GR5242 Final Project Report - No.3 - Feature visualization

YitongHu (yh2875)

1. Introduction

This project is for better understanding how neural network work by doing feature visualization. To be more specific, I would explore the following questions during the project, what particular features various layers represent, given parameters of a trained model, what input image would drive a particular unit/filter/layer's activation the most, and try different optimization method so as to make the images more recognizable.

Data

I mainly used two datasets in this project. MNIST and SVHN. The motivation for choosing them is, firstly, they are good dataset to start, especially with a lot of materials to refer to. Second, it's a consideration of CPU's computational difficulties, these two datasets are not that large and need lot of time to train a model. Another benefit is that these two datasets are all labeled as number digits, one with color channels while the other not, so it's meaningful to compare them.

Related work before coding

Before coding, my main work is focus on reading Google's post about feature visualization[2] and its citations. Besides reading the whole article talking about how to do the visualization, the direction we can explore, the problem we would meet as well as the solutions, I also read the 19 citation papers roughly.

In the paper *Visualizing Higher-Layer Features of a Deep Network*[3], it mentioned maximizing the activation which is the core idea to do the feature visualization. Based on this idea, I made my goal clearly which is to find what input image would drive a particular unit/filter/layer's activation the most. Another paper *Intriguing properties of neural networks* cited in the google post mentioned supplemental perspective, besides treat neuron as a unit to understand neural nets, in real life, combinations of neurons actually work together to represent images in neural networks[4]. This inspired me to see the interaction between neurons while visualizing.

Google post also mentioned when visualizing features by optimizing an image to make neurons fire, the problem I would met is ending up with a kind of neural network optical illusion — an image full of noise and nonsensical high-frequency patterns that the network responds strongly to[2].

Several papers mentioned solutions which can be concluded into three main families of regularization options.

The first one is called Frequency penalization. The paper *Understanding deep image representations by inverting them* introduced some regularisers in its second section, it suggested to explicitly penalize variance between neighboring pixels to deal with high-frequency features[5]. Another *paper Deep neural networks are easily fooled: High confidence predictions for unrecognizable images* introduced several method based on frequency penalization, like adding regularization, using pseudo regularization (set some pixels to 0 or use decay) and blurring the image at each optimization step[6]. A html blog mentioned a way the improve the above method as using a bilateral filter, which preserves edges, instead of blurring, but just idea with no detailed formula or code here[7]. I learned principles of bilateral filter from the other paper as a image processing skill[8].

The second one is Transformation robustness. The idea is easy to understand, during our trying to find images activate the optimization target highly, we can change(jitter, rotate or scale) the image before applying the optimization step. In the google post, it mentioned this adjust maybe effective when combined with a more general regularizer for high-frequencies. Like in paper *Deep Inside Convolutional Networks: Visualizing Image Classification Models and Saliency Maps*, it used image jittering, based on zeroing-out random parts of an image while doing the visualising[9]. Transformation robustness is more like a auxiliary trick to process the image, in order to optimize what we see as the result.

The last one is Learned priors whose idea is to learn a prior and jointly optimize for the prior along with your objective[10]. I gained a rough idea about its principle but have no idea about implement, no enough easy-to-implement instruction findings here.

The challenge of the project

The challenge of this project is to incorporate regularization, such as natural image priors, to generate optimized input that can be interpreted. By now, the academic understanding of the various regularization terms, however, is still limited and may result in unexpected outcomes. Also, in order to qualify its interpretability, we have to manually look at the output of each feature activation. Finally, the ideal training process requires tremendous computational power and the use of GPU which can be difficult to navigate.

2. Experiment

Considering the computational difficulties, I tested on a 3-layer convolutional neural network (conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> pool3 -> fc1 -> dropout -> fc2) using the MNIST and SVHN dataset.

My first intuition is to visualize particular features various layers represent. I used a certain test image as the initialize image and direct visualized for maximizing activation of first, second and third layer.

Figure 1 - 1st convolutional layer

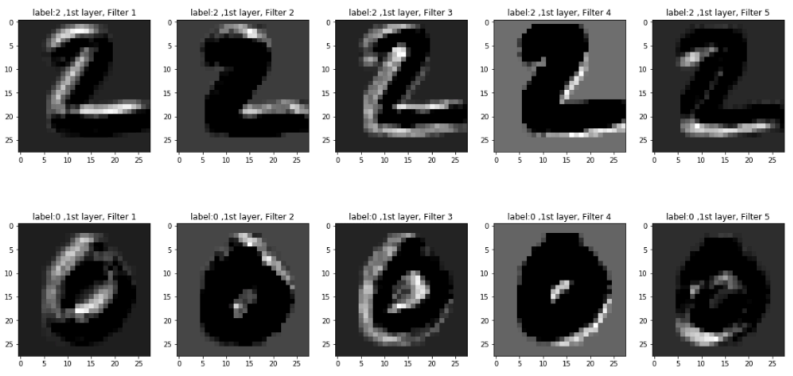


Figure 2 - 2nd convolutional layer

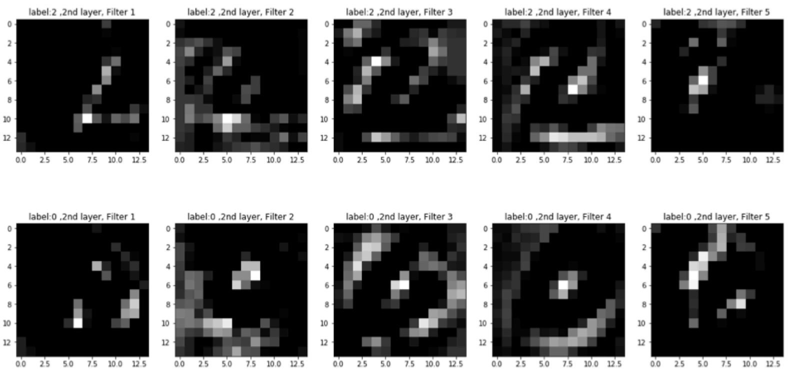
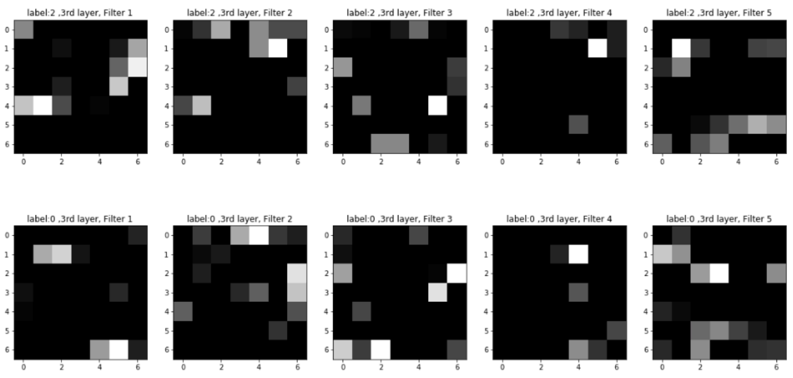


Figure 3 - 3rd convolutional layer

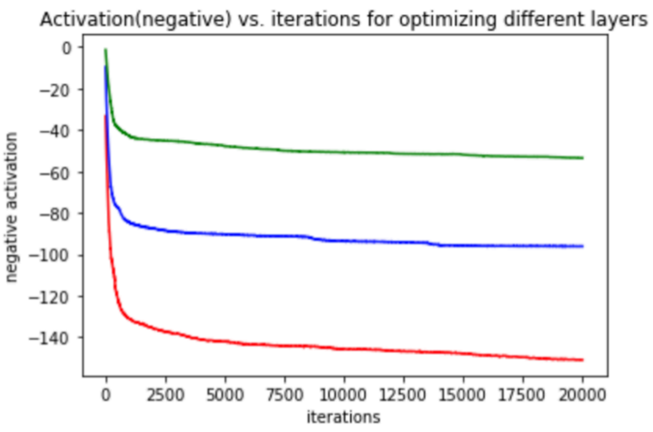


We can see the 'image' (actually it's the activation value) above become more and more magnified and blurred, which is reasonable due to the convolution. The machine, the network can correctly classify the image with these unrecognizable pattern, but we can still not figure out what images the network wants to find. So I tried other methods.

The second idea is inspired by the activation maximization method mentioned in a paper Visualizing higher-layer features of a deep network published in 2009[4] (the third citation of the Google's post[1]). I used the trained net by saving all the parameters, then generated my initializing image by truncated normal and separately maximized the activation of different filters in each layer to found the corresponding images.

For the iteration steps to maximize activation, I checked the loss function with iteration steps increasing for 1st,2nd and 3rd layers while the optimizing process, as the following image shows. I found for whatever layers, after 20000 steps, the loss function just leveled off, so I chose 20000 as my training steps to optimize the activation function.

Figure - 4



I found that the images that maximumly activating lower level layers demonstrate great and clear pattern for both dataset, like edges at different direction, horizontal, vertical or diagonal, and they also have different sizes, like wide stripe and narrow ones. This images could be produced without any optimization.

Figure 7- The images that maximumly activated 1st layer of net on MNIST dataset

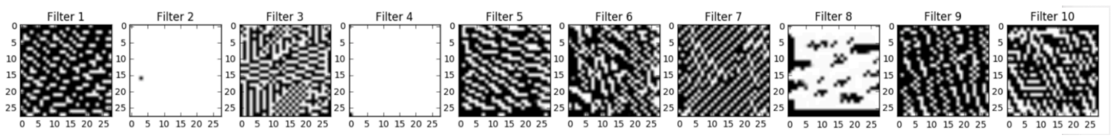
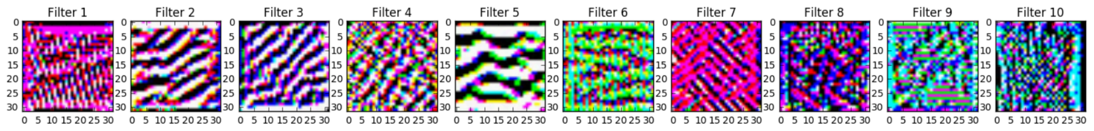


Figure 8 - The images that maximumly activated 1st layer of net on SVHN dataset



Another thing I did for the first layer is to verify the interaction between neurons. As the google post suggested, besides individual neuron as a way to understand neural nets, we can also explore the combinations of neurons working together to represent images in neural networks. Individual neurons are the basis directions of activation space, and it is not clear that these should be any more special than any other direction. So here I tried exploring interaction between filters(neurons) by jointly maximizing the activation of two filters, with weight changing.

Figure 5 - The original images that maximumly activated filter 6 and 9 in the 1st layer

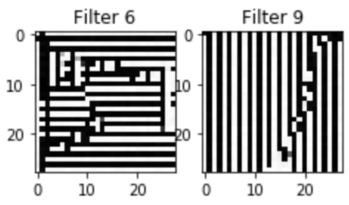
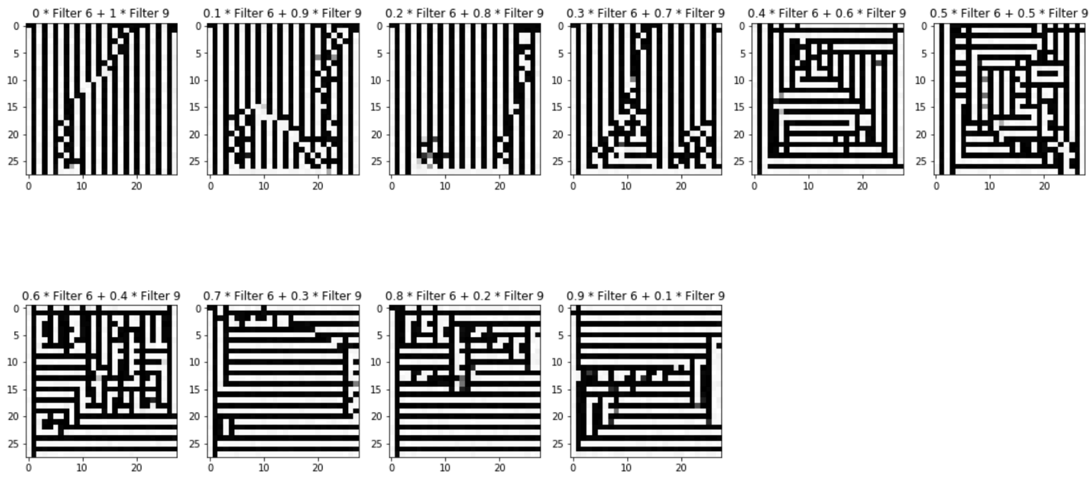


Figure 6 - The images that jointly maximumly activated filter 6 and 9 with different weights


0 * Filter 6 + 1 * Filter 9
0.1 * Filter 6 + 0.9 * Filter 9
0.2 * Filter 6 + 0.8 * Filter 9
0.3 * Filter 6 + 0.7 * Filter 9
0.4 * Filter 6 + 0.6 * Filter 9
0.5 * Filter 6 + 0.5 * Filter 9
0.6 * Filter 6 + 0.4 * Filter 9
0.7 * Filter 6 + 0.3 * Filter 9
0.8 * Filter 6 + 0.2 * Filter 9
0.9 * Filter 6 + 0.1 * Filter 9

According to the above, we can conclude that the filters have interactions, when jointly optimized them, the effect on images is generating their pattern combined with similar styles. The weights for optimizing each filter are mainly reflected on the area of two patterns.

While I used the same method for visualizing intermediate feature layers, directly visualizing the corresponding filters in that convolution layer, does not work for any layer beyond the first one. No more complex texture founded but the noise pattern appeared in some of the filter. Based on the basic activation maximization, I tried several attempt to optimize the feature visualization. Like tunning the image initializer & learning rate, adding regularization term and modifying image after each optimization using clipping, decay or Gaussian blur. But those optimizing method does not have a good effect for intermediate feature layers. I've read a paper, it mentioned we could feed activations of the intermediate layers of CNN to a deconvolution network, so as to map the activations to the pixel space. The deconvolutional network can be thought of as applying the reverse steps of filtering and pooling, in order to map the features back to the pixel space[14]. I tried but failed due to some coding problem, but this can be a future direction which worth a try.

For the output layer, I tried some regularization term or adjust the images like clipping, blurring or use decay at each gradient step. I found regularization term worked better for MNIST and adjusting initial images almost has no effect. While adjusting initial images worked pretty good on SVHN. I guess the reason is that decay avoids bright pixels with very high values in red, green or blue here, so we did not see the complex but messy pattern with different color full of the images, but the function got weaken in the dataset without color.

Figure 9 - The images that maximumly activated output layer of net on MNIST dataset

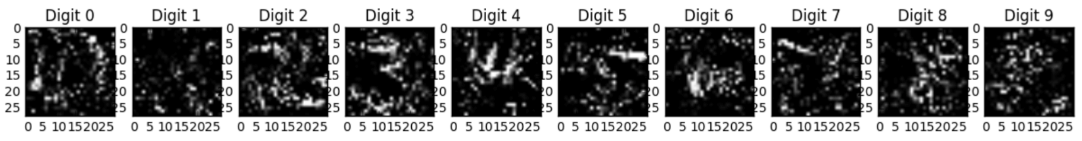
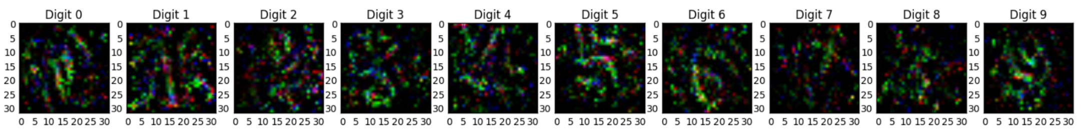


Figure 10 - The images that maximumly activated output layer of net on SVHN dataset



According to the above images, some of the images produced could be recognized as some digits. It shows that the neural network had clearly learned to recognize the general shape of some of the digits, while it was impossible to see how it recognized other digits, since for some digits, it's still unrecognizable.

The reason is perhaps that the network tries to recognize all of the digits simultaneously, and it has found that certain pixels often determine whether the image shows one digit or another. So the neural network has learned to differentiate those pixels that it has found to be important, but not the underlying curves and shapes of the digits. Actually, it's quite similar way in which human recognizes the digits. Another possibility is that the dataset contains mis-classified digits which may confuse the neural network during training. I've checked by printing some of the data and found some of the digits in the dataset are very hard to read even for humans, and this may cause the neural network to become distorted and trying to recognize strange artifacts in the images. Also, I found that these images vary each time I run the code, so it has some randomness.

Further direction

For this project, I only explored the images that maximize the features of the convolutional layers and the final layers, but not fully-connected layer, this could also be a further direction. I could also try adding more layers (convolutional layers or fully-connected layers) and find the input images that maximize their features, so that I could check whether it's useful to add more convolutional layers than three. Actually I tried 2 layers and 3 layers seems to be slightly better.

If there is less computational difficulties, I would explore in those further directions. I could plot the images for all features in each layer instead of just the first 10 features, and see how many of them appear to be unused or redundant. And also check what happens if lowering the number of features in that layer and train the network again, does it still perform just as well.

For the optimization part, there are large space to improve. Like I mentioned, there are some ways to try to improve intermediate layers' feature visualizing, like feed activations of the intermediate layers of CNN to a deconvolution network, so as to map the activations to the pixel space. For the three families regularization, frequency penalization, transformation robustness and learned priors, there are a lot of more work to try.

Experience I learned

First I learned is the importance of GPU. Since I'm doing the project myself, waiting for the code to see the results is so tiring and postponed the whole process. Second, there is a long way between understand a paper's idea and know how to implement it. Coding them in the architecture of Tensorflow also added more difficulties. But at the beginning, this is what I did not expect, so I spent a lot of time in searching for materials and reading paper, and have no enough time to implement what I want, so the thing I implemented actually is so limited. This is also experience gained, that is plan time reasonably.

All in all, reading paper and trying implement as well as optimize the feature visualization process are all interesting. Since every change I made in the programming or parameters can be immediately reflected in the images I produced, I was always inspired by a little improvement of the images(become more recognizable). Moreover, Some of my expectations are verified undoubtedly, while other what I thought the images should be just did not happen, exploring the reason and explanation is also interesting exploring. Therefore, I would say it's a so meaningful project.

Reference

[1] http://ufldl.stanford.edu/housenumbers/

[2] https://distill.pub/2017/feature-visualization/

[3] Visualizing higher-layer features of a deep network [PDF] Erhan, D., Bengio, Y., Courville, A. and Vincent, P., 2009. University of Montreal, Vol 1341, pp. 3.

[4] Intriguing properties of neural networks [PDF] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R., 2013. arXiv preprint arXiv:1312.6199.

[5] Understanding deep image representations by inverting them [PDF]Mahendran, A. and Vedaldi, A., 2015. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5188--5196. DOI: 10.1109/cvpr.2015.7299155.

[6] Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. [PDF] Nguyen, A., Yosinski, J. and Clune, J., 2015. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 427--436. DOI: 10.1109/cvpr.2015.7298640.

[7] https://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html

[8] Fast Bilateral Filtering for the Display of High-Dynamic-Range Images. [PDF] Fr´edo Durand and Julie Dorsey.

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

[10] Plug/play generative networks: Conditional iterative generation of images in latent space [PDF] Nguyen, A., Yosinski, J., Bengio, Y., Dosovitskiy, A. and Clune, J., 2016. arXiv preprint arXiv:1612.00005.

[11] https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/examples/tutorials/mnist/

mnist_deep.py

[12] https://en.wikipedia.org/wiki/Gaussian_blur

[13] http://dev.theomader.com/gaussian-kernel-calculator/

[14] Visualizing and Understanding Convolutional Networks [PDF] Matthew D. Zeiler and Rob Fergus