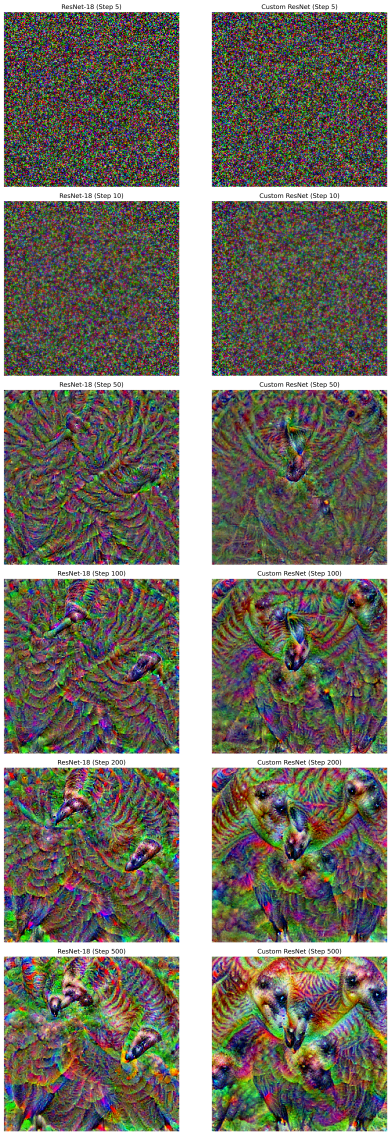


Figure 25: Visualization evolution at 5, 10, 50, 100, 200, and 500 steps. Left: Standard CNN. Right: Modified CNN with improved gradient flow.

9.4 Discussion

The visualizations revealed two prevalent phenomena across all neurons: multifaceted representations and polysemanticity.

Multifaceted representations refer to a neuron's ability to respond to multiple variations of a similar feature, effectively encoding different perspectives or contexts of that feature within its activation [Nguyen et al., 2016]. For example, Neuron 477 in the CNN+SAE model (Figure 23) activated strongly for various perspectives of a dog's snout, such as different angles or levels of detail. This phenomenon highlights the neuron's capacity to capture nuanced variations of a concept, but it also adds complexity to interpretation, as a single visualization may represent multiple facets of the same underlying feature.

Polysemanticity, on the other hand, refers to neurons activating for distinct and unrelated features. For example, Neuron 271 in the simple CNN model (Figure 22) responded to both fur textures and geometric edges, making interpretability more

challenging as unrelated concepts are intertwined within a single neuron.

Although sparsity did not eliminate multifaceted representations or polysemanticity as initially hoped, it significantly improved interpretability by limiting the number of active neurons.

In contrast to the simple CNN, where all neurons contributed to some degree, many neurons in the CNN+SAE model exhibited zero or near-zero activation (see results of section 8) for the husky input. This selectivity reduced redundancy, focusing attention on a smaller, more relevant subset of neurons and emphasizing distinct feature representations aligned with the input.

It is important to note that the input image also contained background elements rich in detail, such as a wooden fence, stair railings, and parquet flooring. Neurons responding to these features might rank among the top activations, complicating the interpretation of neuron specificity in representing the main object—the husky. For example, Neuron 47 and Neuron 382 in Figure 22 appear to activate for features likely linked to these background elements.

10 General discussion

Our methodology builds upon and extends the work of Gorton, advancing neural network interpretability through sparse autoencoders (SAEs) with key differences in focus, pipeline enhancements, and architectural innovations. While Gorton concentrated on the first layers of InceptionV1, uncovering low-level features such as curve detectors, our work applied SAEs to the last convolutional layer of a modified ResNet18, targeting high-level, task-specific features, as illustrated with the husky input image.

Although both approaches share a similar pipeline, we introduced systematic ablation studies on activation filtering methods, including Grad-CAM, maximum and median magnitudes, and the number of activated neurons. These studies refined training efficiency by focusing on the most relevant image regions for predictions. While Gorton utilized standard L1-SAEs, we explored additional modern variants, such as L1 with bias, top-k, and jump-reLU, to evaluate their impact on feature disentanglement.

Our comparison of the base CNN and CNN+SAE configurations further highlighted that SAEs reduce activation redundancy while preserving feature relevance in deeper layers, a critical aspect not explored in Gorton's work. Moreover, we enhanced the ResNet18 model by replacing max-pooling with average pooling and ReLU with LeakyReLU, which improved gradient flow and accelerated the feature visualization process, contributing to the overall efficiency of our approach.

Our findings validate the effectiveness of SAEs across both early and late network layers, bridging the gap between foundational feature disentanglement and task-specific interpretability. The systematic ablation studies identified optimal training strategies, while the additional SAE variants provided no significant improvement over the robustness and simplicity of the standard L1 model.

11 Overall results

The ResNet-18 architecture was modified to improve gradient flow with respect to the input image. After fine-tuning on ImageNet with data augmentation, the model achieved a top-1 accuracy of 65.45%. Neuron activations were filtered using multiple criteria: magnitude, maximum, median, the number of active neurons, and Grad-CAM-based localization. Results showed that magnitude, maximum, median, and the number of active neurons provided visually and quantitatively similar results. Grad-CAM, however, was ineffective at filtering out background patches and did not justify its additional computational cost. Filtering the datasets resulted in better SAE validation loss in fewer iterations.

Sparse autoencoders (SAEs) with different architectures and parameters were trained on layer 4 activations. These SAEs achieved a sevenfold reduction in activation size compared to the non-SAE baseline. Neurons in both the CNN and CNN+SAE models were visualized using ImageNet examples and an optimization procedure. The enhanced visualization approach, which included techniques to penalize high frequencies and enforce transformation robustness, significantly improved the representational fidelity, ensuring the visualized features more closely matched the true concepts represented by the neuron. This method enabled the visualization of eight distinct dog-related features for the CNN+SAE model, compared to only three for the CNN, within the top activating neurons.

12 Limitations

This project focused primarily on analyzing activations from one particular layer, limiting the scope of insights into the representations across other layers. A systematic comparison between layers could provide a deeper understanding of their contributions to the model's decisions. Similar comparisons between different models using the same dataset are also a promising direction for further research. Also, while activation filtering was applied, its effect on SAE training was only evaluated on a training set and validation set of a different sample. Developing more robust metrics for this purpose remains an area for future work. The application of Grad-CAM did not perform as expected, likely due to the limitations introduced by the ReLU function; exploring alternative methods, such as non-ReLU Grad-CAM variants, could be an interesting avenue for future research. Finally, feature visualization was limited to basic techniques, and incorporating advanced methods, such as Fourier Transform-based visualizations, could provide novel insights into the learned features and their spatial or frequency domain characteristics. These limitations highlight several promising directions for further exploration and improvement.

13 Group contributions

The report includes common parts and sections contributed by different individuals. The common parts, including the abstract, introduction, motivation, and background, were contributed by Paul. Blanka added references and images to the first chapters and a discussion of limitations. Finally, Thalis provided the

general discussion and overall results.

In terms of individual contributions, Section 6 and Section 7 were contributed by Blanka, Section 9 was contributed by Thalis, and Section 8 was contributed by Paul.

References

Aharon, M., Elad, M., & Bruckstein, A. (2006). K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, *54*(11), 4311–4322. doi: 10.1109/TSP.2006.881199

Anthropic. (2024). *Transformer Circuits – Interpretability in the Machine Learning World.* https://transformer-circuits.pub/. (Accessed: 2024-09-27)

Azam, S., Montaha, S., Fahim, K. U., Rafid, A. R. H., Mukta, M. S. H., & Jonkman, M. (2023). Using feature maps to unpack the cnn 'black box' theory with two medical datasets of different modality. *Intell. Syst. Appl.*, *18*, 200233. Retrieved from https://api.semanticscholar.org/CorpusID:258629865

Bank, D., Koenigstein, N., & Giryes, R. (2021). *Autoencoders.* Retrieved from https://arxiv.org/abs/2003.05991

Bieder, F., Sandkühler, R., & Cattin, P. C. (2021). *Comparison of methods generalizing max- and average-pooling.* Retrieved from https://arxiv.org/abs/2103.01746

Cammarata, N., Goh, G., Carter, S., Schubert, L., Petrov, M., & Olah, C. (2020). Curve detectors. *Distill*. (https://distill.pub/2020/circuits/curve-detectors) doi: 10.23915/distill.00024.003

Cammarata, N., Goh, G., Carter, S., Voss, C., Schubert, L., & Olah, C. (2021). Curve circuits. *Distill*. (https://distill.pub/2020/circuits/curve-circuits) doi: 10.23915/distill.00024.006

Chattopadhyay, A., Sarkar, A., Howlader, P., & Balasubramanian, V. N. (2017). Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 839-847. Retrieved from https://api.semanticscholar.org/CorpusID:13678776

Cunningham, H., Ewart, A., Riggs, L., Huben, R., & Sharkey, L. (2023). *Sparse autoencoders find highly interpretable features in language models.* Retrieved from https://arxiv.org/abs/2309.08600

Dai, X., Wan, A., Zhang, P., Wu, B., He, Z., Wei, Z., ... Gonzalez, J. E. (2021). Fbnetv3: Joint architecture-recipe search using predictor pretraining. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16271-16280. Retrieved from https://api.semanticscholar.org/CorpusID:237357571

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (p. 248-255). doi: 10.1109/CVPR.2009.5206848

Desai, S. S., & Ramaswamy, H. G. (2020). Ablation-cam: Visual explanations for deep convolutional network via gradient-free localization. *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 972-980. Retrieved from https://api.semanticscholar.org/CorpusID:214604773

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, *abs/2010.11929*. Retrieved from https://api.semanticscholar.org/CorpusID:225039882

Dosovitskiy, A., & Brox, T. (2016). *Inverting visual representations with convolutional networks.* Retrieved from https://arxiv.org/abs/1506.02753

Engan, K., Aase, S. O., & Husoy, J. H. (1999). Method of optimal directions for frame design. In *1999 ieee international conference on acoustics, speech, and signal processing. proceedings. icassp99 (cat. no.99ch36258)* (Vol. 5, pp. 2443–2446). doi: 10.1109/ICASSP.1999.760624

Erhan, D., Bengio, Y., Courville, A. C., & Vincent, P. (2009). Visualizing higher-layer features of a deep network.. Retrieved from https://api.semanticscholar.org/CorpusID:15127402

Ganguli, T., & Chong, E. K. (2024). Activation-based pruning of neural networks. *Algorithms*, *17*(1), 48.

Gao, L., la Tour, T. D., Tillman, H., Goh, G., Troll, R., Radford, A., ... Wu, J. (2024). *Scaling and evaluating sparse autoencoders.* Retrieved from https://arxiv.org/abs/2406.04093

Gao, S., Cheng, M.-M., Zhao, K., Zhang, X., Yang, M.-H., & Torr, P. H. S. (2019). Res2net: A new multi-scale backbone architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*, 652-662. Retrieved from https://api.semanticscholar.org/CorpusID:91184391

Gildenblat, J., & contributors. (2021). *Pytorch library for cam methods.* https://github.com/jacobgil/pytorch-grad-cam. GitHub.

Girshick, R. B. (2015). Fast r-cnn.. Retrieved from https://api.semanticscholar.org/CorpusID:206770307

Girshick, R. B., Donahue, J., Darrell, T., & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 580-587. Retrieved from https://api.semanticscholar.org/CorpusID:215827080

Gorton, L. (2024). The missing curve detectors of inceptionv1: Applying sparse autoencoders to inceptionv1 early vision. *ArXiv*, *abs/2406.03662*. Retrieved from https://api.semanticscholar.org/CorpusID:270286235

Guo, X., Li, Y., & Ling, H. (2017). Lime: Low-light image enhancement via illumination map estimation. *IEEE Transactions on Image Processing*, *26*, 982-993. Retrieved from https://api.semanticscholar.org/CorpusID:5778488

He, K., Gkioxari, G., Dollár, P., & Girshick, R. B. (2017). Mask r-cnn.. Retrieved from https://api.semanticscholar.org/CorpusID:54465873

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778. Retrieved from https://api.semanticscholar.org/CorpusID:206594692

He, L., Wang, J., & Wang, R. (2024). Integrating efficientnet, cosine annealing, and advanced data augmentation for enhanced aircraft detection in satellite imagery. In *2024 3rd international conference on engineering management and information science (emis 2024)* (pp. 219–227).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, *abs/1704.04861*. Retrieved from https://api.semanticscholar.org/CorpusID:12670695

Hu, J., Shen, L., Albanie, S., Sun, G., & Wu, E. (2017). Squeeze-and-excitation networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7132-7141. Retrieved from https://api.semanticscholar.org/CorpusID:140309863

Huang, G., Liu, Z., & Weinberger, K. Q. (2016). Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2261-2269. Retrieved from https://api.semanticscholar.org/CorpusID:9433631

Hugging Face. (2023). *Tiny imagenet dataset.* https://huggingface.co/datasets/zh-plus/tiny-imagenet. (Accessed: 2024-09-27)

Ismail, A. A., Bravo, H. C., & Feizi, S. (2021). Improving deep learning interpretability by saliency guided training. In *Neural information processing systems.* Retrieved from https://api.semanticscholar.org/CorpusID:244715181

Johny, A., & Madhusoodanan, K. (2021). Dynamic learning rate in deep cnn model for metastasis detection and classification of histopathology images. *Computational and Mathematical Methods in Medicine*, *2021*(1), 5557168.

Khalid, M., Baber, J., Kasi, M. K., Bakhtyar, M., Devi, V., & Sheikh, N. (2020). Empirical evaluation of activation functions in deep convolution neural network for facial expression recognition. *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, 204-207. Retrieved from https://api.semanticscholar.org/CorpusID:221118624

Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization.* Retrieved from https://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*, 84 - 90. Retrieved from https://api.semanticscholar.org/CorpusID:195908774

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. doi: 10.1109/5.726791

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, *86*, 2278-2324. Retrieved from https://api.semanticscholar.org/CorpusID:14542261

Lin, C.-J., & Jhang, J.-Y. (2021, 06). Bearing fault diagnosis using a grad-cam-based convolutional neuro-fuzzy network. *Mathematics*, *9*, 1502. doi: 10.3390/math9131502

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9992-10002. Retrieved from https://api.semanticscholar.org/CorpusID:232352874

Liu, Z., Mao, H., Wu, C., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A convnet for the 2020s. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11966-11976. Retrieved from https://api.semanticscholar.org/CorpusID:245837420

Loshchilov, I., & Hutter, F. (2017). *Sgdr: Stochastic gradient descent with warm restarts.* Retrieved from https://arxiv.org/abs/1608.03983

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Neural information processing systems.* Retrieved from https://api.semanticscholar.org/CorpusID:21889700

Mahendran, A., & Vedaldi, A. (2014). *Understanding deep image representations by inverting them.* Retrieved from https://arxiv.org/abs/1412.0035

Mordvintsev, A., Olah, C., & Tyka, M. (2015). *Inceptionism: Going deeper into neural networks.* https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. (Accessed: 2024-11-27)

Nguyen, A., Yosinski, J., & Clune, J. (2016). *Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks.* Retrieved from https://arxiv.org/abs/1602.03616

Nguyen, A., Yosinski, J., & Clune, J. (2019). *Understanding neural networks via feature visualization: A survey.* Retrieved from https://arxiv.org/abs/1904.08939

Nurjannah, A. F., Shafira, A., Kurniasari, D., Sari, Z., & Azhar, Y. (2022). Pneumonia image classification using cnn with max pooling and average pooling. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*. Retrieved from https://api.semanticscholar.org/CorpusID:248467256

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., & Carter, S. (2020a). An overview of early vision in inceptionv1. *Distill*. (https://distill.pub/2020/circuits/early-vision) doi: 10.23915/distill.00024.002

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., & Carter, S. (2020b). Zoom in: An introduction to circuits.. Retrieved from https://api.semanticscholar.org/CorpusID:215930358

Olah, C., Cammarata, N., Voss, C., Schubert, L., & Goh, G. (2020). Naturally occurring equivariance in neural networks. *Distill*. (https://distill.pub/2020/circuits/equivariance) doi: 10.23915/distill.00024.004

Olah, C., Mordvintsev, A., & Schubert, L. (2017). Feature visualization. *Distill*. (https://distill.pub/2017/feature-visualization) doi: 10.23915/distill.00007

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., & Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*. (https://distill.pub/2018/building-blocks) doi: 10.23915/distill.00010

Olshausen, B. A., & Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, *37*(23), 3311-3325. Retrieved from https://www.sciencedirect.com/science/article/pii/S0042698997001697 doi: https://doi.org/10.1016/S0042-6989(97)00169-7

Omeiza, D., Speakman, S., Cintas, C., & Weldemariam, K. (2019). Smooth grad-cam++: An enhanced inference level visualization technique for deep convolutional neural network models. *ArXiv*, *abs/1908.01224*. Retrieved from https://api.semanticscholar.org/CorpusID:199442547

Park, K., Choe, Y. J., & Veitch, V. (2024). *The linear representation hypothesis and the geometry of large language models.* Retrieved from https://arxiv.org/abs/2311.03658

PyTorch. (2024). *resnet18 package reference.* Retrieved from https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html#resnet18 (Accessed: 2024-11-19)

Rajamanoharan, S., Conmy, A., Smith, L., Lieberum, T., Varma, V., Kramár, J., ... Nanda, N. (2024). *Improving dictionary learning with gated sparse autoencoders.* Retrieved from https://arxiv.org/abs/2404.16014

Rajamanoharan, S., Lieberum, T., Sonnerat, N., Conmy, A., Varma, V., Kramár, J., & Nanda, N. (2024). *Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders.* Retrieved from https://arxiv.org/abs/2407.14435

Ren, S., He, K., Girshick, R. B., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*, 1137-1149. Retrieved from https://api.semanticscholar.org/CorpusID:10328909

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536. Retrieved from https://doi.org/10.1038/323533a0 doi: 10.1038/323533a0

Scherlis, A., Sachan, K., Jermyn, A. S., Benton, J., & Shlegeris, B. (2023). *Polysemanticity and capacity in neural networks.* Retrieved from https://arxiv.org/abs/2210.01892

Schubert, L., Voss, C., Cammarata, N., Goh, G., & Olah, C. (2021). High-low frequency detectors. *Distill*. (https://distill.pub/2020/circuits/frequency-edges) doi: 10.23915/distill.00024.005

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., & Batra, D. (2016). Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, *128*, 336 - 359. Retrieved from https://api.semanticscholar.org/CorpusID:15019293

Simonyan, K., & Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition.* Retrieved from https://arxiv.org/abs/1409.1556

Spearman rank correlation coefficient. (2008). In *The concise encyclopedia of statistics* (pp. 502–505). New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-0-387-32833-1_379 doi: 10.1007/978-0-387-32833-1_379

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., ... Rabinovich, A. (2014). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1-9. Retrieved from https://api.semanticscholar.org/CorpusID:206592484

Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, *abs/1905.11946*. Retrieved from https://api.semanticscholar.org/CorpusID:167217261

Targ, S., Almeida, D., & Lyman, K. (2016). Resnet in resnet: Generalizing residual architectures. *ArXiv*, *abs/1603.08029*. Retrieved from https://api.semanticscholar.org/CorpusID:12585246

Tillmann, A. M. (2015). On the computational intractability of exact and approximate dictionary learning. *IEEE Signal Processing Letters*, *22*(1), 45-49. doi: 10.1109/LSP.2014.2345761

Varshney, M., & Singh, P. (2021). Optimizing nonlinear activation function for convolutional neural networks. *Signal, Image and Video Processing*, *15*, 1323 - 1330. Retrieved from https://api.semanticscholar.org/CorpusID:233938052

Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Neural information processing systems.* Retrieved from https://api.semanticscholar.org/CorpusID:13756489

Weng, J., Ahuja, N., & Huang, T. S. (1992). Cresceptron: a self-organizing neural network which grows adaptively. *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, *1*, 576-581 vol.1. Retrieved from https://api.semanticscholar.org/CorpusID:12826256

Wu, J., Zhang, Q., & Xu, G. (2017). Tiny imagenet challenge. *Technical report*.

Xie, S., Girshick, R. B., Dollár, P., Tu, Z., & He, K. (2016). Aggregated residual transformations for deep neural networks.

2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 5987-5995. Retrieved from https://api.semanticscholar.org/CorpusID:8485068

Xu, B., Wang, N., Chen, T., & Li, M. (2015). *Empirical evaluation of rectified activations in convolutional network*. Retrieved from https://arxiv.org/abs/1505.00853

Xu, K., Chen, J., Ning, Y., & Tang, W. (2023). Deep learning in image classification: An overview. *2023 4th International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, 81-93. Retrieved from https://api.semanticscholar.org/CorpusID:259658903

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). *Understanding neural networks through deep visualization*. Retrieved from https://arxiv.org/abs/1506.06579

Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. *ArXiv*, *abs/1605.07146*. Retrieved from https://api.semanticscholar.org/CorpusID:15276198

Zeiler, M. D., & Fergus, R. (2013). Visualizing and understanding convolutional networks. *ArXiv*, *abs/1311.2901*. Retrieved from https://api.semanticscholar.org/CorpusID:3960646

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2017). Shufflenet: An extremely efficient convolutional neural network for mobile devices. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6848-6856. Retrieved from https://api.semanticscholar.org/CorpusID:24982157

Zhao, L., & Zhang, Z. (2024). A improved pooling method for convolutional neural networks. *Scientific Reports*, *14*. Retrieved from https://api.semanticscholar.org/CorpusID:267040420

Zhou, B., Khosla, A., Lapedriza, À., Oliva, A., & Torralba, A. (2015). Learning deep features for discriminative localization. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2921-2929. Retrieved from https://api.semanticscholar.org/CorpusID:6789015