Département de génie électrique et de génie informatique

Visualizing Deep Neural Networks

Apprentissage et reconnaissance (GIF- 7005)

Mathieu Garon, Jinsong Zhang, Zahra Rezaei

September 14, 2017

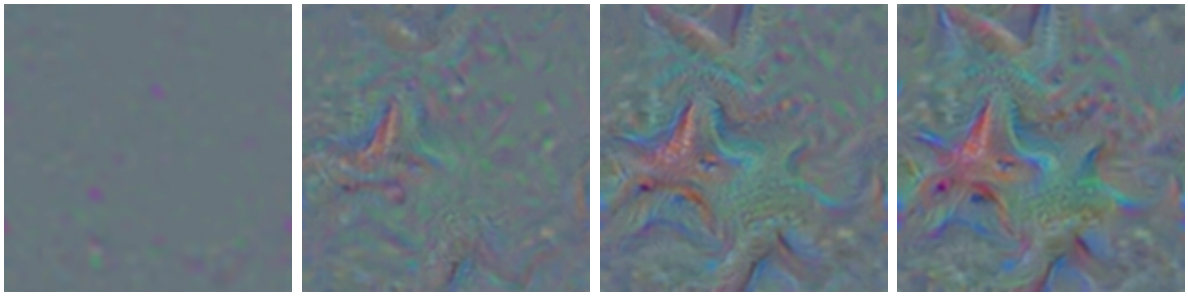


Figure 1.1: Starfish generation by our visualization tool. Starting from a random image(left), over some iterations a starfish(right) image is synthesized.

1 Introduction

Deep learning is a subset and a relatively new area of Machine Learning. Deep Neural Networks have demonstrated state-of-the-art results on many pattern recognition tasks, especially vision classification problems. However it is really difficult to gain intuition on what is happening inside a deep neural network for a particular model. This project aims to visualize the inside of a neural network that would be useful for debugging and model improvement. We propose a visualization tool that let the user glance at the activation of different neurons, and multiple utilities to generate realistic images from a convolutional neural network, see fig.1.1.

2 Related work

Neural networks have been thought of as "black boxes" because it is difficult to understand the neural functions due to the large number of interacting, non-linear parts. Understanding what the model represent is not only interesting but also one key way to debug and improve it. For example, visualizing the features learned by the hidden units suggested an architectural change of smaller convolutional filters that led to state of the art performance on the ImageNet benchmark (Zeiler and Fergus [2014], Yosinski et al. [2015]).

Several approaches for understanding and visualizing Convolutional Networks have been developed. Currently, there is two paradigms in network visualization : Dataset-centric Zhou et al. [2014] and Network-centric Yosinski et al. [2015], Nguyen et al. [2016]. The former would display images from the training or test data that cause high or low activations for individual units. Given an image that is correctly classified this method could use a simplified image to activate the neurons while keep a high classification score. This simplified image indicates the key elements in the image and the key neurons in the network that lead to the classification. For example, it could be possible to pass many faces image to find the neurons with high activation, these neurons may understand face well. It can be less practical to use a data-centric method, and Network-centric methods circumvent the need of data. Those methods aim to use only the model to generate information: for example using back-propagation to generate new data.

The proposition of Yosinski et al. [2015] is to optimize an input image by activating a particular unit at a time. Doing so gives high frequency images that can not be interpreted by humans. They propose different regularization methods to make the image more realistic. More recently, interesting solution uses Generative Adversarial Network to render more realistic images. The general idea of

Clune et al. [2016] is to generate an image that produces high activation of a neuron (from a model's embedding) with the Generator Network. The downside of these methods is the necessity to train a Generative Network which can be cumbersome.

For this project, we propose different techniques to visualize the network's representation[1]. First, given an input image, we visualize the activations at each layer for the network, see fig.2.1. While simple, it can give a lot of insight on how the filter reacts to different data. Furthermore, we implement the activation maximization (AM) to understand how a neural network represent information internally, by synthesizing an image from specified neurons. We implement several regularization methods to generate the visualization image as in Nguyen et al. [2016] and Deep Dream[2].

Moreover, it is possible to use groups of images to monitor the neurons activation as in the the dataset-centric paradigm. We can use the most activated neurons to generate new images and visualize how the network reacts to new classes.

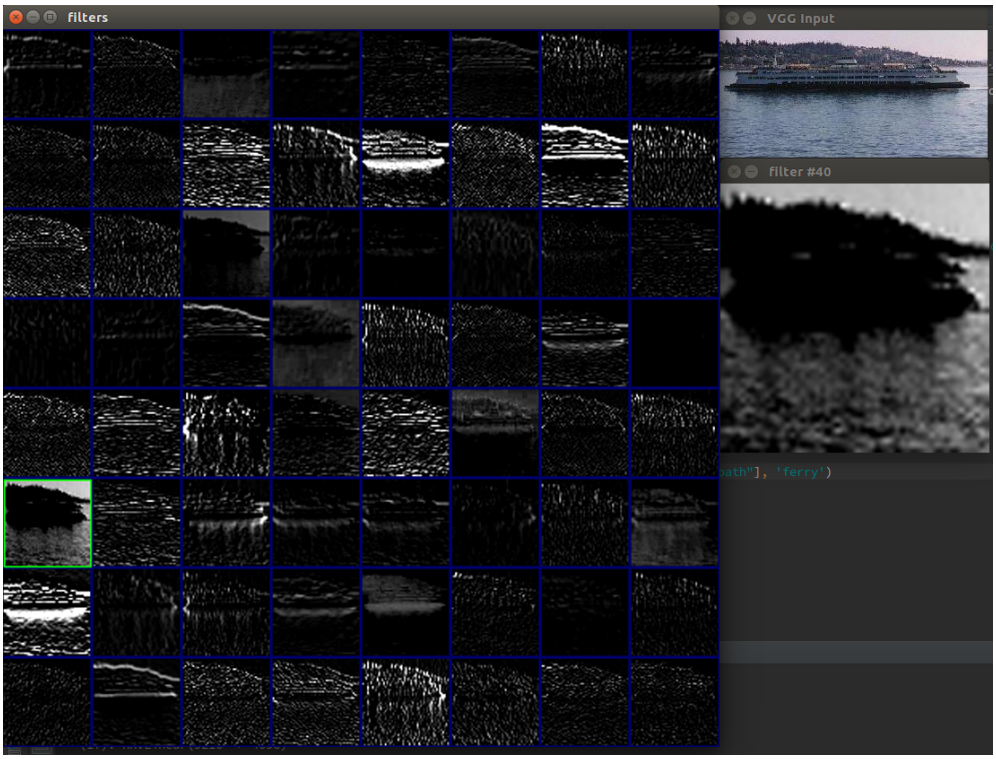


Figure 2.1: Activations. Visualize the activations at one layer in a network. Do a forward pass using the top-right input image, our tool could visualize the activations. The left grid shows a 64-Channel layer, right figures show the input and the zoom-in activation image.

[1]https://github.com/MathGaron/neural-network-visualization–Torch
[2]https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

3 Method

3.1 Dataset-centric

In the Dataset-centric paradigm, the neurons are activated by doing a forward pass, the activation a_j^l of the j^{th} neuron in l^{th} layer is:

$$a_j^l = f\left(\sum_k w_{j,k}^l a_{j,k}^{l-1} + w_{j,k}^l\right)$$

where f is the activation function, w are weights in the neuron network, the sum is over all neurons k in the $(l-1)^{th}$ layer.

Basically this method could be applied to any trained model. We provide a tool to load a pre-trained model, then it will visualize the activation of the convolution filters in the network for an input in real time. The back-end used is Torch(Collobert et al. [2011] or Tensorflow(Abadi et al. [2015]).

3.2 Network-centric

Generating a realistic image from the network is a gradient ascent process that tries to maximize the activation of some particular units. To do so, it is convenient to use regularization technique in the optimization process.

The following optimization problem must be solved for an input image x:

$$x^* = argmax_x(a_i(x) - R_\theta(x))$$

where x is an input image to a network, a_i is the activation function of unit i. Images produced by gradient ascent tend to have high frequencies (see fig.3.1) that needs to be penalized by the regularization function $R_\theta(x)$. We can change the formulation based on the fact that we apply regularization on the output image as well as the output gradient itself:

$$x \leftarrow r_{\theta,1}(x + \eta r_{\theta,2}(\frac{\partial a_i}{\partial x}))$$

where η is a gradient ascent step size, $r_{\theta,1}$ a regularization function applied to the optimized image and $r_{\theta,2}$ a regularization applied to the gradient.

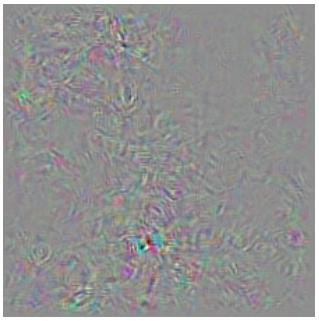


Figure 3.1: High frequency gradient image after gradient ascent.

A common regularization L2 decay penalizes extreme pixels values and is implemented as

$$r_{\theta,1}(x) = (1 - \theta_{decay}) \cdot x$$

Another necessary regularization is to penalize high frequency information. We implement this with a Gaussian blur operation on the gradient. Best results are achieved if the kernel size is randomly chosen at every optimization iteration.

$$r_{\theta,2}(x) = GaussianBlur(x, k)$$

The third regularization method is to compute the norm of each pixel (over red, green, and blue channels) and then set any pixels with small norm to zero. This operation will give us a purely high activation generated object image while removing non important activation.

Another regularization method inspired by deep dream is to add jitter to the input, basically this operation will offset image by a random value. This operation aims to change the fully connected units activity and helps to render full objects. We use the octave idea from deep dream as well, which optimize the image at different pyramid levels.

We also implemented a new regularization step with the gradient propagation process. We observed that even with all the previous regularization combined, the result can converge too fast and output average results. To prevent being stuck in sub-optimal minima, we invert the gradient when the softmax function is saturated. As most of our pipeline got random changes on the output image, this addition largely augment the quality of the results.

3.3 Mixture

We propose to visualize the neurons in the network with Dataset-centric method, and visualize the network by synthesizing an image by Network-centric method. Inspired by both paradigms, we use some specified input images to activate the neurons in the network by a forward pass, then we could find K neurons $\mathcal{N}^K$ that are most activated by the designed input. Instead of using the entire network to synthesize an image, we use the set of $\mathcal{N}^K$ neurons to generate the image by back propagating the gradients. The designed input could be some different type of images that the network was not specifically train for. This tool might be useful to discover the potential ability of the trained networks.

4 Experiment and Explanation

The visualization tool is not constrained by the Deep Learning framework. The front-end is pure python, while the back-end can be Torch or Tensorflow. We use a pre-trained VGG16 model for this work, but any CNN can be used for most features.

The Dataset-centric method requires the input network and an input image to fire the neurons, see fig.2.1. Given an input image or a batch, the neurons get triggered by a forward pass. As we look deeper into the model's filters, we see that neurons get fired by some specific object parts. The proposed framework can load any dataset or an attached webcam to see the activation in near real-time[3].

[3] 15 fps, torch, VGG16, GTX 970M GPU

(a) blackwidow (b) flamingo

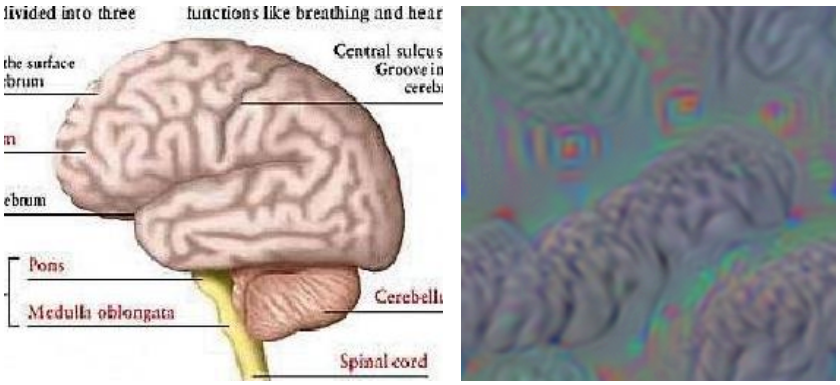
(c) flowerpot (d) monarch

(e) snails (f) volcano

Figure 3.2: Network Visualization. With a pre-trained model (VGG16), given a random noise image, we maximize the Softmax layer.

The Network-centric implementation lets the user synthesize images easily from the network's

last layer (Softmax for VGG), see fig.3.2. We could visualize how the network responses to a kind of images by updating the input image to maximize the response of specified neurons.



(a) (b)

Figure 4.1: Visualization from neurons highly activated by brain images(Fig.(a)). Fig.(b) show the optimized image. While brains are not part of imagenet, those features are probably learned from the brain coral which has similar texture with different colors.

(a) (b)

Figure 4.2: Visualization for neurons activated by faces. Fig.(a) shows faces part, mostly eyes and noses. At some point in the optimization we saw ears and mouths. There is no classes for human faces in ImageNet but probably many pictures with people.

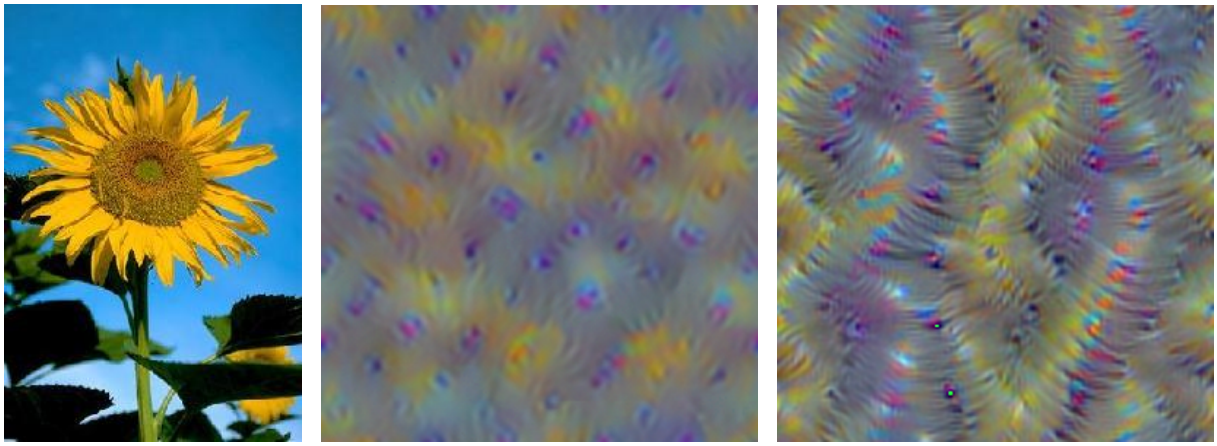
To better understand the ability of neuron network, we use Caltech101 dataset to trigger the VGG16. The Caltech101 dataset contains pictures of objects belonging to 101 categories[4]. About 40 to 800 images per category. Most categories have about 50 images. The size of each image is roughly 300 x 200 pixels. We choose some classes that are not contained in ImageNet.

We use a group of images from the same class to trigger the neurons. We select k filters with the highest activation. Then we maximize the activation of these neurons to optimize a random noise image. The converged result represents what activates most the selected filters, see fig.4.1 and

[4]Dataset link: `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`

fig.4.2. Using filters from early layers tend to give too much emphases on low level textures, while maximizing neurons from fully connected layer tend to render noisy and unrecognizable objects. With these observations, more weights are given to the middle layers which is the best compromise for the current algorithm.

We also visualize the features that learned at different layers in a network, the higher layers usually combine the features from the lower layers, see fig.4.3.



(a) (b) (c)

Figure 4.3: Visualize the activation of sunflower from different layers. This figure shows that at different layer, the network combines different structures. The sun flower is never seen by the pre-trained VGG16. Fig.(b) shows a visualization result only using the lower level neurons to update the random image. It contains basis components of the flower that was seen by VGG16. Fig.(c) is an example generated with higher level neurons, which is able to combine the basis components that could be interpreted as petal-like structures.

5 Conclusion

In this project we implement both Data-centric and Network-centric visualization methods for deep neuron network. Moreover, inspired from both paradigm, we use a group of images to activate particular neurons to synthetize a similar image. This shows the incredible power of representation that the CNNs can model and that they can represent things they were not originally trained for. The current results shows that the network can return us some high level features from the inputs images (nose, eyes, petals) but still can't represent higher level features like the whole objects. The gradient ascent implementation could be improved, and it would be an interesting path to find a regularization that penalize activation contribution from non objects.

References

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*, 1, 2015.

Jeff Clune, Jason Yosinski, and Thomas Brox. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. (Nips):1–29, 2016.

R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *arXiv preprint arXiv:1605.09304*, 2016.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.