Visualization of Feature Evolution During Convolutional Neural Network Training

Arjun Punjabi
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL, USA
arjunpunjabi2015@u.northwestern.edu

Aggelos K. Katsaggelos
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL, USA
aggk@eecs.northwestern.edu

***Abstract*—Convolutional neural networks (CNNs) are a staple in the fields of computer vision and image processing. These networks perform visual tasks with state-of-the-art accuracy; yet, the understanding behind the success of these algorithms is still lacking. In particular, the process by which CNNs learn effective task-specific features is still unclear. This work elucidates such phenomena by applying recent deep visualization techniques during different stages of the training process. Additionally, this investigation provides visual justification to the benefits of transfer learning. The results are in line with previously discussed notions of feature specificity, and show a new facet of a particularly vexing machine learning pitfall: overfitting.**

Keywords—deep learning; convolutional neural network; feature visualization; transfer learning

I. Introduction

Convolutional neural networks (CNNs) have provided state-of-the-art performance in a variety of computer vision and image processing applications [1]. Recent developments in hardware, namely GPUs, have caused an inundation of CNN-based methods. That said, a discrepancy exists between knowledge of how to construct such algorithms and knowledge of how these algorithms operate. One major criticism of CNNs in general refers to the treatment of the algorithm as a "black box", with the ultimate result of the training procedure shrouded in mystery. Although the process of backpropagation used to modify filter weights has been thoroughly discussed, describing the function of these features has been less explored.

Algorithms that fall under the category of deep visualization strive to address such issues. At their core, these methods attempt to bridge the gap between human and machine perception by illustrating CNN features in a visual manner. This paradigm differs from some traditional views on CNN analysis that are primarily results oriented. It is common practice to judge the efficacy of any modifications to a network or dataset by the capacity to increase performance. Of course, this is a functionally logical approach to CNN design; however, not observing changes to the network features themselves is another example of the "black box" methodology. Such thinking may inhibit progress towards the next breakthrough in machine learning. It is the intention of deep visualization to aid in combatting the esoterica of CNNs.

II. Related Work

Deep visualization encompasses several approaches that have been described in the literature. This analysis will focus on a technique called activation maximization. The term was perhaps first coined in a 2009 publication in which the authors describe "qualitative interpretations of high level features" [2]. They produce visualizations from a deep belief network (DBN) trained on the classic MNIST digit classification dataset that confirm intuitions held about the learned representations. Since then, several authors have employed activation maximization and modified the procedure or usage. Yosinksi et al. [3] applied the method to a more complex classification problem and developed an accompanying software toolbox for interactive visualization. A 2015 investigation at Google described a technique that modified activation maximization with the purpose of creating art as "inceptionism" [4]. In [5], the algorithm was modified to highlight the multifaceted nature of specific network neurons. Mahendran and Vedaldi [6] created a generalized algorithm to perform activation maximization as well as another deep visualization method: inversion.

Inversion produces a different kind of visualization that is primarily used to quantify the loss of information at increasingly deep network layers. In essence, the ability for a network to reconstruct an input image from features at a given layer signify the information retained in those layers. Mahendran and Vedaldi first described their inversion method in [7], and Dosoviskiy and Brox supply a different approach in [8]. Inversion is related to another type of visualization that uses a "deconvolutional" network to identify stimuli of individual feature maps [9]. This identification is akin to locating the receptive field of a feature, a concept also explored in [10].

A third class of deep visualization algorithms can be described as sensitivity or saliency maps, which illustrate the support of a particular feature in a given image. Simonyan et al. [11] compare this method with a form of activation maximization. In [12], the authors show sensitivity maps with evidence both for and against a particular class, while [13] develops heatmaps showing relevance or importance of image regions.

All of these methods yield complementary views of the information in neural network features. Because this analysis focuses on activation maximization, a more detailed explanation of the procedure is outlined in the next section.

ISBN 978-0-9928626-7-1

III. Activation Maximization

The following explanation of the activation maximization method will synthesize information from [3] with additional description supplied by [6]. As previously suggested, the algorithm aims to create visual representations of CNN features, either at the convolutional filter level or object class level. In this manner, the method can be cast as an inverse problem that is solved using an optimization approach. To begin, consider an RGB image that produces some activation when passed through a CNN. Yosinksi formulates the problem as in [3]:

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

where x^* is the final visualization, x is a candidate input image to the network, $a_i(x)$ is the activation for some particular unit i, and $R_\theta(x)$ is some parameterized regularization function. In general, the unit i to be maximized can be the index of a filter or element in any layer of the network; however, in this case, the following analysis will only concern maximizing indices representing classes in the last layer of the network. The final visualization will be a synthetic RGB image of the same size as the input. One can also formulate a minimization to accomplish the same task, as Mahendran and Vedaldi do in [6], that is:

$$x^* = \arg\min_x \big(l(\Phi(x), \Phi_0) + R_\theta(x)\big) \tag{2}$$

where $l(\Phi(x), \Phi_0)$ is a loss function between the feature representation of the input $\Phi(x)$ and the target feature representation Φ_0. Φ_0 can either be the weights of the filter one wishes to visualize, or in this analysis, the final feature vector of the target class. In this case, the loss function is usually defined as the Euclidean distance between the two vectors. Alternatively, although the logic is somewhat circular, the loss function can be defined as the negative of the similarity, typically calculated using a dot product. This analysis will opt for the simpler case defined by Yosinski [3].

The optimization can be effectively solved using a gradient descent procedure. The pixels in x are modified in the direction of the gradient of $a_i(x)$. Consequently, the regularization is usually applied to the gradient step rather than in the objective function itself. Several regularizers are suggested in [3] and [6], with the overall goal of restricting the visualizations to natural-looking images. Without such a condition, the resulting images will not be semantically interpretable to humans, even if they are reasonable solutions to the optimization. The authors in [6] present two bounds on pixel range and variation, which have some corollaries in [3]. Some more complex functions that involve pixel shifts and texture regularizers are also presented. There is not a clear consensus on the optimal regularization methods; therefore, this analysis opts for two relatively simple conditions. Pixel changes that fall outside the normal range are clipped, and a 5x5 median filter is applied every four gradient steps. It was experimentally found that these conditions were satisfactory to produce semantically interpretable visualizations.

IV. New Applications: Feature Evolution and Transfer Learning

At this point, activation maximization as a method for deep visualization has been thoroughly discussed, both in usage as well as in implementation. Yet, there is much untapped potential in this domain. One key assumption that predicates the use of the algorithm is the existence of a fully trained network. This condition is a natural one: it is logical to visualize features after their modifications during training. However, perhaps visualizing the evolution of features during the training process would be even more enlightening. Most observations of neural network training have involved tracking values of loss functions or validation accuracies; now, there is an opportunity to visualize the actual features at play. By visualizing features at several time points during training, the evolution of features can be compared to improvements in performance and shed light on the otherwise obfuscated learning procedure.

This new line of thought also presents the chance to observe another somewhat enigmatic facet of neural networks: transfer learning. As described in [14], the generality of low-level features suggests that a network trained on one task may only need to slightly modify those features in order to perform an entirely different task on new data. The authors argue that it is the deep layer features that are task specific and thus require greater changes. In practice, this manifests itself when a standard CNN architecture is initialized with weights from one task and then fine-tuned with a new dataset. It can be seen that the training procedure will converge faster, and in some cases the accuracy may even be higher than if the starting weights were randomly initialized. With this new paradigm of using activation maximizations to visualize features during learning, perhaps a greater understanding of this phenomenon will emerge.

V. Results and Discussion

Two experiments were designed to examine visualizations that arise during the training of a CNN. In one instance, the filters weights in the network were randomly initialized in the usual fashion. The other case began with weights trained on the ImageNet ILSVRC 2012 dataset for 1000 class object classification [15]. The CNN architecture used in both cases is the VGG-16 network described in [16]. The network was implemented in Theano using Keras as a front-end [17][18]. Some additional references were used in the compilation of the code [19][20]. The Adadelta optimizer was used in the training procedure [21], and categorical cross entropy was used for classification. The network was trained using an NVIDIA Titan Z, with total training times on the order of several hours. The activation maximization implementation also made use of the Titan Z, where each visualization took 2.5 minutes to complete.

In both instances, the classification task was to differentiate between a small subset of the ImageNet data. Namely, only four classes were used: tree frog, flamingo, pool table, and hamburger. Because there are only four classes in this new task, the last layer of the VGG network was changed from a length of 1000 to a length of four. As a result, the weights from this layer could not be transferred in the pretraining experiment. Each class contains 1300 images, yielding a total dataset of 5200 images. 400 of these images were put aside in a validation set.

ISBN 978-0-9928626-7-1 © EURASIP 2017

Epoch	1	2	4	6	8	12	13	18	23
Validation Accuracy	60%	80%	83%	88%	89%	90%	91%	92%	93%

Table 1. Validation accuracy during training; no pretraining

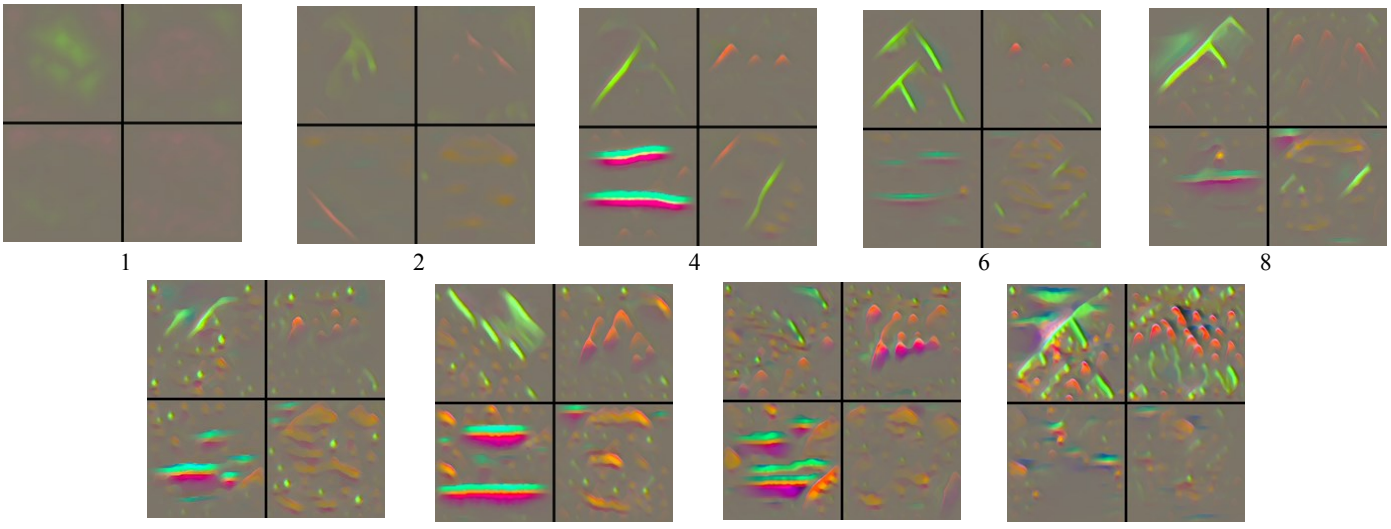

Fig. 1. Visualizations of network at each training epoch; no pretraining. Classes from upper left (clockwise): tree frog, flamingo, hamburger, pool table

Epoch	1	2	6	7	8	14	20
Validation Accuracy	59%	89%	91%	95%	96%	96%	93%

Table 2. Validation accuracy during training; pretrained on full ImageNet

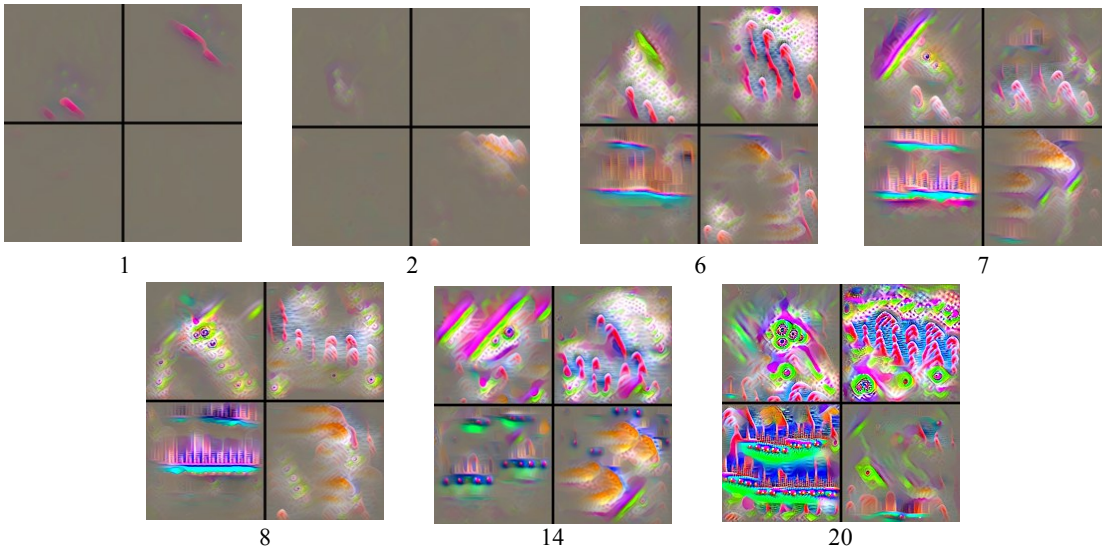

Fig 2. Visualizations of network at each training epoch; pretrained on full ImageNet. Classes from upper left (clockwise): tree frog, flamingo, hamburger, pool table

ISBN 978-0-9928626-7-1 © EURASIP 2017

A. Trained from randomly initialized weights

Table 1 shows the validation accuracy during training at each epoch. After the training set is passed through the network a single time, the model performs classification on the validation set with 60% accuracy. The model is fully trained after 23 epochs and achieves 93% accuracy at this time. Only epochs in which the validation accuracy increases are shown. The corresponding activation maximizations at each epoch are shown in Fig. 1. Each image is divided into four sections, each corresponding to one class. The classes, starting from the upper left section in clockwise order are: tree frog, flamingo, hamburger, and pool table.

To begin, it is clear that the visualizations at the early layers of the network are not very informative. At this stage, the convolutional filters have not been fully developed, nor is the validation accuracy high enough to justify their efficacy. That said, within a few epochs one can see some salient features forming. In epoch 4, it appears that the tree frog class is represented by a series of green lines, the flamingo class by some pink shapes, and the hamburger class by similar brown shapes. The pool table class is more strongly defined, with the visualization showing a very prominent horizontal colored line detector. Perhaps this shows an early understanding of the discontinuity between the colored felt of a pool table and the wooden rails. After epoch 12, the validation accuracy exceeds 90% and while the features do appear to increase in complexity, they are not nearly representative of their corresponding classes.

Based on the results in the literature of activation maximization applied to networks trained on the full ImageNet dataset, one would expect the visualizations to more closely resemble the original objects. Fig. 3 shows the visualizations of such a network; in this case, activation maximization was applied to the VGG network fully trained on the entire ImageNet set and without any fine-tuning with the small four-class subset. One can clearly see notions of the objects in these visualizations, from frog eyes and flamingo necks to pool balls and hamburger buns. It appears that the discriminatory power of a feature is heavily dependent on the difficulty of the task: a simpler classification task will yield simpler features even when the constituent data is the same.

B. Pretrained with full ImageNet

This observation is further supported by the results in Fig. 2, showing the visualizations of a network pretrained on the full ImageNet dataset. The corresponding validation accuracies at each epoch are shown in Table 2. In this instance, the network converges to high validation accuracies sooner than previously. This is to be expected, given the transferred knowledge already in the network. The eventual maximum accuracy is higher, reaching 96% by epoch 8. Again, given the study of transfer learning in [14], this result is unsurprising. The features are also more complex; but, yet again, the features are not as complex as in Fig. 3. It may be argued that the tree frog features resemble eyes by epoch 8, the flamingo shapes are more pronounced, and the hamburger buns are more discernable. The pool table features are much more apparent; in fact, the visualization in epoch 14 does seem to show red pool balls lined up on blue or green felt. Many deductions can be made from these results. For one, it may be argued that the additional accuracy from pretraining is most likely due to the added feature complexity. Furthermore, these results still support the theory that simplifying the classification task will result in less complex and well-defined features.

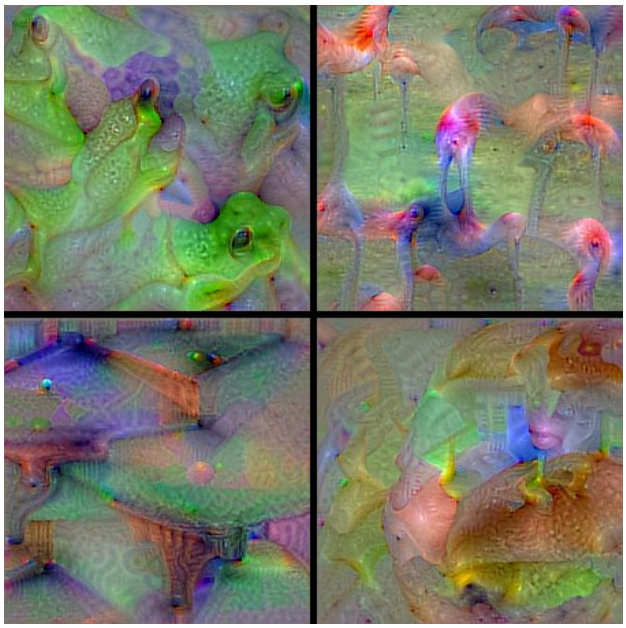

Fig. 3. Visualization of network trained on full ImageNet. Classes from upper left (clockwise): tree frog, flamingo, hamburger, pool table

One final modification was made in this experiment: a visualization was shown for epoch 20, where the validation accuracy slightly decreases. Given that the training loss at this time was still decreasing, this suggests that the network may have begun to overfit the data. It seems that even though the complexity of the features is still increasing, the images themselves are less clear. For example, it appears the tree frog eyes that exist in epochs 8 and 14 begin to manifest themselves in the visualizations of other classes by epoch 20. One particularly notable case is in the bottom section of the flamingo visualization. In addition, the pink neck shapes that define the flamingo class appear in both the tree frog and pool table visualizations. Perhaps this confusion of features is an illustration of the mechanism behind which overfitting can degrade the discriminative power of a network. More testing would be required to fully investigate this phenomenon.

VI. Summary and Future Work

Deep visualization of feature evolution, especially in the case of transfer learning, is a nascent approach to understanding CNNs. In this paper, activation maximization was used to experimentally show the following:

- Feature complexity increases with validation accuracy, but can continue to increase even after accuracy saturates
- Discriminative classification power of a network is a function of the number of classes; i.e. a CNN automatically generates features of just enough complexity to perform the task at hand, even when the network is pretrained on a more challenging task

ISBN 978-0-9928626-7-1 © EURASIP 2017

- Training on a more challenging task (e.g. larger number of classes) will yield features that are more informative and archetypal of the representative class members
- Unchecked feature complexity leads to feature confusion, a potential precursor to overfitting

As discussed, additional testing into confusion of features may lead to greater understanding of the ever-present caveat of overfitting. Larger datasets with greater variety, such as increasing the subset of classes used in ImageNet, are a logical next step. Some modifications to the optimization algorithm for the visualizations may also prove useful, such as more intricate loss functions or regularizers. Freezing the fine-tuning of certain layers during pretraining is discussed in [14] and supplying visuals for this analysis may be enlightening. Finally, subjecting other deep visualization techniques, such as inversion, to this temporal analysis may enhance the information gleaned from these methods.

ACKNOWLEDGMENT

The authors would like to acknowledge the Integrated Data Driven Discovery in Earth and Astrophysical Sciences (IDEAS) program at Northwestern University for support (NSF Research Traineeship Grant 1450006).

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.

[2] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. "Visualizing higher-layer features of a deep network." *University of Montreal* 1341 (2009): 3.

[3] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. "Understanding neural networks through deep visualization." *arXiv preprint arXiv:1506.06579* (2015).

[4] A. Mordvintsev, C. Olah, and M. Tyka. "Inceptionism: Going deeper into neural networks." *Google Research Blog. Retrieved June* 20 (2015): 14.

[5] A. Nguyen, J. Yosinski, and J. Clune. "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks." *arXiv preprint arXiv:1602.03616* (2016).

[6] A. Mahendran and A. Vedaldi. "Visualizing deep convolutional neural networks using natural pre-images." *International Journal of Computer Vision* 120.3 (2016): 233-255.

[7] A. Mahendran and A. Vedaldi. "Understanding deep image representations by inverting them." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.

[8] A. Dosovitskiy and T. Brox. "Inverting visual representations with convolutional networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[9] M. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer International Publishing, 2014.

[10] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. "Object detectors emerge in deep scene cnns." *arXiv preprint arXiv:1412.6856* (2014).

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

[12] L. Zintgraf, T. Cohen, and M. Welling. "A new method to visualize deep neural networks." *arXiv preprint arXiv:1603.02518* (2016).

[13] W. Samek, A. Binder, G. Montavon, S. Bach, and K. Muller. "Evaluating the visualization of what a deep neural network has learned." *IEEE Transactions on Neural Networks and Learning Systems* (2016).

[14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?." *Advances in neural information processing systems*. 2014.

[15] O. Russakovsky, et al. "Imagenet large scale visual recognition challenge." *International Journal of Computer Vision* 115.3 (2015): 211-252.

[16] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

[17] J. Bergstra, et al. "Theano: A CPU and GPU math compiler in Python." *Proc. 9th Python in Science Conf*. 2010.

[18] F. Chollet. "Keras." (2015).

[19] F. Tence. "Visualizing Deep Neural Networks Classes and Features." *Ankivil — Machine Learning Experiments*. N.p., 7 July 2016. Web. 27 Feb. 2017.

[20] F. Chollet. "How convolutional neural networks see the world." *The Keras Blog*. N.p., 30 Jan. 2016. Web. 27 Feb. 2017.

[21] M. Zeiler. "ADADELTA: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).