Seein' In:
Peering into the depths of the convolutional neural network

Tanner Schmidt

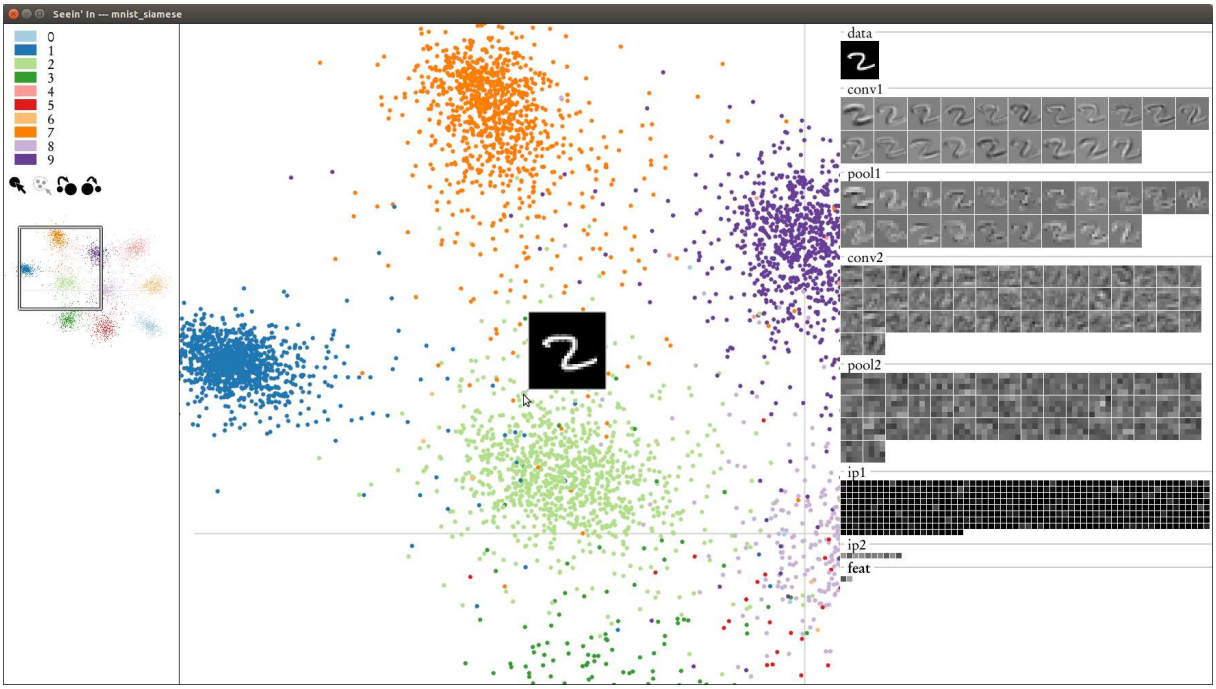

Fig. 1. The Seein' In user interface

Abstract—This paper presents Seein' In, an interactive tool for visualizing trained convolutional neural networks. The core visualization is a layer-by-layer scatter plot matrix, plotting pairs of dimensions of the intermediate feature maps produced by the network given a large input dataset. This display is coordinated with another display showing at once all activations produced by a single image, or the average activation produced by a subset of the images. A variety of navigation techniques support and encourage exploration of the vast amount of data contained in these displays. This tool should prove useful for machine learning researchers wishing to make more informed decisions about network architectures and learning parameters.

1 Introduction

Convolutional neural networks, or CNNs, have been around for quite some time [6]. However, just in they last few years they have enjoyed a massive increase in popularity, due in part to the abundance of parallel computation resources available with modern graphics processing units (GPUs) as well as their success on the ImageNet Large Scale Visual Recognition Challenge [4]. Part of a larger recent trend within the machine learning community towards so-called 'Deep Learning,' the CNN model is in essence a hierarchical decomposition of the general problem of learning a function from an input space to an output space. The function proceeds in stages or 'layers,' in which a number of 'hidden' intermediate representations of the input are computed as a function of the previous layer's representation, beginning with the input itself, until the output layer is reached. These functions are defined by a collection of weights or parameters which are learned from a large corpus of data.

While the architecture of the CNN is rather straightforward and easy enough to understand in general, the results produced by any one specific network can be frustratingly difficult to grasp. This is essentially a problem of dimensionality; applied to computer vision, for example, the input space is generally an image, the representation of which can easily stretch into the tens of thousands of dimensions. Given the high dimensionality of the data and depth of the networks typically used in the vision domain, it's not uncommon to see networks with tens of millions of parameters in the literature. Not surprisingly, even when networks perform objectively well, it is extremely difficult even for the researchers who designed them to describe the function it has learned with any degree of specificity.

As a direct result, decisions about the architecture of a network or the parameters used in the learning algorithm that determines its weights are generally made with little to no insight as to how they will affect the internal workings of the network. This leads to what has been described as 'graduate student descent,' a process with which the author is himself all to familiar, in which a graduate student makes iterative changes to the network or its learning parameters and observes the affect on some performance measure, aiming to increase performance by improving on the networks that perform best. Graduate student descent, of course, typically has the same problems with local minima as automated gradient descent methods, and is in any case neither enjoyable nor intellectually stimulating.

tws10@cs.washington.edu

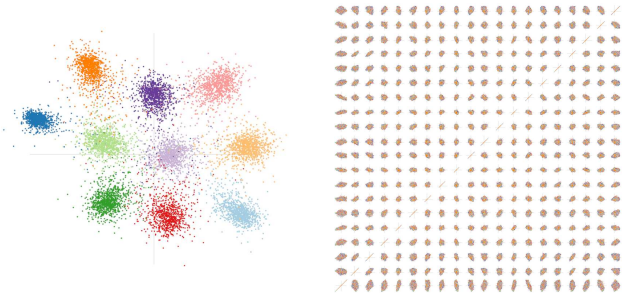

Fig. 2. The embedding view, which can show data with anywhere from two (left) to twenty (right) or more dimensions.

The goal of this project is to do better. With Seein' In, computer vision practitioners can directly observe the internal workings of their networks by seeing how a set of images is represented in any of the intermediate layers, in essence uncovering the 'hidden' layers. The dimensionality problem is addressed using a scatter plot matrix for layers of dimensionality greater than two, where every dimension of the representation is plotted against every other dimension. While these matrices quickly become very large, overwhelming, and somewhat difficult to parse, the hope is that any using window into the network operation is better than treating it purely as a black box. Some specific potential usages of the tool will be outlined in a case study in section 4. In addition to the core visualization of raw, high dimensional data, the tool also incorporates a number of basic visualization principles such as brushing and linking, focus+context, details on demand, and multiple coordinated displays as will be discussed section 3, after a brief review of related work.

2 Related Work

A traditional approach to visualizing convolutional neural networks involves selecting a particular feature of interest (e.g. a single dimension of the output responsible for a particular class label in the case of a classification network), and computing the input that would lead that feature to have a maximal value. This technique has recently been used by Le et al. [5] and by Simonyan et al. [7], where soft or hard constraints on the magnitude of the input are also enforced to ensure regularity. In the former, this is coupled with a visualization showing a collection of actual input images which cause the feature of interest to be activated most strongly. In the latter, the authors also present the ability to generate a saliency map showing how important each pixel in a particular input image is to its classification.

While these visualizations are compelling and do teach us something about the corresponding networks, their use is limited by the fact that they are still treating the network as a black box; the network is simply being driven backwards such that inputs are produced mysteriously from outputs rather than the usual outputs being produced mysteriously from inputs. Zeiler and Fergus [8] attacked this problem by using a similar approach to the saliency map, but for any dimension at any layer in the network, including the intermediate representations or 'feature maps.' This allows the viewer to see how high-level features might be constructed from lower-level features. By coupling this technique with the N-highest activation technique, they show a number of image patches that fire most strongly at internal network layers in an attempt to tease out what the individual dimensions in the intermediate representations might *mean*.

Meanwhile, there has been some previous work on interactive tools for CNN visualization, most notably deepViz by Bruckner et al. [1]. They built a server-client system for online visualization of a learned network, allowing the a number of user interactions:

1. The user can select an image, and then select any layer in the network to visualize the feature map for that image at that layer.
2. The user can view the confusion matrix, seeing which classes in the dataset are confused with which other classes and how often.
3. The user can view images from the dataset clustered by Euclidean distance in the last layer of the network.
4. The user can select two points in time and see the network parameters as learned at those points in time, side by side.

This system also allows for some degree of introspection into the network as they are able to directly display the feature maps as grayscale images. However, the feature maps are incredibly difficult to interpret in this format, especially at higher levels, where they mostly appear gray and blob-like. However, the interactivity of the system is compelling, as it allows the user to pick out from the massive amount of data exactly what they are interested in learning more about.

Seein' In incorporates the interactivity and feature map displays of deepViz, while also pushing the introspective powers of previous work by showing how a large number of images map onto any of the features dimensions. As a result, investigating any particular feature is as easy as looking at which images map to which parts of the various representation spaces.

3 Interface

In contrast to the web-based (and therefore highly distributable) platform of deepViz, Seein' In is a standalone C++ application. By giving up the broad distribution of a web application, we gain the ability to push the graphics rendering pipeline to the limit by rendering massive amounts of data using low-level OpenGL calls. Furthermore, deepViz assumes the data and trained network are already resident on a server, whereas a user wishing to inspect her own network and/or data would be required to perform bandwidth-intensive uploads.

Seein' In interfaces with the popular CNN implementation 'caffe' [3]. To use the tool, a user must first define her network as specified by caffe and train the network. The learned weights will automatically be saved by caffe. This weight file and the network specification file are then passed to Seein' In to launch the interface, along with a pointer to the test images to be displayed, and some additional information such as the image class labels and class names, if applicable.

3.1 The Embedding View

At the core of Seein' In is the embedding view, centrally located in the interface shown in figure 1, so called because the intermediate representation of the dataset in any layer of the network can be viewed as an embedding of the data into the representation space. At any given point in time, the embedding view shows the input data as represented in the active layer, which can be any layer in the network. Depending on the dimensionality of the representation in the active layer, the embedding view will show either a two dimensional scatter plot or a D by D matrix of scatter plots, where D is the dimensionality. Both cases are shown in figure 2.

In the case of the scatter plot matrix, each individual entry in the matrix plots one dimension of the representation against another; that

Fig. 3. The receptive field associated with a representation of an MNIST digit. Note that features firing for this particular receptive field could equally point towards identification as a two or a seven, and that further context will need to be incorporated in further layers to disambiguate the digit.

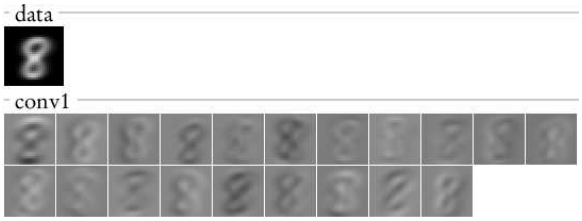

Fig. 4. The average input and average first layer response for every 8 in the MNIST test set.

is, the entry in the second row, third column will plot the third dimension on the x axis and the second dimension on the y axis. Thus, along the diagonal, we have straights lines, as a dimension plotted against itself results in a one dimensional dot plot. The plots are arranged such that all plots in a row share a y axis and all plots in a column share an x axis.

For each image, there is an associated vector of length D for the active layer, which is determined by the function the network has learned to map from the image space to the output space. Therefore, in a D by D scatter plot matrix, each image appears D^2 times, once per plot (or sometimes more, as will be explained shortly). The points associated with an image are colored according to the class label of the image, as specified by the user. If there are no labels in the dataset, the data are simply shown in the same color.

The embedding view is endowed with traditional pan and zoom navigation to enable exploration of the data at varying scale. The point size can be adjusted to trade off between density estimation and point resolution.

3.2 Details on Demand

Given a scatter plot showing the representations of a large number of images, a user may naturally desire to get more information about the image associated with a particular data point. Hovering the mouse over a data point in the embedding view will display the image associated with that point, as can be seen in figure 1. Now for a quick aside on structured representations and receptive fields.

Receptive Fields

One of the key insights of the convolutional aspect of the CNN is weight sharing — that is, that a learned function for recognizing a particular pattern in, for example, an 8 pixel by 8 pixel subregion of an input image is equally applicable to recognizing the same pattern in another part of the image. When using convolutions, rather than having a simple D dimensional representation in the resulting feature map, as is the case of the fully-connected layers of traditional neural networks, we actually have structured representations that themselves have a width W and height H in addition to depth D (think of an RGB color image with depth $D = 3$). Each entry in the feature map has associated with it a receptive field, which is the subregion of the original input that influenced or could have influenced that value of the map via all intermediate layers (thus the receptive field of the feature maps are monotonically increasing as information propagates through the network). Receptive fields are constant across the D dimension but vary along the W and H dimensions. One way to view an embedding in these intermediate layers is to simply unwrap the feature map into a length $D \times W \times H$ vector. Seein' In takes a different approach by instead interpreting an input image as represented in a structured feature map (i.e. one in which W and/or H are greater than 1) as having $W \times H$ distinct embeddings in a shared D dimensional space. Thus, when a user hovers over a point in a layer with a structured embedding, the point is associated not with an entire image but with a *part* of the image. This is shown to the user in a bounding box as shown in figure 3.

3.3 Coordinated Feature Map Display

In addition to seeing the image associated with a data point, the user may be interested in how that image is represented in other layers, or even in other dimensions of the same layer that are currently off the screen. To facilitate such explorations, there is a feature map view coordinated with the embedding view, as shown on the right side of the interface in figure 1. Clicking on an image will update the feature map view to show the response in every feature map in the entire network for the selected digit (the view can be scrolled and zoomed to handle extremely large networks). This is similar to the deepViz interface, but more extensive, as deepViz shows only one layer of feature maps at a time. In addition, selecting multiple images will compute and display the *average* response for the set of images at every layer, including the input, as in figure 4. Structured feature maps are shown as images in a grid, with one image per depth dimension of the feature map. Unstructured maps are simply shown as single-pixel images.

The feature map display can likewise be used to navigate within the embedding view. Clicking on the name of any layer in the feature map display will update the currently active layer in the embedding view accordingly. Clicking on a particular pixel in a feature map image will do the same but additionally navigate the user to the scatter plot on the diagonal of the matrix associated with that feature.

3.4 Focus + Context

Because the size of the scatter plot matrices can become massive with even moderately large dimensionality of embeddings, some context is required to ensure the user does not get completely lost in the data. Therefore, an overview of the entire matrix for the active layer is rendered and shown to the left of the embedding view as shown in figure 1. A box on the overview shows the current viewport in the embedding view, and dragging or scrolling in the overview performs the corresponding navigation in the embedding view as well. Because the scale changes can span multiple orders of magnitude, the box switches to a crosshairs overlay showing only the viewport center once the user has zoomed in so far that a box would not be resolvable.

3.5 Brushing and Linking

Seein' In also supports the brushing and linking technique for analyzing data shown in multiple displays. The user can select points one at a time, using an arbitrarily drawn polygon with the lasso tool, or by class label. If there is a selection, unselected points are made more faint and smaller while selected points are made larger. Furthermore, the render order is modified such that selected points are not hidden behind unselected points in dense displays. Once made, a selection is linked across all scatter plots in all layers, as shown in figure 5, allowing the user to select points of interest and see their representation throughout the network.

4 Discussion

To evaluate the potential utility of Seein' In, we'll explore some interesting aspects of a particular network that are brought to light through

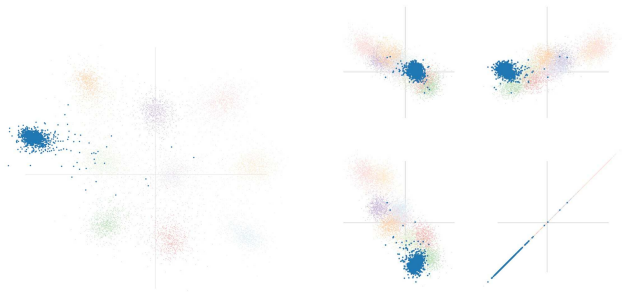

Fig. 5. A selection of points, shown across multiple plots in the same (right) and different layers.

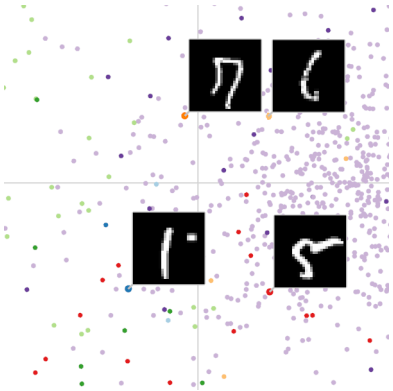

Fig. 6. A number of degenerate digits cluster around the origin of the two dimensional output embedding.

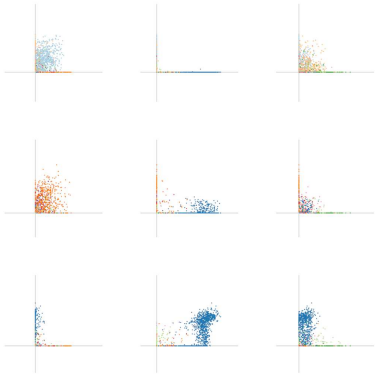

Fig. 7. The effect of the rectified linear unit: embeddings are constrained to the first quadrant of all scatter plots.

its introspective powers. This specific network was designed by Hadsell et al. to map images from the MNIST dataset, a collection of tens of thousands of images of handwritten digits, into a two dimensional space, with the goal of mapping instances of the same digit near each other but separated from instances of every other digit [2]. The network was taken directly from the 'siamese' example in caffe.

In fact, this is the data that has been used for all images shown herein. The data points are colored according to which digit the image depicts, as shown in the interface legend in figure 1. The images are 28 by 28 pixels, and there are 10,000 test images which the network does not get to see at training time. It is these images which are used for the analysis. The network has 7 layers: a convolution, a pooling, another convolution, another pooling, and finally three fully connected layers. There is only one non-linearity: a rectified linear unit (ReLU) between the first fully connected layer and the second, which sets all negative feature responses to zero.

4.1 Degenerate Digits

We'll start our analysis with the final result of the network. As you can see from the 2D scatter plot in figure 2, the images cluster quite nicely by label and are fairly well separated. A closer inspection, however, shows that there is one cluster that seems more permeated with other digits than other clusters, which are otherwise rather clean; this is the '8' cluster. It also so happens that the '8' cluster is situated closest to the origin of the embedding space. I'll hypothesize that the corruption of this cluster is do to what I'll call degenerate digits — these are digits which have something which is at least subjectively wrong with them and therefore are on the tails of the distribution of possible digits. It is unlikely that the network saw anything like these digits while training. A sampling of such digits can be seen in figure 6. With no training to go on, one might expect a roughly mean-zero output, resulting in a corruption of the '8' cluster which is nearest the zero vector. This seems to visually highlight a deficiency in the network, which is the fact that it has no way to encode uncertainty about the embedding of a particular digit.

4.2 Uneven Clustering Performance

Similarly, there is a cluster which appears to have much smaller variance than all the others, an indicator of greater recognition by the network. This is the '1' cluster. In order to see why this is, we'll delve deeper into the network. Shown in figure 8 is a scatter plot showing two out of the 50 dimensions of the feature map produced by the second pooling layer, before entering the fully connected layers. For the most part, the digits seem fairly thoroughly intermixed, excepting for one visually obvious dense cluster of blue points ('1's). Hovering over these points reveals that they are predominantly subregions of images with a vertical line on the right and nothing to the left — a feature which fires very frequently for '1's, fairly often for '7's, and less frequently for other digits. This clustering mid-way through the network surely makes the job of separating '1's from everything else much easier upstream. I'll hypothesize that this is happening because '1's more or less require a very simple edge detection which can be done in a layer or two, whereas other digits are more complicated combinations of lines and curves. This might suggest that a we would be better off with a deeper network, such that other digits might have a chance to achieve similarly tight lower-level clustering.

4.3 Feature Interpretation

Imagine now that the user clicks the '1' row in the legend, causing all '1's to be selected and causing the feature map view to show the average features maps for all instances of the digit. One thing she may notice is that there are few features produced by the first inner product layer which fire quite strongly, on average, for the instances of the digit '1', as indicated by their brightness relative to surrounding features. Clicking on one of these feature will transport her to the one dimensional dot plot showing where all 10,000 images map onto this one dimension in this particular feature map. The first thing she might notice is that not only is this feature activated strongly by most '1's, it is also rarely activated strongly by any other digit.

Our user can now drill down further, using the lasso tool to select only the images that cause this feature to be activated most strongly. This result is shown in the top of figure 9, with the selection shown at left and the average digit shown at right. It seems that this feature is activated most strongly by not just any instance of the digit '1', but more specifically by instances with a strong slant towards the right. Moving down towards more moderately activating images (second row in figure 9), we see an average image of a more upright '1'. Finally, more towards the middle we see an average images which seems to be a number of '1's slanting back the other way, with some '7's and '8's mixed in giving a faint crossing line. This is fascinating — we seem to have discovered a single intermediate feature which has learned to encode the slant of a drawing of the digit '1'!

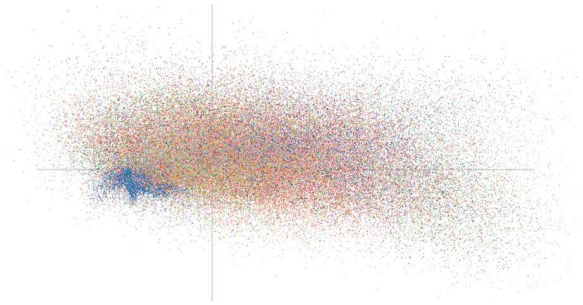

Fig. 8. A scatter plot demonstrates a part of representation space that is dominated by subregions from drawings of the digit '1'.

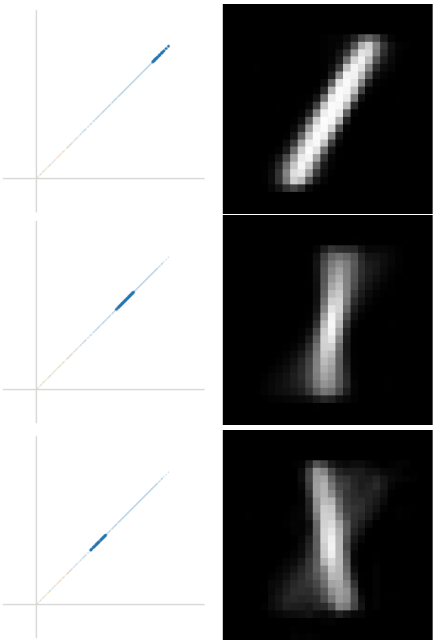

Fig. 9. Selections of different regions of an intermediate feature (left) show average images (right) with angles related to the feature value.

4.4 Network Trade-offs

Another visually striking effect in this data is the vast difference in the appearance of the feature maps produced by the first inner product layer and all other feature maps. This is due to the rectified linear units causing all points to lie in the first quadrant of all scatter plots. It seems to be that the ReLU allows this layer of the network develop groups of features that fire together for one or two classes to the exclusion of all others, as can be seen in figure 7. This might lead one to wonder what would happen without the rectification. To find out, we'll just train a new network without the ReLU layer and see what happens!

The feature maps produced by the first inner product layer will of course look completely different as the data will again occupy all four quadrants of the scatter plots. The final output embedding produced by the third fully connected layer is not as well separated as before, but it looks similar (not surprising, as it is directly encouraged to do so by the loss function). What is interesting is the effect on the second fully connected layer, shown in figure 10. These embedding appear quite different, with the mapping computed without the ReLU riddled with linear dependences between features, a highly undesirable trait of a network.

This example brings us back to the original motivation for Seein' In — in this example, the input was the same, one parameter was changed, and the output was quantitatively worse but appears similar. With graduate student descent, the unfortunate graduate student would simply carry on with the network with the ReLU and discard the network without. With Seein' In, CNN researchers can hopefully gain some insight as to how parameters are affecting the internals of their networks and make more informed decisions about how to move forward.

5 Conclusion

There are a variety of directions to take for future work. The A/B comparison of two networks is a compelling use case, but currently requires running two different instances of the interface. It would be nice to be able to load two sets of weights simultaneously such that navigation and/or selection can be automatically coordinated between the displays of both networks. Similarly, it would be nice to add a temporal feature as exists in deepViz, with which a user can hit a play button or a stepper to see how their network evolves over time. In the extreme, one can even imagine a more thorough integration of Seein' In with caffe such that the output of a network can be observed even as it's being trained.

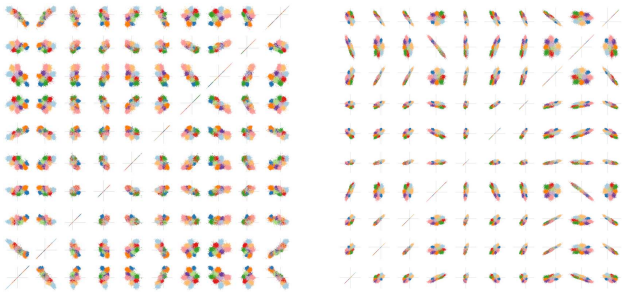

Fig. 10. The output feature maps of the second fully connected layer with (left) and without (right) the rectified linear units.

Another straightforward extension would be to see what could be done with 3D scatter plots. These are notoriously tricky to interpret, but with proper interaction (e.g. rotation), they could prove useful.

Finally, from the layer-by-layer scatter plots, it seems like it is possible to try to gain some understanding of what certain features are representing. However, what the current implementation does not help with is figuring out *how* those representations are learned. One straightforward extension would be to add support for comparing the representation in one feature map on one axis of a scatter plot matrix against the representation in another layer on the other axis. This would give some sense of causality, showing which features activated in earlier layers are correlated with features activated in later layers.

In conclusion, I've presented an exploratory tool for investigating the internal workings of trained convolutional neural networks. The key idea was to focus on showing large amounts of high-dimensional data exactly as it is at every step in the process of mapping from inputs to outputs. The amount of data can be overwhelming, but the hope is that with the suite of interactive tools provided – details on demand, brushing and linking, focus+context, and coordinated displays – users will be able to begin to make some sense of what's happening inside their networks.

References

[1] D. Bruckner, J. Rosen, and E. Sparks. deepviz: Visualizing convolutional neural networks for image classification. 2014.

[2] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

[5] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *Proc. of the International Conference on Machine Learning (ICML)*, 2012.

[6] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[7] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[8] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 818–833. Springer, 2014.