Understanding Neural Networks Through Deep Visualization

Jason Yosinski YOSINSKI@CS.CORNELL.EDU
Cornell University

Jeff Clune JEFFCLUNE@UWYO.EDU
Anh Nguyen ANGUYEN8@UWYO.EDU
University of Wyoming

Thomas Fuchs FUCHS@CALTECH.EDU
Jet Propulsion Laboratory, California Institute of Technology

Hod Lipson HOD.LIPSON@CORNELL.EDU
Cornell University

Abstract

Recent years have produced great advances in training large, deep neural networks (DNNs), including notable successes in training convolutional neural networks (convnets) to recognize natural images. However, our understanding of how these models work, especially what computations they perform at intermediate layers, has lagged behind. Progress in the field will be further accelerated by the development of better tools for visualizing and interpreting neural nets. We introduce two such tools here. The first is a tool that visualizes the activations produced on each layer of a trained convnet as it processes an image or video (e.g. a live webcam stream). We have found that looking at live activations that change in response to user input helps build valuable intuitions about how convnets work. The second tool enables visualizing features at each layer of a DNN via regularized optimization in image space. Because previous versions of this idea produced less recognizable images, here we introduce several new regularization methods that combine to produce qualitatively clearer, more interpretable visualizations. Both tools are open source and work on a pre-trained convnet with minimal setup.

Published in the *Deep Learning Workshop, 31st International Conference on Machine Learning*, Lille, France, 2015. Copyright 2015 by the author(s).

1. Introduction

The last several years have produced tremendous progress in training powerful, deep neural network models that are approaching and even surpassing human abilities on a variety of challenging machine learning tasks (Taigman et al., 2014; Schroff et al., 2015; Hannun et al., 2014). A flagship example is training deep, convolutional neural networks (CNNs) with supervised learning to classify natural images (Krizhevsky et al., 2012). That area has benefitted from the combined effects of faster computing (e.g. GPUs), better training techniques (e.g. dropout (Hinton et al., 2012)), better activation units (e.g. rectified linear units (Glorot et al., 2011)), and larger labeled datasets (Deng et al., 2009; Lin et al., 2014).

While there has thus been considerable improvements in our knowledge of how to create high-performing architectures and learning algorithms, our understanding of how these large neural models operate has lagged behind. Neural networks have long been known as "black boxes" because it is difficult to understand exactly how any particular, trained neural network functions due to the large number of interacting, non-linear parts. Large modern neural networks are even harder to study because of their size; for example, understanding the widely-used AlexNet DNN involves making sense of the values taken by the 60 million trained network parameters. Understanding what is learned is interesting in its own right, but it is also one key way of further improving models: the intuitions provided by understanding the current generation of models should suggest ways to make them better. For example, the deconvolutional technique for visualizing the features learned by the hidden units of DNNs suggested an architectural change of smaller convolutional filters that led to state of the art performance on the ImageNet benchmark in 2013 (Zeiler & Fergus, 2013).

We also note that tools that enable understanding will especially benefit the vast numbers of newcomers to deep learning, who would like to take advantage of off-the-shelf software packages — like Theano (Bergstra et al., 2010), Pylearn2 (Goodfellow et al., 2013), Caffe (Jia et al., 2014), and Torch (Collobert et al., 2011) — in new domains, but who may not have any intuition for why their models work (or do not). Experts can also benefit as they iterate ideas for new models or when they are searching for good hyperparameters. We thus believe that both experts and newcomers will benefit from tools that provide intuitions about the inner workings of DNNs. This paper provides two such tools, both of which are open source so that scientists and practitioners can integrate them with their own DNNs to better understand them.

The first tool is software that interactively plots the activations produced on each layer of a trained DNN for user-provided images or video. Static images afford a slow, detailed investigation of a particular input, whereas video input highlights the DNNs changing responses to dynamic input. At present, the videos are processed live from a user's computer camera, which is especially helpful because users can move different items around the field of view, occlude and combine them, and perform other manipulations to actively learn how different features in the network respond.

The second tool we introduce enables better visualization of the learned features computed by individual neurons at every layer of a DNN. Seeing what features have been learned is important both to understand how current DNNs work and to fuel intuitions for how to improve them.

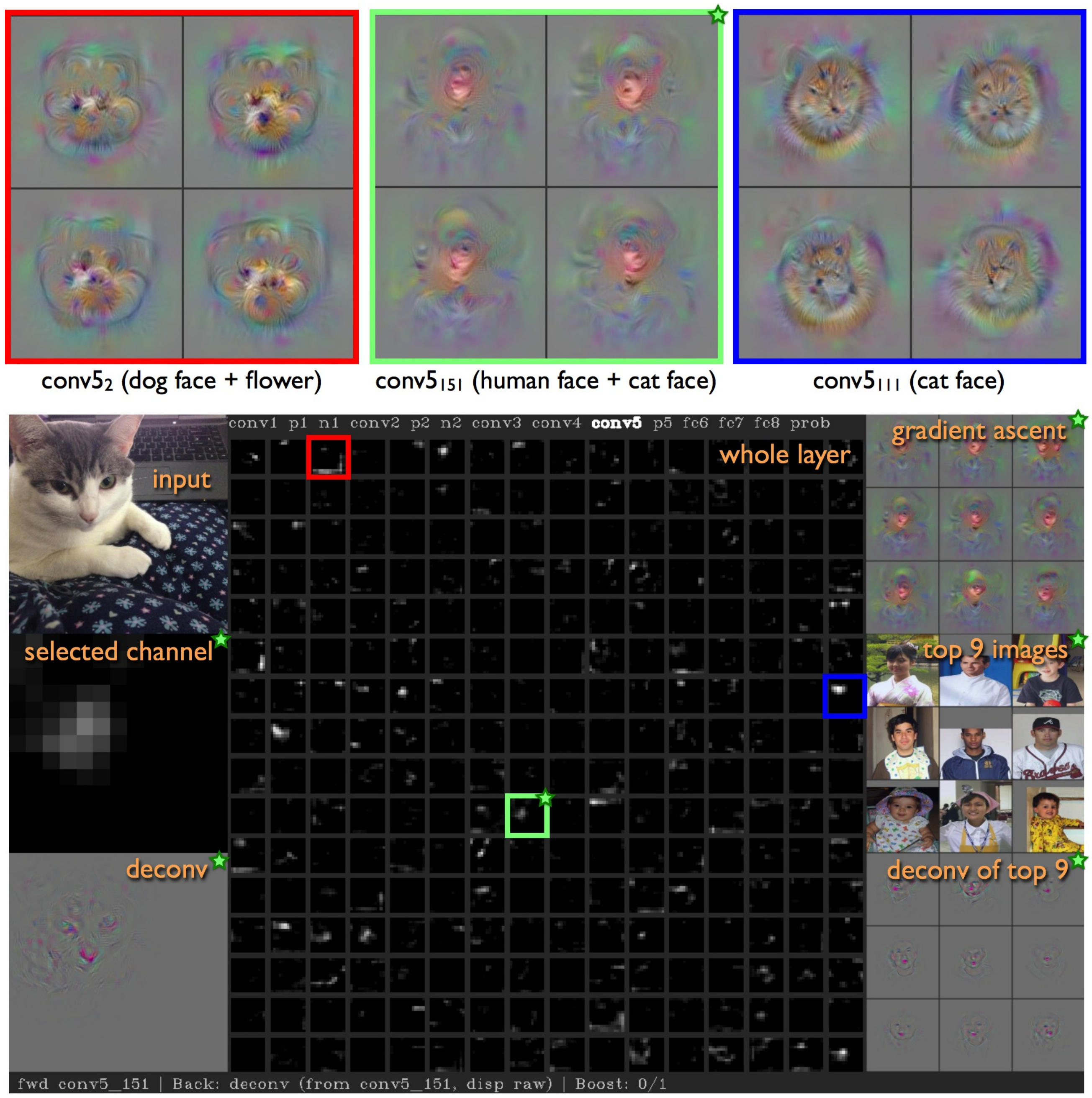
Attempting to understand what computations are performed at each layer in DNNs is an increasingly popular direction of research. One approach is to study each layer as a group and investigate the type of computation performed by the set of neurons on a layer as a whole (Yosinski et al., 2014; Mahendran & Vedaldi, 2014). This approach is informative because the neurons in a layer interact with each other to pass information to higher layers, and thus each neuron's contribution to the entire function performed by the DNN depends on that neuron's context in the layer.

Another approach is to try to interpret the function computed by each individual neuron. Past studies in this vein roughly divide into two different camps: *dataset-centric* and *network-centric*. The former requires both a trained DNN and running data through that network; the latter requires only the trained network itself. One dataset-centric approach is to display images from the training or test set that cause high or low activations for individual units. Another is the deconvolution method of Zeiler & Fergus (2013), which highlights the portions of a particular image that are responsible for the firing of each neural unit.

Network-centric approaches investigate a network directly without any data from a dataset. For example, Erhan et al. (2009) synthesized images that cause high activations for particular units. Starting with some initial input $\mathbf{x} = \mathbf{x_0}$, the activation $a_i(\mathbf{x})$ caused at some unit i by this input is computed, and then steps are taken in input space along the gradient $\partial a_i(\mathbf{x})/\partial \mathbf{x}$ to synthesize inputs that cause higher and higher activations of unit i, eventually terminating at some $\mathbf{x}^*$ which is deemed to be a preferred input stimulus for the unit in question. In the case where the input space is an image, $\mathbf{x}^*$ can be displayed directly for interpretation. Others have followed suit, using the gradient to find images that cause higher activations (Simonyan et al., 2013; Nguyen et al., 2014) or lower activations (Szegedy et al., 2013) for output units.

These gradient-based approaches are attractive in their simplicity, but the optimization process tends to produce images that do not greatly resemble natural images. Instead, they are composed of a collection of "hacks" that happen to cause high (or low) activations: extreme pixel values, structured high frequency patterns, and copies of common motifs without global structure (Simonyan et al., 2013; Nguyen et al., 2014; Szegedy et al., 2013; Goodfellow et al., 2014). The fact that activations may be effected by such hacks is better understood thanks to several recent studies. Specifically, it has been shown that such hacks may be applied to correctly classified images to cause them to be misclassified even via imperceptibly small changes (Szegedy et al., 2013), that such hacks can be found even without the gradient information to produce unrecognizable "fooling examples" (Nguyen et al., 2014), and that the abundance of non-natural looking images that cause extreme activations can be explained by the locally linear behavior of neural nets (Goodfellow et al., 2014).

With such strong evidence that optimizing images to cause high activations produces unrecognizable images, is there any hope of using such methods to obtain useful visualizations? It turns out there is, if one is able to appropriately regularize the optimization. Simonyan et al. (2013) showed that slightly discernible images for the final layers of a convnet could be produced with L_2-regularization. Mahendran and Vedaldi (2014) also showed the importance of incorporating natural-image priors in the optimization process when producing images that mimic an entire-layer's firing pattern produced by a specific input image. We build on these works and contribute three additional forms of regularization that, when combined, produce more recognizable, optimization-based samples than previous methods. Because the optimization is stochastic, by starting at different random initial images, we can produce a set of opti-


conv5₂ (dog face + flower)
conv5₁₅₁ (human face + cat face)
conv5₁₁₁ (cat face)
conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
input
whole layer
gradient ascent
selected channel
top 9 images
deconv
deconv of top 9

```
fwd conv5_151 | Back: deconv (from conv5_151, disp raw) | Boost: 0/1
```

Figure 1. The **bottom** shows a screenshot from the interactive visualization software. The webcam *input* is shown, along with the *whole layer* of conv5 activations. The *selected channel* pane shows an enlarged version of the 13x13 conv5_{151} channel activations. Below it, the *deconv* starting at the selected channel is shown. On the right, three selections of nine images are shown: synthetic images produced using the regularized *gradient ascent* methods described in Section 3, the *top 9 image* patches from the training set (the images from the training set that caused the highest activations for the selected channel), and the *deconv of the those top 9* images. All areas highlighted with a green star relate to the particular selected channel, here conv5_{151}; when the selection changes, these panels update. The **top** depicts enlarged numerical optimization results for this and other channels. conv5_{2} is a channel that responds most strongly to dog faces (as evidenced by the top nine images, which are not shown due to space constraints), but it also responds to flowers on the blanket on the bottom and half way up the right side of the image (as seen in the inset red highlight). This response to flowers can be partially seen in the optimized images but would be missed in an analysis focusing only on the top nine images and their deconv versions, which contain no flowers. conv5_{151} detects different types of faces. The top nine images are all of human faces, but here we see it responds also to the cat's face (and in Figure 2 a lion's face). Finally, conv5_{111} activates strongly for the cat's face, the optimized images show catlike fur and ears, and the top nine images (not shown here) are also all of cats. For this image, the softmax output layer top two predictions are "Egyptian Cat" and "Computer Keyboard." All figures in this paper are best viewed digitally, in color, significantly zoomed in.

mized images whose variance provides information about the invariances learned by the unit.

To summarize, this paper makes the following two contributions:

1. We describe and release a software tool that provides a live, interactive visualization of every neuron in a trained convnet as it responds to a user-provided image or video. The tool displays forward activation values, preferred stimuli via gradient ascent, top images for each unit from the training set, deconv highlighting (Zeiler & Fergus, 2013) of top images, and backward diffs computed via backprop or deconv starting from arbitrary units. The combined effect of these complementary visualizations promotes a greater understanding of what a neuron computes than any single method on its own. We also describe a few insights we have gained from using this tool. (Section 2).

2. We extend past efforts to visualize preferred activation patterns in input space by adding several new types of regularization, which produce what we believe are the most interpretable images for large convnets so far (Section 3).

Both of our tools are released as open source and are available at `http://yosinski.com/deepvis`. While the tools could be adapted to integrate with any DNN software framework, they work out of the box with the popular Caffe DNN software package (Jia et al., 2014). Users may run visualizations with their own Caffe DNN or our pre-trained DNN, which comes with pre-computed images optimized to activate each neuron in this trained network. Our pre-trained network is nearly identical to the "AlexNet" architecture (Krizhevsky et al., 2012), but with local reponse normalization layers after pooling layers following (Jia et al., 2014). It was trained with the Caffe framework on the ImageNet 2012 dataset (Deng et al., 2009).

2. Visualizing Live Convnet Activations

Our first visualization method is straightforward: plotting the activation values for the neurons in each layer of a convnet in response to an image or video. In fully connected neural networks, the order of the units is irrelevant, so plots of these vectors are not spatially informative. However, in convolutional networks, filters are applied in a way that respects the underlying geometry of the input; in the case of 2D images, filters are applied in a 2D convolution over the two spatial dimensions of the image. This convolution produces activations on subsequent layers that are, for each channel, also arranged spatially.

Figure 1 shows examples of this type of plot for the conv5 layer. The conv5 layer has size $256\times13\times13$, which we depict as 256 separate 13×13 grayscale images. Each of the 256 small images contains activations in the same spatial x-y spatial layout as the input data, and the 256 images are simply and arbitrarily tiled into a 16×16 grid in row-major order. Figure 2 shows a zoomed in view of one particular channel, conv5_{151}, that responds to human and animal faces. All layers can be viewed in the software tool, including pooling and normalization layers. Visualizing these layers provides intuitions about their effects and functions.

Although this visualization is simple to implement, we find it informative because all data flowing through the network can be visualized. There is nothing mysterious happening behind the scenes. Because this convnet contains only a single path from input to output, every layer is a bottleneck through which all information must pass en-route to a classification decision. The layer sizes are all small enough that any one layer can easily fit on a computer screen.[1] So far, we have gleaned several surprising intuitions from using the tool:

- One of the most interesting conclusions so far has been that representations on some layers seem to be surprisingly local. Instead of finding distributed representations on all layers, we see, for example, detectors for text, flowers, fruit, and faces on conv4 and conv5. These conclusions can be drawn either from the live visualization or the optimized images (or, best, by using both in concert) and suggest several directions for future research (discussed in Section 4).

- When using direct file input to classify photos from Flickr or Google Images, classifications are often correct and highly confident (softmax probability for correct class near 1). On the other hand, when using input from a webcam, predictions often cannot be correct because no items from the training set are shown in the image. The training set's 1000 classes, though numerous, do not cover most common household objects. Thus, when shown a typical webcam view of a person with no ImageNet classes present, the output has no single high probability, as is expected. Surprisingly, however, this probability vector is noisy and varies significantly in response to tiny changes in the input, often changing merely in response to the noise from the webcam. We might have instead expected unchanging and low confidence predictions for a given scene when no object the network has been trained to classify is present. Plotting the fully connected layers (fc6 and fc7) also reveals a similar sensitivity to small input changes.

[1]The layer with the most activations is conv1 which, when tiled, is only 550x550 before adding padding.

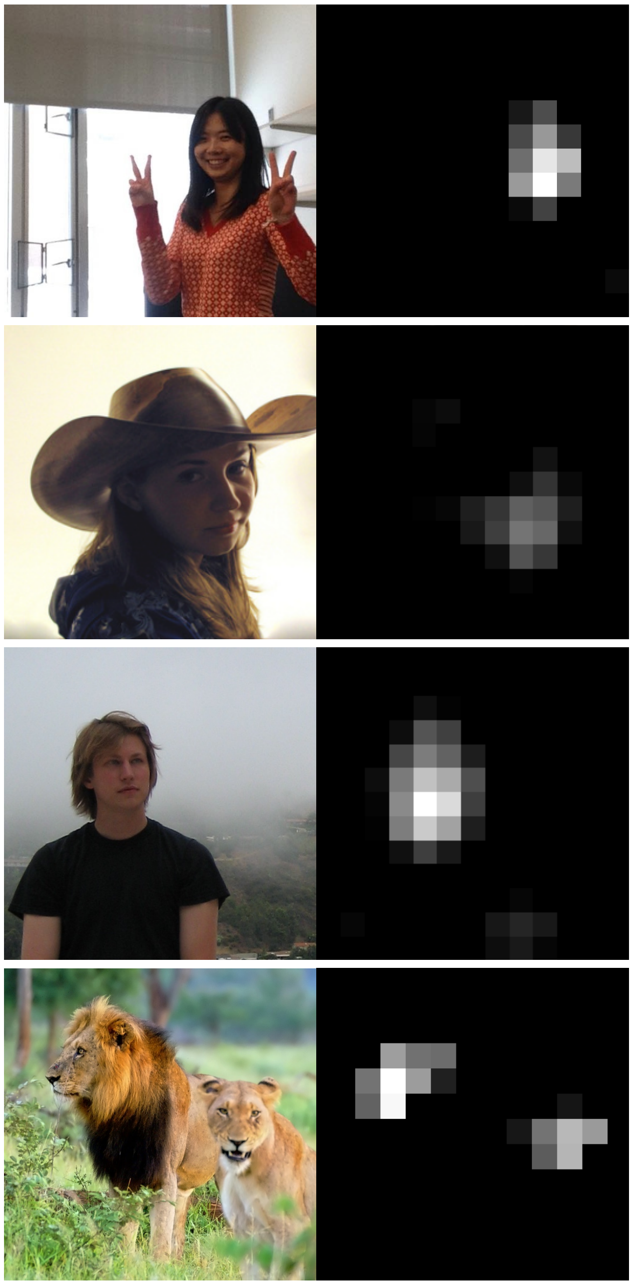


Figure 2. A view of the 13×13 activations of the 151st channel on the conv5 layer of a deep neural network trained on ImageNet, a dataset that does not contain a face class, but does contain many images with faces. The channel responds to human and animal faces and is robust to changes in scale, pose, lighting, and context, which can be discerned by a user by actively changing the scene in front of a webcam or by loading static images (e.g. of the lions) and seeing the corresponding response of the unit. Photo of lions via Flickr user arnolouise, licensed under CC BY-NC-SA 2.0.

- Although the last three layers are sensitive to small input changes, much of the lower layer computation is more robust. For example, when visualizing the conv5 layer, one can find many invariant detectors for faces, shoulders, text, etc. by moving oneself or objects in front of the camera. Even though the 1000 classes contain no explicitly labeled faces or text, the network learns to identify these concepts simply because they represent useful partial information for making a later classification decision. One face detector, denoted conv5_{151} (channel number 151 on conv5), is shown in Figure 2 activating for human and lion faces and in Figure 1 activating for a cat face. Zhou et al. (2014) recently observed a similar effect where convnets trained only to recognize different scene types — playgrounds, restaurant patios, living rooms, etc. — learn object detectors (e.g. for chairs, books, and sofas) on intermediate layers.

The reader is encouraged to try this visualization tool out for him or herself. The code, together with pre-trained models and images synthesized by gradient ascent, can be downloaded at `http://yosinski.com/deepvis`.

3. Visualizing via Regularized Optimization

The second contribution of this work is introducing several regularization methods to bias images found via optimization toward more visually interpretable examples. While each of these regularization methods helps on its own, in combination they are even more effective. We found useful combinations via a random hyperparameter search, as discussed below.

Formally, consider an image $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$, where $C = 3$ color channels and the height (H) and width (W) are both 227 pixels. When this image is presented to a neural network, it causes an activation $a_i(\mathbf{x})$ for some unit i, where for simplicity i is an index that runs over all units on all layers. We also define a parameterized regularization function $R_\theta(\mathbf{x})$ that penalizes images in various ways.

Our network was trained on ImageNet by first subtracting the per-pixel mean of examples in ImageNet before inputting training examples to the network. Thus, the direct input to the network, $\mathbf{x}$, can be thought of as a zero-centered input. We may pose the optimization problem as finding an image $\mathbf{x}^*$ where

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} (a_i(\mathbf{x}) - R_\theta(\mathbf{x})) \tag{1}$$

In practice, we use a slightly different formulation. Because we search for $\mathbf{x}^*$ by starting at some $\mathbf{x_0}$ and taking gradient steps, we instead define the regularization via an operator $r_\theta(\cdot)$ that maps $\mathbf{x}$ to a slightly more regularized version of itself. This latter definition is strictly more expressive, allowing regularization operators r_θ that are not the gradient of any R_θ. This method is easy to implement within a gradient descent framework by simply alternating between taking a step toward the gradient of $a_i(\mathbf{x})$ and taking a step in the direction given by r_θ. With a gradient descent step size of η, a single step in this process applies the update:

$$\mathbf{x} \leftarrow r_\theta \left(\mathbf{x} + \eta \frac{\partial a_i}{\partial \mathbf{x}} \right) \tag{2}$$

We investigated the following four regularizations. All are designed to overcome different pathologies commonly encountered by gradient descent without regularization.

L_2 **decay**: A common regularization, L_2 decay penalizes large values and is implemented as $r_\theta(\mathbf{x}) = (1-\theta_{\text{decay}}) \cdot \mathbf{x}$. L_2 decay tends to prevent a small number of extreme pixel values from dominating the example image. Such extreme single-pixel values neither occur naturally with great frequency nor are useful for visualization. L_2 decay was also used by Simonyan et al. (2013).

Gaussian blur: Producing images via gradient ascent tends to produce examples with high frequency information (see Supplementary Section S1 for a possible reason). While these images cause high activations, they are neither realistic nor interpretable (Nguyen et al., 2014). A useful regularization is thus to penalize high frequency information. We implement this as a Gaussian blur step $r_\theta(\mathbf{x}) = \text{GaussianBlur}(\mathbf{x}, \theta_{\text{b\_width}})$. Convolving with a blur kernel is more computationally expensive than the other regularization methods, so we added another hyperparameter $\theta_{\text{b\_every}}$ to allow, for example, blurring every several optimization steps instead of every step. Blurring an image multiple times with a small width Gaussian kernel is equivalent to blurring once with a larger width kernel, and the effect will be similar even if the image changes slightly during the optimization process. This technique thus lowers computational costs without limiting the expressiveness of the regularization. Mahendran & Vedaldi (2014) used a penalty with a similar effect to blurring, called *total variation*, in their work reconstructing images from layer codes.

Clipping pixels with small norm: The first two regularizations suppress high amplitude and high frequency information, so after applying both, we are left with an $\mathbf{x}^*$ that contains somewhat small, somewhat smooth values. However, $\mathbf{x}^*$ will still tend to contain non-zero pixel values everywhere. Even if some pixels in $\mathbf{x}^*$ show the primary object or type of input causing the unit under consideration to activate, the gradient with respect to all other pixels in $\mathbf{x}^*$ will still generally be non-zero, so these pixels will also shift to show some pattern as well, contributing in whatever small way they can to ultimately raise the chosen unit's activation. We wish to bias the search away from such behavior and instead show only the main object, letting other regions be exactly zero if they are not needed. We implement this bias using an $r_\theta(\mathbf{x})$ that computes the norm of each pixel (over red, green, and blue channels) and then sets any pixels with small norm to zero. The threshold for the norm, $\theta_{\text{n\_pct}}$, is specified as a percentile of all pixel norms in $\mathbf{x}$.

Clipping pixels with small contribution: Instead of clipping pixels with small norms, we can try something slightly smarter and clip pixels with small *contributions* to the activation. One way of computing a pixel's contribution to an activation is to measure how much the activation increases or decreases when the pixel is set to zero; that is, to compute the contribution as $|a_i(\mathbf{x}) - a_i(\mathbf{x}_{-j})|$, where $\mathbf{x}_{-j}$ is $\mathbf{x}$ but with the j^{th} pixel set to zero. This approach is straightforward but prohibitively slow, requiring a forward pass for every pixel. Instead, we approximate this process by linearizing $a_i(\mathbf{x})$ around $\mathbf{x}$, in which case the contribution of each dimension of $\mathbf{x}$ can be estimated as the elementwise product of $\mathbf{x}$ and the gradient. We then sum over all three channels and take the absolute value, computing $|\sum_c \mathbf{x} \circ \nabla_\mathbf{x} a_i(\mathbf{x})|$. We use the absolute value to find pixels with small contribution in either direction, positive or negative. While we could choose to keep the pixel transitions where setting the pixel to zero would result in a large activation increase, these shifts are already handled by gradient ascent, and here we prefer to clip only the pixels that are deemed not to matter, not to take large gradient steps outside the region where the linear approximation is most valid. We define this $r_\theta(\mathbf{x})$ as the operation that sets pixels with contribution under the $\theta_{\text{c\_pct}}$ percentile to zero.

If the above regularization methods are applied individually, they are somewhat effective at producing more interpretable images; Figure 3 shows the effects of each individual hyperparameter. However, preliminary experiments uncovered that their combined effect produces better visualizations. To pick a reasonable set of hyperparameters for all methods at once, we ran a random hyperparameter search of 300 possible combinations and settled on four that complement each other well. The four selected combinations are listed in Table 1 and optimized images using each are shown for the "Gorilla" class output unit in Figure 4. Of the four, some show high frequency information, others low frequency; some contain dense pixel data, and others contain only sparse outlines of important regions. We found the version in the lower-left quadrant to be the best single set of hyperparameters, but often greater intuition can be gleaned by considering all four at once. Figure 5 shows the optimization results computed for a selection of units on all layers. A single image for every filter of all five convolutional layers is shown in Supplementary Figure S1. Nine images for each filter of all layers, including each of the 1000 ImageNet output classes, can be viewed at `http://yosinski.com/deepvis`.

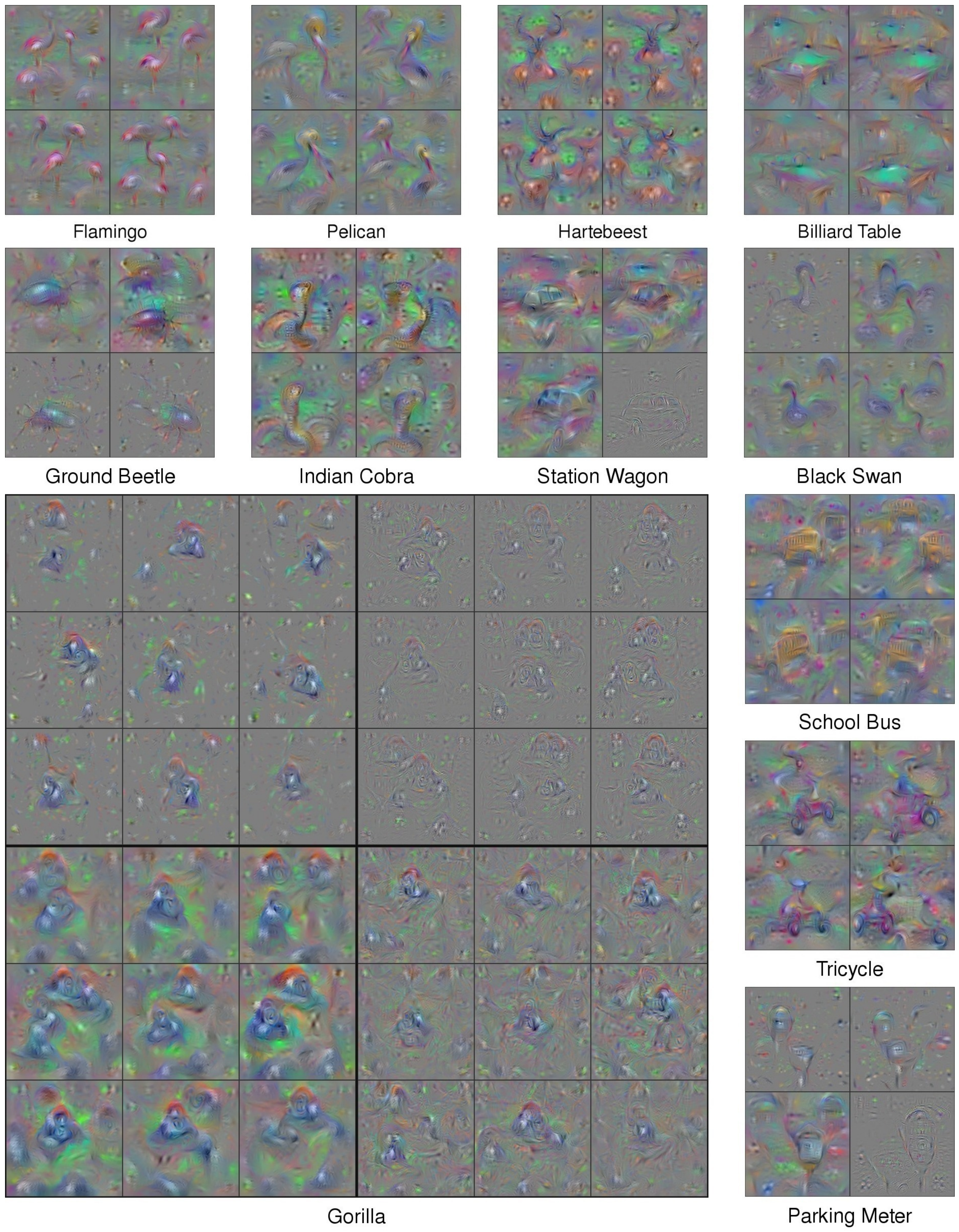


Figure 4. Visualizations of the preferred inputs for different class units on layer fc8, the 1000-dimensional output of the network just before the final softmax. In the lower left are 9 visualizations each (in 3×3 grids) for four different sets of regularization hyperparameters for the Gorilla class (Table 1). For all other classes, we have selected four interpretable visualizations produced by our regularized optimization method. We chose the four combinations of regularization hyperparameters by performing a random hyperparameter search and selecting combinations that complement each other. For example, the lower left quadrant tends to show lower frequency patterns, the upper right shows high frequency patterns, and the upper left shows a sparse set of important regions. Often greater intuition can be gleaned by considering all four at once. In nearly every case, we have found that one can guess what class a neuron represents by viewing sets of these optimized, preferred images. Best viewed electronically, zoomed in.

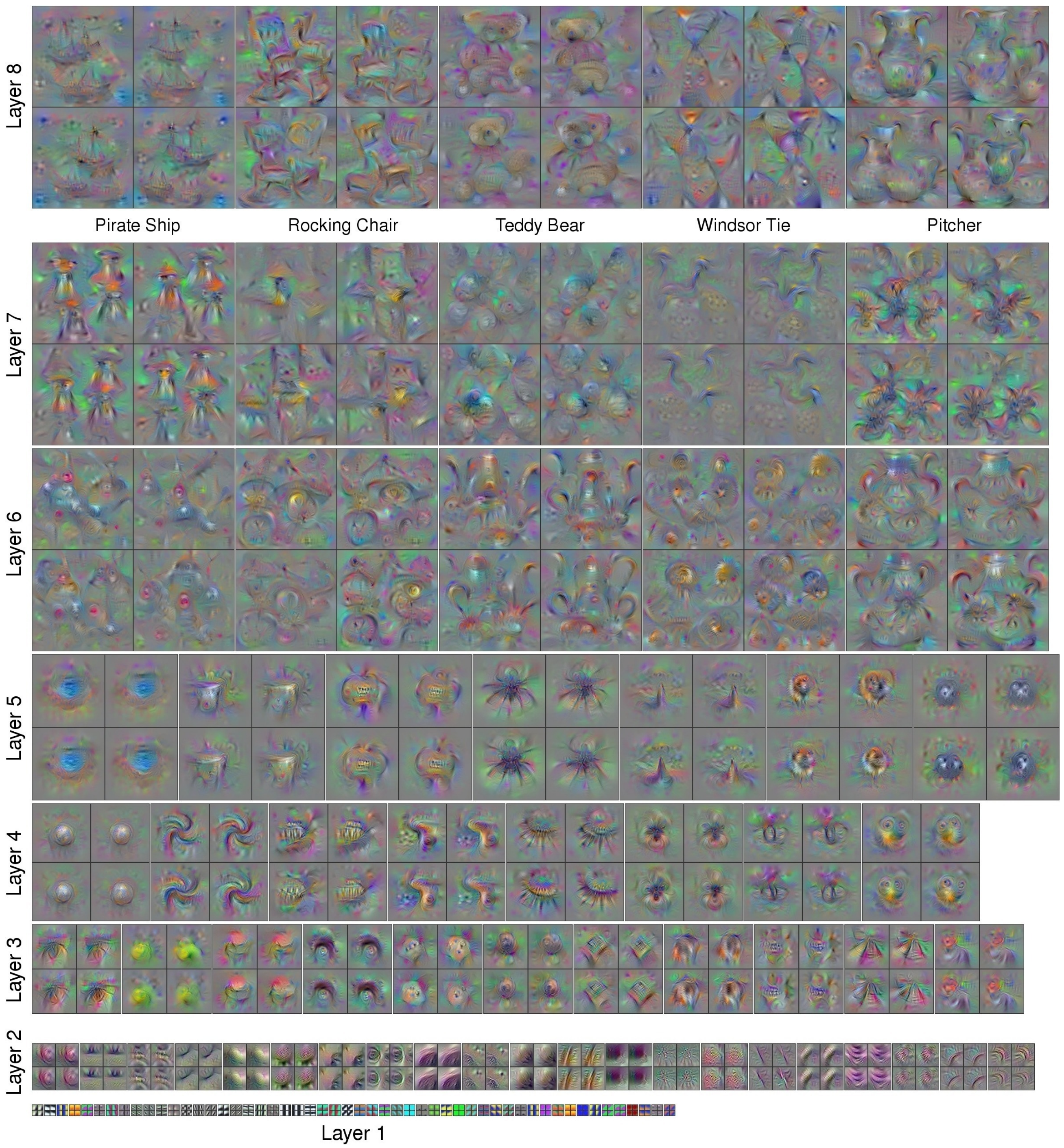


Figure 5. Visualization of example features of eight layers of a deep, convolutional neural network. The images reflect the true sizes of the features at different layers. In each layer, we show visualizations from 4 random gradient descent runs for each channel. While these images are hand picked to showcase the diversity and interpretability of the visualizations, one image for each filter of all five convolutional layers is shown in Figure S1 in supplementary information. One can recognize important features of objects at different scales, such as edges, corners, wheels, eyes, shoulders, faces, handles, bottles, etc. The visualizations show the increase in complexity and variation on higher layers, comprised of simpler components from lower layers. The variation of patterns increases with increasing layer number, indicating that increasingly invariant representations are learned. In particular, the jump from Layer 5 (the last convolution layer) to Layer 6 (the first fully-connected layer) brings about a large increase in variation. Best viewed electronically, zoomed in.

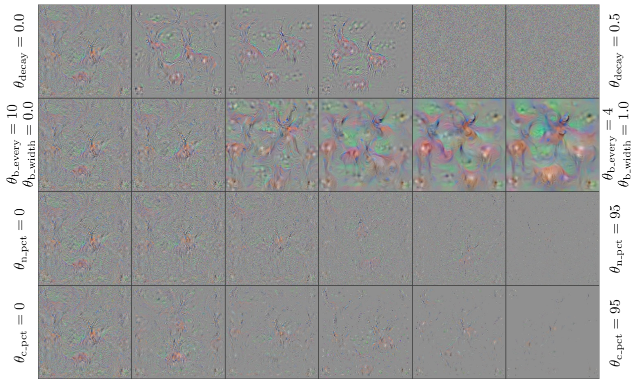

$\theta_{\text{decay}} = 0.0$
$\theta_{\text{decay}} = 0.5$
$\theta_{\text{b\_every}} = 10$
$\theta_{\text{b\_width}} = 0.0$
$\theta_{\text{b\_every}} = 4$
$\theta_{\text{b\_width}} = 1.0$
$\theta_{\text{n\_pct}} = 0$
$\theta_{\text{n\_pct}} = 95$
$\theta_{\text{c\_pct}} = 0$
$\theta_{\text{c\_pct}} = 95$

Figure 3. The effects of each regularization method from Section 3 when used individually. Each of the four rows shows a linear sweep in hyperparameter space from no regularization (left) to strong regularization (right). When applied too strongly, some regularizations cause the optimization to fail (e.g. L_2 decay, top row) or the images to be less interpretable (small norm and small contribution clipping, bottom two rows). For this reason, a random hyperparameter search was useful for finding joint hyperparameter settings that worked well together (see Figure 4). Best viewed electronically, zoomed in.

Table 1. Four hyperparameter combinations that produce different styles of recognizable images. We identified these four after reviewing images produced by 300 randomly selected hyperparameter combinations. From top to bottom, they are the hyperparameter combinations that produced the top-left, top-right, bottom-left, and bottom-right Gorilla class visualizations, respectively, in Figure 4. The third row hyperparameters produced most of the visualizations for the other classes in Figure 4, and all of those in Figure 5.

θ_{decay}	$\theta_{\text{b\_width}}$	$\theta_{\text{b\_every}}$	$\theta_{\text{n\_pct}}$	$\theta_{\text{c\_pct}}$
0	0.5	4	50	0
0.3	0	0	20	0
0.0001	1.0	4	0	0
0	0.5	4	0	90

4. Discussion and Conclusion

We have introduced two visual tools for aiding in the interpretation of trained neural nets. Intuition gained from these tools may prompt ideas for improved methods and future research. Here we discuss several such ideas.

The interactive tool reveals that representations on later convolutional layers tend to be somewhat local, where channels correspond to specific, natural parts (e.g. wheels, faces) instead of being dimensions in a completely distributed code. That said, not all features correspond to natural parts, raising the possibility of a different decomposition of the world than humans might expect. These visualizations suggest that further study into the exact nature of learned representations — whether they are local to a single channel or distributed across several — is likely to be interesting (see Zhou et al. (2014) for work in this direction). The locality of the representation also suggests that during transfer learning, when new models are trained atop the conv4 or conv5 representations, a bias toward sparse connectivity could be helpful because it may be necessary to combine only a few features from these layers to create important features at higher layers.

The second tool — new regularizations that enable improved, interpretable, optimized visualizations of learned features — will help researchers and practitioners understand, debug, and improve their models. The visualizations also reveal a new twist in an ongoing story. Previous studies have shown that discriminative networks can easily be fooled or hacked by the addition of certain structured noise in image space (Szegedy et al., 2013; Nguyen et al., 2014). An oft-cited reason for this property is that discriminative training leads networks to ignore non-discriminative information in their input, e.g. learning to detect jaguars by matching the unique spots on their fur while ignoring the fact that they have four legs. For this reason it has been seen as a hopeless endeavor to create a generative model in which one randomly samples an x from a broad distribution on the space of all possible images and then iteratively transforms x into a recognizable image by moving it to a region that satisfies both a prior $p(x)$ and posterior $p(y|x)$ for some class label y. Past attempts have largely supported this view by producing unrealistic images using this method (Nguyen et al., 2014; Simonyan et al., 2013).

However, the results presented here suggest an alternate possibility: the previously used priors may simply have been too weak (see Section S1 for one hypothesis of why a strong $p(x)$ model is needed). With the careful design or learning of a $p(x)$ model that biases toward realism, one may be able to harness the large number of parameters present in a discriminately learned $p(y|x)$ model to generate realistic images by enforcing probability under both models simultaneously. Even with the simple, hand-coded $p(x)$ models we use in this paper as regularizers, complex dependencies between distant pixels already arise (cf. the beetles with structure spanning over 100 pixels in Figure 4). This implies that the discriminative parameters also contain significant "generative" structure from the training dataset; that is, the parameters encode not only the jaguar's spots, but to some extent also its four legs. With better, learned probabilistic models over the input and activations of higher layers, much more structure may be apparent. Work by Dai et al. (2015) shows some interesting results in this direction. While the images generated in this paper are far from being photo-realistic, they do suggest that transferring discriminatively trained parameters to generative models — opposite the direction of the usual unsupervised pretraining approach — may be a fruitful area for further investigation.

Acknowledgments

The authors would like to thank the NASA Space Technology Research Fellowship (JY) for funding, Wendy Shang, Yoshua Bengio, Brian Cheung, and Andrej Karpathy for helpful discussions, and Freckles the cat for her feline countenance.

References

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clément. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

Dai, Jifeng, Lu, Yang, and Wu, Ying Nian. Generative modeling of convolutional neural networks. In *International Conference on Learning Representations (ICLR)*, 2015.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Erhan, Dumitru, Bengio, Yoshua, Courville, Aaron, and Vincent, Pascal. Visualizing higher-layer features of a deep network. Technical report, Technical report, University of Montreal, 2009.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume*, volume 15, pp. 315–323, 2011.

Goodfellow, Ian J, Warde-Farley, David, Lamblin, Pascal, Dumoulin, Vincent, Mirza, Mehdi, Pascanu, Razvan, Bergstra, James, Bastien, Frédéric, and Bengio, Yoshua. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

Goodfellow, Ian J, Shlens, Jonathon, and Szegedy, Christian. Explaining and Harnessing Adversarial Examples. *ArXiv e-prints*, December 2014.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. Deep Speech: Scaling up end-to-end speech recognition. *ArXiv e-prints*, December 2014.

Hinton, Geoffrey E, Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoff. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.

Lin, Tsung-Yi, Maire, Michael, Belongie, Serge, Hays, James, Perona, Pietro, Ramanan, Deva, Dollár, Piotr, and Zitnick, C. Lawrence. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL `http://arxiv.org/abs/1405.0312`.

Mahendran, A. and Vedaldi, A. Understanding Deep Image Representations by Inverting Them. *ArXiv e-prints*, November 2014.

Nguyen, Anh, Yosinski, Jason, and Clune, Jeff. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. *ArXiv e-prints*, December 2014.

Schroff, F., Kalenichenko, D., and Philbin, J. FaceNet: A Unified Embedding for Face Recognition and Clustering. *ArXiv e-prints*, March 2015.

Simonyan, Karen, Vedaldi, Andrea, and Zisserman, Andrew. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034, presented at ICLR Workshop 2014*, 2013.

Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian J., and Fergus, Rob. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

Taigman, Yaniv, Yang, Ming, Ranzato, Marc'Aurelio, and Wolf, Lior. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1701–1708. IEEE, 2014.

Torralba, Antonio and Oliva, Aude. Statistics of natural image categories. *Network: computation in neural systems*, 14(3): 391–412, 2003.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3320–3328. Curran Associates, Inc., December 2014.

Zeiler, Matthew D and Fergus, Rob. Visualizing and understanding convolutional neural networks. *arXiv preprint arXiv:1311.2901*, 2013.

Zhou, Bolei, Khosla, Aditya, Lapedriza, Àgata, Oliva, Aude, and Torralba, Antonio. Object detectors emerge in deep scene cnns. *CoRR*, abs/1412.6856, 2014. URL `http://arxiv.org/abs/1412.6856`.

Supplementary Information for: Understanding Neural Networks Through Deep Visualization

Jason Yosinski YOSINSKI@CS.CORNELL.EDU
Jeff Clune JEFFCLUNE@UWYO.EDU
Anh Nguyen ANGUYEN8@UWYO.EDU
Thomas Fuchs FUCHS@CALTECH.EDU
Hod Lipson HOD.LIPSON@CORNELL.EDU

S1. Why are gradient optimized images dominated by high frequencies?

In the main text we mentioned that images produced by gradient ascent to maximize the activations of neurons in convolutional networks tend to be dominated by high frequency information (cf. the left column of Figure 3). One hypothesis for why this occurs centers around the differing statistics of the activations of channels in a convnet. The conv1 layer consists of blobs of color and oriented Gabor edge filters of varying frequencies. The average activation values (after the rectifier) of the edge filters vary across filters, with low frequency filters generally having much higher average activation values than high frequency filters. In one experiment we observed that the average activation values of the five lowest frequency edge filters was 90 versus an average for the five highest frequency filters of 5.4, a difference of a factor of 17 (manuscript in preparation)[2,3]. The activation values for blobs of color generally fall in the middle of the range. This phenomenon likely arises for reasons related to the $1/f$ power spectrum of natural images in which low spatial frequencies tend to contain higher energy than high spatial frequencies (Torralba & Oliva, 2003).

Now consider the connections from the conv1 filters to a single unit on conv2. In order to merge information from both low frequency and high frequency conv1 filters, the connection weights from high frequency conv1 units may generally have to be larger than connections from low frequency conv1 units in order to allow both signals to affect the conv2 unit's activation similarly. If this is the case, then due to the larger multipliers, the activation of this particular conv2 unit is affected more by small changes in the activations of high frequency filters than low frequency filters.

[2]Li, Yosinski, Clune, Song, Hopcroft, Lipson. 2015. How similar are features learned by different deep neural networks? In preparation.

[3]Activation values are averaged over the ImageNet validation set, over all spatial positions, over the channels with the five {highest, lowest} frequencies, and over four separately trained networks.

Seen in the other direction: when gradient information is passed from higher layers to lower layers during backprop, the partial derivative arriving at this conv2 unit (a scalar) will be passed backward and multiplied by larger values when destined for high frequency conv1 filters than low frequency filters. Thus, following the gradient in pixel space may tend to produce an overabundance of high frequency changes instead of low frequency changes.

The above discussion focuses on the differing statistics of edge filters in conv1, but note that activation statistics on subsequent layers also vary across each layer.[4] This may produce a similar (though more subtle to observe) effect in which rare higher layer features are also overrepresented compared to more common higher layer features.

Of course, this hypothesis is only one tentative explanation for why high frequency information dominates the gradient. It relies on the assumption that the average activation of a unit is a representative statistic of the whole distribution of activations for that unit. In our observation this has been the case, with most units having similar, albeit scaled, distributions. However, more study is needed before a definitive conclusion can be reached.

S2. Conv Layer Montages

One example optimized image using the hyperparameter settings from the third row of Table 1 for every filter of all five convolutional layers is shown in Figure S1.

[4]We have observed that statistics vary on higher layers, but in a different manner: most channels on these layers have similar average activations, with most of the variance across channels being dominated by a small number of channels with unusually small or unusually large averages (Li, Yosinski, Clune, Song, Hopcroft, Lipson. 2015. How similar are features learned by different deep neural networks? In preparation.)

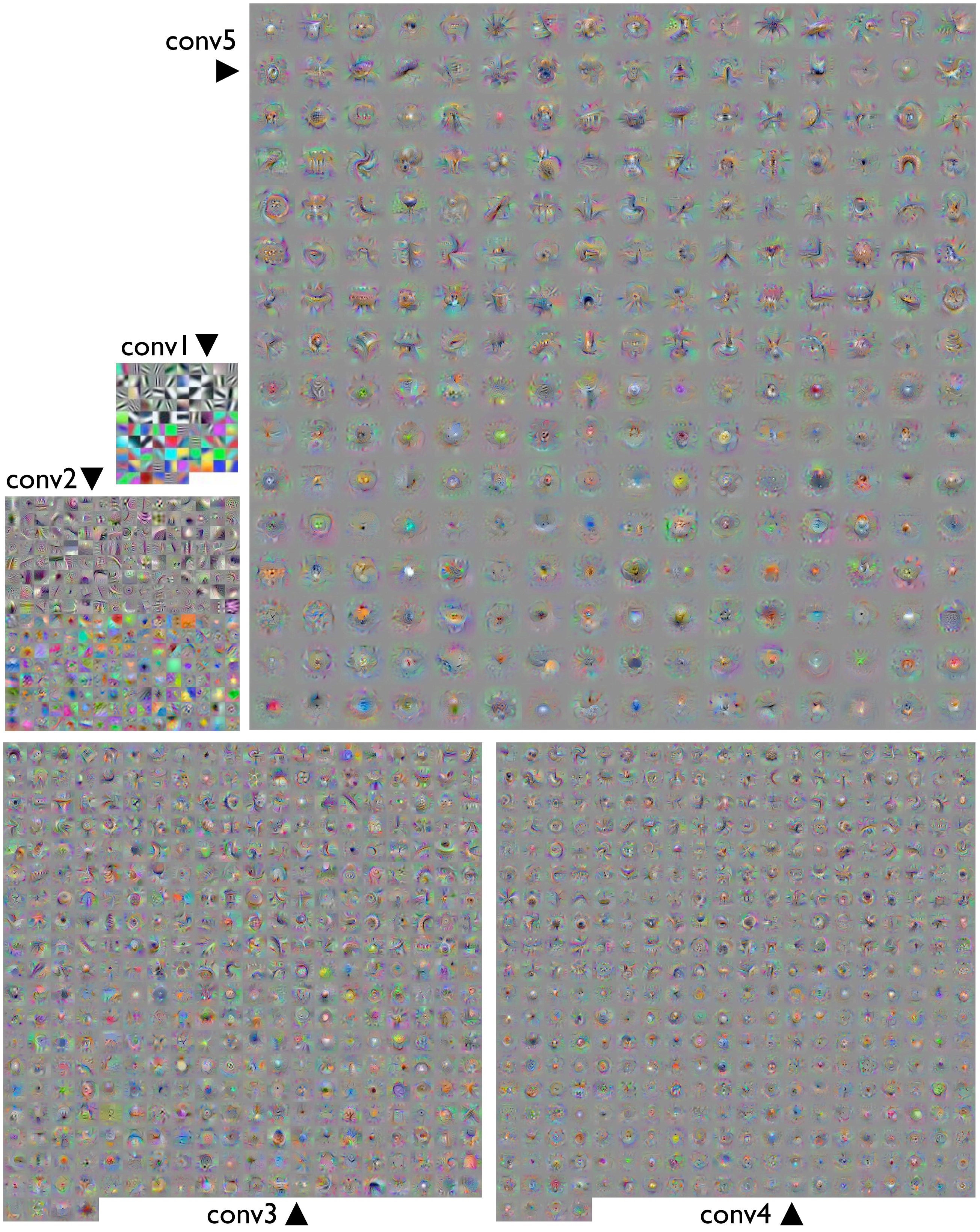


Figure S1. One optimized, preferred image for every channel of all five convolutional layers. These images were produced with the hyperparameter combinations from the third row of Table 1. Best viewed electronically, zoomed in.