Interpretability in Computer Vision: Unveiling the Inner Workings of Deep Learning Models

Akhila Nenavath, 2019CS50417*
Devanshi Khatsuriya, 2019CS10344*
cs5190417@cse.iitd.ac.in
cs1190344@cse.iitd.ac.in
Indian Institute of Technology Delhi
New Delhi, India

ABSTRACT

Convolutional neural networks learn features and concepts from images that seem abstract to humans. Here we attempt to visualize the same using two intuitive methods, one standard approach and three state of the art novel approaches for deep neural network visualization. We dig deeper and further into how these methods were motivated from previous work. We present the mathematical workings of these models and also release our code for public usage.

1 MOTIVATION

Explainability in deep learning models is an area of active research. With the growing applicability and usefulness of such models in all disciplines of life, it becomes necessary to account for the damage the usage of such models might cause. Such damages can be wide ranging - ranging from direct impact on under-represented/misrepresented groups to indirect impact by the propagation of biases that over time would detriment the standards for some groups and society as overall. Apart from damages targeted at particular groups of society, there can be seemingly trivial errors made by learning the wrong patterns and associations from data. Such mistakes are common because of high correlation also occurring between features that are not causally related.

This calls for transparency and understanding of the "thinking process" of such machine learning and deep learning models. We target to analyze and implement some ways to do the same in the domain of computer vision.

2 EXPERIMENTAL SETUP

We use a model with the architecture of ResNet50 v2 model [5], which is a CNN based state of the art computer vision model trained on the ImageNet Database [3]. We finetune it on the Cats v/s Dogs dataset [proposed in [4]. Available here and via HuggingFace API] to improve its accuracy on the dataset. The dataset has 25k images labelled as cat or dog. We add a 2 way classification head on top of the ResNet50 v2 penultimate layer so that it can be used for the classification task. For fine-tuning, we train the model on the cats v/s dogs classification task for 10 epochs on a P100 GPU.

We apply various interpretability methods on this fine-tuned model. All the code has been written using the Tensorflow Keras module [7] and is available at https://github.com/devanshikhatsuriya/Visualizing-CNNs.

*Both authors contributed equally to this research.

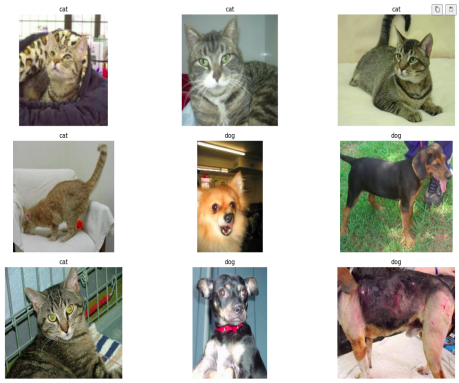

Figure 1: Some images from our dataset.

3 EXPERIMENTS

Convolutional neural networks learn features and concepts from images that seem abstract to humans. Here we attempt to visualize the same using two intuitive methods, one standard approach and three state of the art novel approaches for neural network visualization.

3.1 Visualizing Learned Filters

The first and most basic method to visualize the inner workings of a CNN would be to visualize the parameters that it learns - that is the filters. Visualizing learned filters refers to the process of generating visual representations of the filters or kernels learned by convolutional neural networks (CNNs). In CNNs, filters are small matrices applied to input data to extract specific features. Each filter learns to detect different patterns or characteristics within the input. By visualizing the learned filters, we can gain insights into what the network is looking for in the input data. It helps us understand the types of features that the network is sensitive to and provides an interpretability aspect to the network.

Visualizing learned filters can reveal interesting patterns and structures that the network has learned to detect. For instance, in early layers of CNNs, filters may resemble simple patterns like edges, corners, or textures. As we move deeper into the network, the filters can become more complex, capturing higher-level representations relevant to the task the CNN is trained for. Figures 2, 3, 4 and 5 describe the filters learned at layers *conv_1*, *conv_2_block_1*, *conv_2_block_1* and *conv_2_block_1* respectively. We can clearly see in Figure 2 the filters for detecting vertical lines, horizontal

lines, slanted lines, diagonal lines, crosses, dots, and circles. Similar patterns can be observed in filters at other layers too. We can infer from here that the model learns to extract such features from each patch of the image and combines the results over the layers to get indicators from specific identifiers to classify the image.

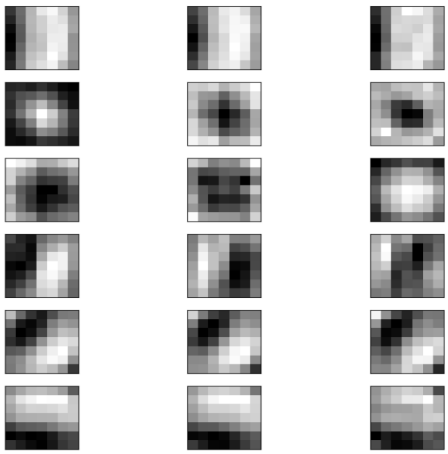

Figure 2: Learned filters for *conv1_conv* layer.

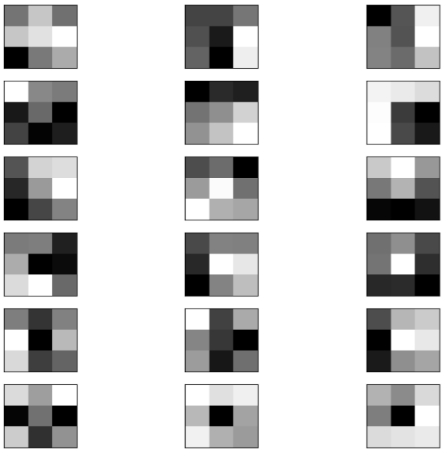

Figure 3: Learned filters for *conv2_block1_2_conv* layer.

3.2 Feature Maps: Visualizing Intermediate Representations

On similar lines as filter visualizations, we can also visualize the intermediate representations that are created for an input image, which are also called *feature maps*. Feature maps, also known as activation maps, are the intermediate representations of input data within a convolutional neural network (CNN). They are generated by applying a set of learned filters or kernels to the input data. Each filter detects specific patterns or features in the input and produces a corresponding feature map. Figure 6 shows this for the input image of a cat on the left for layers *conv_1* and *conv_3*.

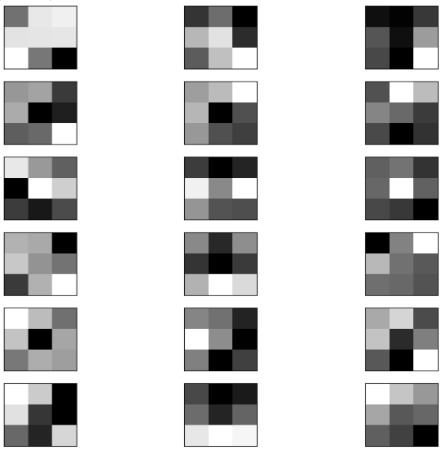

Figure 4: Learned filters for *conv2_block2_2_conv* layer.

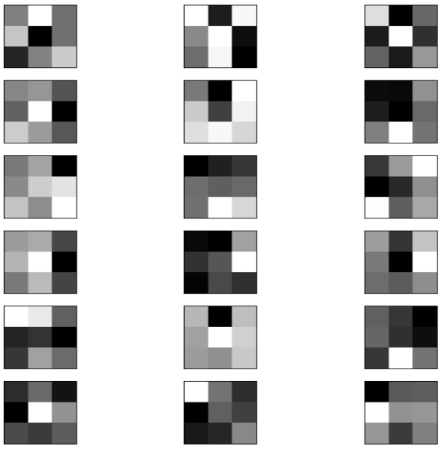

Figure 5: Learned filters for *conv3_block2_2_conv* layer.

Feature map visualization can be done by taking an input image and passing it through the network until the desired layer's activation. By extracting and visualizing the activations of different filters within that layer, we can observe the presence or absence of specific features in the input. These visualizations can provide valuable information about how the network is processing and transforming the input data. The intermediate representations indicate what the image is perceived as along different filters. Each feature map highlights a different feature of the cat. As compared to *conv_1*, the feature maps in the *conv_3* layer depict more high level features which cannot be perceived as identifiable cat features by humans, but it is what the model uses to make the classification.

3.3 Feature Visualization by Activation Maximization

Activation maximization is a technique where we find the image that maximizes the activation for some neurons. To achieve this,

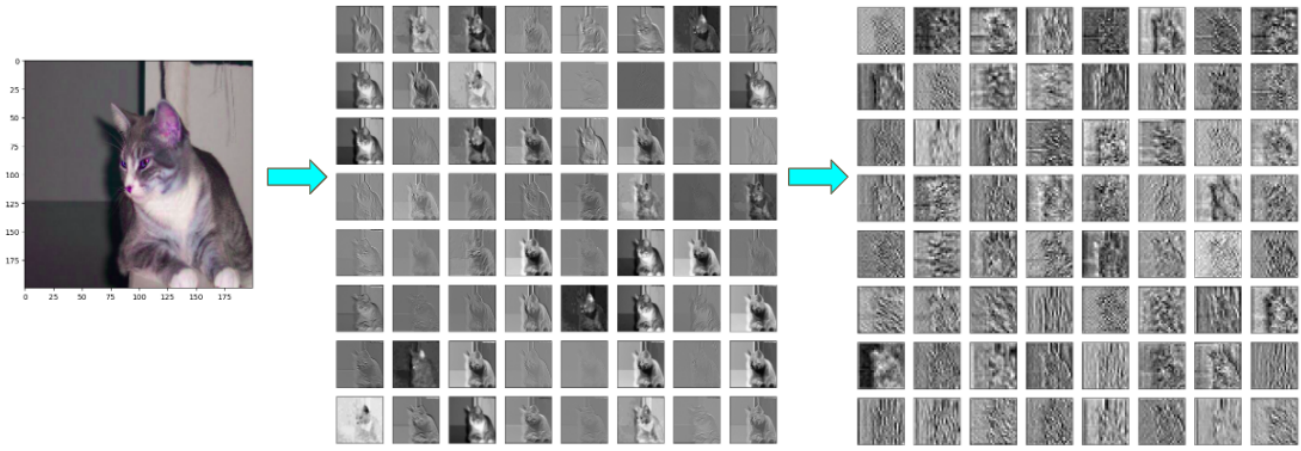

Figure 6: Representations used by different filters in layers *conv1_conv* and *conv3_block2_2_conv* respectively for the input image (left to right)

more precisely, we choose a random subset of filters at a convolutional layer and aim to generate the image that maximizes the activations for those filters. Neurons get excited by features present in input, so they learn to be feature detectors. Usually, basic or low level features (lines, curves, textures, colors) excite layers near input, while high level feature (objects: faces, vehicles, animals) excite deeper layers.

Feature visualization is the process of making the learned features of a neural network explicit [9]. It involves finding the input that maximizes the activation of a specific unit in the network. The unit can refer to individual neurons, channels (feature maps), entire layers, or the final class probability in classification.

Mathematically, feature visualization is an optimization problem where the goal is to find an image (referred to as img^*) that maximizes the activation of a unit.

$$img^* = arg \max_{img} \sum_{x,y} h_{n,x,y,z}(img)$$

The activation function is denoted as "h" and depends on the neuron's position (x, y), the layer (n), and the channel index (z). To visualize an entire channel in a layer, the mean activation across all spatial positions (x, y) is maximized. There are different ways to approach this optimization problem. One approach is to generate new images starting from random noise, imposing constraints on the image to ensure meaningful visualizations. Regularization techniques such as jittering, rotation, scaling, frequency penalization, or incorporating priors from generative adversarial networks (GANs) or denoising autoencoders can be used to reduce noise in the feature visualization.

Feature visualization can be applied to various types of data, not just images. It can be extended to fully connected neural networks for tabular data or recurrent neural networks for text data. In these cases, the "feature" refers to the input data itself, such as tabular features or sequences of characters in text. For text data, recurrent neural networks have been shown to learn interpretable features, with specific neurons being activated for certain patterns like opening and closing brackets or URLs.

The Network Dissection method is used to link human concepts with individual neural network units. It requires additional datasets labeled with human concepts to establish connections between the concepts and the network units.

Overall, feature visualization is a technique for understanding and interpreting the learned features of neural networks, and it can provide insights into how the network processes and represents information.

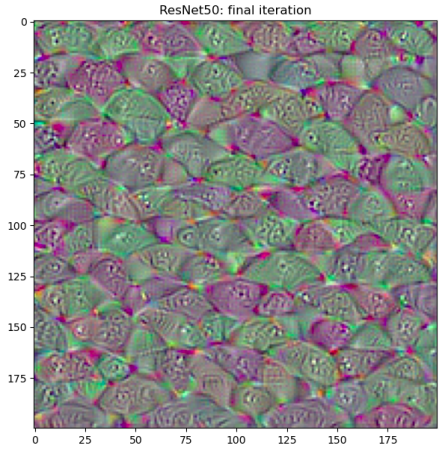

Figure 7: Input image optimized for *conv3_block4_3_conv* layer.

3.4 Saliency Map Visualization

We can visualize the gradients of the model loss with respect to the input image pixels. This gives us a idea of which pixels contribute

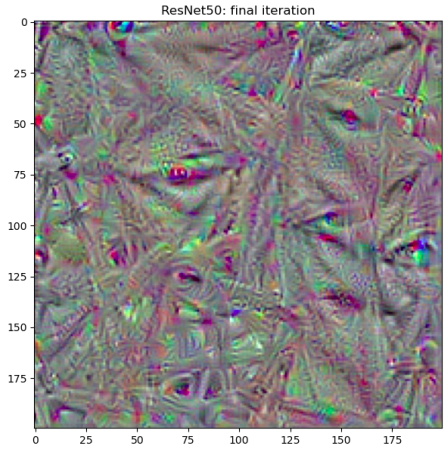

Figure 8: Input image optimized for *conv4_block6_2_conv* layer.

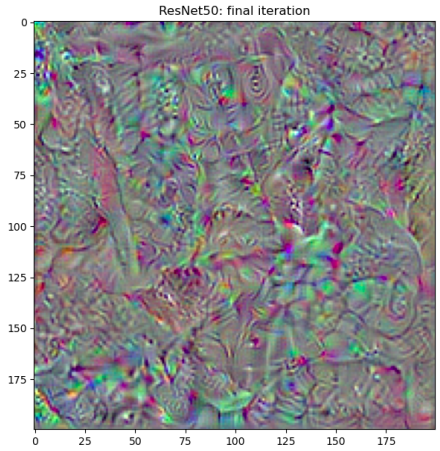

Figure 9: Input image optimized for *conv5_block3_3_conv* layer.

more to the loss gradients and hence more the model parameters update. Hence, this method is also called *pixel attribution*. This method was introduced in the paper *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps* [13]. In the paper, the authors propose a method to rank the pixels of an image based on the influence of their class score. Given an image I_0 with m rows and n columns, the class saliency map $M \in R_{m \times n}$ is computed as follows. First, the derivative w is found using back-propagation. Then, the saliency map is obtained by rearranging the elements of the vector w so that it can be viewed as an image. For a grayscale image, where the number of elements in w is equal to the number of pixels in I_0, the saliency map is computed as:

$$M_{ij} = |w_{h(i,j)}|$$

In the above equation, $h(i, j)$ is the index of the element of w corresponding to the image pixel in the i^{th} row and j^{th} column.
For a multi-channel image, such as an RGB image, we take the maximum across all color channels:

$$M_{ij} = max_c |w_{h(i,j,c)}|$$

Here, c denotes the color channel. For this method, only a single backward pass is required per image, hence computation is very fast. Saliency maps can be extracted using a trained classification CNN, so there is no need for additional annotated data.
The paper also describes a method for weakly supervised object localization using the class saliency maps. These maps encode the object's location in the image and can be used for localization despite being trained only on image labels. The authors propose a simple object localization procedure that involves computing the object segmentation mask using GraphCut color segmentation [1]. The foreground and background color models are estimated using Gaussian Mixture Models, and the segmentation is performed using the GraphCut algorithm. The object segmentation mask is then set to the largest connected component of the foreground pixels.

Saliency maps for some input images in our dataset can be seen in Figures 10, 11, 12 and 13.

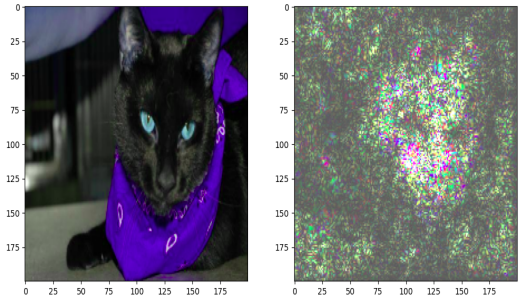

Figure 10: Saliency map for example image.

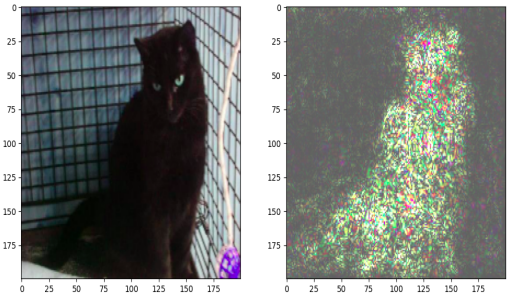

Figure 11: Saliency map for example image.

3.5 Grad-CAM Visualization

This method was introduced in the paper *Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization* [12]. Heat maps are 2D grids of scores associated with specific output class, that, for every input pixel, indicate how important the pixel was with respect to the class considered. The importance of a pixel with respect to a class can be computed by computing the gradient of the score produced for that class with respect to the input pixels.

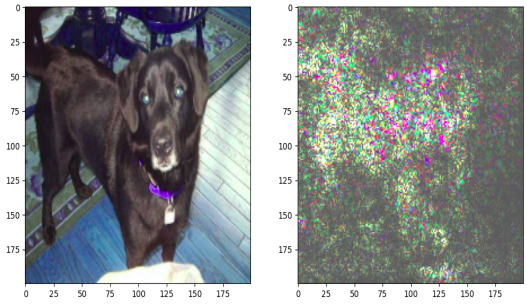

Figure 12: Saliency map for example image.

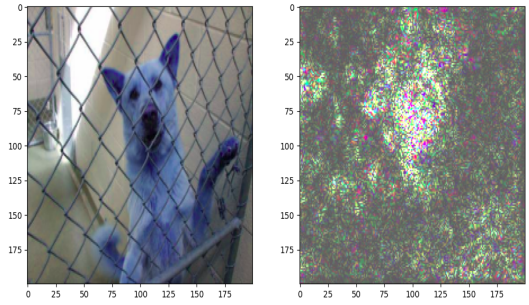

Figure 13: Saliency map for example image.

The more the gradients, the more the pixel contributes to the score for that class. In order to obtain the heat map of the image with respect to any class C, first the gradient of the score y^c for class c before the final softmax is computed with respect to the feature maps (the intermediate representation of the image in the network) A^k. This is back-propagated iteratively through all the convolutional layers and is globally average-pooled across feature maps to obtain the *neuron importance weights* α_k^c. Then, the feature maps are combined using the obtained weights α_k^c to obtain the required Grad-CAM heat map. This process is described in equations 1 and 2 below.

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

$$H_{Grad-CAM}^c = ReLU\left(\sum_k \alpha_k^c A^k \right)$$

It is a very general method that can be applied to a wide variety of CNN model-families - fully connected layers, multi-modal inputs, etc. without re-training or architectural changes.
Grad-CAM visualizations for some input images in our dataset can be seen in Figures 14 and 15.

A relevant approach to Grad-CAM is call Class Activation Mapping (CAM) [17] approach to localization. This approach, as opposed to Grad-CAM, modifies the architecture and replaces the fully-connected layers with convolutional layers and pooling to obtain class-specific feature maps. A drawback of this approach would be that it requires softmax layers before convolutional layers. Such a condition limits it's applicability to other CNN based models which might be better performing. Grad-CAM, on the other hand, has introduced a new way to combine feature maps using gradients without any changes to the architecture. Grad-CAM is a generalization of CAM.

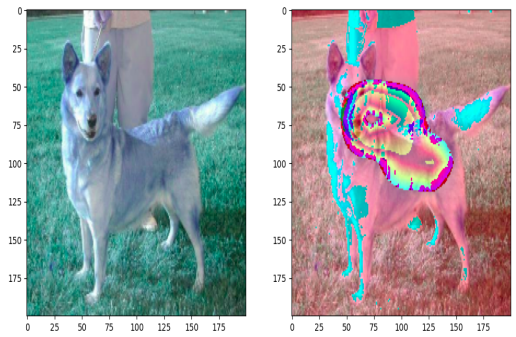

Figure 14: Grad-CAM visualization for example image.

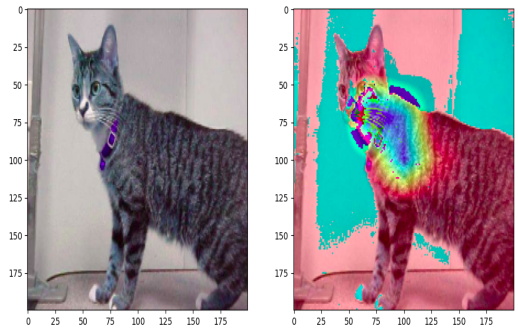

Figure 15: Grad-CAM visualization for example image.

Grad-CAM is a faster approach compared to other techniques like *occlusion* [16] and *patch-wise classifications* [11] since they require multiple forward passes while Grad-CAM requires only a single forward pass and a partial backward pass per image.
Grad-CAM visualizations can help in debugging the decision processes of convolutional neural networks, especially in multi-class scenarios. This can be done by analysing the Grad-CAM visualizations and ensuring whether the focus of the model was on correct features while making the predictions or not. This differentiates it from previous methods like Saliency Maps which are not class-discriminative.
The authors conducted a study to measure if Grad-CAM explanations help users to gain trust in outputs given by the model and if they enable them to distinguish between strong and weak models even when predictions are identical.
One limitation of this method is the inability to visualize the complete dataset at once and check if the model learns the correct associations of features to classes and not some noise that might be present in the dataset. Motivated by this, the method in the next subsection aims to address this issue and visualizes the complete dataset at a large scale.

3.6 Activation Atlases

Activation atlases were introduced in [2]. The paper, code and demo are available at [6].

So far, methods that we saw were limited to seeing only how the network sees a single input. There are some other techniques that give more global views, but they tend to have other downsides. For example, [10] organizes all images of a dataset by their activation values, but this visualization isn't very useful as it just indicates which images the network responds to equally. Activation Atlases give a bigger picture of the network. They plot feature visualizations (that can be obtained by optimizing the image for a layer) of averaged activations of a layer as a huge atlas. Combining neuron activations give us a bigger picture by showing common combinations of neurons. They reveal visual abstractions as well as high level misunderstandings.

The process to generate activation atlases can be summarised as - first we collect activations from many (millions) of images corresponding to each filter, then we sample one spatial activation per image, then we use dimensionality reduction techniques like t-SNE [15] or UMAP [8] to further reduce the sample size while preserving the local structure and finally we draw a grid and average all activations that lie within the grid to obtain the atlas representation.

Attribution vectors for each activation vector are also computed, which indicate how much an activation vector influences the logit for each class. This is computed by the dot

$$h_{x,y} \cdot \frac{\delta logit_c}{\delta h_{x,y}}$$

where $h_{x,y}$ is the activation vector at position x, y and $logit_c$ is the class logit for the desired class. This is similar to Grad-CAM [12], but without the spatial averaging of gradient over filters. The attribution shown for cells in the activation atlas is the average of attribution vectors for activations in that cell.

The authors use the model InceptionV1 GoogLeNet [14] on the Imagenet database [3]. An activation atlas that was generated on this is shown in Figure 16. Different places in the atlas identify different features - fur, feet, waterbodies, sand, faces, shapes, colors. Across layers, atlases tend to represent more specific and complex activations.

We can also look at the class-wise display of the atlas by identifying how much each activation contributes to a class and displaying it as opacity of the grid cell so that areas for that particular class are clearly visible. We can compare this display for different class to see which activations are shared across classes and are later combined to form more complex activations.

To further isolate classes, we can also generate something called *class activation atlas*. Here rather than dimming other activations, we completely remove other activations before running the dimensionality reduction step. This results in an atlas that shows the activations that contribute mostly to the class we have considered. These can be used to identify wrong kind of correlations if any also to compare classes. The authors have used this to identify wrong correlations to in turn construct adversarial examples by adding patches to bias the image towards making the wrong prediction. This gives concrete examples of where the model would fail to make a correct prediction and can helps users know what wrong associations the model has learnt.

Figure 16: An activation atlas on the ImageNet dataset.

4 LIMITATIONS AND DISCUSSION

We saw a lot of interpretability methods for CNNs, which can be useful in a lot of scenarios and can help to identify systemic errors the model might be making. However, there still might be mistakes that we miss to identify. That is because there is still a lack of exhaustive explanation for the interpretations being done. Hence, we cannot fully rely on the present deep learning models for problem solving due to the lack of proper explanations. We need to be able to compare models using these methods, showing similarities and differences beyond error rates. In essence, present methods of neural network analysis do not suffice and this calls for more rigorous deep learning model explanalibilty methods.

We are moving towards this goal gradually, with methods such as *Grad-CAM* enabling us to qualitatively (rudimentarily) compare CNN-based models and *Activation Atlases* paving the way towards larger scale analyses and towards model comparison.

Future work also includes investigating interpretability in decisions made by deep networks in the domains of reinforcement learning, natural language processing and video applications.

REFERENCES

[1] Yuri Boykov and Marie-Pierre Jolly. 2001. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, 1, 105–112 vol.1.

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

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, K. Li, and Li Fei-Fei. 2009. Imagenet: a large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

[4] Jeremy Elson, John (JD) Douceur, Jon Howell, and Jared Saul. 2007. Asirra: a captcha that exploits interest-aligned manual image categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., (Oct. 2007). https://www.microsoft.com/en-us/research/publication/asirra-a-captcha-that-exploits-interest-aligned-manual-image-categorization/.

[5] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

[6] A. Karparthy. 2019. Introducing activation atlases. Retrieved 2014 from https://openai.com/research/introducing-activation-atlases.

[7] Martín Abadi et al. 2015. TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. (2015). https://www.tensorflow.org/.

[8] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. 2018. Umap: uniform manifold approximation and projection. *J. Open Source Softw.*, 3, 861.

[9] Christoph Molnar. 2019. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable.*

[10] Chris Olah and Ludwig Schubert. 2014. T-sne visualization of cnn codes. https://cs.stanford.edu/people/karpathy/cnnembed/.

[11] Maxime Oquab, Léon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and transferring mid-level image representations using convolutional neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 1717–1724.

[12] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. 2016. Grad-cam: visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128, 336–359.

[13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: visualising image classification models and saliency maps. *CoRR*, abs/1312.6034.

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9.

[15] Laurens van der Maaten and Geoffrey E. Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9, 2579–2605.

[16] Matthew D. Zeiler and Rob Fergus. 2013. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*.

[17] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. 2015. Learning deep features for discriminative localization. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2921–2929.