Hidden Figures: Using Activation Mapping and Maximization to Visualize Hidden Layers in Convolutional Neural Networks

A senior thesis, presented in partial fulfillment of

the Bachelor of Science Degree with Honors

Kiana Khozein

Department of Computer Science

Under the supervision of Jordan Pollack

Brandeis University

May 2017

Contents

Acknowledgments

There are many people without whom I would not have been able to complete this thesis, and I would be remiss in not expressing my profound gratitude towards them.

Thank you:

Professor Pollack, my advisor, for allowing me the opportunity to conduct my senior research under you this year. You introduced me to a field of computer science I had no previous experience with, and I will always value the lessons and knowledge gained from my time as your student.

Nick Moran, for teaching me everything I know about CNNs and Tensorflow. Without your help, advice, and guidance I would have been very lost in just about every aspect of this research. I cannot thank you enough.

My friends, family, and especially my parents, for all your support in this and throughout my Brandeis career.

Jacob Regenstein, Talie Massachi, and Karishma Reddy Khan, for constantly providing sympathetic ears, hot tea, and the encouragement I needed to make it through this process.

Abstract

Convolutional neural networks (CNNs) have been proven able to recognize and categorize images with near-human accuracy once trained - in fact, for some tasks the state of the art for CNNs outperforms human ability. However, the methods by which CNNs achieve this level of accuracy are largely unclear. In a CNN consisting of several hidden layers of feature maps (also called filter layers), when these filters convolve over a given input image, only the output of the first can be mapped directly to the pixel space. As a result, several different methods for rendering these inner layers of CNNs have been put forth in the past few years. One such technique is to graph each activation produced when a layer of filters convolves over an image. Visualizing these activations allows us to see the way subsequent filters select for image characteristics changes over time, giving insight into both how the network learns classifications and how significant elements of reconstructions built from these CNNs are determined. It has further been shown that activation mapping and maximization produces useful visualizations even when applied various kinds of input, and can be used to make advanced and open-ended queries of CNNs while bypassing costly search operations.

1 Introduction

The intuition behind visualizing activation layers comes from the question of how the network's filters interact with the input the networks are given to classify. After the advancements of the past several years in the development of neural nets, the state of the art for image classification using CNNs performs at near-human accuracy, but due to the general mystery of how exactly these classifications are performed, improvements in performance have often come about as the results of a combination of trial and error and lucky "hacks." As such, the study of deep learning as a whole would greatly benefit from a formalized method for analyzing the internal process of image classification.

One of the most salient ways to accomplish this is to map the activations produced when a set of filters convolves over a given input. Several methods have been recently introduced that provide different approaches to this task; this paper serves to collate and examine three of the most promising as a means for supporting the practice as a whole. Notably, research done by Zeiler and Fergus (2013) introduces the practice of linking deconvolutional network to each convolutional layer of a CNN. Zeiler and Fergus propose that these linked deconvolutional nets would serve as a visualizer for the otherwise uninterpretable output produced by the hidden convolutional layers by inverting the convolution and mapping the result of this inversion back to the image space. Following my study of this paper, I took a convolutional autoencoder (which differs in design from Zeiler and Fergus's model, but also utilizes inverted convolutional layers to reconstruct its given input) and visualized the activations from the convolutional encoding and deconvolutional decoding steps for comparison.

Branching out from the use of deconvolutional nets to facilitate activation mapping, I then discuss two more papers which apply activation mapping to different kinds of images. Yosinki et al. (2015) propose a method for using synthesized inputs in a way that takes advantage of their potential for high activation excitement while also regularizing the input to be more visually interpretable, whereas Mahendran and Vedaldi (2016) present a way for creating more easily visualized natural pre-images in such a way that they interact favorably with activation maximization. Finally, in my conclusion I discuss one or two further areas in which activation visualization can help improve the generative power of CNNs.

1.1 A note on the CNN models used by each experiment

In order for the different methods of mapping activations to be comparable, the sophistication of the CNN producing the activations had to be uniform. However, testing all methods mentioned on the same CNN and dataset would compromise the integrity of the research by exposing it all to the same specific quirks and flaws. Thus, to supply a healthy variation, most of the papers discussed here use different state-of-the-art

CNNs to test their visualization methods. These include:

- The architecture as described in Krizhevsky et al. (2012), trained on the ImageNet 2012 dataset (Zeiler and Fergus)
- The architecture as described in Krizhevsky et al. (2012) but with normalization layers added after pooling, trained on the ImageNet 2012 dataset. (Yosinki et al.)
- More well known state-of-the-art networks such as AlexNet, VGG-M, and VGG-VD (Mahendran and Vedaldi)

For the original work as explained in Section 2.2, we build an autoencoder using convolutional encoding and deconvolutional decoding steps using Google's Tensorflow, and trained it on images from the Cifar-10 dataset.

2 Using attached deconvolutional networks to visualize activations for CNNs

2.1 Formalization of this method by Zeiler and Fergus

"Visualizing and Understanding Convolutional Networks" (2013) by Zeiler and Fergus is perhaps one of the most groundbreaking papers for studying the evolution of filters as per their activation layers. Zeiler and Fergus proposed utilizing deconvolutional neural networks (DCNNs) to shed insight on the inner machinations of CNNs. The deconvolutional nets used by Zeiler and Fergus can be thought of as a convolutional model that uses the same elements as a typical CNN, such as filtering and pooling, but in reverse, thus projecting the intermediate activations of a particular layer of an associated CNN back to the pixel space. To visualize the hidden layers of a trained CNN, Zeiler and Fergus proposed linking a deconvolutional net to each layer, which would take the convolved output of that filter layer as input. Then, the DCNN would make a series of operations (unpooling, rectifying, and filtering) on that un-visualizable input, exactly mirroring operations performed by the CNN to generate the input, but in reverse until the dimensions of the input signal agree with pixel space.

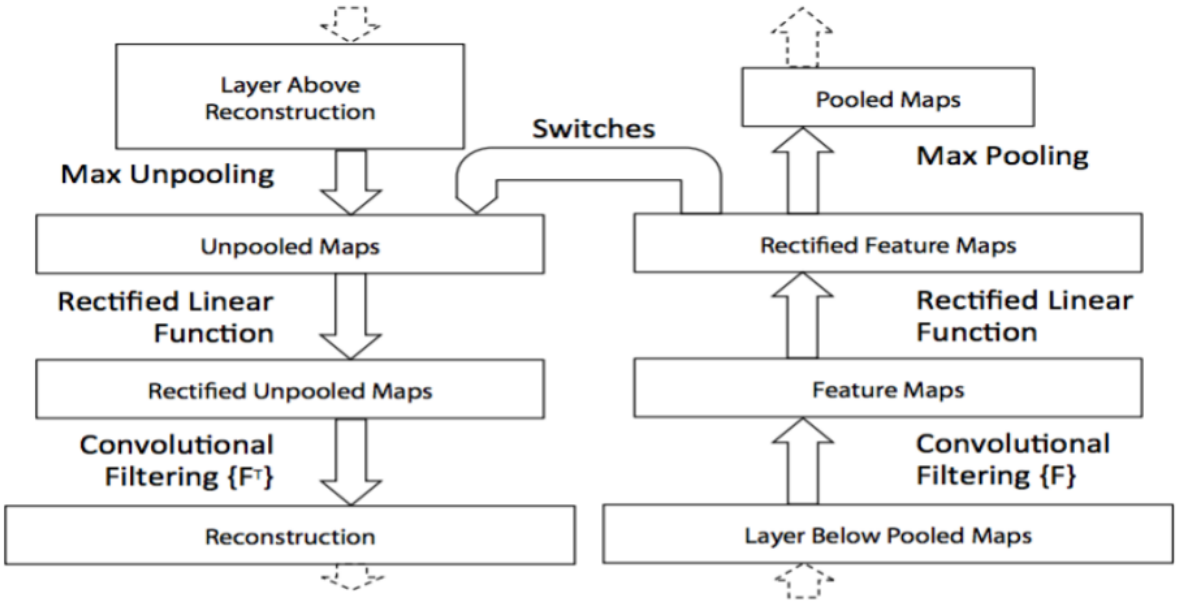


Figure 1. The architecture of Zeiler and Fergus's deconvolutional model. Shown are the processes composing a convolutional layer (right) and those of the attached deconvolutional net (left). (Zeiler and Fergus. 2013. *Visualizing and Understanding Convolutional Neural Nets*, p. 3).

For Zeiler and Fergus's method to work properly, the DCNNs need to invert (or approximately invert) the operations made by the normal CNN to convolve over an input. A standard operation for CNNs is to insert a pooling step in between each convolutional step. After the filter layer convolves over its input, a pooling filter (typically of size 2x2) is applied to the convolutional layer's output and takes the largest

element (practically speaking, a pixel value) from the selected window. The pooling filter moves over this input in strides typically equal to its width and height, and a new, smaller feature map is constructed out of these max values. Repeated successively after every convolutional and ReLU non-linearity function, this process reduces the spatial size of the feature maps, thus reducing the computational load placed on the CNN and making representations of the feature maps more manageable as a whole. It also makes the networks invariant to small changes to the input image as it only propagates the most significant pixel values to the following layers. However, max-pooling is also usually a non-invertible operation, in equal parts because all but the max value within a window is thrown out and because the operation does not inherently record the maximum's original position. To combat this, Zeiler and Fergus propose a system of "switch variables" to track the original location of the max value prior to max pooling in the CNN. The DCNN looks at these switch variables when reconstructing the image to determine where to place each maximum it finds, so that the relative dispersion of activity to non-activity is preserved in the reconstructed image.

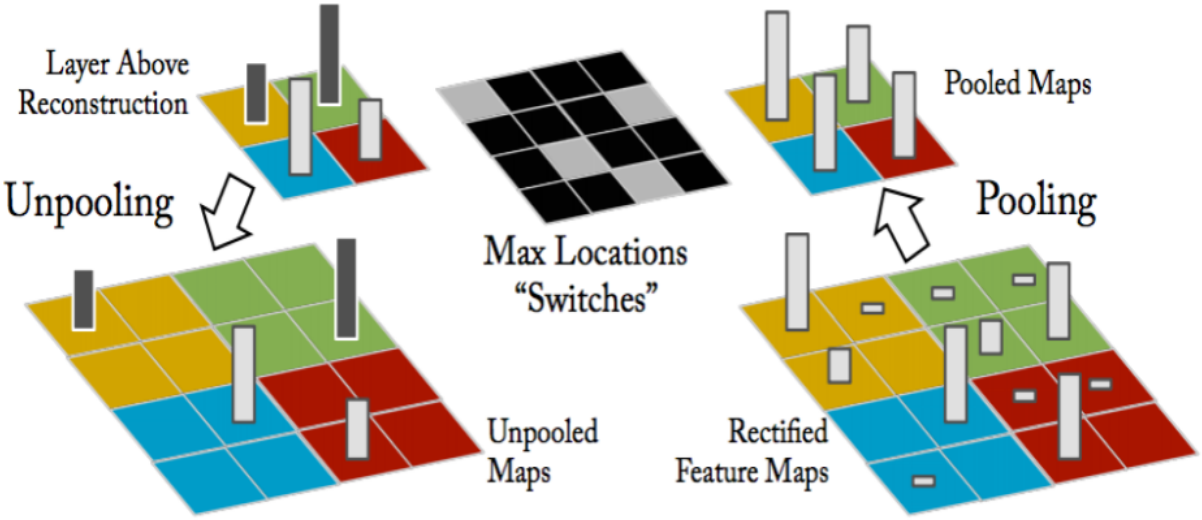
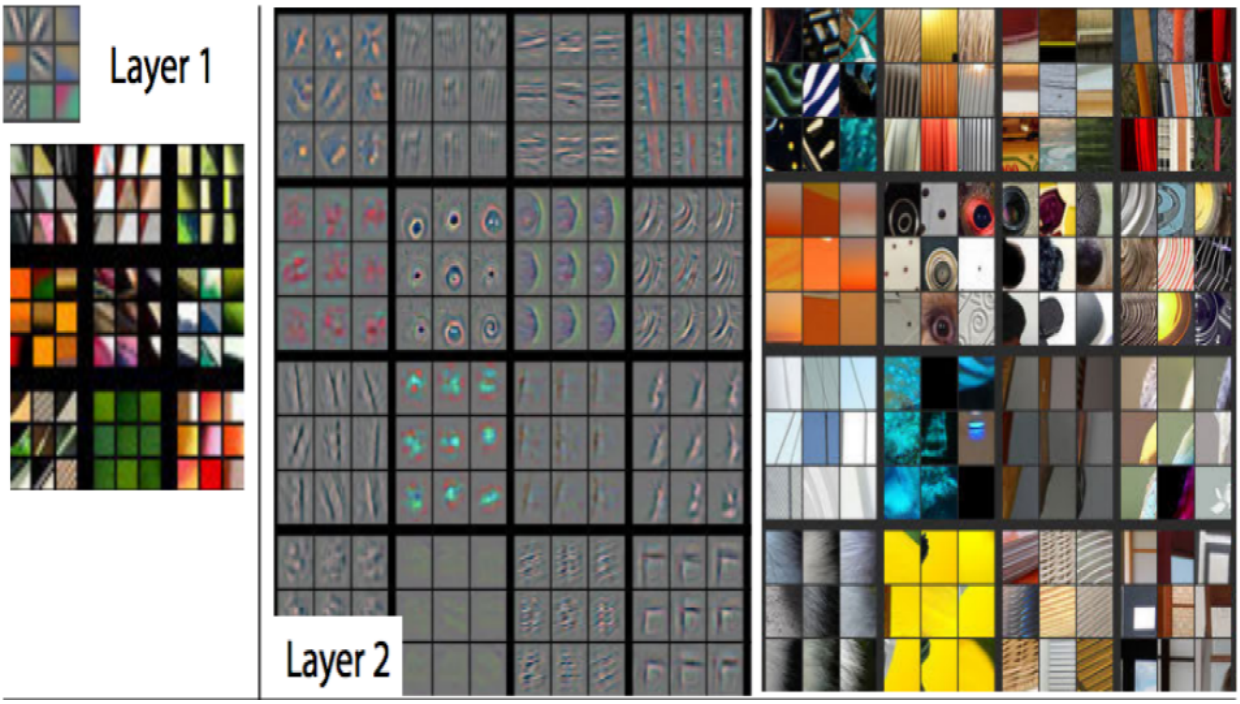


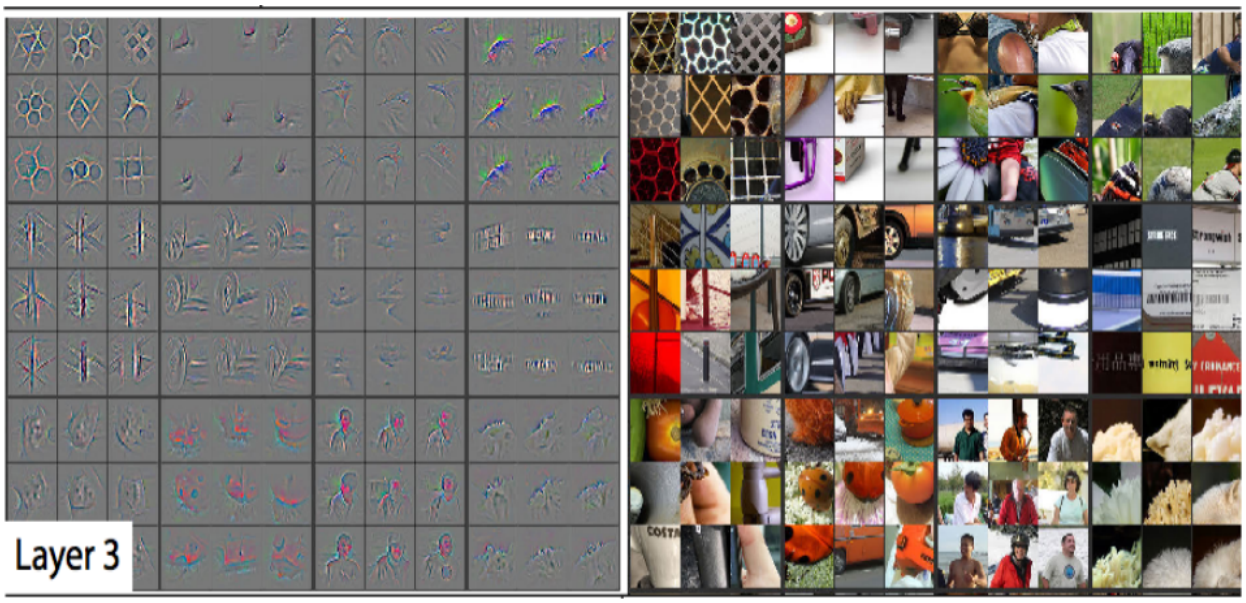
Figure 2. A diagram of the pooling and unpooling operations. Max values as found within the selected windows are represented by bars of varying height. Switch locations are preserved separately in the pooling step, allowing for a locational reconstruction of the max pixel values in the deconvolutional net's unpooling step. (Zeiler and Fergus. 2013. *Visualizing and Understanding Convolutional Neural Nets*, p. 3).

Without the addition of some non-linear activation function, purely linear neural networks are unable to approximate ambiguous neuronal activity (such as when a convolved value is negative). So, after the convolutional step, the result are fed into a ReLU (short for Rectified Linear Units) non-linearity ($f(x) = \max(0,x)$ where x is the input matrix) prior to max-pooling. This provides a simple gradient signifying neural activity, and as a side effect ensures that every output of the non-linearity is positive. Zeiler and Fergus argue that in the interest of preserving total inversion of the convolutional process, the deconvolutional filtering step should be applied to rectified feature maps as well, so the unpooled feature maps are fed through

the same ReLU non-linearity as in the convolutional layer. The final step, filtering, inverts the attached filter layer by flipping each filter horizontally and vertically, and then applies these transposed filters to the rectified maps produced by the ReLU non-linearity. This is not exactly a direct inversion of the analogous process CNNs use to convolve over feature maps because the CNN will apply the non-inverted filters to the feature map produced by the previous layer. It is important to recognize that the DCNN at each layer only inverts the convolutional layer it is connected to, as it is only attempting to reconstruct activations produced by the filters at that layer.

The activations produced by this method clearly show which structures in a given input excite the activated feature map, and when compared side-by-side to the input for the activations, the shapes within the activations become recognizable. It is worth noting how each layer highlights a different aspect of the given inputs. Early layers are devoted to detecting simple image attributes, like the edges seen in layer 1. As we move to deeper layers, the image attributes they identify become more and more sophisticated - layer 2 begins to isolate localized shapes and corners, and layer 3 is capable of recognizing delicate and intricate textures within the given images. We can see how these complex features are simply combinations of simpler features in the layers below, giving us critical insight into how CNNs learn to categorize images.





Figures 3 and 4. Visualizations of Zeiler and Fergus's fully-trained model. The top 9 activations for the given test images are shown, next to the feature map activations that have been reconstructed with the deconvolutional method. Note how the complexity of features identified increases between layers 1 and 2 (top) and 3 (bottom), and the strong grouping of features within the feature maps. (Zeiler and Fergus. 2013. *Visualizing and Understanding Convolutional Neural Nets*, p. 4).

2.2 An original application of using deconvolutional nets to visualize hidden layers in an autoencoder

I chose to extend Zeiler and Fergus's method of visualization to an autoencoder trained on the Cifar-10 image set. The architecture of the autoencoder is such that there are four encoding layers represented as 2d convolutional layers that take the original input, a 32x32 image with 3 channels, and convolve it at each layer so that the image dimensions are reduced by half at each layer as the number of channels doubles, eventually resulting in an encoded value matrix of dimension 2x2x128. The decoding layers are direct inversions of the analogous encoding steps, upsampling from the final encoded output to produce larger images with fewer channels until the original input size of 32x32x3 is reached. This inversion is accomplished by flipping the expected input and output channels in the filter construction for the decoding layers and taking the deconvolution before passing the input to the non-linearity function. Both the encoder and decoder feed their given inputs through a ReLU non-linearity, like Zeiler and Fergus, and have a stride length 2 in both the vertical and horizontal direction (this stride length being the step that physically produces the upsampling). At this point, the autoencoder can be run and the activations for each of the encoding and decoding layers can be saved.

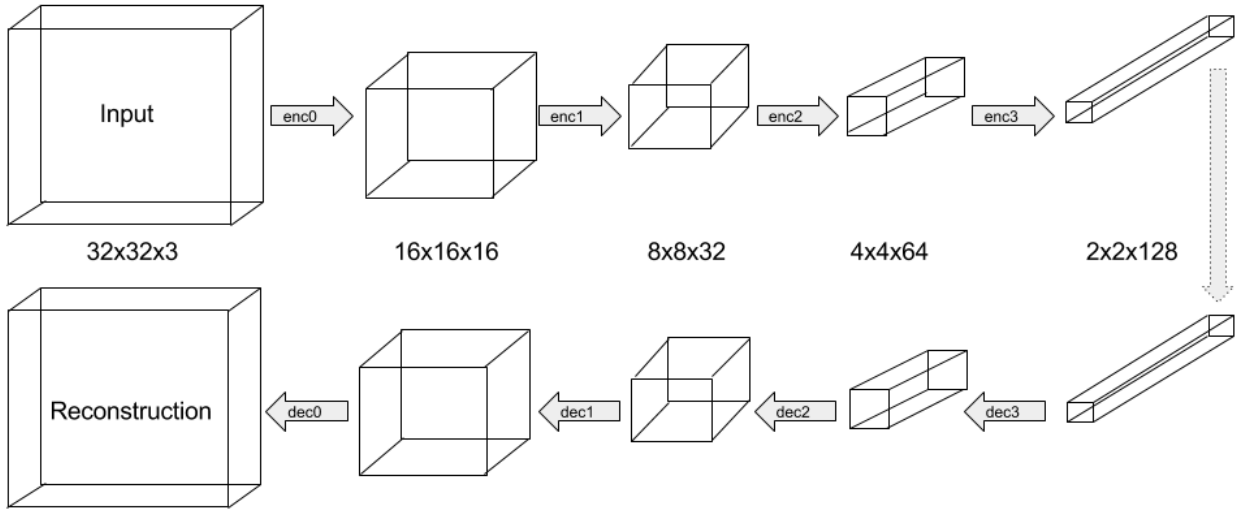
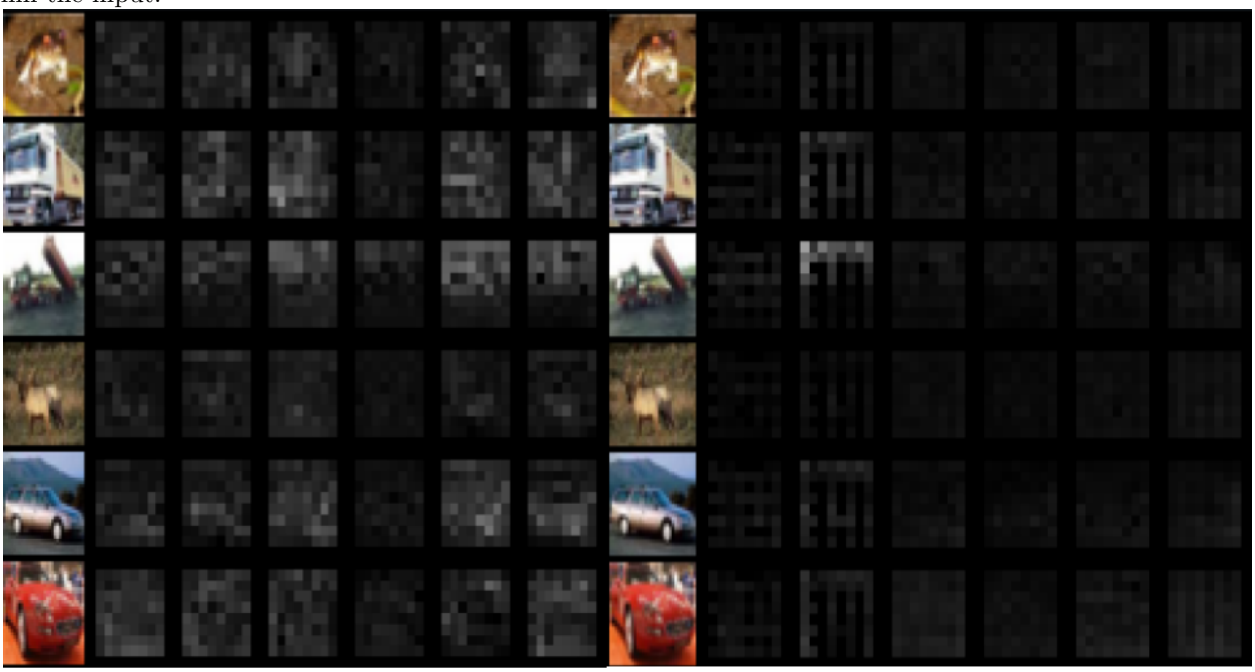
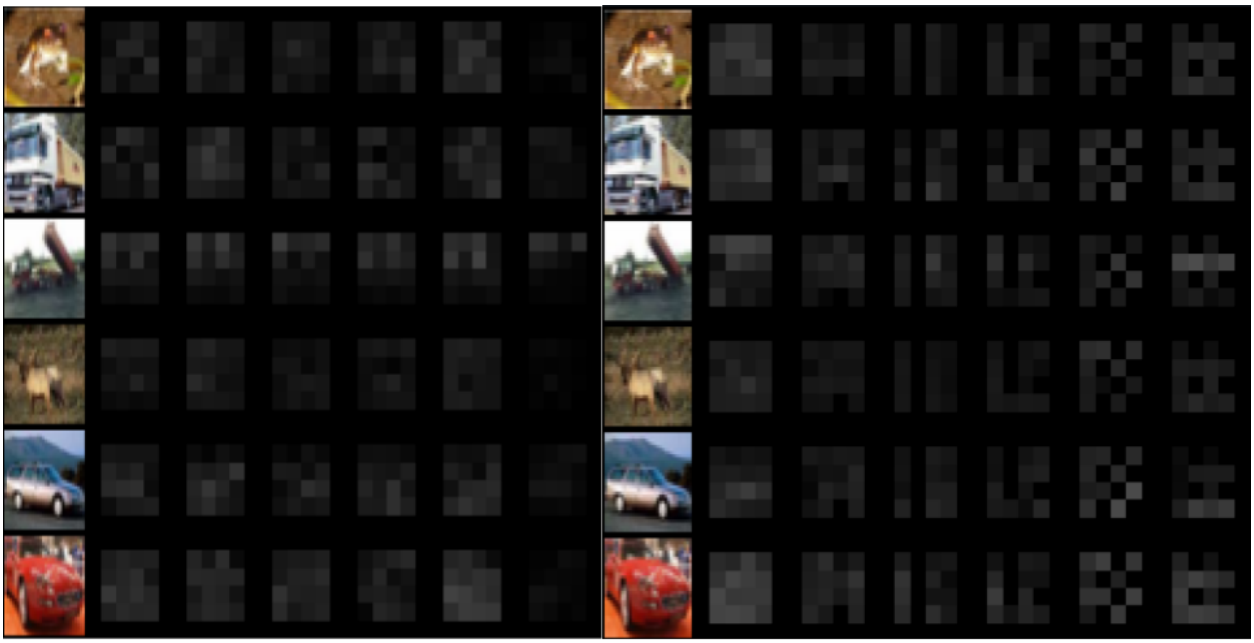


Figure 5. A diagram of the architecture of the autoencoder from Section 2.2. Note that the 2x2x128 signal output of the enc3 layer is the same one being input to the dec3 layer. When imaging the activations at each step between the encoding and decoding layers, we pair the layers that produce the same output dimensions for ease of visualization (e.g. (enc0, dec1), (enc1, dec2), (enc2, dec 3)).

The most useful visualizations came from the first layer of the encoder and the penultimate layer of the decoder. To benefit from a more accurate network, the images shown were selected from the last training epoch. Considering first the activations from mid-high level encoding layers (such as enc1 and enc2) we can see a wider distribution of activity from the input image, and the filters that are activated by the

image have a lot of intricate detail that provides a recognizable rendition of the input. When considering the paired decoder activations (dec2 and dec3, respectively), we can see that fewer elements activate the transposed filters, but that elements that provoke an activation incite a stronger response, indicated by the saturation of the activated areas. As this value matrix is compressed through the autoencoder, by the time it reaches higher layers of the encoder and decoder (such as enc2 and dec3) the activations lose most of their descriptiveness as result of the small image size. Still, if we regard active pixels as locational signposts for image activity we can assume that these filters are classifying elements of the signal relative to their position within the input.





Figures 6-7. Paired encoding and decoding layers visualizing activations of the first 6 filters for inner layers of the autoencoder (top: enc1 (left), dec2 (right); bottom: enc2 (left), dec3 (right)). The activation maps have been enlarged to match the input image size for ease of viewing.

The most interesting result of the visualizing activations from the autoencoder can be seen in the comparison between activations from enc0 and dec1. The encoding layer visualizations show a somewhat typical pattern of filter activations, similar to those seen in the higher layers (albeit in much greater detail). The decoding layer visualization, however, shows a strange checkerboarding pattern that regularly interrupts the visualizations. This pattern is not present at any of the other visualized layers, and is due to a filter size larger than that of its stride. When the filter length outgrows the stride length, at each movement across the input the filter window overlaps an area it has previously classified. This redundancy could then cause artifacts in the reconstruction, so to avoid this we set some of the input values given to the decoding layer to 0. These 0 values are what cause the checkerboard pattern seen in the dec1 visualizations. This result is profound, because not only does this visualization give information about what elements are being identified in the filters - it also gives us the information needed to determine their shape.

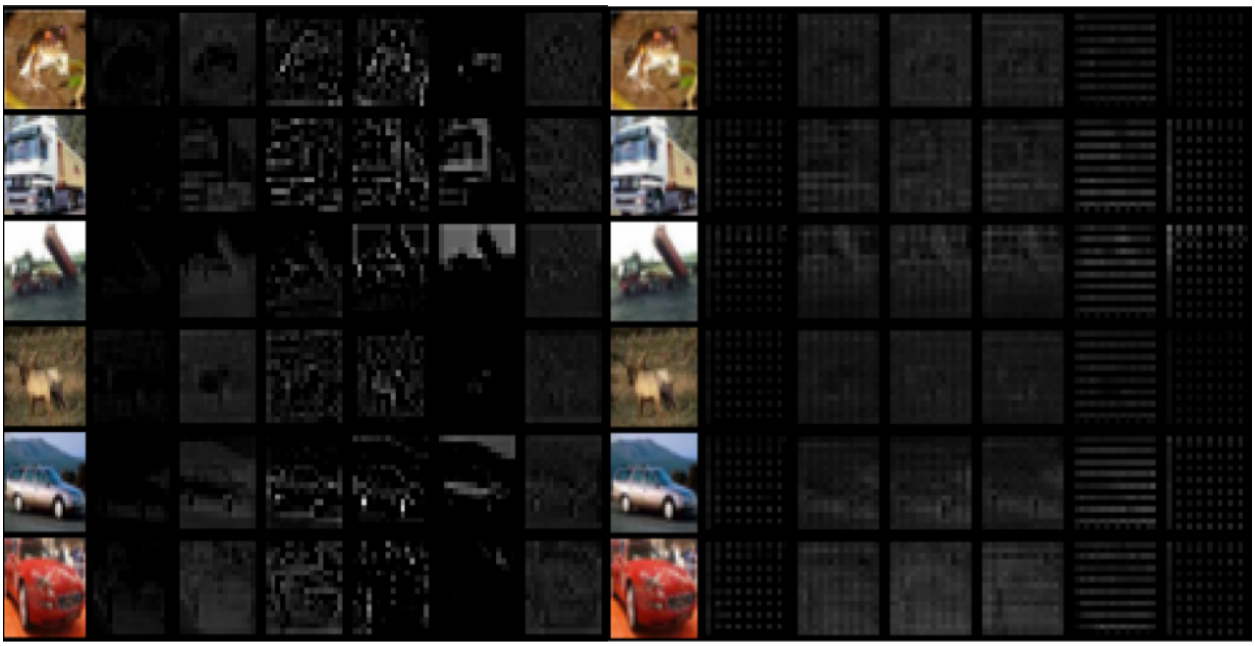


Figure 8. Visualizations of activations from the enc0 and dec1 layers of the autoencoder. Note the strong checkerboarding pattern present in the decoded activation visualizations, which is due to a stride length longer than that of the filter in the deconvolutional step.

3 Visualizing hidden layers via maximizing activations and regularizing optimization with synthetic inputs

Zeiler and Fergus's method presents a way to understand how each layer of filters in the CNN interprets a piece of the original image, allowing us to see which features of a given input image are identified at various points throughout the classification process. This gives insight into what features are prioritized in the higher layers, and thus which elements of the image are discriminative to the CNN. However, this approach is intrinsically tied to the data of the input image - if the input does not contain a feature that the CNN has learned to recognize, this method does not allow us to visualize that feature. If one wishes to investigate a network directly without the medium of a dataset, the alternative is to provide some kind of synthetic input meant to produce certain activations. While promising in theory, the methods that comprise this approach exploited the way activations are produced, such as feeding the neural nets a series of extreme pixel values or high frequency patterns based on an original natural image that all but guaranteed activity from the filter layers. This technique was not without faults, however; as research cited by Yosinki shows, when certain "imperceptible non-random perturbation[s]" are applied to a given test image and given to a trained state-of-the-art network, the network will correctly classify the unaltered image but will misclassify the perturbed image. (Szegedy et al. 2014). It was also shown that attempting to visualize the optimized input produced images that were unrecognizable to humans, thus defeating the purpose of mapping activations in the first place.

In their paper "Understanding Neural Networks Through Deep Visualization" (2015), Yosinki et al. propose four different methods for regularizing these optimized inputs so that they produce recognizable images once visualized. The method they use to create the synthetic input first introduces an initial image from which an initial activation of some filter is computed. Then, gradient descent is applied in steps within the input space to synthesize input that produces ever greater activations of that same filter. After the gradient descent has finished generating a potential image, some combination of the four regularization methods is applied to the generated image to improve its interpretability. The four regularization methods – L2 decay, Gaussian blur, clipping pixels with small norm, and clipping pixels with small contribution – bias optimized images found to produce good activations so that they can be visually interpreted with more ease. According to the authors, all four are valid methods for synthesizing interpretable images, but work best when applied to an image in different useful combinations. L2 decay and Gaussian blur both regularize by respectively eliminating unnaturally high pixel amplitudes and frequencies in the input (which may occur when the images are produced via gradient descent). The resultant image of these two procedures will thus be comprised of relatively small, smooth values; natural images have pixel values that are similarly smooth

and non-extreme, so generated images with these properties are more likely to look natural. However, the issue of noise in the image remains unsolved.

To combat this, one may choose to eliminate pixels that either do not contribute to network classification of the image (clipping with respect to norm) or do not contribute greatly to the activation of the chosen unit (clipping with respect to contribution). For norm, a simple regularization is applied to the image that computes the norm of each pixel over its RGB channels, which then sets the value of any pixel with a small norm to 0. The reason behind this is to prevent unnecessary contribution to the unit's activation; even if the primary object needed for activation is present in the input, these non-zero pixel values will shift to show some random pattern to raise the activation, resulting in a weirdly noisy image. The second approach, clipping pixels with small contribution to the activation, has the same result as clipping w.r.t. the norm but does so in a slightly more direct way. Instead of eliminating pixels that might yield small contributions to the activation due to their norm, this method computes how the activation changes when each pixel is set to 0. Pixels that greatly change the total activation can be classified as having a large contribution, and thus are left in the input; pixels shown to have a small contribution are left at 0 to denoise the final input. While both more thorough and straightforward, this process is also ridiculously slow as it must compute the partial activation for each individual pixel. Yosinki et al. determined a method for approximating this by linearizing the activation around the input, allowing the contribution of each layer of the input to be computed by taking the product of each element of the input and the gradient used to produce it. Then, the absolute value of the sum of the activations across all three of the color channels is taken, which reveals which pixels give a small contribution to the total activation. These pixels are then clipped from the image to reduce meaningless noise from the hidden layer visualizations.

Applying various combinations of these optimizations produce highly recognizable features at upper layers of the CNN. The complexity in each of these visualizations is much greater than those produced by Zeiler and Fergus's method, and we can assert that this is because of the use of synthetic input primed to activate certain feature maps, with the recognizability of these images in these activation maps despite their high activity being a direct result of the optimization methods mentioned above.

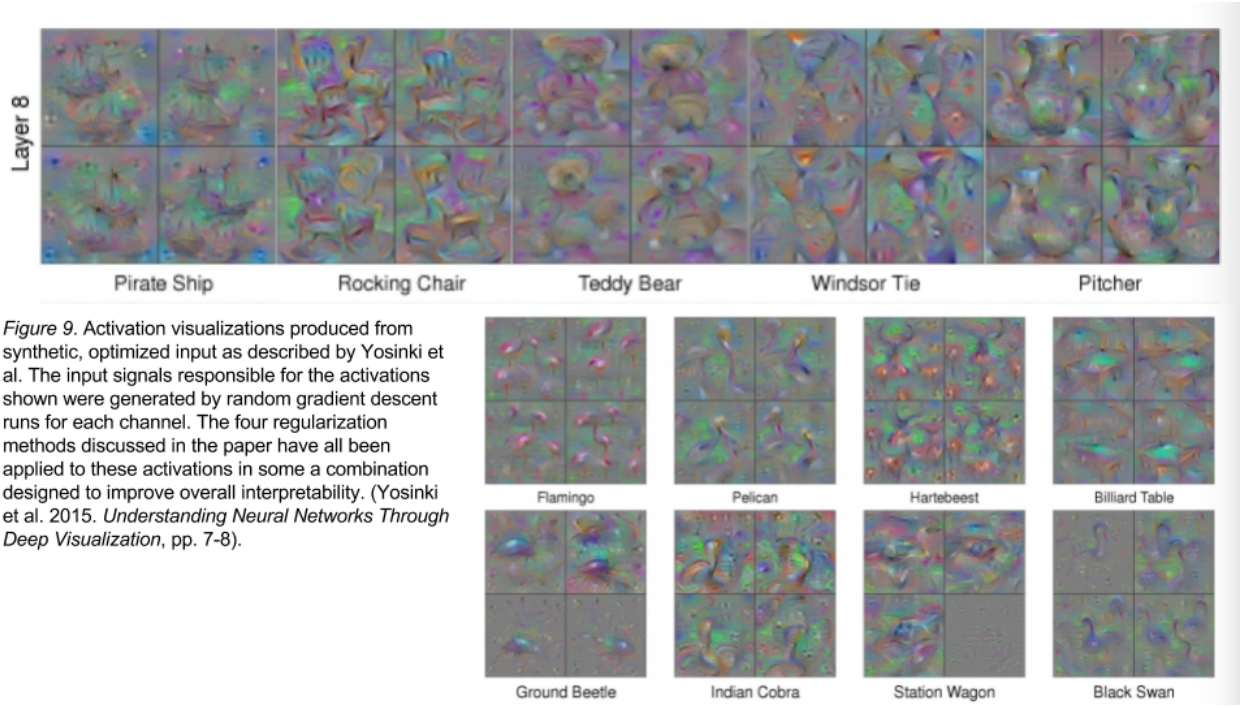


Figure 9. Activation visualizations produced from synthetic, optimized input as described by Yosinki et al. The input signals responsible for the activations shown were generated by random gradient descent runs for each channel. The four regularization methods discussed in the paper have all been applied to these activations in some a combination designed to improve overall interpretability. (Yosinki et al. 2015. *Understanding Neural Networks Through Deep Visualization*, pp. 7-8).

4 Applying activation maximization to natural pre-images

Activation maximization has thus far proven exceedingly useful for visualizing hidden layers within CNNs that are either actively classifying organic images (Zeiler and Fergus) or that have been given optimized input (Yosinki et al.). Yet another way activation maximization can be used to reveal the internal workings of a CNN is by applying this technique to a CNN that uses "natural pre-images" as its input, as described by Mahendran and Vedaldi in their paper "Visualizing Deep Convolutional Neural Networks Using Natural Pre-Images." Natural pre-images are here defined as natural-looking images that contain some notable property. By prioritizing the natural appearance of the input, Mahendran and Vedaldi guarantee that the input images are visualizable, eliminating the issues regarding uninterpretable input faced in the Yosinki paper.

To create these pre-images, Mahendran and Vedaldi first consider a sample input image that exhibits some desired property. Then, they select a set of images with the same dimensions as the sample input, and model a function that compares the potential inputs' activations with that of the original sample image.

This comparison is done via a loss function that does the bulk of the activation comparison mentioned above, two regularizing terms (regularizers) that capture natural image priors, and a constant that which governs the tradeoff between the primary loss and regularization terms. The nature of this equation allows all components except the constant term to be fixed for all regularization and visualization types; thus, for activation maximization, the loss function is such that it uses the original image's representation as a weight vector for biasing the potential image selection towards samples that maximally activate the chosen image component(s). The regularizers are used as a way for generating the prior images used to model potential sample input; of the various regularizing terms that Mahendran and Vedaldi list, there are two that specifically affect activation maximization. The first of these, bounded range, encourages pixel intensity in the image to stay bounded by normalizing the image by its area and by a scalar set to a typical norm of the pixel color vector. Then a combination of soft and hard constraints are applied to the image to limit pixel intensity to the typical norm of the color vector. This is particularly important when applying activation maximization to networks that do not contain normalization layers, as increasing the image range will directly increase neural activity by the same factor. The second regularizer, jitter, randomly shifts the input image before giving it to the representation. Given a random horizontal and vertical translation, at each iteration of gradient descent the translation is applied to the potential input and the expectation of the loss produced by this jittered image is factored into the weight vector. The shifting within the image produced by the translations injects some noise back into the potential input, thus counterbalancing overly robust downsampling performed in the earlier layers of the CNN and generally yielding sharper and more distinctive pre-images.

The above describes how natural pre-images are selected specifically for visualization by activation maximization. Further on in their paper, Mahendran and Vedaldi explain the result of applying this visualization method to the selected pre-images. They found that the complexity of the activations increased dramatically per layer; while visualizations of activations produced by the first convolutional layer in one of their test subjects only captured general colors and amorphous figures, it was possible to recognize distinct aspects of the images (such as eyes, faces, or wheels).

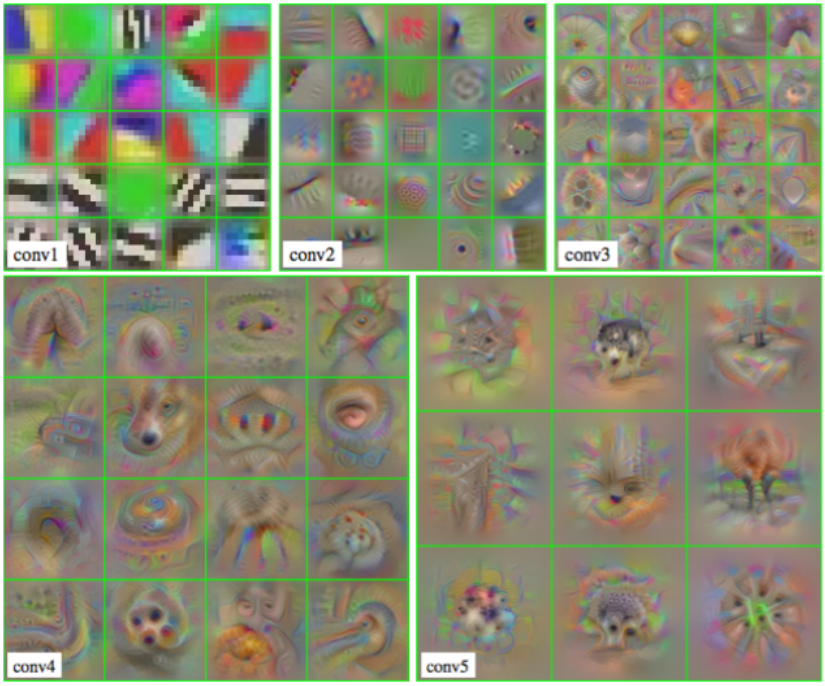


Figure 10. Activation mapping applied to Mahendran and Vedaldi's natural image prior technique as trained on the VGG-M model. (Mahendran and Vedaldi. 2016. *Visualizing Deep Convolutional Networks Using Natural Pre-Images*, p. 19).

In another test case that examined to the same level of convolutional layers, this increasingly complexity was replicated, but it was noted that the colors of the reconstructed images were more saturated (potentially due to a lack of normalization layers in that specific CNN) and that these reconstructions contained more fine details. The increase in detail is likely because the small, blob-like features activate very strongly in the first few layers of the network.

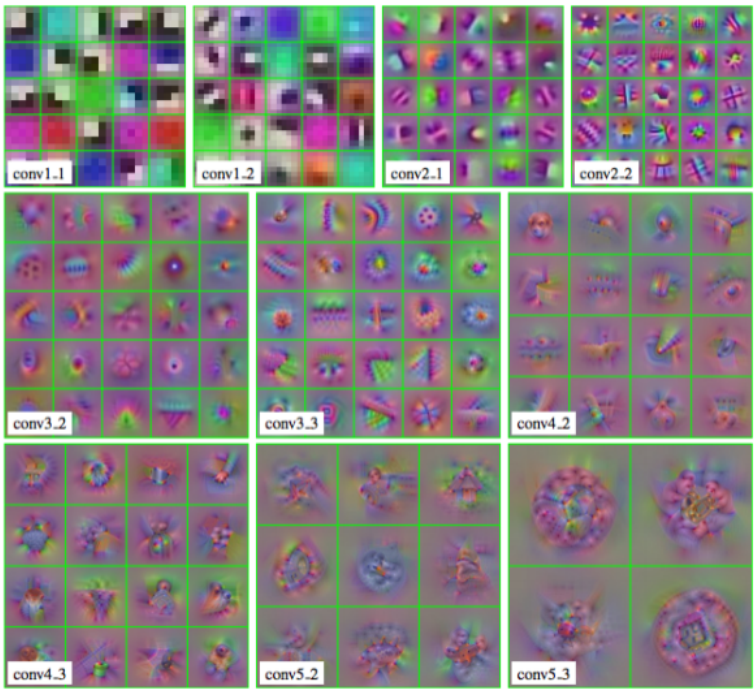


Figure 11. Activation mapping applied to Mahendran and Vedaldi's natural image prior technique as trained on the VGG-VD-16 model. (Mahendran and Vedaldi. 2016. *Visualizing Deep Convolutional Networks Using Natural Pre-Images*, p. 19).

5 Conclusion and further uses

Activation mapping can also be used for more than simply shedding light on how the hidden layers of a CNN go about interpreting and classifying a given input. As put forth in Wang et al.'s paper "A Backward Pass Through a CNN Using a Generative Model of Its Activations" (2016), one practical use for visualizing a CNN's activations is to build a generative model of those activations. This model can then be used to query the robustness of a CNN's classification abilities. Examples given in the paper include asking the CNN to identify the locations of the objects that were used to classify an already-classified image, or giving it an image of an unclear known object and asking it to fit that object class to the occluded image. The general architecture of this problem can be explained as tracing down the CNN's visual hierarchy from a high-level classification down into the image space given an unclear or ambiguous input. This is a sharp departure from methods such as Zeiler and Fergus's, which are intentionally non-generative with respect to activations so as to interact with the CNN in a purely observational capacity. By expanding on the practice of mapping activations, and extending it to a generative model, there exists a way by which we can get useful results from querying CNNs with more vague or subjective tasks, as explained in more detail in Wang et al.'s paper.

Mapping activations is in general a very powerful tool for understanding how convolutional nets go about classifying their input. While there is still somewhat of a black box surrounding the intricacies of these networks, exposing them to diverse forms of input allows us to make general hypotheses and predictions as to their operations. From raw training data as seen with Zeiler and Fergus to Yoskini's synthetic inputs and Mahendran's natural pre-images, in endeavoring to map the activations these images produce two outcomes are created. From a purely observational capacity, these exercises give us a better understanding of where within a given CNN image elements are classified (such as basic shape vs. more complex features, such as texture). Looking beyond that to a generative approach, evidence suggests that pursuing this line of inquiry will lead to more powerful convolutional nets that are better able to handle subjective queries without undergoing heavy computational cost.

References

[1] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[2] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *CoRR*, vol. abs/1506.06579, 2015.

[3] A. Mahendran and A. Vedaldi, "Visualizing deep convolutional neural networks using natural pre-images," *CoRR*, vol. abs/1512.02017, 2015.

[4] H. Wang, A. Chen, Y. Liu, D. George, and D. S. Phoenix, "A backward pass through a CNN using a generative model of its activations," *CoRR*, vol. abs/1611.02767, 2016.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[7] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *International conference on artificial neural networks*, vol. 60, pp. 53–60, Perth, Australia, 1995.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.