# Visualization of Neural Networks

Jana Cavojska
Freie Universität Berlin
Institute for Computer Science
Research group: Database Systems

*17. December 2015*

## ABSTRACT

In the past few years, large convolutional networks have achieved remarkably good results when dealing with image classification problems. However, in order to further improve the classification accuracy, an insight into these networks is needed. By understanding how specific architecture choices influence the overall performance of the networks, we can make improvements that will lead to better classification scores. One way to gain such an understanding is to visualize what goes on in a network while and after we have trained it. In this paper, we give an overview of the methods used to visualize convolutional neural networks, as well as a more detailed description of three of them. We also mention some secondary applications of network visualization that have developed over the past few years.

## 1. INTRODUCTION

When using convolutional neural networks to solve problems such as image classification, usually networks with rather complex architectures (several dozen layers, maybe a hundred filters per layer) can lead to satisfactory results. This complexity makes it difficult for humans to understand which role each network parameter plays exactly, and where there is room for improvement.

One approach that addresses this issue is neural network visualization.
It is common practice (Zeiler at al., 2013 [1]) among visualization approaches to project the activations of the first convolutional layer back into the image space. This means isolating image features that excited neurons of the first convolutional layer and then highlighting these features (lines, color blobs...) in the input image. Such an approach, however, is unsuitable for visualizing higher network layers.

This paper aims to provide a short overview of the existing visualization techniques even for these higher network layers. It can serve as a starting point for readers interested in this topic, mentioning some of the notable works in this field that can serve as material for further reading.

Beside gaining insight into neural networks in order to improve their performance, there is another area where neural network visualization found its applications, and that is generating new images, which are interesting in themselves. Such images are, for example, landscapes created originally to represent the network's understanding of a certain class (Mordvintsev et al., 2015 [2]), which became popular due to their dream-like nature. Another such example could be images that resemble works of famous painters created by applying the "style representation" of a well known work of art to some input image (Gatys et al., 2015 [3]). This "style representation" can be isolated from an image using a convolutional neural network, and is therefore, in a way, a graphical representation of features learned by this network. Such a wide range of practical uses creates a need to better understand not only neural networks, but also the visualization process itself, so that we might discover other such applications.

*Structure of the paper:* In chapters 2 and 3 we give a short introduction into Neural Networks and Convolutional Neural Networks, respectively. In chapter 4, we briefly summarize the existing visualization approaches. Chapter 5 focuses on the deconvolution method by (Zeiler and Fergus, 2013 [1]), chapter 6 on the method by (Mordvintsev et al., 2015 [2], a.k.a "DeepDream"), chapter 7 on the method studying network invariances by (Mahendran and Vedaldi, 2014 [4]). Chapter 8 contains the Conclusion, chapter 9 the References, and chapter 10 is the Appendix.

## 2. NEURAL NETWORKS

A neural network can be thought of as a series of feature extractors stacked on top of each other, with a classifier on top. A classifier is an algorithm which receives as its input a data sample and outputs the answer to the question which one of a series of predefined classes this data sample belongs to.
*Architecture:*
A neural network consists of so called neurons, which are organized in layers (these correspond to the feature extractors). The first layer, the "input layer", is special in that is does not compute anything, it only receives the data of the input sample. The intermediate layers, "hidden layers", perform the feature extraction. The last layer, the "output layer", performs the classification and outputs the class predicted by the network. The neurons of each layer are connected to the neurons of the layer directly above it via a series of weighted edges. The outputs of the input layer's neurons and the outputs of each hidden layer's neurons provide, multiplied by the edge weights, the inputs for the neurons of the layer above it. It follows that the feature extraction is hierarchical and the output layer performs classification based on the features extracted by the last hidden layer.
Several supervised and unsupervised training algorithms for neural networks exist. For an overview, see for example

(Wilamowski, 2003 [5]).

One of the most widely used training algorithms is the supervised algorithm known as "**backpropagation**". Its popularity is due to the facts that it is conceptionally simple, computationally efficient and often works (LeCun et al., [6]). The basic idea is as follows: We initialize the edge weights of our network to some random values (these cannot be zeros), and then perform gradient descent on the surface of the error function, until we, ideally, find its global minimum and hence the best possible classifier for our given training set. When presented with an input data sample, the network passes it in a feed-forward manner through the layers, until a classification is computed by the output layer. The training error, that is, how much the computed class prediction $y'$ differs from the actual class $y$ depends on the weights we have chosen to interconnect neurons of neighbouring layers. This is why the training error can be thought of as a function of the chosen weights (and, of course, of the input data). During backpropagation, we look to diminish the error by gradually adjusting the weights until the error is acceptable. After a random initialization of the weights, the backpropagation algorithm performs iteratively:

1. The feed-forward step: We send the input data sample through the network and compute the error $E$
2. The backprop step: We send the error $E$ backwards through the network and adjust every weight according to the derivative of $E$. The new weights $W$ of a layer in the training iteration $t+1$ will be computed as

$$W(t+1) = W(t) - \eta \frac{dE(W)}{dW}$$

where
$\eta$ is the learning rate and $\frac{dE(W)}{dW}$ is the derivative of the error function in respect to the weights in iteration $t$.

# 3. CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks make the assumption that their input data are always images. This results in some characteristic differences in their architectures when compared to classical neural networks, such as:

- The feature extraction is implemented as image convolution. Image convolution is a process during which an image is multiplied with a smaller matrix (the "convolution kernel"), the centre of which is placed on every pixel $x_i, y_i$ (if no spatial reduction is done) of the input image and multiplied with the subimage at this location. All the numbers in the matrix resulting from this multiplication are summed up (or some other operations are performed with them) and written on the position $x_i, y_i$ of the output image. This yields an output image which, depending on the kernel used, might have sharpened edges, or be blurred, or have some color manipulation done to it. Network layers which perform convolution of their input are called convolutional layers.
- Each convolutional layer applies several such filters. Each of these filters will, during training, learn to recognize different features (simple oriented edges and color blobs in the first hidden layer, simple textures in the second, ... and even whole objects like human faces in the last hidden layer).
- Each neuron in one of the hidden layers receives connections only from a small portion of the neurons in the below itself, as opposed to all of them, which is often the case with less complex networks. This portion is called the "receptive field" of a neuron. The receptive fields of neighbouring neurons within the same layer usually overlap a little, but if the distances of their centres in the layer below are $> 1$ then the dimension of the output of the current layer will be smaller than the dimension of the layer serving as input. This greatly reduces the computational complexity of such networks. If we were to work with fully connected layers, each having a number of neurons equal to the number of input image pixels, the computational cost would make convolutional networks unusable.
- There are so called "pooling layers", the purpose of which is solely downsampling (a max-pooling layer typically passes on only the highest one out of 4 neighbouring input values, discarding all the rest)
- Unlike with normal networks, the network layers and their outputs are multi-dimensional. The first two dimensions of a hidden layer are the width and height of the output volume it will produce, and the third dimension is the number of filters in this layer. The dimension of the output volume is equal to the dimension of the layer that produced it.

(Karpathy, 2015 [7])

# 4. VISUALIZATION APPROACHES

As mentioned in the introduction, visualizing the higher network layers is the most challenging part, because direct projections of the neuron activations back into pixel space (e.g. the input image) are not possible. Some of the existing methods for visualizing even the higher network layers include:

- Performing gradient ascent on the input image itself to find out how the image would need to look like to maximize the neuron outputs (Erhan et al., 2009 [8], Mordvintsev et al., 2015 [2], Yosinski et al., 2015 [9]). This, however, does not provide much information about the neurons' invariances (it does not answer the question how else the input image could look like in order to elicit a similar response).
- (Le et al, 2010 [10]) choose to compute numerically a quadratic approximation for the higher layer neurons. A potential shortcoming of this approach is that a neuron's invariances in the higher levels are far too complex to be captured by a simple quadratic approximation.
- The visualizations used by (Donahue et al., 2013 [11]) show patches within the input picture that are responsible for strong activations at higher network layers.

- The deconvolutional technique used by (Zeiler and Fergus, 2013 [1]) differs from the approach used by (Donahue et al., 2013 [11]) in that they do not show crops of input images, but rather top-down projections that capture the essence of the structures within each image patch that stimulate particular neurons.
- The visualization approach by (Mahendran and Vedaldi, 2014 [4]) gives insight into network invariances. It focuses on determining via gradient descent how exactly the input image could have looked like, given its representation computed by the network.

# 5. DECONVOLUTION METHOD PROPOSED BY ZEILER AND FERGUS

In their 2013 paper [1], Zeiler and Fergus propose a visualization method that interprets the activity on intermediate layers. They map the activity back into the input pixel space, showing the patterns that originally caused a given neuron activation.
This method first lets the input image go through the convolutional network in a feed-forward manner, and then go through the network in the opposite direction, this time using the output of the forward pass as the input. An image that results from this reversed pass should tell us something about the architecture of the network the image has been through. *Figure 1* in the Appendix illustrates this process. The reversed network that the output image from the forward pass goes through is called a "deconvolutional network (deconvnet)". To examine a convolutional network (convnet), a deconvnet is attached to each of its layers.
The network architecture chosen for this visualization is a slight variation of the AlexNet architecture by (Krizhevsky et al., 2012 [12]). One of the differences to this architecture is that the sparse connections used in Krizhevsky's architecture are replaced by dense connections.

For example: After choosing the layer we want to visualize, in order to examine the output of a specific neuron, we would set the outputs of all other neurons in the same layer to zero and pass the image obtained from the convnet at this layer to the attached deconvnet layer. Then this image is sent through the rest of the deconvnet network, going through all the layers it encountered in the convnet, but in reverse.
The AlexNet architecture contains different types of layers, each of which requires special measures to "reverse" it for the purposes of deconvolution:

- **Convolutional layers:** reverting these consists simply of convolving the input image with the inverse image filter (to reverse the convolution with this filter on the forward pass)
- **Max-Pool-Layers:** The pooling operation discards 3/4 of the information of an image, by only keeping the maximum value out of 4 neighbouring pixels. This is a lossy operation and hence cannot be fully reversed. However, the amount of information retained after the reversion can be maximized by recording the positions of such maxima during the forward pass in a set of *switch* variables.

In order to visualize the network, not only the strongest activation (neuron with the highest output value) per layer is sent through the deconvnet and highlighted in the input image, but the top 9 activations. These are projected separately down into the input pixel space. Such projections reveal the different structures that lead to strong activations in the layer being analysed. *Figure 2* shows these projections for each layer next to the pictures being analysed. Simply put, this visualization method "draws" into a picture whatever features were recognized by the network as classification-relevant.

As we can see in *Figure 2*, the features extracted by the different layers show a clearly hierarchical structure:

- Layer 1 features are simple lines, only differing in direction, count, and width.
- Layer 2 neurons have been trained to recognize slightly more complex features like corners and other edge/color conjunctions.
- Layer 3 captures simple textures, like mesh patterns.
- Layer 4 shows significant class specificity (dog faces etc.)
- Layer 5 shows entire objects

It could be said that each layer recognizes objects that consist of features recognized by the previous layer. This makes sense, as each layer's input is also the output of the layer before it.

Another benefit of being able to look into a network is being able to judge at which point the training reached a certain level and no longer needs to continue. *Figure 3* shows the strongest activations of our network after different numbers of epochs (complete passes over the whole training set). While for the first few layers a small number of training epochs is sufficient for the layer activations to converge, the higher layers do not recognize anything notable until after over 40 epochs.

# 6. INCEPTIONISM (GOOGLE'S "DEEPDREAM")

The visualization approach described on Google's research blog (Mordvintsev et al., 2015 [2]) belongs to the group of techniques that perform gradient ascent in the input image space to find a modified version of the image that would maximize the neuron activations.[13]

- One way to visualize neural networks the research blog post describes results in an image showing the network's understanding of a *typical* representation of a certain class.
  For example, if the network was trained to differentiate between different kinds of fruit, we might ask it to "draw" a typical representation of a banana. For this purpose, we start out with an image consisting of random noise, and then gradually change the image towards what the network considers a typical representation of a banana. *Figure 4* shows the input and output images of this process.

  This would not lead to satisfactory results without

imposing some constraints on the way the new image is generated. The resulting image would be one which the network would assign a high score to for being a banana, but the image would not resemble much (Yosinski et al., 2015 [14]). This shortcoming can be addressed by imposing a so called *prior* constraint that the image should have similar statistics to natural images. In the case of Google's visualization, this means that the neighbouring pixels of the output image should be correlated. For some other examples of this visualization technique, see *Figure 5*.

One of the reasons why it is relevant to see what a network considers a typical representation of a class is that we need a way to verify whether or not a network learned what we intended to teach it. During training, we show a network thousands of examples of objects from different classes, hoping it will extract their essence and look for it in the pictures it is supposed to classify. The method by (Mordvintsev et al., 2015 [2]) helps answer the question whether or not the network learned the correct features during training.
*Figure 6* shows a network's typical representations of a dumbbell. Note that none of the pictures show dumbbells not being held by a human arm. The network failed, in this case, to extract the essence of what a dumbbell is supposed to look like. The fault could be with the training set, maybe none of the pictures presented to the network during training contained dumbbells without an arm holding them. Training mistakes like these can be uncovered using visualization.

- Another way to visualize networks proposed by (Mordvintsev et al., 2015 [2]) is to start out with a regular input image (that is, a photograph instead of random noise) and to gradually enhance whatever features were detected. For this purpose, a layer is picked and the image features that caused the strongest activations at this layer are enhanced in the input image. As mentioned earlier, higher layers tend to detect more complex features, lower layers simple features like strokes and edges. A picture where such low level features have been highlighted is shown in *Figure 7*.
Enhancing the input image according to higher layers' activations results in whole objects emerging in the input image. This works especially well due to the iterative nature of the process: We enhance whatever features the network layer detected, then rerun the detection, causing the network layer to recognize these enhanced features even more strongly. This process is repeated until clearly visible objects emerge in the input image, seemingly out of nowhere. *Figure 8* shows a simple input image (a sky) and then the same image, only with the therein recognized features enhanced. In *Figure 9*, which contains zoomed in portions of the image from *Figure 8*, we can clearly see the complex higher layer patterns (in this case, animals) that were recognized in the image.

We could now apply the feature enhancing iteratively on its own outputs, followed by some zooming after each iteration. This way, we get an endless stream of new impressions, going recursively deeper and deeper. We can start this process even from a random-noise image, so that the result becomes purely the result of the neural network, as can be seen in *Figure 10*.

*Runtime:* In practice, enhancing even higher level features is quite fast - using an online tool such as the one at http://deepdreamgenerator.com/,
highlighting the features in an $1100 \times 622$ pixel image took 15 seconds.

# 7. MAHENDRAN AND VEDALDI'S METHOD

The method by (Mahendran and Vedaldi, 2014 [4]) focuses on answering questions about the network invariances. When training a classifier, we want it to learn to focus on recognizing aspects of objects that always stay the same (a person will usually have two eyes), and ignoring aspects that often vary (a field of grass can be a darker or a lighter shade of green, depending on illumination).

Like with Zeiler and Fergus, their method starts out with a natural image $x_0$. They send this image through the network and obtain its representation $\phi(x_0)$. Then they reconstruct an image that produces a similar representation. This method provides insight into the activation of a whole layer rather than individual neurons.
The reconstruction is, in its essence, a regularized regression problem. Given a representation function $\phi : \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}$ ($H$ - height, $W$ - width, $C$ - channel) and representation $\phi_0 = \phi(x_0)$ we want to invert, the reconstruction will try to find the image $x$ that minimizes

$$x^* = argmin\ loss(\phi(x), \phi_0) + \lambda R(x)$$

where *loss* compares the image representation $\phi(x)$ to the target representation $\phi_0$ and $R$ is the regularizer capturing a natural image prior (meaning that it ensures the produced image has some of the statistical qualities of natural images to make it more recognizable to humans).
Minimizing $x^*$ results in an image $x^*$ that resembles $x_0$ (from the view of the representation). There is usually more than one solution to this problem (which makes sense, as several different images should be classified as a "cat", not only one). Sampling the space of possible reconstructions can reveal these. For the different results such a reconstruction can yield, see *Figure 11*.

The search for optimal solutions for this problem is inspired by neural network learning algorithms and uses gradient descent with

$$E(x) = loss(\phi(x), \phi_0) + \lambda R(x)$$

as its objective function.
Mahendran and Vedaldi show that several layers in convolutional neural networks retain photographically accurate information about the input image, with different degrees of geometric and photometric invariance. However, especially the higher layers capture progressively deformed form of the objects (see *Figure 12*). The last layer, for example, inverts back to multiple copies of the object/parts at different positions and scales. Note that this higher network layer only seems to capture a sketch of objects, which, evidently, suffices for classification.

# 8. CONCLUSION

## 8.1 Summary

In this paper, we have presented a short overview of the state-of-the art neural network visualization techniques. These differ both in the way they handle the visualization problem as well as in their understanding of the specific objective. The visualization objectives range from showing which features a network saw in a picture over which image patches were most relevant for the classification to what an ideal representation of an object would be, according to a network.

## 8.2 Remarks

The wide range of the presented visualization objectives suggests that the correct visualization technique should be chosen according to the specific needs of whoever creates and uses a network for a particular purpose.
All of these methods together provide the possibility to look into networks in a way that enables us to comprehend what goes on inside these networks not only on a numerical, but also on a semantic level. This could be extremely helpful when deciding in what way to change a network architecture in order to improve a network's classification scores. Instead of blindly trying or guessing whether or not to put the pooling layers at certain positions, visualizing might help us make these decisions faster.

## 8.3 Future Research

Despite the diversity of the described methods, one thing that all of the publications studied in this paper seem to be lacking is describing the relationship between the results of the visualization and the classification scores. It is obvious now that trained networks retain some information about how the input images looked like and can reproduce them, to a degree. But how does the ability to reproduce the pictures correlate with the ability to classify them correctly? In a certain layer, how much variability in the extracted features is optimal, and at which point becomes the loss of detail in a layer's representation of a class a problem? When is it beneficial? We hope to see some publications in the future that attempt to answer these questions, as well.

# 9. REFERENCES


[1] Matthew D. Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. Dept. of Computer Science, Courant Institute, New York University. 2013.

[2] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. *Inceptionism: Going Deeper into Neural Networks*. URL: http://googleresearch.blogspot.de/2015/06/inceptionism-going-deeper-into-neural.html.

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style". In: *CoRR* abs/1508.06576 (2015). URL: http://arxiv.org/abs/1508.06576.

[4] Aravindh Mahendran and Andrea Vedaldi. "Understanding Deep Image Representations by Inverting Them". In: *CoRR* abs/1412.0035 (2014). URL: http://arxiv.org/abs/1412.0035.

[5] Bogdan M. Wilamowski. "Neural Network Architectures and Learning". In: vol. 1. IEEE, 2003.

[6] Yann LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–48.

[7] Andrej Karpathy. *Convolutional Neural Networks (CNNs / ConvNets)*. URL: http://cs231n.github.io/convolutional-networks/.

[8] Dumitru Erhan et al. *Visualizing higher-layer features of a deep network*. Technical report, University of Montreal, 2009.

[9] Jason Yosinski et al. "Understanding Neural Networks Through Deep Visualization". In: *Deep Learning Workshop, International Conference on Machine Learning (ICML)*. 2015.

[10] Le et al. "Tiled Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. 2010.

[11] Donahue et al. *DeCAF: A deep convolutional activation feature for generic visual recognition*. 2013. URL: http://arxiv.org/abs/1310.1531.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. NIPS, 2012.

[13] Jason Yosinski. *Understanding Neural Networks Through Deep Visualization*. URL: http://yosinski.com/deepvis.

[14] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images". In: *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.


All links were last followed on December 17, 2015.

# 10. APPENDIX

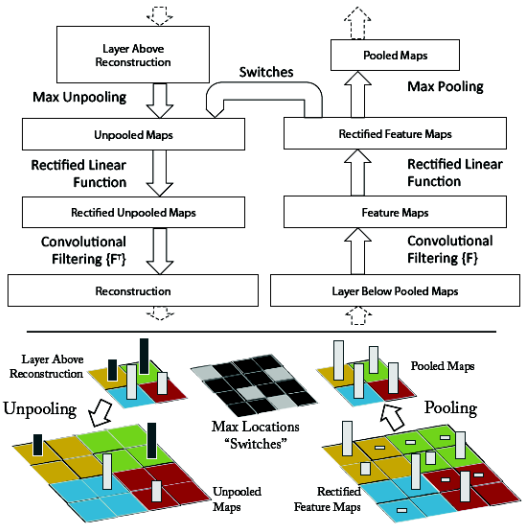


**Figure 1:** *Convolution and Deconvolution (Zeiler, 2013 [1]). Top: A deconvnet layer (left) attached to a convnet layer (right). The deconvnet will reconstruct an approximate version of the convnet features from the layer beneath. Bottom: An illustration of the unpooling operation in the deconvnet, using switches which record the location of the local max in each pooling region (coloured zones) during pooling in the convnet.*


**Figure 2:** *Visualization of features in a fully trained model with the deconvolution method. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are not samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. (Zeiler, 2013 [1]).*

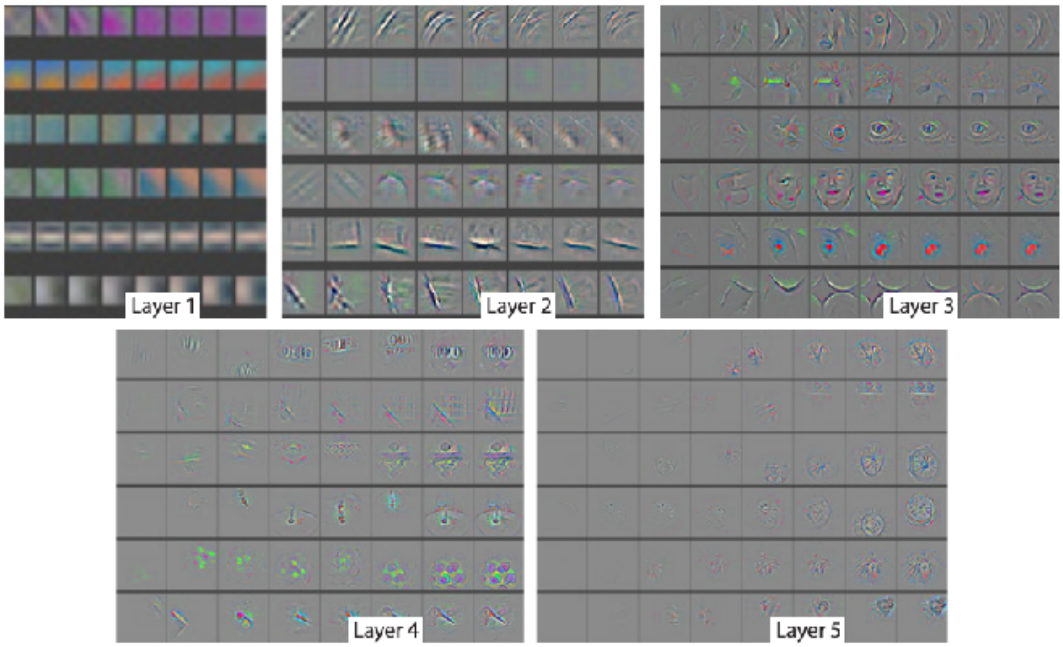


**Figure 3:** *Evolution of a randomly chosen subset of model features through training. Each layer's features are displayed in a different block. Within each block, we show a randomly chosen subset of features at epochs [1,2,5,10,20,30,40,64]. The visualization shows the strongest activation (across all training examples) for a given feature map, projected down to pixel space using our deconvnet approach. Color contrast is artificially enhanced and the figure is best viewed in electronic form. (Zeiler, 2013 [1]).*

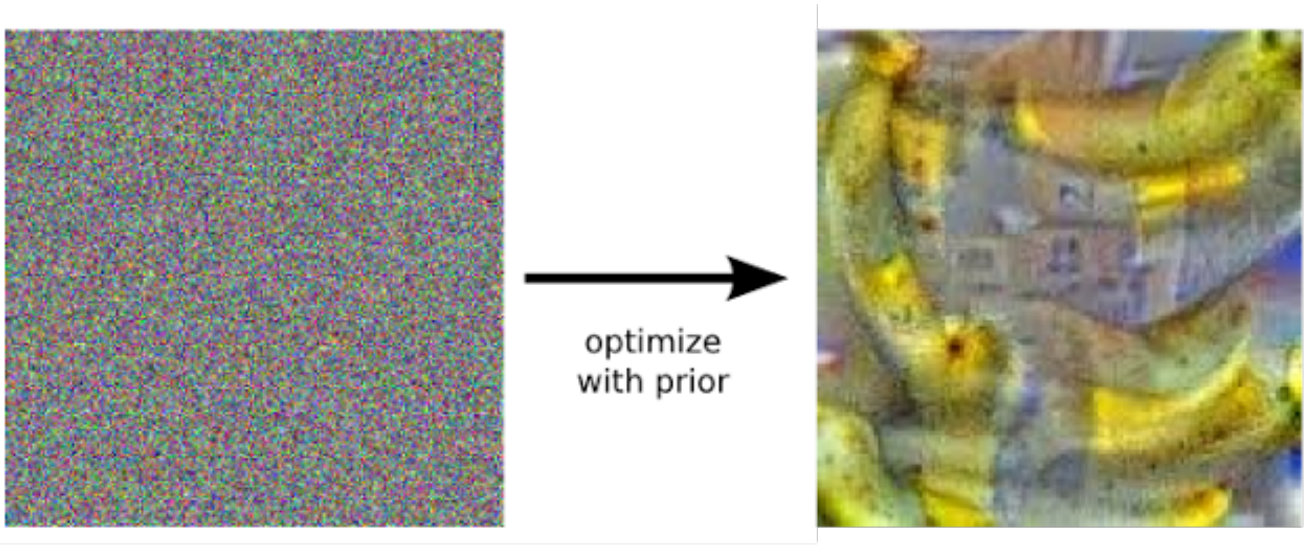


**Figure 4:** *A network produces the representation of a typical banana gradually out of an input picture of random noise (Mordvintsev et al., 2015 [2]).*

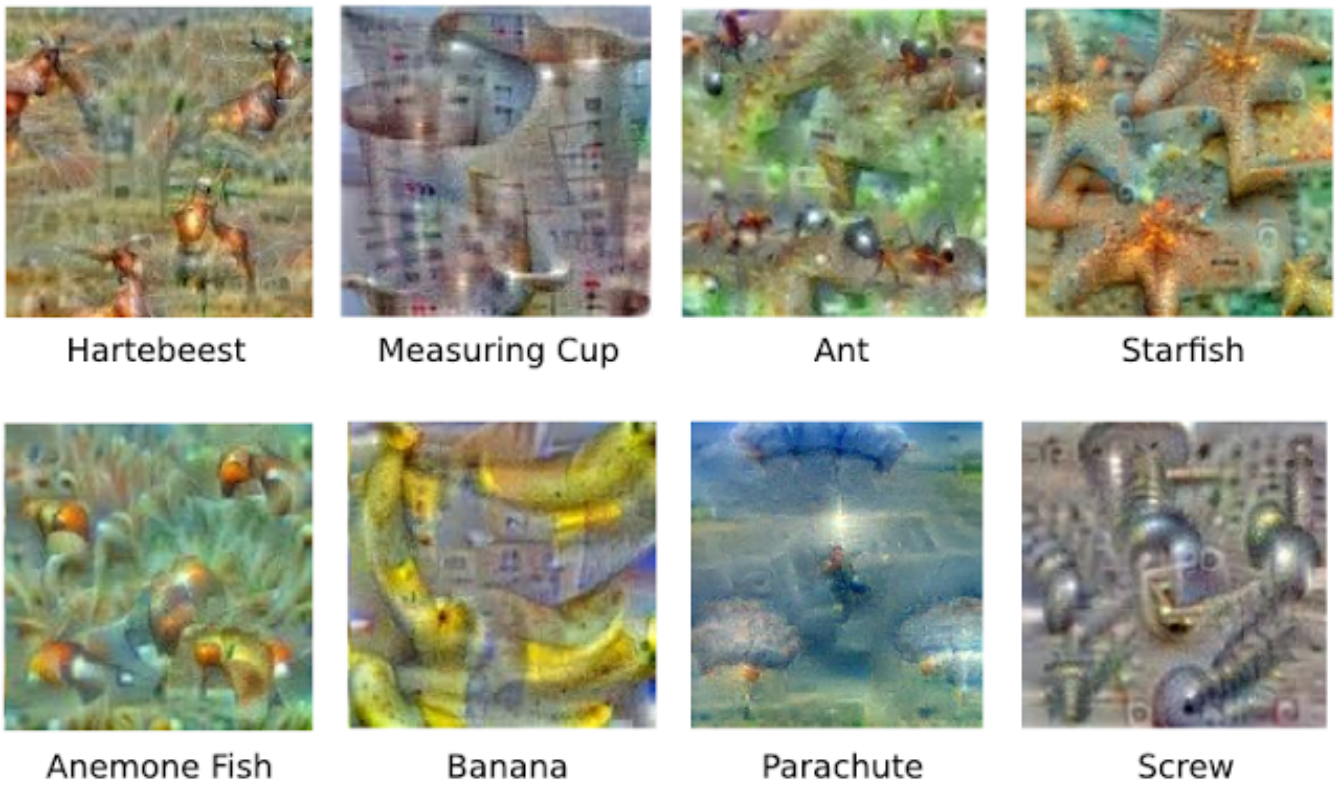


**Figure 5:** *Visual representation of some other selected image classes (Mordvintsev et al., 2015 [2]).*

**Figure 6:** *Some typical representations of a dumbbell (Mordvintsev et al., 2015 [2]).*

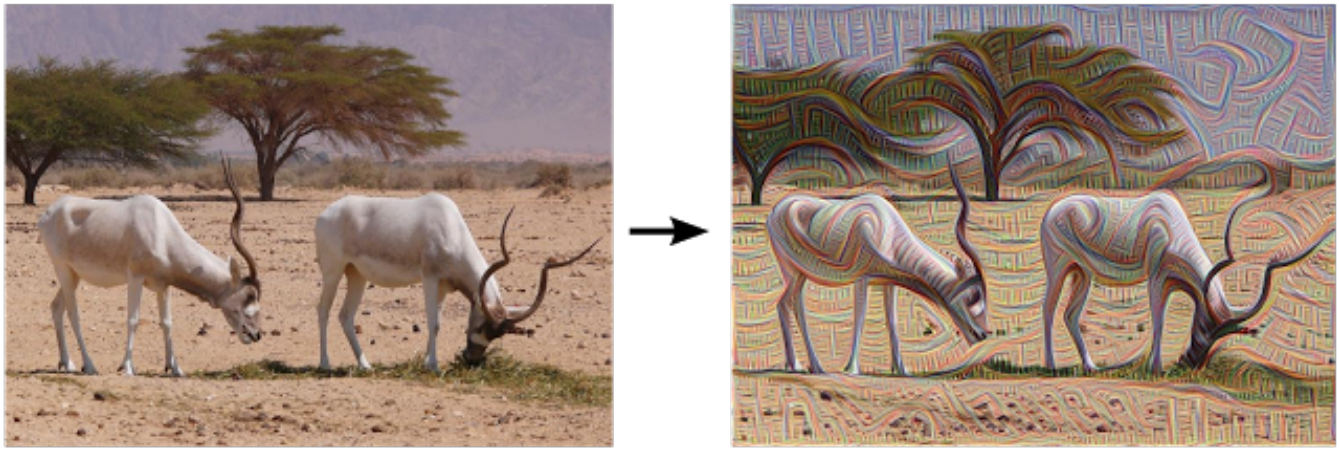

**Figure 7:** *Left: Original photo by Zachi Evenor. Right: Processed by Günther Noack, Software Engineer (a picture with highlighted low level features). (Mordvintsev et al., 2015 [2]).*

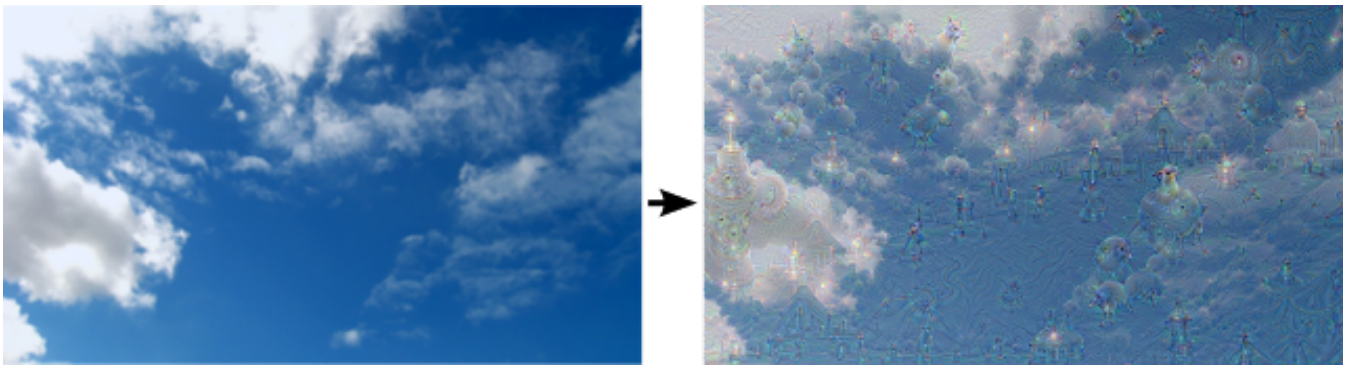

**Figure 8:** *Left: Photo of a sky. Right: Processed photo with enhanced higher layer features (Mordvintsev et al., 2015 [2]).*


**Figure 9:** *Close-ups of detected and enhanced higher layer features from Figure 8 (Mordvintsev et al., 2015 [2]).*

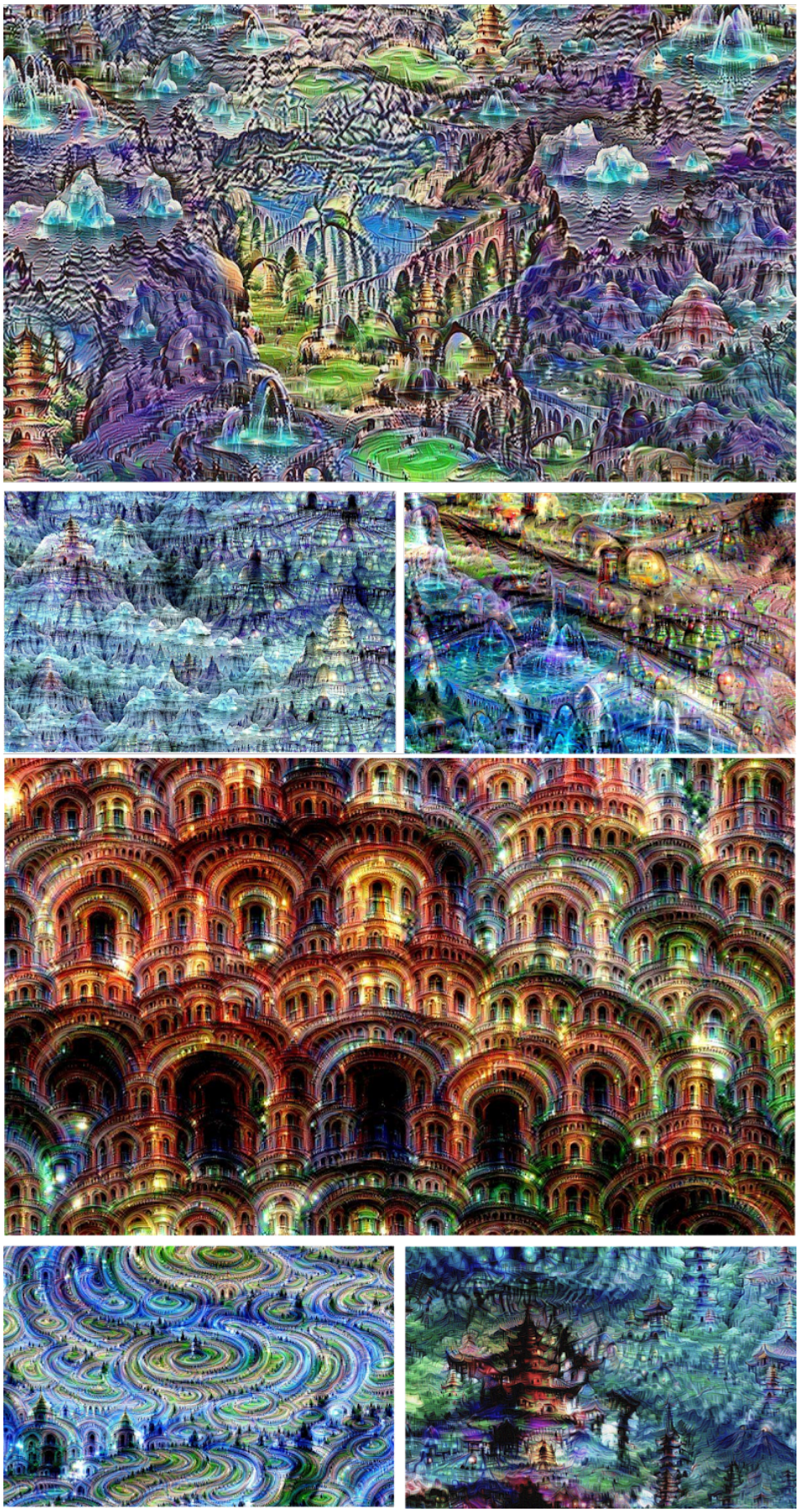

**Figure 10:** *Iterative feature enhancing combined with zooming after each iteration (Mordvintsev et al., 2015 [2]).*

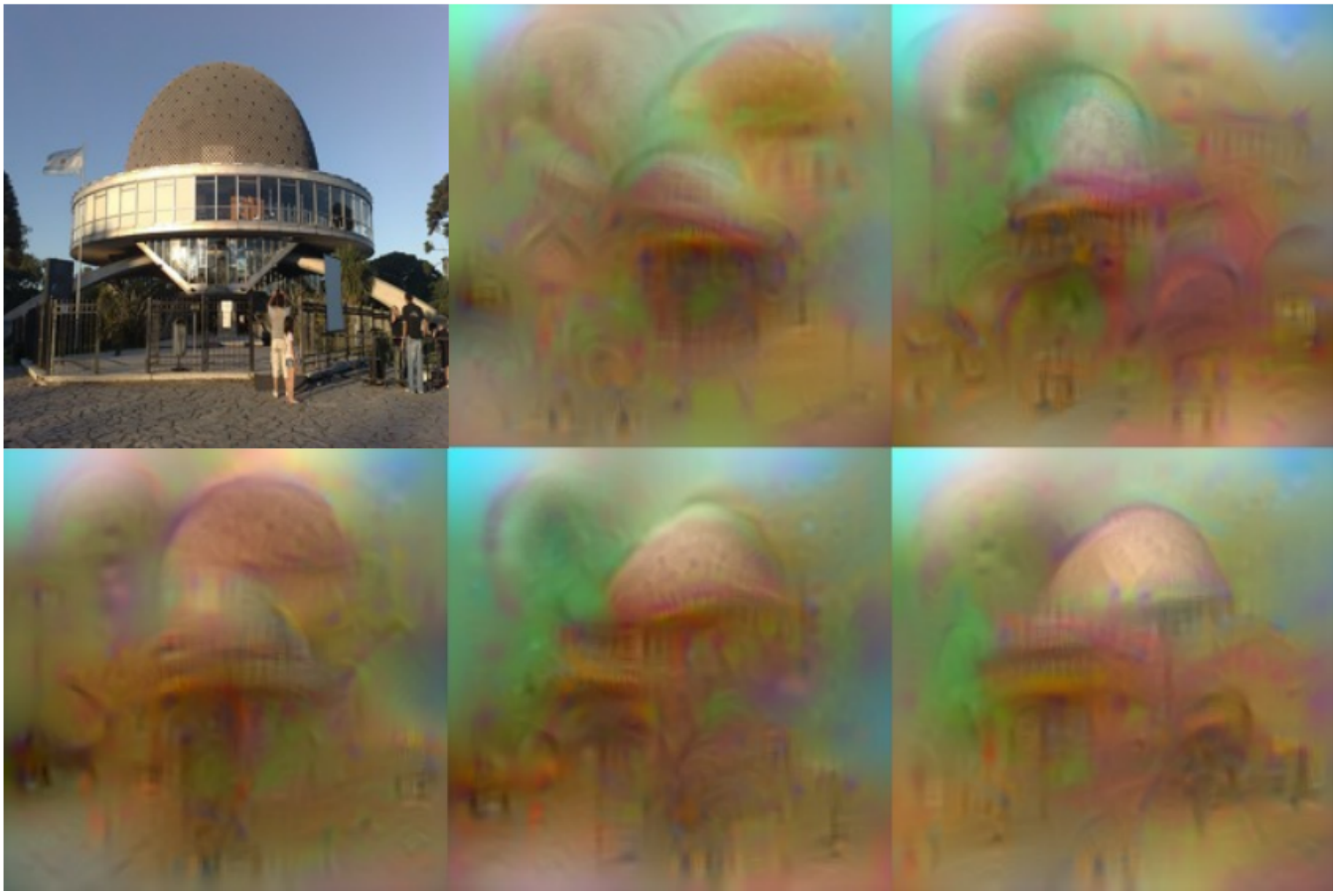

**Figure 11:** *What is encoded by a CNN? The figure shows five possible reconstructions of the reference image obtained from the 1,000-dimensional code extracted at the penultimate layer of a reference convolutional network (before the softmax classifier in the output layer is applied) trained on the ImageNet data. From the viewpoint of the model, all these images are practically equivalent (Mahendran and Vedaldi, 2014 [4]).*


**Figure 12:** *This figure examines the invariance captured by the CNN model by considering multiple reconstructions out of each deep layer. A careful examination of these images reveals that the codes capture progressively larger deformations of the object. In particular, fc8 inverts back to multiple copies of the object/parts at different positions and scales. (Mahendran and Vedaldi, 2014 [4]).*