Project Report

Interpretability of MLP Layers in Vision Transformers

Mika Allert, Ngoc Ahn Trung Pham, Ramazan Özdemir, Moritz Schwerdt

February 21, 2024

Abstract

Image classification is a historic task in machine learning and only recently have Transformers been introduced to the playing field. While a variety of interpretability methods exist for CNNs, we investigate the methods available for Transformers and adapt CNN methods to the Transformer. We will focus only on the MLP layers in Transformer blocks and build upon previous work, allowing us to interpret the 3-layer MLP as key-value vector pairs. Based on the foundation that the key vectors represent highly interpretable class prototypes, we focus on investigating the weighting thereof and how that affects the classification result. Specifically, we analyze how much impact very few of these value vectors can have and how their weighting schema allows us to make global and local associations between a class and the key and value vectors. We will demonstrate that this approach to the MLP layer not only allows us to find patches in an image relevant to a class but also allows us to generate class-specific information when maximizing the corresponding key vectors for certain value vectors. Altogether we show just how much information is actually contained within the weights and activation of MLP layers and how that can be leveraged to interpret this part of a Transformer. The code for this project alongside accompanying notebooks is publicly available under https://github.com/SoulofAkuma/dlcv-vit-mlp-explainability.

1 Introduction

After Transformers [1], a model architecture built mostly on the attention paradigm [2], started to gain some traction in the computer vision domain [3], it was quickly adapted to various tasks besides basic image classification [4]. Actually, before the mere classification of single images, the Transformer architecture had already been adapted to object detection [5] and has subsequently been introduced to image synthesis [6], multi-object tracking [7] and has recently come as far as video synthesis [8]. However, while all these advancements have been made in adapting Transformers to all these tasks, less *attention* has been given to explain why specifically the Transformer is so capable in these domains and how the Transformer structure on a local and global scale operates to achieve these state-of-the-art results.

In the following, we will focus on further explaining the Transformers capabilities on one of the earlier tasks it was adapted to, namely image classification. Structurally, we will focus on the MLP layer within a vision Transformer. We base our work upon research conducted by Vilas et al. [9] who came to the conclusion that projecting value vectors from MLP layers, using the pre-trained MLP head of the Vision Transformer (ViT) onto the class embedding space, revealed that most classes have a value vector that strongly predicts said class and that these value vectors can predominantly be found in one specific block. Dividing up the 3-layer MLP in Transformer blocks into key-value vector pairs is a concept that originated the MLP interpretability research of Transformers in NLP [10].

In this report, we aim to demonstrate our investigation into these MLP layers with four different approaches, ranging from transferable interpretability methods to methods adapted from CNN image classifiers. We will explain our approaches and present results, that suggest, that these value neurons, and their corresponding weighting via their key neurons, not only manage to predict specific classes, but are often capable of locating subject-relevant image tokens and contain, in their place within the Transformer, class-specific visually interpretable information.

2 Previous Work

2.1 Research Foundation

Our work is based on the research foundation [9]. This section summarizes their work and briefly go through how one can effectively interpret the parameters in the feed-forward layer, as proposed in [9, 10]

In general, they explained the model's behavior, in this case a Vision Transformer, in terms of its internal parameters by projecting them onto the class embedding space. Then from there quantified how much the parameters in each block encodes the class prototype representations, which have been learned by the model during training. As a result, most of the time the parameters in the penultimate block yield a clearly higher alignment with the class prototype representation than in any other block. This inspire us to investigate the parameters in this specific block.

Even though in the research foundation both the attention layer and the MLP layer in each block are investigated, we will only discuss the interpretability of the MLP layer in this report. According to [10] the MLP layer in a Vision Transformer block can be interpreted as key-value memories. This essentially means the following: (1) The *key vectors* are the neurons of the first layer in the MLP and act as a pattern detector for the inputs; each key vector/neuron is responsible for a specific pattern. (2) The *value vectors* are the neurons in the second layer of the MLP and can be interpreted as the resulting hidden representations that emerge upon detecting the patterns that are encoded in their corresponding key vectors. By examining the way matrix multiplication works, one can see that each key neuron produces a so-called *memory coefficient* [9], which quantifies how much this specific pattern encoded in the key neuron is present in the inputs. As a result, the final output of the MLP layer is a sum of value vectors, where each is weighted by the corresponding memory coefficient. Thus, the greater the presence of a pattern from a key vector in the inputs, the stronger the influence of its corresponding value vector on the final output.

2.2 Network Dissection

Network Dissection as a well-known proposed Framework for the interpretability of CNNs, focuses on the disentanglement of hidden logical units in large NNs. Those disentangled representations, allow finding interpretal internal units that make up the internal representation structure. A key point of the proposed Research [11] was the observation and factorization of the disentangled hidden representations, as well as their alignment in the feature space. To analyze the hidden representations, individual unit activations were aligned with a set of semantic concepts. This process takes numerous images through the trained NN and records the activation for each hidden unit. Utilizing these unit activations, a segmentation map is generated by creating a binary mask according to some threshold. The results of [11] which were derived from applying the proposed framework to common CNN architectures, were promising. Especially remarkable was the discovery of varied degrees of interpretability throughout the different layers. Here deeper layers tended to align with specific, sometimes surprisingly detailed, semantic concepts, that could be mapped back to meaningful image parts.

For our Project we want to map the key ideas of Network Dissection to the Vision Transformer Architecture. Since Transformers are built up of multiple complex parts, we focus on the analysis of the MLP close to the Output Layer in the Transformer. Here we focus on the Key Neuron parts of the MLP representation through the Blocks. This Idea based upon [10], who prior to the project proposed the dissection of the MLP Layers into Key and Value

Vectors. Like [10] we will map back the Key Neurons into the Class Embedding Space to determine the importance of certain Key Neurons in the Decision Making for a Classification Tasks.

2.3 MILAN

Mutual-information-guided linguistic annotation of neurons (MILAN) [12] is a procedure that allows the automatic labeling of individual model neurons with natural language descriptions. It builds on recent approaches with the goal of explaining the behaviour of deep networks.

To give annotations to each neuron, MILAN requires 2 steps:
It first represents each neuron via a set called exemplar. An exemplar set consists for each neuron with all the images that have activation value above a certain threshold parameter. Since this set would become way too large, it's restricted to only contain the top n (n=15 was used) images that cause the greatest activation. Ultimately, these sets contain the top n images with their activation masks. These masks highlight the regions of the images in which the neuron fired the most.

MILAN then gives descriptions to the neurons based on these exemplar sets and the images and image regions it contains.
To train the annotation model for MILAN, they used the approach of first obtaining a set of image regions using different models' neurons. The image regions were then presented to multiple human annotators with the task of describing their commonality. These descriptions were subsequently used to train the annotation models. Because of these human-annotated exemplar sets MILAN is able to generalize and give descriptions to neurons in new models and new datasets.

2.4 Lucid

Lucid [13] is a Tensorflow library for neural network interpretability. Specifically, Lucid is able to generate images that maximize a specific objective within a given model. These objectives are tailored to CNNs, which will have to be adapted to Transformers [1], but more on that in the implementation section of the report. This objective might be the activation value of internal network layers, the class scores, or class probabilities. Although Lucid applies transformations to the gradient as well as the image that is being generated, the objective is the major defining factor of what the generated image actually contains. Considering this importance, for example choosing an objective that optimizes the class score, instead of the class probabilities is a decision that can improve the result from noise to meaningful class representations and vice versa. In this particular case, the authors of Lucid have found that optimizing the probabilities leads to worse results because maximizing one probability, can also be achieved by minimizing others [14], while the class score would only maximize for this one class, regardless of other classes. Lucids implementation offers many pre-implemented objectives, ranging from single neurons in a single channel, to a neuron at a specific location, entire channels, and local groups [13]. All of these have limited application to the aforementioned perspective of Transformer MLPs as key-value vector pairs [10].

Another important aspect, besides the objectives, is the transformations Lucid applies to the gradient and to the image. Although the user never has to interact with these transformations while using Lucid, understanding what these are and how they lead from noise to interpretable results is valuable when using the library. There is one major modification applied to the gradient, or rather the image when the gradient is being applied, which is a Fourier transform over the pixel space to reduce high-frequency noise in the resulting image. Additionally, Lucid applies several other transformations to an image before every optimization step. These

include, in this specific order, padding the image with gray pixels, jittering all pixels by the same amount, scaling the image slightly, rotating the image slightly, and finally jittering the image again. These transformations lead to significantly more interpretable results, as demonstrated by [14], and improve the sensibility to transformations of the final image.

3 Visualization Implementation

3.1 Heatmap Visualization

One of the most simple ways to interpret a neuron unit is to visualize its activation on a given image as a heatmap. In this section, we describe the process of applying this method to the key neurons in the Vision Transformer. Furthermore, we will use the pretrained Vision Transformer base architecture with patch size 16x16, which was reported in [4].

When considering images of a specific class, not every key neuron is interpretable. Some key neurons could barely produce any activation value for a given image. This is due to the fact that the presence of the encoded pattern from these key neurons is simply very low in the input image. Thus, for every class, we evaluate every key neuron based on the alignment of their corresponding value vector with the class prototype representation. We then select the top key neurons for each class, that is, the key neurons whose value vectors has the highest alignment value with the class prototype representation, and use them for visualization.

Formally, according to [9], let C be the set of classes, d be the embedding dimension, and M be the set of key-value pairs, then the projection in the class embedding space of the value vectors is defined as

$$\mathbf{W}_{out} \cdot \mathbf{E}$$

where $\mathbf{W}_{out} \in \mathbb{R}^{|M| \times d}$ is the the value vectors and $E \in \mathbb{R}^{d \times |C|}$ is the class embedding matrix. The resulting matrix can be interpreted as an *alignment matrix* whose every entry is an alignment value between a value vector and a class prototype representation. The process of finding the top key neurons for a class is very straightforward: For every column in the alignment matrix we store the indices of the value vectors that yield the highest alignment value. These indices are also the indices of the key neurons. Note that the whole process is carried out for a single block, meaning that the 'top key neurons' refer specifically to the top ones within this particular block. However, we should be able to generalize this process to all 12 blocks of the Vision Transformer.

After obtaining a top key neuron for a specific class, we use the pretrained Vision Transformer as a feature extractor to retrieve the activation of this key neuron on every image tokens/-patches. The activation is a 197-dimensional vector, since there are 196 image tokens and an additional class token in total. We omit the class token and reshape the 196-dimensional vector into 14×14 tensor. Then bilinear interpolation is performed to upscale the activation map to match the original image size. Finally, we simply convert the enlarged activation map into a heatmap.

3.2 Network Dissection

3.2.1 Key Neuron Mask Activation

As mentioned by [10], the MLP layers in the ViT can be thought of as a key-value pared representation.

$$\mathrm{MLP}(\mathbf{X}) = \mathrm{GELU}(\mathbf{X}\mathbf{W}_{\mathrm{inp}})\mathbf{W}_{\mathrm{out}}$$

To transfer the Idea of Network Dissection to the ViT Architecture, we want to observe the Key Neuron Values, for a given Image. This can be done by examining the Top Value Neurons $\mathbf{W}_{\text{out-top}}$ of the alignment matrix for the specific class and then extracting the matching Top Key Neurons $\mathbf{W}_{\text{inp-top}}$. Those matching Top Key Neurons $\mathbf{W}_{\text{inp-top}}$ are utilzied for the Top Key Neuron Activation $\mathbf{X}\mathbf{W}_{\text{inp}}$, for a specific Image $\mathbf{X}$. The Key Neuron Activation gives insight into what Image Parts are activated, for which class, and in which MLP Block. Utilizing the ViT Architecture, we can use this Image Key Neuron Activation to overlay with the input image and obtain a binary mask. This binary mask gives the Top Key Neuron Activation Parts of a specific Image above a treshold, for specific Class Top Key Neurons, for a specific MLP Block.

3.2.2 Additional Implementation

To grasp a more quantitative understanding of the Key Neuron Activation along the Blocks, we experimented with multiple Questions that we answered with Visualizations.

1. How do the Top Key Neuron Activations of Blocks interact to produce the Output?
    - We plot the Running Mean of Top Key Neuron Activations over Blocks.
2. How are the Top Key Neuron Activations Distributed in the Blocks?
    - We plot the Block Index of the Most Activated Key Neuron in the alignment Matrix for each Class.
3. How do the Mean Top Key Neuron Activation for Images of a Class look in each Block?
    - We average the Top Key Neuron Activation over the Class Dimension, for Test Images.
4. How do the Mean Top Key Neuron Activation between Classes differ?
    - We compute the Image Pairwise L2 Distance for the Class Average Top Key Neuron Activations.

3.3 Annotating Neurons

The model used for the implementation is the Vision Transformer vit_base_patch16_224 model with pre-trained weights from the timm library [15]. This Vision Transformer was pre-trained on the ImageNet-21k dataset at resolution 224x224 with a patch size of 16 and fine-tuned on the ImageNet-1k dataset.

For the dataset the ImageNet-1k validation dataset is used to find the top images because this set only contains 50k images.

To give descriptions to the extracted top images, the base model for MILAN is used. This model was trained on all available data, which are all combinations of the models {alexnet, resnet152, biggan} and the datasets {imagenet, places365}.

As for the models layers since we are only interested in the MLPs FC2-layer only the top-activating images for these layers for each of the 12 blocks are computed. Since the computation of these exemplars needed a lot of computational power (MILAN required a lot of RAM) and storage (for each neuron, MILAN saved the top images and their masks inside the results folder) we were only able to inspect the first 100 neurons of each block in our implementation.

The forked repository with our added model and dataset can be found under Section 7.

3.4 Stimulative Image Generation

Because Lucid [13] is implemented in Tensorflow, we will be using Lucent [16] for the implementation, a PyTorch implementation of the original Lucid. The main challenge that comes with using Lucid on an image Transformer is, as already mentioned, that the objectives defined by the library do not apply to Vision Transformers. Hence, the goal was to implement an objective for Lucid, that optimizes an image with respect to the research question guiding our project: 'How much information do the value vectors contain, that most predict a class?'. The very functional nature of the Lucent implementation then allows hijacking the objective API to implement our own objective function that Lucent will use to optimize the image.

To develop the objective, it is useful to step through the underlying math in the Transformer MLP. As already mentioned, we have exactly 3 layers in this MLP, which is why we can describe the output of the final MLP layer as $\text{GeLU}(XW_{\text{inp}})W_{\text{out}}$ [10, 9]. To dive a little deeper, X is row-wise made up of individual patches $\{p_i\}_{i=1}^{N}$ with $p_i \in \mathbb{R}^d$, where d is the embedding dimension used by the Transformer and we have N patches. Thinking of X in terms of row-wise patches allows us to define the hidden vector $h_i = p_i W_{\text{inp}}$. If we consider each column of W_{inp} a key vector, then each scalar value $h_{i,j}$ in h_i is the dot product between the patch i and the value vector j where $1 \leq j \leq h$ and h is the hidden dimension of the MLP hidden layer and hence the number of key and value vectors. We can then collect this dot product over all patches for this key vector j as the key vector activation $a_j = [h_{1,j} \quad \dots \quad h_{N,j}]$. In simpler terms, a_j is the j-th column in the matrix XW_{inp}. Now taking a step back, a row of XW_{inp} is the vector that contains the dot product between one patch and all key vectors and a column of W_{out} is called a value vector. When looking at the final output for each patch $\text{GeLU}(XW_{\text{inp}})W_{\text{out}}$, we see that for each patch i, the result is a sum of the columns of W_{out} weighted with $h_{i,j}$ for the j-th column, in other words a weighted sum of value vectors.

With that in mind, we can finally define the objective we want to optimize for. Let's consider the value vector at some block b with column index j that has, when projected with the final MLP head of the Transformer, the highest probability for a class c. As we want to generate an image that maximizes the usage of this value vector j in the weighted sum result for each patch, the objective is the same as maximizing all entries in $a_j \in \mathbb{R}^h$. As we want to maximize these values and do gradient ascent, the optimization loss, that is backpropagated to the input pixels, can then be defined as

$$\mathcal{L} = -\frac{1}{h}\sum_{k=1}^{h} a_{j,k}^{(b)}.$$

In practice, this then involves the following steps to successfully generate an image for a class: (1) extract all value vectors in the model, (2) determine the most predictive value vector for the class, (3) pass the maximization of said value vector as the objective to Lucent. One very important aspect, that we will touch upon later in the Experiments section, is the difference in quality when applying the activation function to a_j in step (3), which turns out to have a negative effect on the interpretability of the generated images.

4 Experiments

4.1 Heatmap Visualization

In this section we will describe and discuss the results of this specific method.

We start off by extracting the top 5 key neurons for each class across all 12 blocks of the Vision Transformer and display their heatmaps. Figure 10 shows the best results, i.e., the

images for which the heatmap of the top 1 key neuron almost perfectly highlights the object of interest. This is because the image represents almost exactly the pattern that is encoded in the top key neuron. For the lower-ranked key neurons, they either produce high activation against every possible location in the images, or they activate to a very small extent. This is understandable, since lower-ranked value vectors do not align highly with the class prototype representation, and thus their behavior can be considered somewhat random.

However, there are some image instances (Figure 11) where the top 1 key neuron failed to capture the important features/patterns of the object of interest. This suggests that although the top value vector is highly aligned with the class prototype representation, its corresponding key vector encodes certain abstract concepts/patterns that are not human-understandable. In the figure, we see that it is actually the second-ranked key vector that highlights the object better than the first-ranked. We did not have the chance to perform the experiment on every image, but we hypothesize that the lowest possible rank of a key neuron that actually highlights the object of interest better than the first-ranked is very small; for example, it is very unlikely that a rank 10 key neuron would highlight the object better than the first. The result indicates that for these examples, the model performs predictions by aggregating information from different value vectors, and it is not always the top value vector that is dominant in the final output.

As a conclusion, Heatmap Visualization of Neuron Activation is a simple method for explaining the neuron's behavior when predicting the presence of a class. This method gives us a rough idea of what the neurons are looking for in an image. For the images that represent the encoded feature almost perfectly, the top key vector will produce a nearly perfect saliency map. However, most of the time, the activation maps are not interpretable. This is to be expected due to the following reasons: (1) Images come in different variations with respect to viewpoint, scale, orientation, lighting, etc. Therefore, no single key neuron suffices to encode all variations. (2) It is likely that the metrics we use for evaluating how good a key vector is based on the alignment of its value vector with the class prototype representation are not optimal and hence do not fully explain the model's behavior. This is because there are some low-ranked key vectors that actually encode more human-understandable patterns than the higher-ranked key vectors.

4.2 Network Dissection

While providing a suitable Implementation for the Top Activation Image Mask Generation based on the Key Neurons, we were not able to extend the original Network Dissection Framework with the Transformer implementation. Without access to the Vast amount of pixel-wise annotated pictures for the Imagenet Dataset, we can however still make some statements and observations for the Key Neurons in the MLP. In the following, we display the proposed Network Dissection Mask for an Example Image.

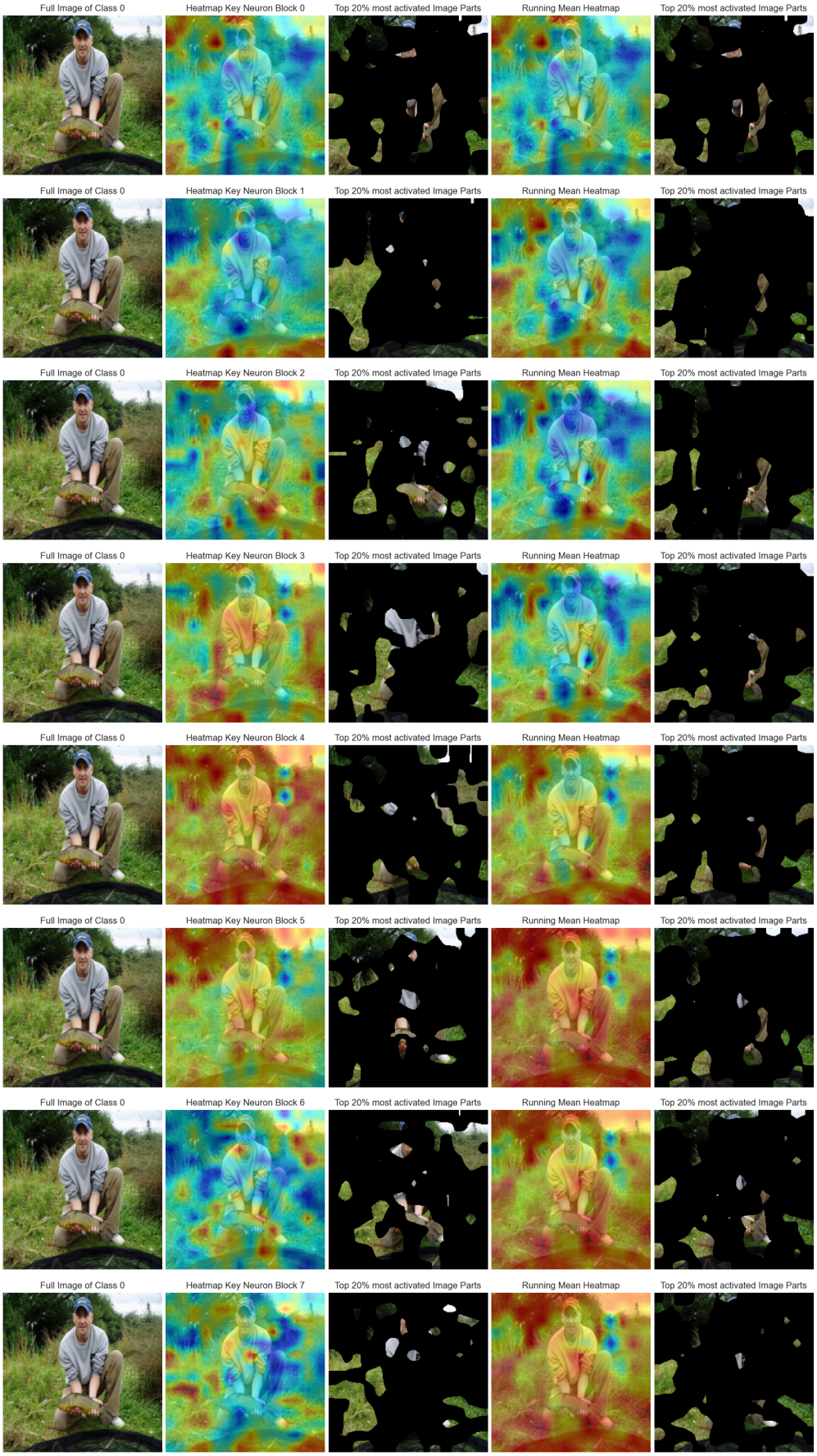

Figure 1: Top Key Neuron Activation of Class Tench Image throughout the Blocks 0-7

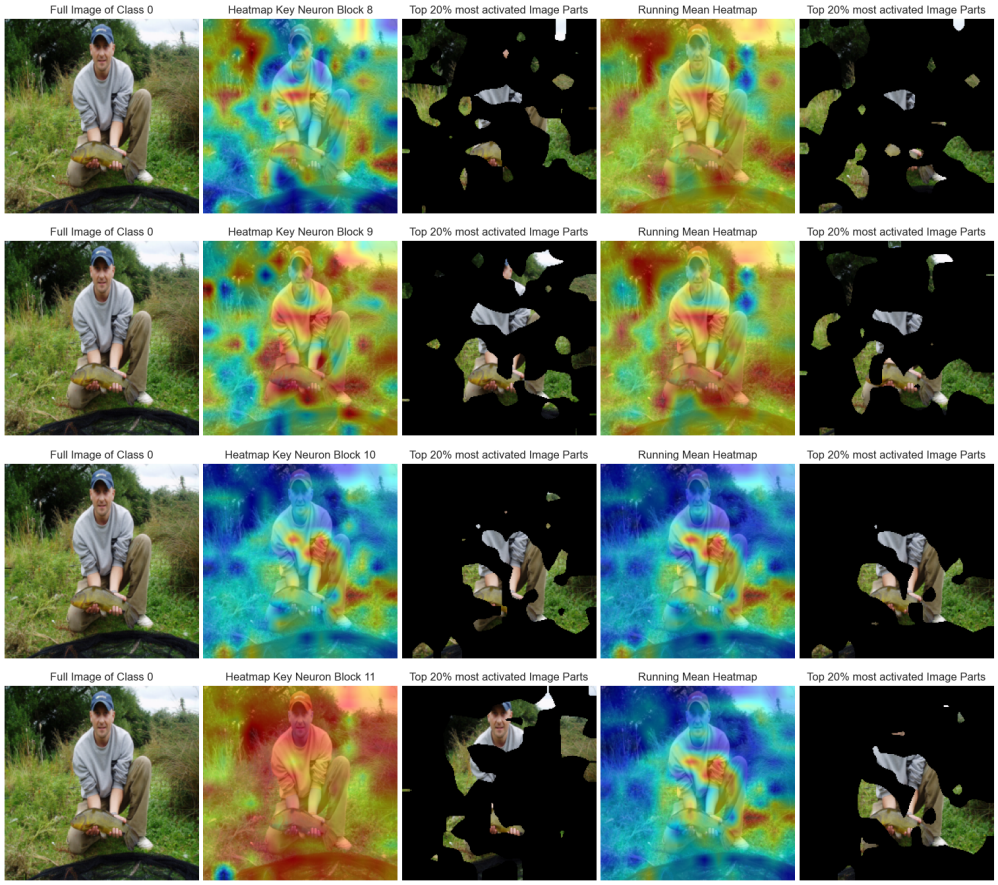

Figure 2: Top Key Neuron Activation of Class Tench Image throughout the Blocks 8-11
From Left to Right: Original Image, Heatmap of Top Key Neuron Activation, 80th Percentile of Activation Values, Heatmap of Average Seen Blocks Key Neuron Activation, 80th Percentile of Averaged Activation Values from Seen Blocks

We can observe how the ViT Key Neurons Activation of the Image changes over the Blocks. Here in Lower Blocks, the Top Key Neuron for the Class does not seem to grasp a real pattern. In later Blocks, a noticeable Focus Pattern of the Key Neurons seems to emerge.

In extension to Vilas et. al [9] Analysis of the Top Decision Forming Blocks e.g. Blocks with the highest Key Neuron Activation for the corresponding Class, we can visualize the overall Distribution for the ViT base 21k pre-trained model in Figure 3.

The most activated Key Neuron for each class tends to subside in Block 10 of the ViT. We observed that typically for Classes where the Top Activated Key Neuron was present before Block 10, the ViT did not capture the Class reliably.
To Gather some further knowledge of the actual Distribution of those Top Key Neuron Indexes per Block, we display the average Distribution of Top Key Neuron Activations for all Images of a Class in Figure 4.

We observe that for most classes, a small number of Top Key Neurons capture the Activation of the Class in the most activated Block, for that specific Class. For later Blocks, the Activations tended to be significantly higher. For relatively similar Classes like sub-breeds of Dogs, similar Key Neurons were activated in earlier Blocks.

In the following Figure 5, we want to observe how the Average Key Vector Activation of dif-

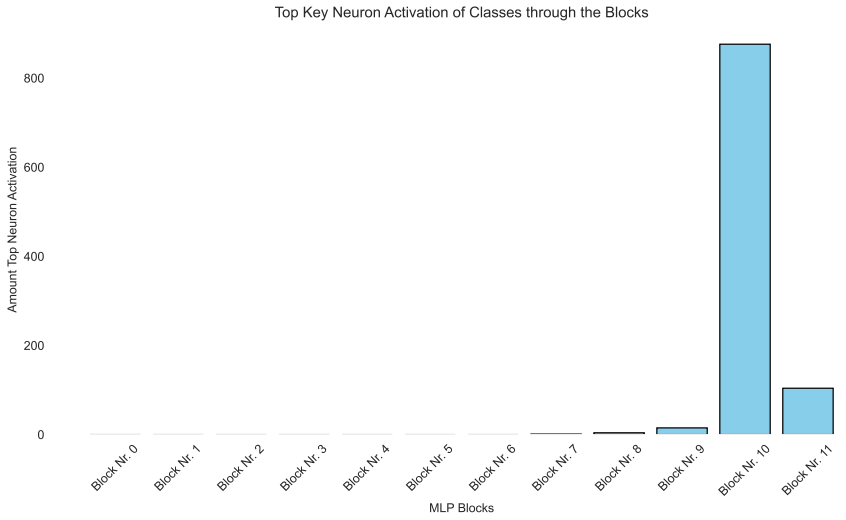

Figure 3: Distribution of Block Index for Top 1 Key Neuron Activation of ImageNeT classes

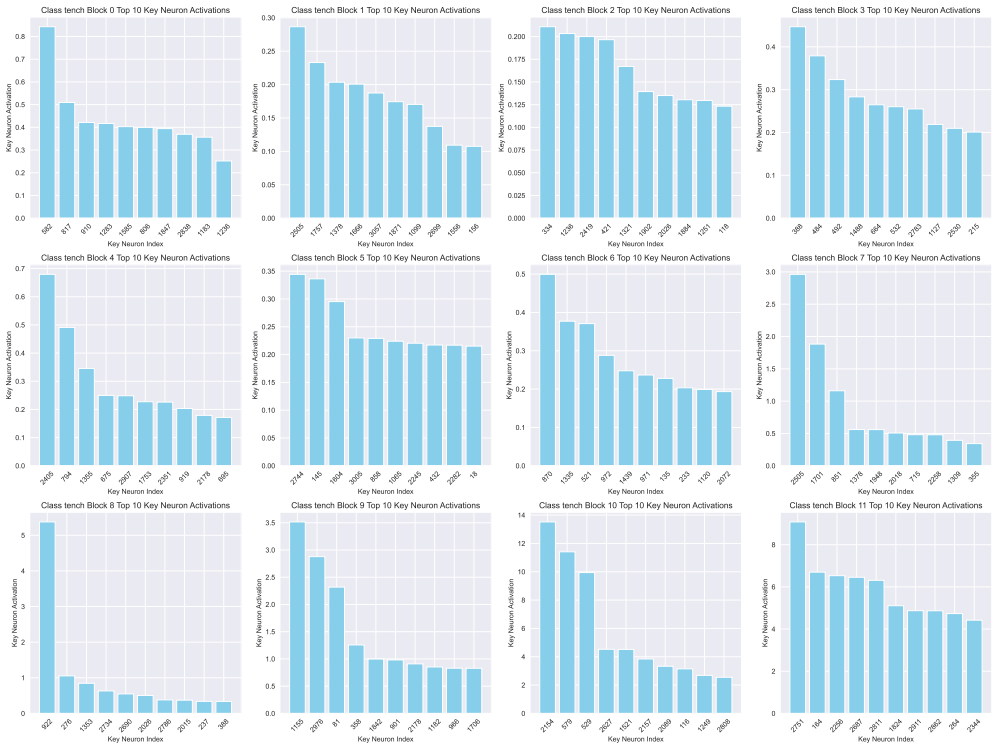

Figure 4: Distribution of Class Averaged Key Neuron Activation Index of Class tench throughout the Blocks

ferent Classes changes, throughout the Blocks. Here observe how the change in the Amount of Top Key Neurons to consider, changes the Class distances.

For all compared Classes, the Top Activated Key Neuron for that Class resided in Block 10. We observe that the Top Key Neuron Activation for relatively similar classes is relatively similar encoded. Dogs, for example, seem to appear closest together. In Block 10 Tench seems to be more similar to Fireboat than to a Dog.

We observe that for lower-level Blocks, the relative average pairwise distance of Classes does change 6. The use of more Top Key Neuron Activations seems to graph a bigger detail in

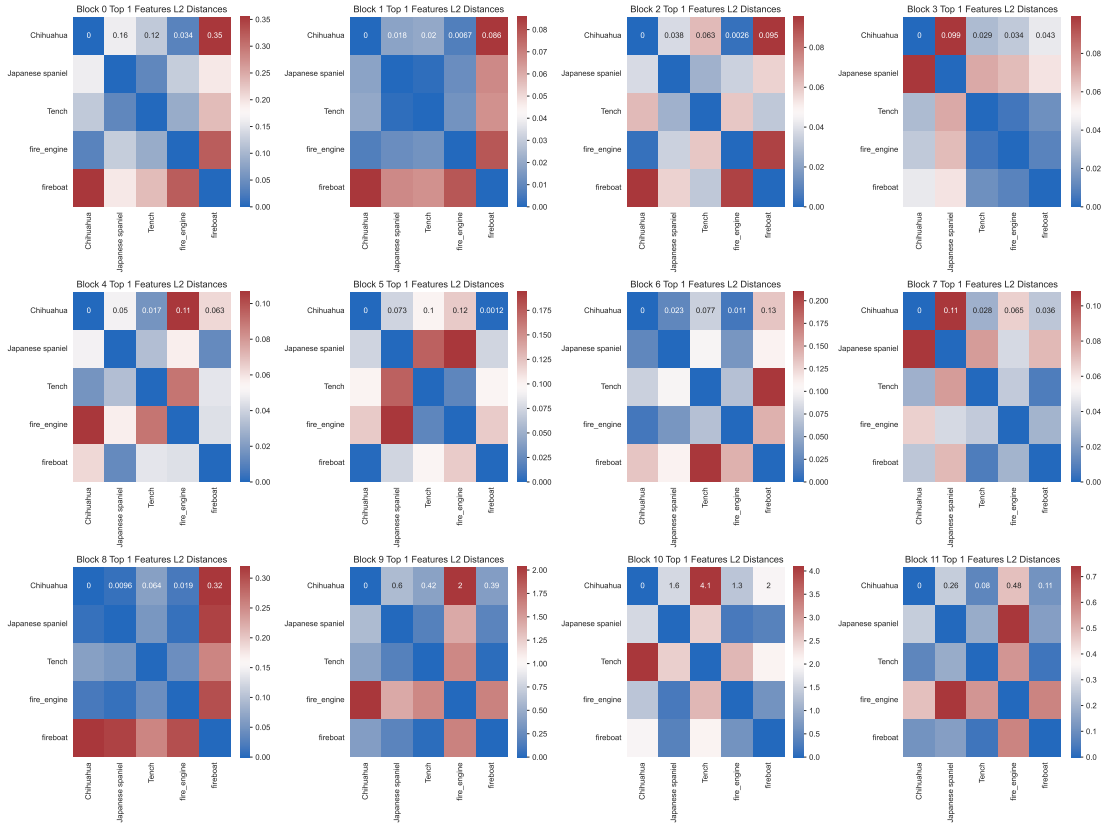

Figure 5: Average Class Block-wise L2 Distance utilizing Top1 Key Neuron

difference. Those bigger details, however, seem to not only grasps the main subject. Fire Boat for example in Block 10, seems to be relatively more similar when using 3 Key Neuron Activations. This might give a hint, that some Classes are so to say "more complex" and only grasped by more Top Key Neurons. Other Classes however, seem to be mostly grasped by a small amount of Top Key Neurons

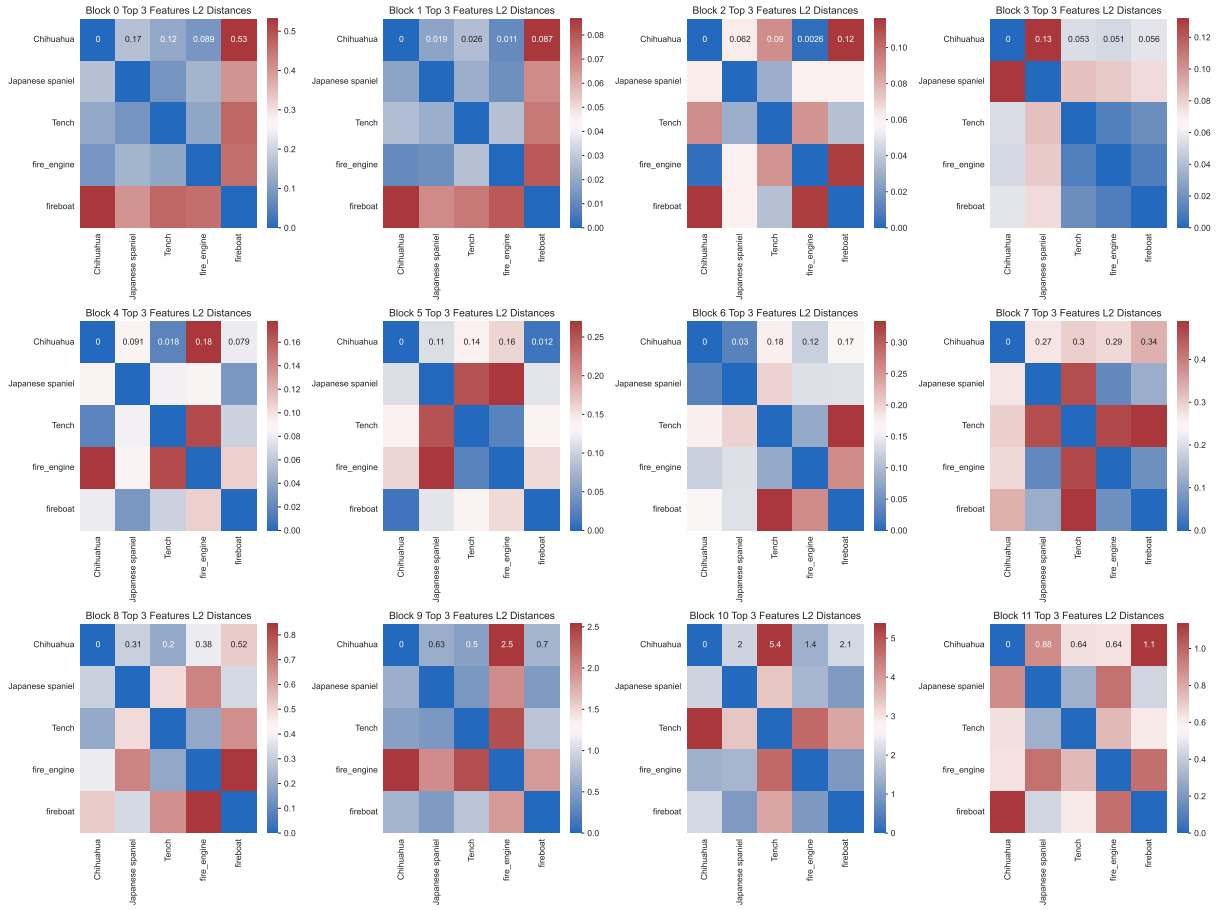

Figure 6: Average Class Block-wise L2 Distance utilizing Top3 Key Neurons

4.3 Annotating Neurons

After adding both our model and dataset to the files and running both the script to extract the exemplars and the script to annotate these exemplars we are able to look at the resulting masked images and the descriptions given to each of the individual neurons. The inspection of the results was mostly done manually. We tried to look over the different blocks by first finding decent descriptions and then looking at the images of that neuron.

From our observations, we noticed that earlier blocks contained a lot of general descriptions like *Lines, Stripes, Dots* and other not very interpretable descriptions. While the neurons in the first block still observed image regions that fit the (still general) descriptions some of the following blocks were observing image regions that are hard to interpret for a human.

Later blocks contained a decent number of neurons that not only observed interpretable image regions but also had descriptions that fit these regions. For example, the 56th neuron in block 10 seems to be focusing on penguins which MILAN accordingly gave the description "Penguin, animal".

A noticeable pattern across all blocks is that most of the time descriptions didn't match the image regions where the neuron had the highest activity. This was especially perceivable for non-general descriptions in the later blocks.

We tried a few more experiments by only using a limited number of classes as our dataset or trying to find images that are present across multiple blocks. Unfortunately, in our observations, these didn't produce any interpretable results.

In summary, while MILAN faces the problem of giving out a lot of general descriptions or descriptions that don't fit the observed image regions it allows a good, small, and easy first insight into what neurons might be looking at in images. Promising descriptions can be used to find out what an individual neuron focuses on.
Since we faced the problem of having limited resources we were only able to inspect a limited amount of neurons. Further inspections on more neurons per block would be required to give out a more detailed conclusion.

4.4 Stimulative Image Generation

The experiments conducted on the generated images can be subdivided into two sections. The qualitative analysis of images for individual classes, as well as the quantitative analysis over several images. All the following analyses will be conducted on the images optimized for a_j before applying the GeLU activation function to it, as optimizing for the activations after the activation function often leads to images entirely consisting of noise, as can be seen in Figure 7. We encourage to download a dataset after the activation function though (suffixed with `-act`) from here and see the results yourself. More information about this can be found in the Stimulative Image Generation Notebook in the main repository.

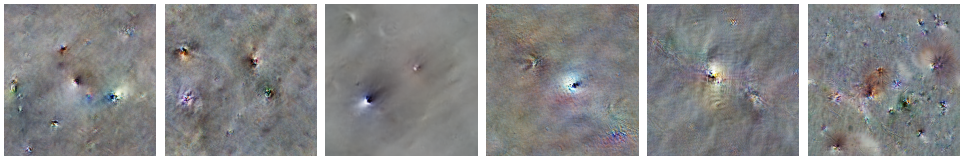

Figure 7: Failure cases optimized with Lucid after applying the activation function

We conducted two major quantitative experiments. One was to predict an image optimized for the objective of the most predictive value vector of that class for each class and see whether this image would be enough to re-classify the image. We evaluated the results for one image over all classes optimized for $250, 500$, and 750 iterations respectively and got a correct prediction rate of 68.4% to 68.6% and 68.9% respectively. That means from 250 to 750 iterations with the Adam [17] optimizer, only 5 more images were classified correctly, which shows how quickly the generated image represents a very interpretable state to the transformer. This accuracy is approximately only 15% lower than the evaluation dataset accuracy, which is 83.37%. The other quantitative experiment was finding the images, that had the highest activation of their key vector. These can be seen in 8. These images include animals that have some disproportional body shapes, like the sloth with its long arms and fingers and the camel with its hump. Others can mostly be defined by their unique textures, like the pineapple, artichoke, jigsaw, and the butcher shop.

Qualitatively we conducted 4 experiments for each class: (1) optimizing for the single most predictive value vector, (2) optimizing for the single most predictive value vector and a diversity term, (3) optimizing for the k most predictive value vectors, (4) optimizing for the k most predictive value vectors and a diversity term. The results for (1) and (3) can be seen in Figure 9.

In these generated images we can see, that the value vectors for some classes appear to contain a remarkable amount of information about that class. So much in fact, that one might be able to guess the class this image was optimized for, just by looking at the generated image. The pirate ship, the vest, and the cassette player stand out, particularly as inanimate objects that were almost fully generated just through this one value vector. Going from one to many value vectors does not clearly enhance these images, as the top value vector appears to capture most of the class and introducing more appears to introduce a certain level of noise.

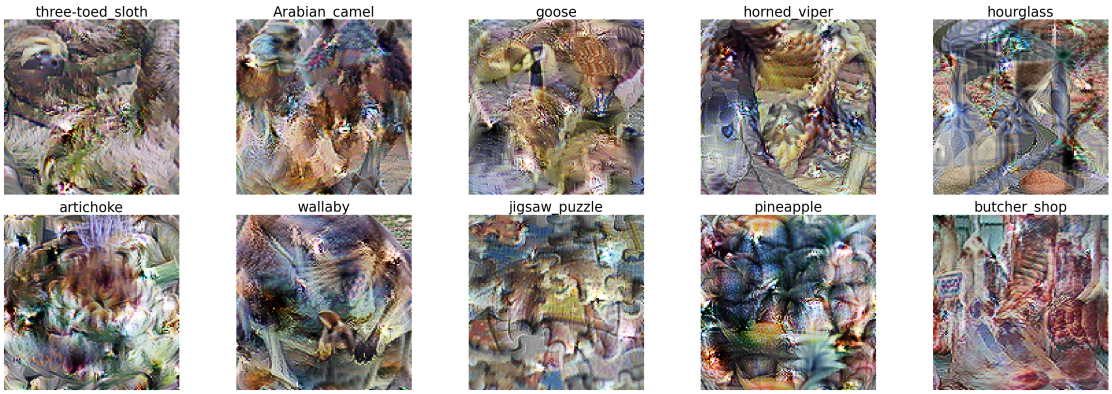

Figure 8: Images of resolution 128×128 optimized for 750 steps that achieved the highest mean key vector activation value

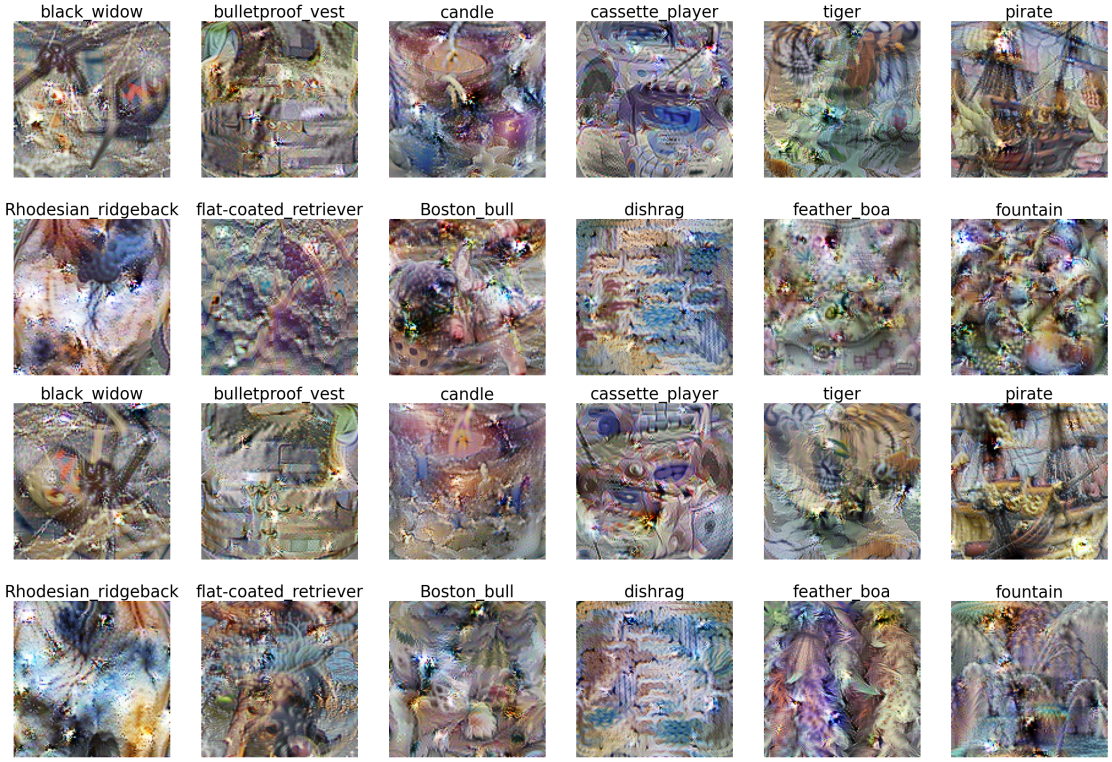

Figure 9: 128×128 Images optimized for a single class with the top value vector (first two rows), top $k = 5$ value vectors (last two rows). The first of each row pair contains classes that visualize well, the second classes that do not visualize particularly well.

Certain classes that didn't visualize at all with just one value vector, like the feather boa or the fountain, are improved significantly from one to multiple value vectors.

Last but not least, images generated with an additional diversity objective are shown in Figures 12 and 13. These show that the diversity objective is indeed capable of creating multiple different interpretable versions of the object, that meaningfully vary from each other. For a more detailed analysis of the individual contents of the images, please refer to the Stimulative Image Generation Notebook.

5 Future Work

For future work, we would like to investigate the small bright dots that are spread randomly throughout the heatmap (Figure 10, 11). We speculate that dissecting and interpreting the way the attention layer computes the output would shed light on this problem. Besides visualizing the neuron activation, there are many other ways to visualize what an intermediate neuron looks for in an image, which we would like to try out in the future as well. Examples of such methods would be gradient-based saliency map and guided backpropagation. Most of these methods use gradient information to help highlight the relevant pixel, and hence they are very likely to reveal interesting regions that Neuron Activation Visualization cannot detect.

Additionally, now that we have investigated where the subject is analyzed by the Vision Transformer, it would follow to investigate where the context and the surroundings are being analyzed. Which key neurons are responsible for the background, which for the subject and how some may be mixed, poses an interesting research question. A lot of our negative examples, where our methods didn't achieve interpretable results, had a most predictive value vector that was shared among multiple classes and it would be interesting to attempt identifying 'base classes', that these classes are made up of and these shared value vectors identify. In this context of mixed classes, it could also be considered to analyze the sum of pairs of value vectors and if they predict a class very well, to then interpret these pairs with the methods outlined here, but this might prove computationally very expensive, as for $\mathcal{O}(h)$ value vectors, there are $\mathcal{O}(h^2)$ pairs.

Finally, further research would include more quantitative analyses. For once about the generated images and heatmaps, as well as for these shared value vectors outlined here. For example, the here used ImageNet-1k Dataset is biased towards dogs, which represent over 10 percent of all images. We have seen instances where these dogs share a common value vector and ones where they do not and quantitatively analyzing the classes and value vectors, might lead to interesting results.

6 Conclusion

Concluding all our investigative methods, each of them has confirmed that there are value neurons for a class, that predict said class disproportionately well and that the corresponding key neurons for these value vectors (1) respond highly to the subject being classified, (2) respond relatively little to the background, (3) are highly interpretable. As these neurons almost exclusively resign in later blocks, the later blocks seem to be responsible for the correct detection of an object, and hence the correct classification as a whole. Value neurons in earlier blocks, on the other hand, appear not to be interpretable with respect to whole objects and generally have a noisy focus.

To advance the interpretability of MLP layers in Vision Transformers, we have successfully visualized the influence of key neuron activations for certain value vectors from different perspectives. These include visualizing which patches trigger high activations to visualize what these value vectors are focussing. Dissecting individual value vectors from the perspective of the class to be predicted to see whether these vectors reliably focus on the subject. Annotating neurons with the concept they are focussing on and trying to confirm the network dissection results from the perspective of the neuron. And ultimately generating images just based on the activation value of the key vector of the value vectors most predicting a class and hence visualizing what the other investigations have led us to assume: that key vectors and their activation, corresponding to these value vectors, do indeed contain class-specific and visually human interpretable information.

7 Contributions

For more detailed information see the GitHub repository contributions GitHub repository contributions, as well as the two repository forks: MILAN fork, Lucent fork.

- **Mika Allert**: Abstract, 1 Introduction, 2.4 Lucid, 3.4 Stimulative Image Generation, 4.4 Stimulative Image Generation, Future Work, code foundation of project, code for image generation and analysis
- **Ramazan Özdemir**: 2.3 MILAN, 3.3 Annotating Neurons, 4.3 Annotating Neurons
- **Ngoc Anh Trung Pham**: 2.1 Research Foundation, 3.1 Heatmap Visualization Implementation, 4.1 Heatmap Visualization Experiment, implementation of Heatmap visualizer
- **Moritz Schwerdt**: 2.2 Network Dissection Background, 3.2 Network Dissection, 4.2 Network Dissection & Additional

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," Advances in neural information processing systems, vol. 30, 2017.

[2] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," arXiv preprint arXiv:1410.5401, 2014.

[3] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," Advances in neural information processing systems, vol. 32, 2019.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., "An image is worth 16x16 words: Transformers for image recognition at scale," arXiv preprint arXiv:2010.11929, 2020.

[5] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in European conference on computer vision, pp. 213–229, Springer, 2020.

[6] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis. 2021 ieee," in CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 12868–12878, 2020.

[7] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer, "Trackformer: Multi-object tracking with transformers," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 8844–8854, 2022.

[8] L. Yu, Y. Cheng, K. Sohn, J. Lezama, H. Zhang, H. Chang, A. G. Hauptmann, M.-H. Yang, Y. Hao, I. Essa, et al., "Magvit: Masked generative video transformer," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10459–10469, 2023.

[9] M. G. Vilas, T. Schaumlöffel, and G. Roig, "Analyzing vision transformers for image classification in class embedding space," Advances in Neural Information Processing Systems, vol. 36, 2024.

[10] M. Geva, R. Schuster, J. Berant, and O. Levy, "Transformer feed-forward layers are key-value memories," arXiv preprint arXiv:2012.14913, 2020.

[11] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 6541–6549, 2017.

[12] E. Hernandez, S. Schwettmann, D. Bau, T. Bagashvili, A. Torralba, and J. Andreas, "Natural language descriptions of deep visual features," 2022.

[13] C. Olah, A. Mordvintsev, and L. Schubert, "Lucid." https://github.com/tensorflow/lucid, 2021.

[14] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," Distill, 2017. https://distill.pub/2017/feature-visualization.

[15] R. Wightman, "Pytorch image models." https://github.com/rwightman/pytorch-image-models, 2019.

[16] C. Lim Swee Kiat, Olah, A. Mordvintsev, and L. Schubert, "Lucent." https://github.com/greentfrapp/lucent, 2021.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

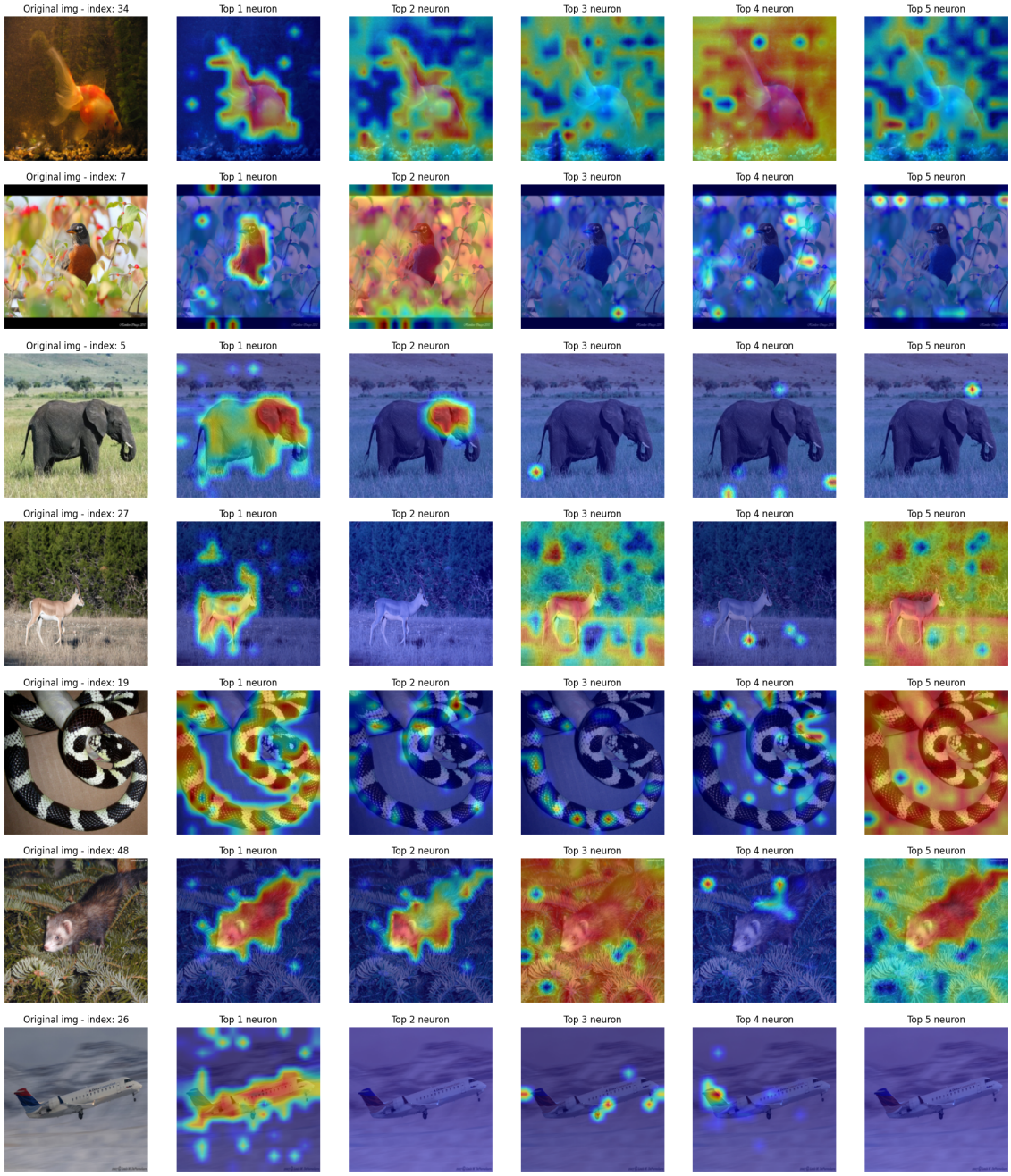

Figure 10: Images for which the top 1 key neuron almost perfectly highlight the object. For the lower-ranked key neuron, they either produce high activation against every possible location in the images, or they activate to a very small extent. This is understandable, since lower-ranked value vectors do not align highly with the class prototype representation. Observe that there are small bright dots that spread randomly through out the heatmap. We have the following two hypotheses: (1) Numerical errors which possibly arised during the normalization of the activation map when converting it into a heatmap. (2) The input to the MLP layer is the sum of the attention layer's output and a skip connection. Perhaps it is related to how the attention layer computes the final output. Unfortunately this is out of the scope of this report.

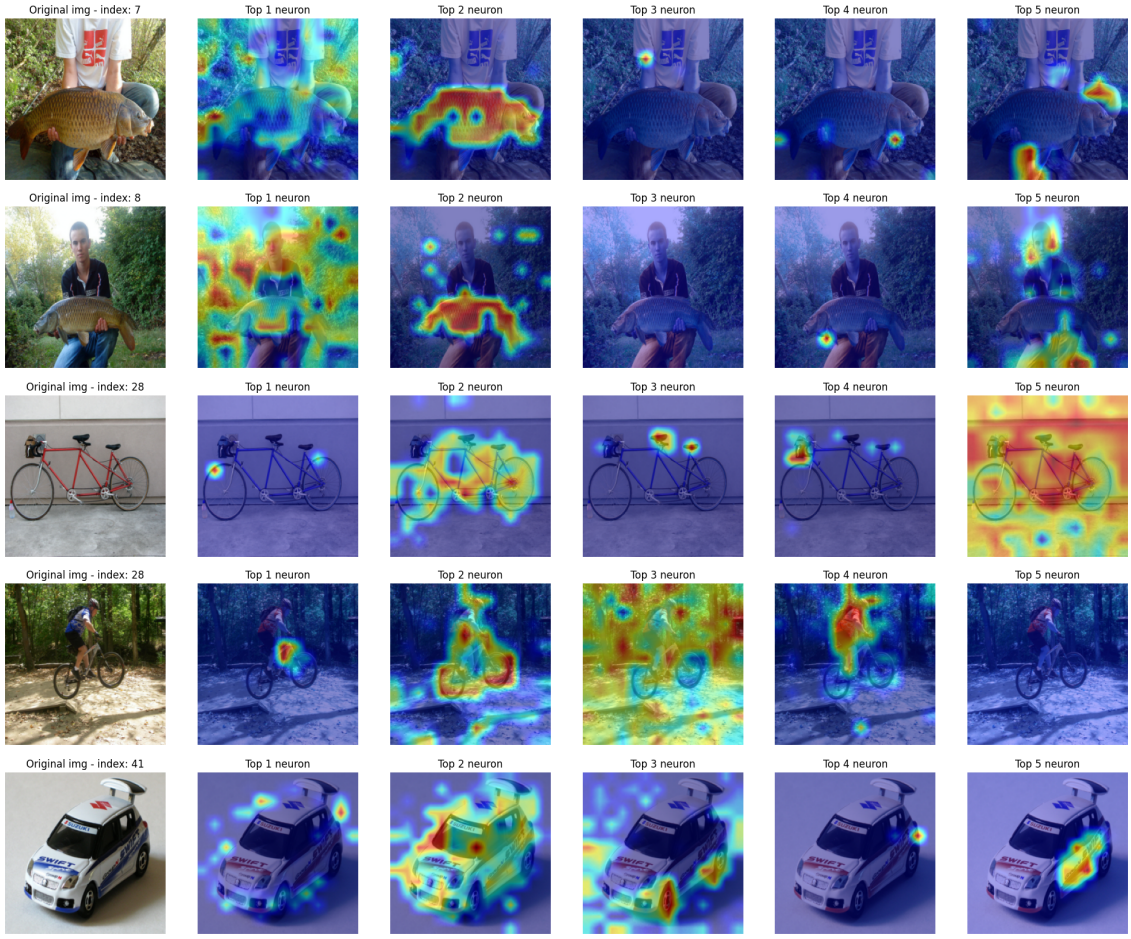

Figure 11: Image instances where the top 1 key neuron failed to capture the important features/pattern of the object of interest. The second ranked key neuron actually highlights the object of interest better than the first. This suggests that although the first ranked value vector has higher alignment value, the corresponding key vector did not encode the important features of the object. While the second ranked value vector has lower alignment value, its key vector captures more of the object's pattern. This bring us to believe that the final prediction is not always depends on the first ranked value vector but rather is a composition of different value vectors. Unfortunately, we did not have the time to look into the difference of the alignment value between the first ranked and the second ranked value vector. However, we hypothesize that their difference is not very large since the model's predictions are correct for all of these images.

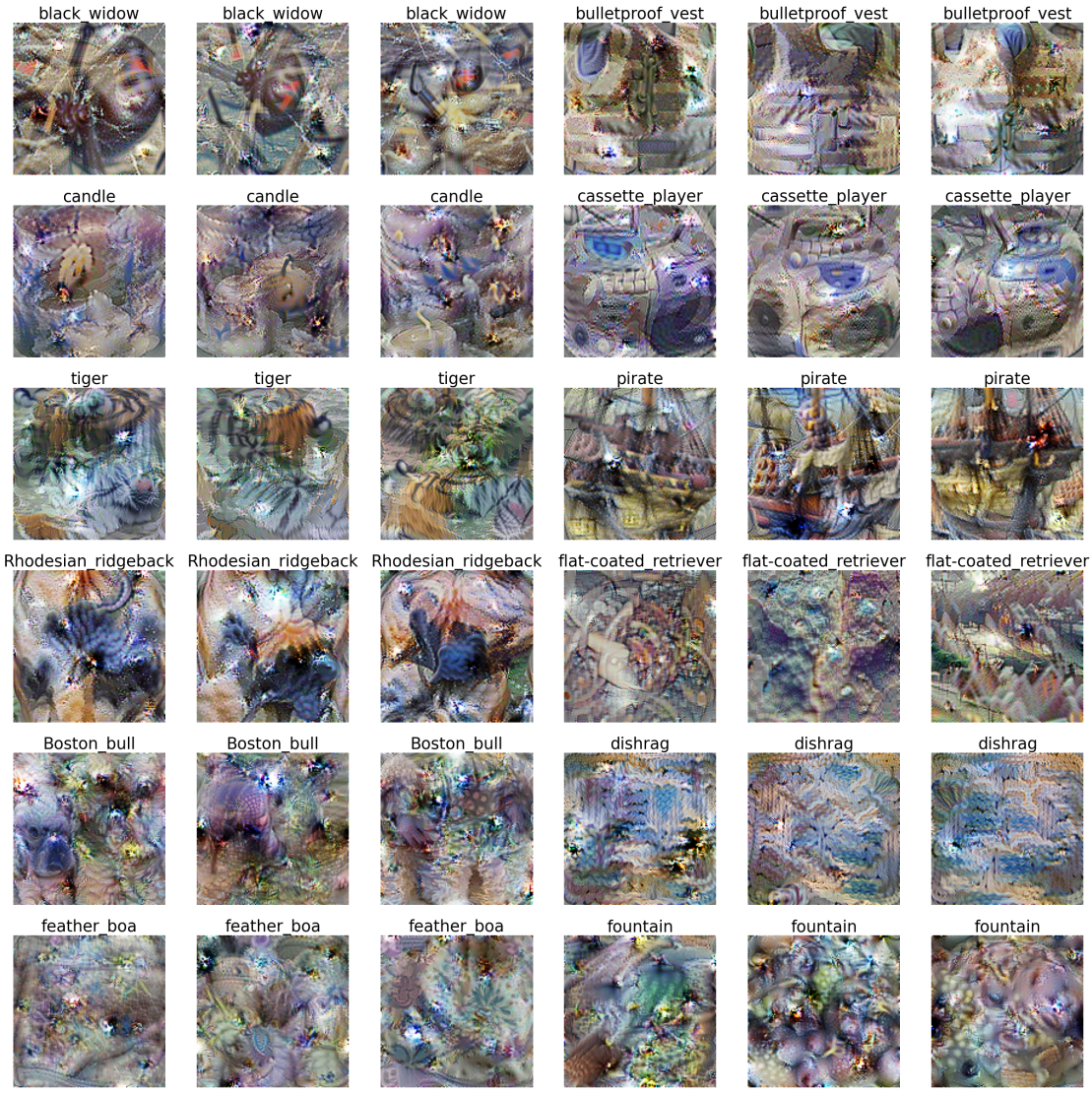

Figure 12: 3 Diversity images for each of the classes optimized for the top value vector objective and an additional diversity term. The first three rows of visually interpretable classes and the last three of visually less interpretable classes.

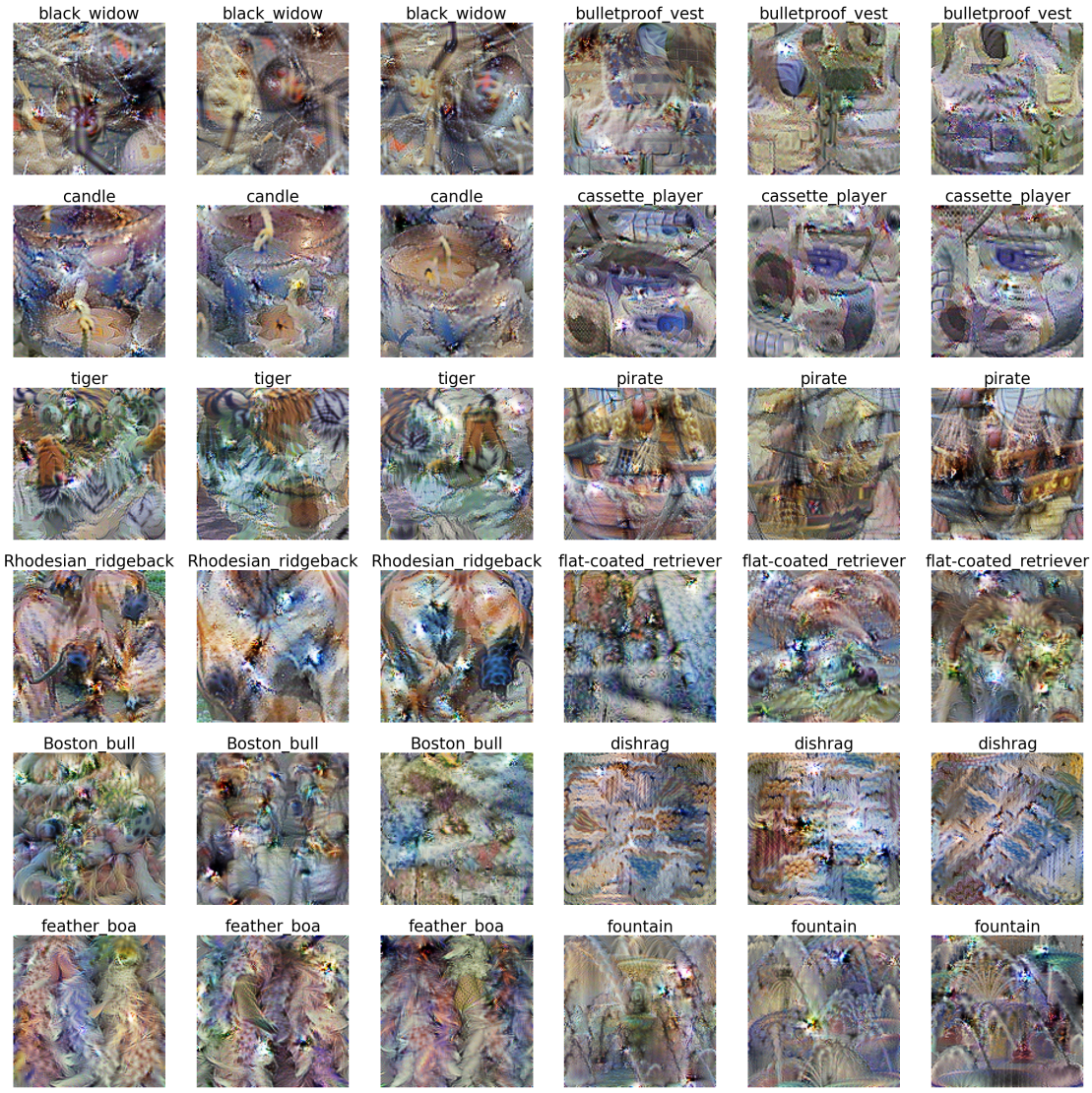

Figure 13: 3 Diversity images for each of the classes optimized for the top $k = 5$ value vector objectives and an additional diversity term for each value vector. The first three rows of visually interpretable classes and the last three of visually less interpretable classes.