Feature Visualization on VGG-16

Final Project

Jing Zhao (jz2786), Mengya Zhao (mz2593)

Columbia University in the city of New York

1. Introduction

Thanks to the power of GPU and its availability of large datasets, neural network becomes increasingly popular. However, it is still not easy to explain neural network by words and understand how it works inside. Thus, implementing feature visualization is very helpful to explore this 'black box' and make neural networks more understandable for people.

In general, during the training process, we usually use backpropagation to optimize the weights. However, for here we used pretrained model VGG16 with saved fixed weights. We start with a gray noise image, then optimized the input pixels with fixed weights and finally visualize the image that can activate the a given layer the most.

2. Model -- VGG16

VGG16[2] is a convolutional neural network with 13 convolutional layers and 3 fully connected layers introduced by Simonyan and Zisserman in 2014. This model is trained on ImageNet database and achieved 90.0% top-5 accuracy for the ILSVRC-2014 competition.

Although it takes a lot of time to train VGG16, this high-accuracy model is easy to understand and apply. Thus, we decide to use it as target model for our following feature visualization work.

3. Methods -- Gradient Ascent

While implementing feature visualization, the main idea is to feed a random picture to a pre-trained model, and generate the output of a certain layer which can maximally activate the neuron. In in this report, we implemented gradient ascent methods and based on that, we also took some techniques to improve the performance of visualization.

3.1 Gradient Ascent

For this project, we first tried to visualize filters in different layers to find what the filters capture. Then we applied gradient ascent to generate images for each layer.

In this filter visualization part, we can find which pattern in a given image that filters are trying to recognize [4]. In other word, how that image activates the neurons in the convolutional layer. With the deeper layer in a neural network, filters can capture much more complicated patterns.

In Gradient Ascent, we keep fixed weights which gained from training process and optimize the input pixels. Specifically, the naive gradient ascent method is to calculate the gradient of a neuron for a specific channel, and change the pixels of the input image by using the calculated gradient. Then repeat this process and here we set number of iterations to 200. It basically as follows:

1. Prepare an initial input image (we use gray scale random noise as default)
2. Choose the layer before softmax and the feature channel to visualize

Iterate:

3. Calculate current scores (we use reduce_mean as the score object)
4. Calculate gradient of neuron value by backpropagation, with respect to input image pixels
5. Update the input image

However, the naive gradient ascent performs not as good as we expected. It may because the gradient run to local maximum are hard to get out [7]. Thus, we improved our algorithms by adding regularizations.

3.2 Regularization

3.2.1 Common Methods: Clip, Decay, Blur and Clip with small norm

a) Clip

Clip means we do preprocessing to the input image which is to ensure its pixels are between 0 and 255[3]. But notice that in the VGG16 model, it has a preprocess tensor to make zero-mean input, so we need to reverse it before doing the clip.

b) L2 decay

L2 decay theoretically discourages large values which means this regularization can avoid some extreme pixel value in images. Therefore, L2 decay regularization can make all pixel values more concentrated and the images look smoother.

c) Gaussian blur

Blur is a gaussian filter which penalizes high frequencies and make the image smoother, considering that the gradient ascent method itself can produce high activations in image.

d) Clip with small norm

Clip with small norm is an idea which proposed by Jason Yosinski et al. The core idea of it is to compute norm of each pixel over RGB channels and set small norm to 0. [1] By applying this way, we can make the visualization focus on the main object and discard those trivial parts.

.

What's more, we tried to do further improvement by applying some other optimizing methods from other's articles on top of the regularization results. For example, recursive computing and Laplacian pyramid decomposition. In this way, we are expecting the images become smoother and features are more recognizable.

3.2.1 Improved Methods

For improved gradient ascent part, we have tried many different ways to improve the performance of naive gradient ascent keeping regularizations applied previously.

First, we add gaussian blur to gradient with different variance and then add all of them at each iteration in the naive gradient ascent method. Through this way, the color of output images would become more pastel.

Then based on this structure, we do this improved gradient ascent recursively, which means we blur the gradient and the input image more than once.

At last, inspired by DeepDream[6], we applied Laplacian pyramid decomposition which computes difference of blurred gradients between every two levels. It shares similar idea with the method we used before. They both increase the low frequencies of gradient and decrease the high frequencies.

4. Result

4.1 TensorBoard

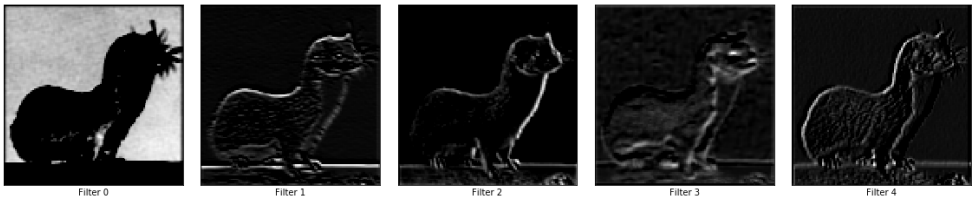
To begin with, we use tensorboard to draw the graph of the pretrained VGG16 model we choose. It is quite straightforward to learn the structure of our model on tensorboard. Also, we can easily view the names as well as dimensions for each block. There are 13 convolutional layers, 4 pooling layers, 3 fully connected layers and 1 softmax in VGG16, with initial input dimension (?, 224, 224, 3).

4.2 Activation of Filters

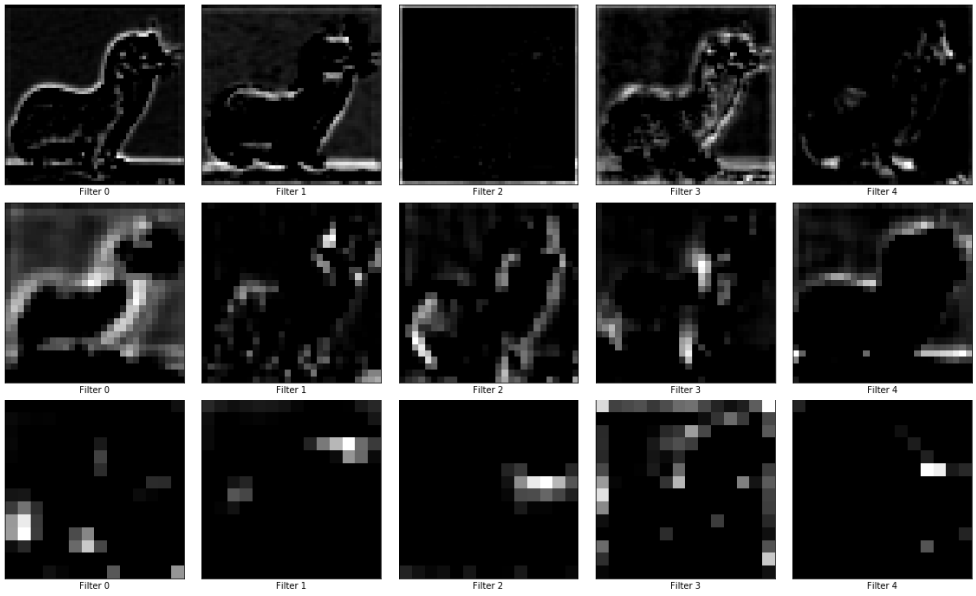
Then by visualizing some filters in a list of layers [conv1_1, conv2_1, conv3_1, conv4_1, conv5_1], we get the activations at a given layer for a given input image of Laska. The following images are activations of first five filters for each layer in the list.

We can see that at first few layers the activation plots are more like our original input, without high-frequency noise or distinguish structures. As we go deeper to the network, the activation plots become "blur" and filters in a layer are learning to activate for different features of the image.

← Conv1_1, filter 0:4



← Conv2_1, filter 0:4



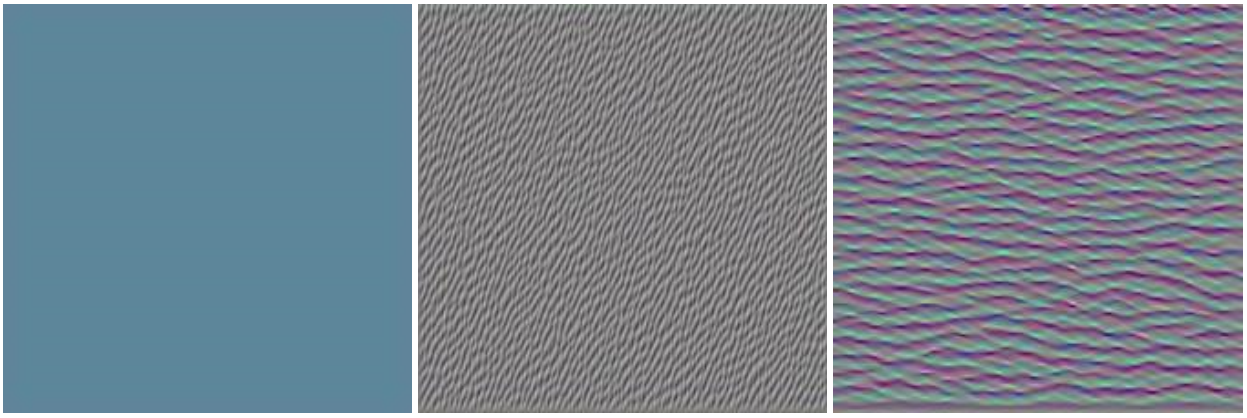
← *Conv3_1, filter 0:4*

← *Conv4_1, filter 0:4*

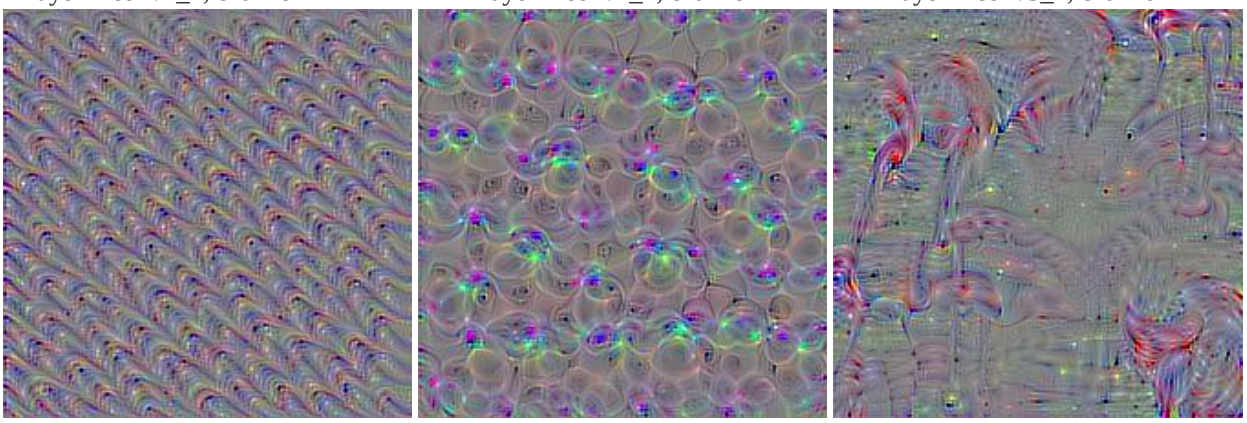
← *Conv5_1, filter 0:4*

4.3 Naive Feature Visualization

In our early experiment, we simply visualize channels in layer conv2_2 and finding very less structures. As what we have done in the filter visualization, we then tried higher layers. It turns out that more structures, more color and more detailed textures are showing up when going deeper. Here are feature visualization images we get by naive model. Five for features in different layers and one image for the last fully connected layer with label 130.



layer = conv1_1, channel = 1 | layer = conv2_1, channel = 1 | layer = conv3_1, channel = 1



layer = conv4_1, channel = 1 | layer = conv5_1, channel = 1 | layer = fc3/BiasAdd, channel = 1

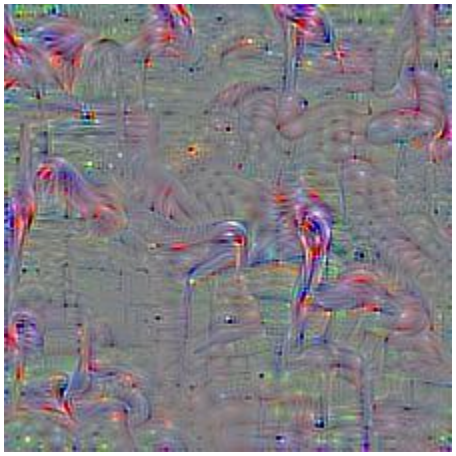
Most of visualization images generated are not recognizable as some certain objects, instead they are just conveying information of what kind of feature can activate this channel the most. For example, the conv3_1 channel 1 may be activated most by texture like water wave or curves, as shown. Feature visualization for conv4_1 channel 1 shows more complex texture, more color included. In the visualization for conv5_1 channel 1, some highlight spots with high frequency appear and the colors become brighter. Textures in the visualization plots for the lower layer conv1_1 and conv2_1 are not obvious.

The label we choose for fully connected is “flamingo” however we can hardly find it on this result. Thus, adding regularizations can help.

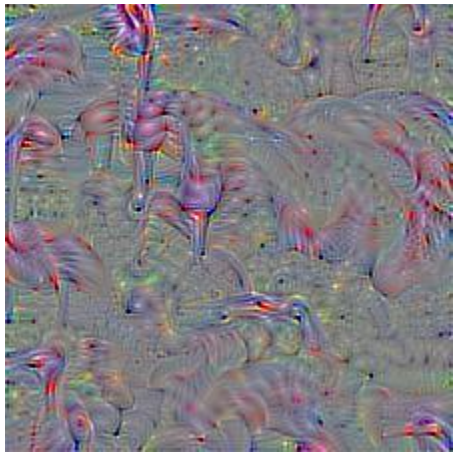
4.4 Regularization

4.4.1 Clipping, Decay, Blur and Clipping with Small Norm

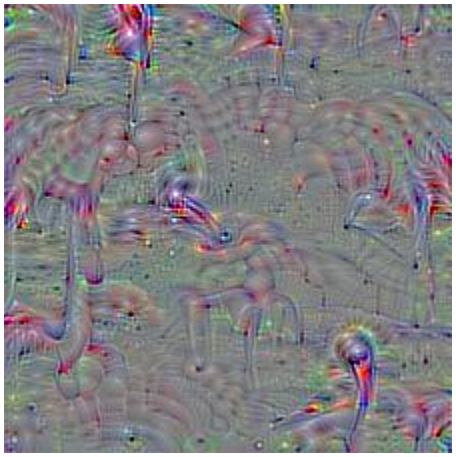
By adding regularization, we are expecting for smooth the image and optimize frequency to make the image distinguishable. The images of a each method are output with keeping all previous regularizations. We apply the methods to layer fc_3 label =130 with order of Clip, Decay, Blur and Clip with small norm.



Clipping



+ Decay



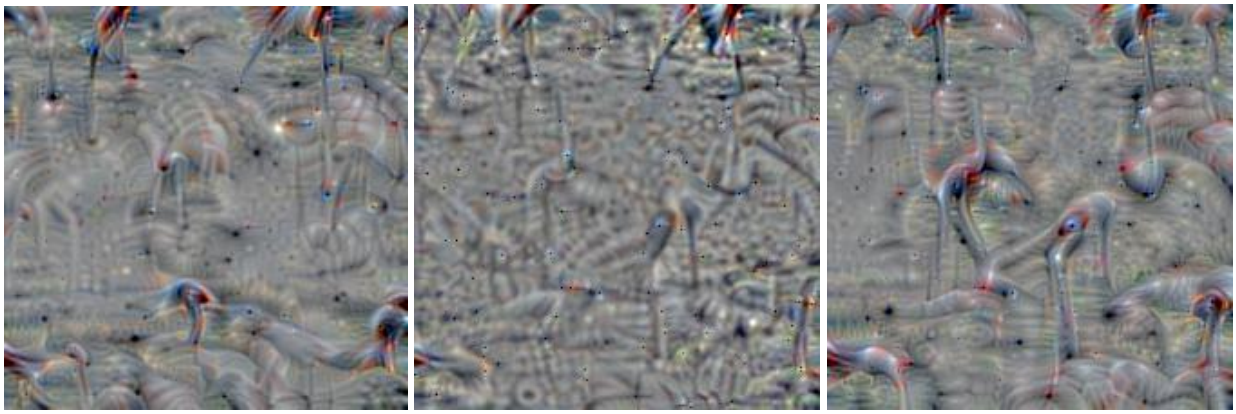
+ Blur

+ Clipping with small norm

Although patterns in the images become somehow more recognizable, the improvement by adding these regularizations are not very obvious.

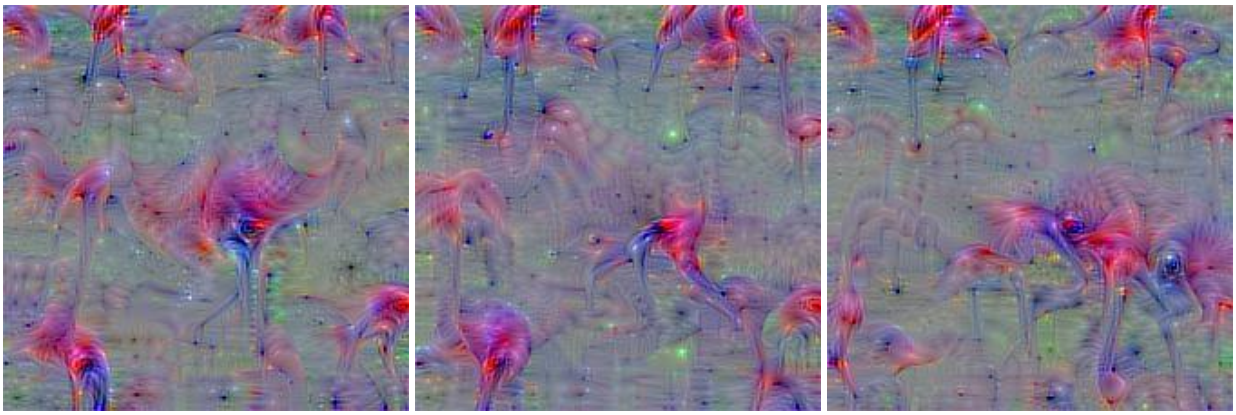
4.4.2 Improved Methods

In the improved methods, we take the post-regularization results of previous part 4.4.1as input here. Then as we expected, the color of output image becomes more pastel and grayish. Based on this, we recursively apply the method and output after 1st and 5th recursive are as shown. We can easily tell the heads, necks and eyes of flamingo in image c).



a) Added Gaussian Blur b) Recursive Gaussian Blur_1 round c) Recursive Gaussian Blur_5 round

Parallelly, we also apply Laplacian pyramid decomposition on post-regularization images, which can increase the low frequencies of gradient and decrease the high frequencies. Comparing with images in 4.2.1 of naïve method, we can see after Laplacian Method the color becomes less grayish. It has been improved significantly, for the low frequency places get enhanced and the high frequency spots are lowered (especially in plot d).



d) Laplacian method with 4 splits e) Laplacian method with 4 splits f) Laplacian method with 4+5 splits

We also discover an interesting result that with splits number = 4, the visualization result of flamingo for Laplacian method are more sensitive with eyes and mouths. While for splits number =5, it turns to be more legs visualized. So, we try overlapping the plots and are expecting for a whole flamingo. Unfortunately, it's not that case. It maybe what we can do in further study to reveal the reason behind.

5. Limits and Further Study

Due to the limitation of pretrained model VGG16, the size of input images must be fixed at (224, 224) and we fail to change it. Therefore, we are not able to implement tiled gradient ascent which is a good alternative of naive gradient ascent to save running time. Similarly, neither downscaling nor upscaling is realizable when applying Laplacian pyramid decomposition and recursive computing. So, for further study, we can try other advanced pretrained models which are more flexible to feed different size of input images, and apply them with some other methods like Deconvent.

References

[1] Yosinski, Jason, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. 2015. "Understanding Neural Networks Through Deep Visualization". *Deep Learning Workshop, 31St International Conference On Machine Learning*.

[2] "VGG In Tensorflow · Davi Frossard". 2017. *Cs.Toronto.Edu*. http://www.cs.toronto.edu/~frossard/post/vgg16/.

[3] "Visualizing Deep Neural Networks Classes And Features – Ankivil". 2017. *Ankivil.Com*. http://ankivil.com/visualizing-deep-neural-networks-classes-and-features/.

[4] "Visualizing Neural Network Layer Activation (Tensorflow Tutorial)". 2017. *Medium*. https://medium.com/@awjuliani/visualizing-neural-network-layer-activation-tensorflow-tutorial-d45f8bf7bbc4.

[5] "Deepdream". 2017. *Alexander Mordvintsev*. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/tutorials/deepdream.

[6] "TensorFlow Tutorial #14_DeepDream". 2017. *Magnus Erik Hvass Pedersen*. https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/14_DeepDream.ipynb

[7] "Visualizing GoogLeNet Classes". 2017. *Audun M. Øygard*. https://www.auduno.com/2015/07/29/visualizing-googlenet-classes/