How the Deep Convolutional Networks see the World

Yili Tang, Jingtian Yao, Chenfei Jiang, Ziqi Liu

Abstract

Feature visualization of deep Convolutional Networks (ConvNets) can help us understand how ConvNets select features and how neuron interacts with each other. This paper describes a set of concrete techniques for feature visualization on ConvNets built on MNIST and ImageNet dataset. We introduce a visualization technique, saliency map, that gives insight into the convolutional layers and fully connected layers of a complicated ConvNet and generalize this technique to relatively simple ConvNets. It turns out that with the complication of images and network architecture, it becomes more difficult for human to interpret the visualization. As the layer goes deeper, more complex features such as invariant patterns are generalized. Although the visualization on fully connected layers of ConvNets solving classification problems remains a large part of unexplainable features, it does to some extent capture significant features of objects that help distinguish this class from another.

Part one: Introduction

1.1 Introduction

Due to its high performance and scalability, deep Convolutional Networks (ConvNets) now has significant stature in computer vision, including object detection, object recognition[1], image classification[2], image recognition[3] and segmentation. As it is never satisfying treating the model as a black box, the interpretability of the architecture becomes particularly relevant.

Feature visualization and attribution are two main approaches for interpreting those networks. By visualizing features generated in different layers, we attempt to understand what features are extracted and how each neuron interacts with each other.

1.2 Related work

Previous work on feature visualization are mostly limited to the first layer where projections to pixel space are possible. In higher layers, alternate methods are needed. To find the optimal stimulus for each unit, [4] gradient ascend is performed in image space to maximize the unit's activation. This requires a careful initialization while does not give any information about the unit's invariances. To overcome this shortcoming, the Hessian of a given unit may be computed numerically around the optimal response, giving some insight into invariances. [5]

However, for higher layers, the invariances are extremely complex so are poorly captured by a simple quadratic approximation. Girshick et al. show visualizations that identify patches within a dataset that are responsible for strong activations at higher layers in the model. [6] Recently, the problem of ConvNet visualization was addressed by Zeiler et al. [7] They proposed the Deconvolutional Network (DeconvNet) architecture, which aims to approximately reconstruct the input of each layer from its output for convolutional layer visualization.

1.3 implementation details

Our visualization experiments were carried out using a single deep ConvNet, using MNIST dataset (50M) to train the simple version. For the final implementation, training image(82.9M) were selected from the ImageNet dataset and labelled into 5 classes: gold fish, husky, lemon, ostrich and snake. Our ConvNet is a modification of the model built by Visual Geometry Group, University of Oxford. The weight layer configuration is: conv-conv-conv-conv-full-full, where each conv represents a combination of convolutional-maxPooling-normalization and full stands for fully-connected layer. On validation dataset, it achieves an accuracy of 0.801, not far from the performance of VGG model (0.801 accuracy).

Part two: Simple CNN Visualization

2.1 Model Summary

Our model is built on MNIST dataset, which contains 60000 28 * 28 grayscale handwritten digit images of 10 classes. The test set contains 10,000 images, which can be loaded from Keras directly. [8]

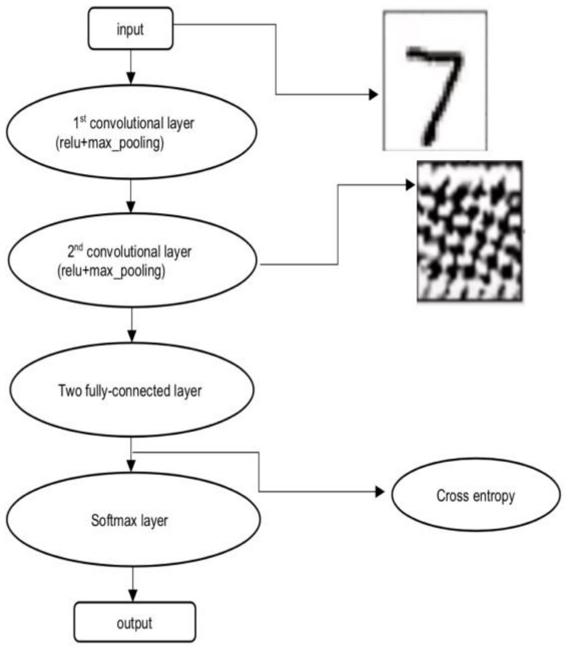
input
1st convolutional layer (relu+max_pooling)
2nd convolutional layer (relu+max_pooling)
Two fully-connected layer
Softmax layer
Cross entropy
output

Fig2.1 model architecture

Architecture of the model is shown above, which contains two convolutional layers and two fully connected layers. It maps the input image array to a probability vector over 10 different classes. The detailed information for the model layers is as follows:

- Convolutional layer: The filter sizes of both convolutional layers are 3 * 3 with stride size 2. Both layers apply Relu as the activation function. The output channels of the first and second convolutional layers are 32 and 64 respectively.
- Max Pooling layer: Each convolutional layer is followed by a max pooling layer with window size 2 * 2 and stride size 2.
- Fully Connected layer: Output size of two fully connected layers are 128 and 10 respectively with Relu and Softmax as the activation function. dropout method is implemented here with 30% dropout rate.

The model is trained using cross-entropy as loss function and Adam as optimizer. After training, the model accomplishes the final training accuracy of 0.991 and testing accuracy of 0.982.

2.2 Visualization

Number 7 is chosen as the input image and outputs of two convolutional layers are visualized by plotting directly. Since the number of output channels are 32 and 64 respectively, there are 32 and 64 images within the 8 * 8 grid shown as below:

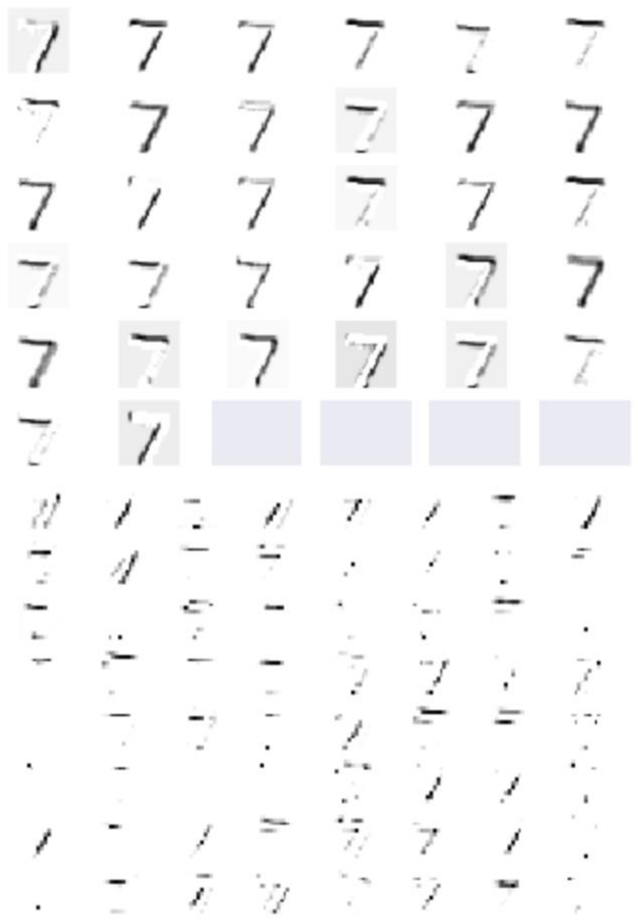

Fig 2.2 cnn layer visualization based on number 7

According to the upper output plot, the first convolutional layer simply creates several variations of the input image, as if light shining from different angles and casting shadows in the image. From the lower output plot, the second convolutional layer seems to detect certain lines and patterns, which are less sensitive to local variations in the original input images. Meanwhile, some neurons seem to have limited contribution to the next layer since there is seldom information shown on the plot. While it is still implicit for us to understand the whole mechanism, some functionality may be similar to the way human eye works.

In short, considering the simplicity of both image and network architecture, we could visualize what each convolutional layer learned by plotting their outputs directly. However, as shown above, although similarities do exist between the way neural network and human beings "seeing" the world, there still are some ambiguous parts for us to understand how neural network actually works.

Part Three: Visualization with gradient saliency map based on VGG16 model

3.1 Overview

(1) Method overview: since visualization is more difficult with neural network becoming deeper and more complicated, more advanced methods are required to achieve our goal. In this paper, we apply a method called gradient based saliency map. The general idea is to fix the weight and bias values of the pre-trained model as constant and take gradient of the loss function based on initialized input image X. By carrying out the gradient ascent back-propagation optimization, we are able to find an input image which maximizes the neuron activity of interest in the image space. [9][10]

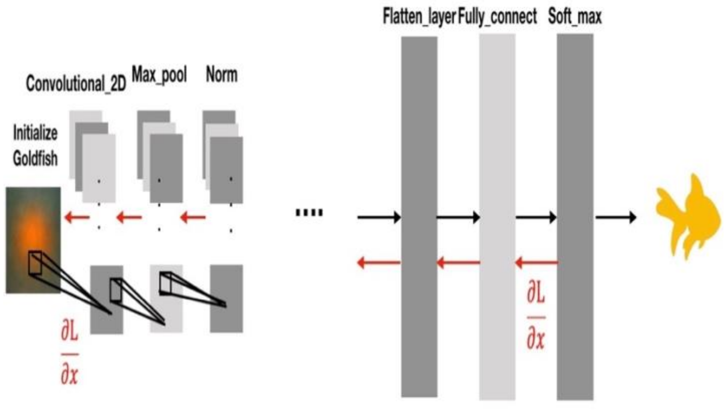

Fig3.1 Method architecture

(2) Pre-trained model overview: We implement above method to the VGG16 pre-trained model on ImageNet dataset. VGG16 is a convolutional neural network architecture named after the Visual Geometry Group from Oxford. To this day, it is still considered as an excellent vision model. The pre-trained model has also been saved in Keras framework and we are able to load it directly.

3.2 Application to the first, second and final convolutional layers

This part aims at finding an input image which maximizes the neuron activity in the convolutional layer. Firstly, the 224 * 224 * 3 input images are initialized by generating random number following the standard normal distribution. Then, the loss function is set to be the mean of each channel's output values. Thirdly, gradient ascent method is applied iteratively to optimize the loss function. The number of iterations is an important tuning parameter to ensure that loss function is converged. Since the converge speed of each channel is different from each other and the standard for converging is hard to well defined, it's challenging for us to choose an appropriate value. In the end, by comparing output under different iteration number, we empirically set the iteration number as 20. During iteration process, we simply ignore channels which get stuck in 0 and skip to the next ones. Finally, we visualize patterns of input that maximize the loss function of channels within the first, second and final convolutional layers in VGG16 model. The results are shown below:

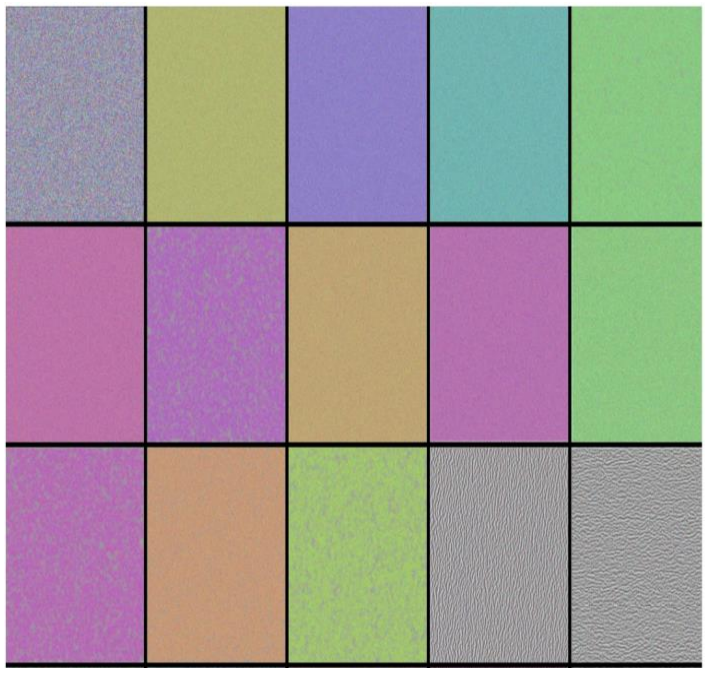

Fig3.2.1: A few of 64 channels within 1st convolutional layer

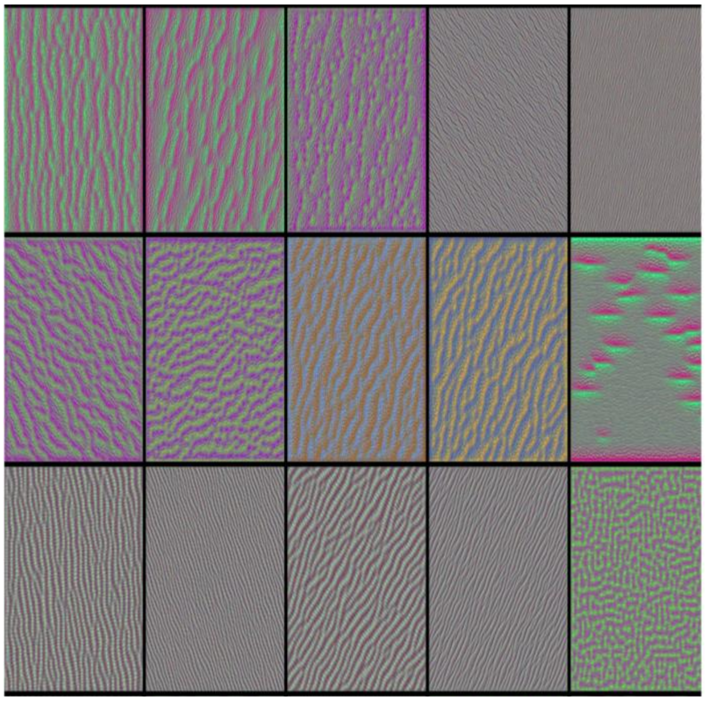

Fig3.2.2: A few of 128 channels within 2nd convolutional layer

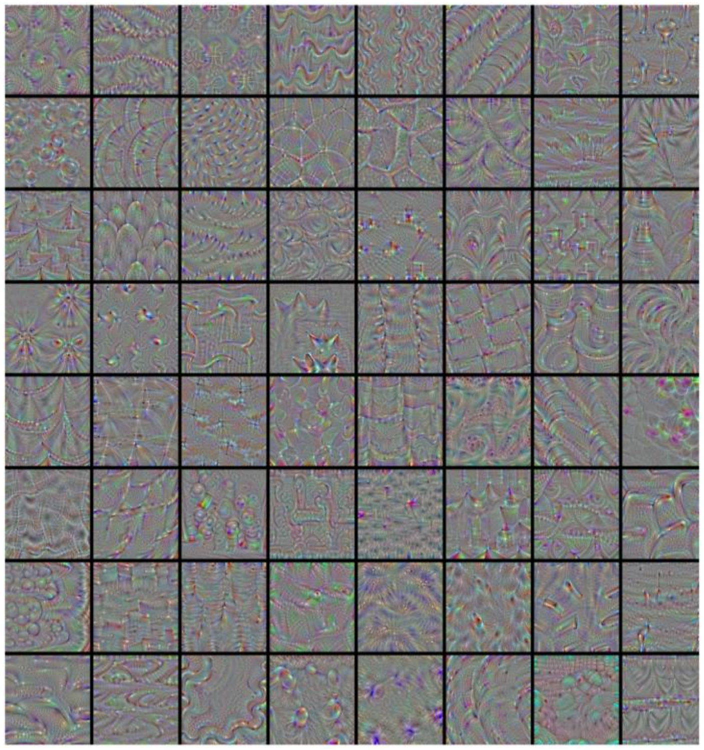

Fig3.2.3: A few of 512 channels within last convolutional layer

Based on the visualization, it's shown that features captured by the first convolutional layer are mainly color (RGB) information. Starting from the second convolutional layer, certain simple patterns seems to be captured by some of channels. With layer getting deeper, patterns become more and more complicated. In the last plot, some textures similar to certain part of those objects which network is trained on can be recognized, such as feathers, eyes, etc. However, meanings of many other patterns shown in some channels remain a mystery.

3.3 Application to the final fully connected layer

In this part, an input image which maximizes the neuron activity corresponding to certain image class at the final fully connected layer is attempted to be found. Although the general idea and steps are basically the same as previous part, a few adjustments need to be made.

(1) Loss function: let $Sc(I)$ be the score of the class c, computed by the classification layer of the convolutional neural network for an image I. We would like to find an L1-regularized image with the highest score Sc :

$$arg\ max\ S_C(I) - \alpha \parallel I \parallel \tag{1}$$

A locally-optimal I can be found by back-propagation. It should be noted that we use the class scores Sc before normalization, rather than the class posteriors, returned by the softmax layer.

(2) Initialization: besides random initialization as previous, we also initialize our input image by taking the mean of all images' pixel values which belongs to the class c.

(3) Tuning parameters: the step-size lambda and regularization coefficient alpha are two important tuning parameters. By comparing output based on different combination of these two, we empirically set the step-size as 0.5 and penalized coefficient as 0.1.

The results are shown as below:

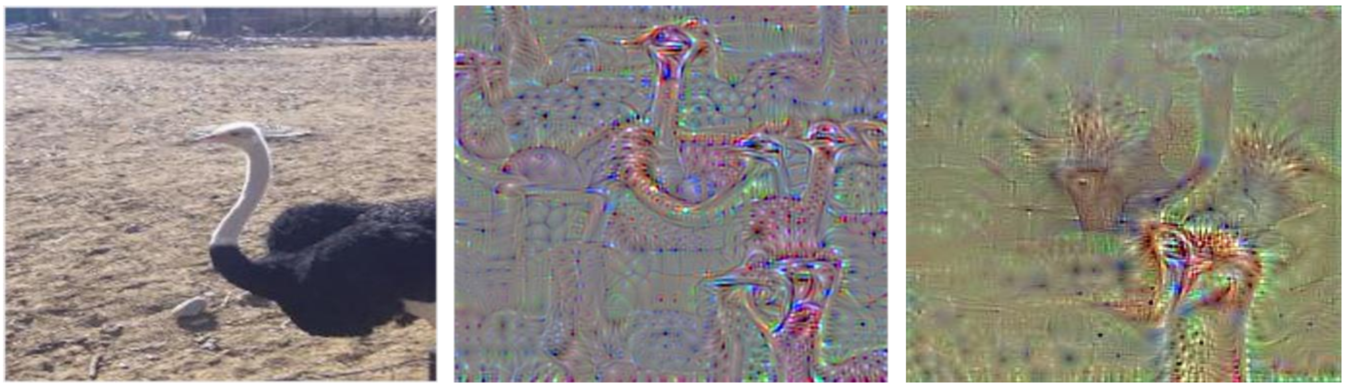

Fig3.3.1 ostrich: original image and saliency map result

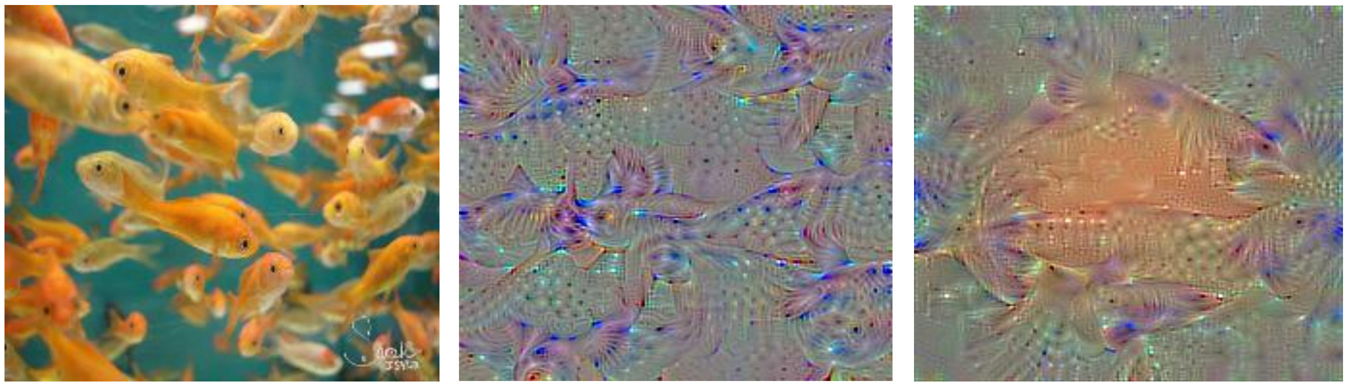

Fig3.3.2 goldfish: original image and saliency map result

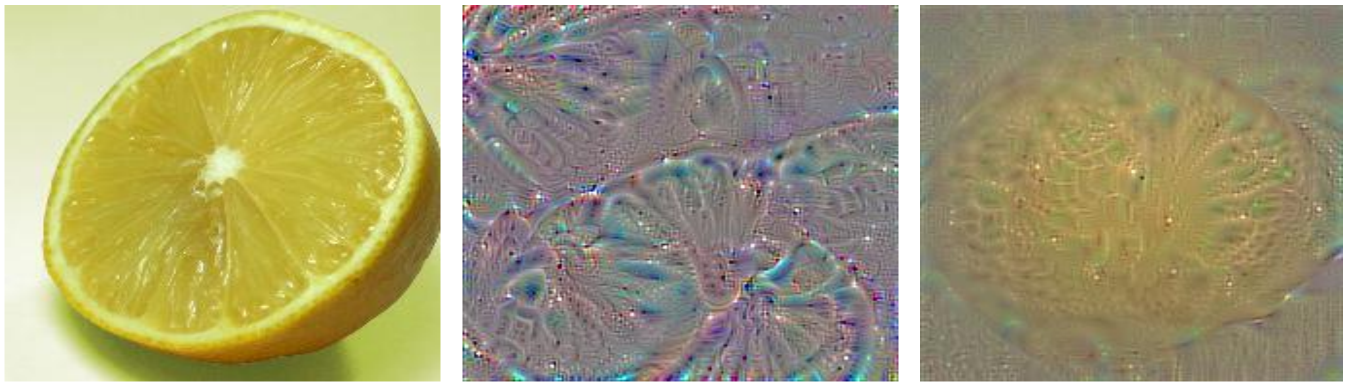

Fig3.3.3 lemon: original image and saliency map result

(Note: The left side is the original image; middle is saliency map with random initialization; right is with mean initialization)

As the result shown above, saliency maps with both initialization methods show important features of the original image class. Compared to the random initialization, mean initialization captures more color features and losses some texture patterns instead. However, both saliency maps include some noise unavoidable.

Part Four: Method generalization to self-building model

4.1 Model Summary

In this part, our own deep CNN model is built to test the method generalization ability. We select 5 classes from ImageNet dataset instead of using all categories to train and test the model. These 5 classes are goldfish, Siberian husky, lemon, ostrich and green snake. On average, over 600 images are collected for each class and the size of total training and testing data are 2312 and 578 respectively.

The model we built contains four convolutional layers, each followed by a max pooling layer. And finally, two fully connected layers. Inside the convolutional layer, we implement batch normalization on the output of a previous activation layer to reduce internal covariate shift. Some of the related details are provided as below:

- Convolutional layer: The filter sizes of all convolutional layers are 3 * 3 with stride size 2. Relu is applied as the activation function on all the layers. The output channels of the four convolutional layers are 32, 64, 128 and 128 respectively.

- Max Pooling layer: Each convolutional layer is followed by a max pooling layer with window size 2 * 2 and stride size 2.

- Fully Connected layer: output size of two fully connected layers are 128 and 5 with Relu and Softmax as the activation function respectively. Dropout method is implemented as well with 30% dropout rate.

Our model is trained with cross-entropy as the loss function and Adam as the optimizer. Consider the small size of our datasets, only 5 iterations are implemented during the training period to prevent overfitting. Finally, the model achieved a training accuracy of 0.879 and test accuracy of 0.801. Which is not bad even compared to the VGG model (with 0.927 test accuracy).

4.2 Application to the final fully connected layer of the self-building model

The gradient based saliency map method is then applied with random initialization and the results are shown below:

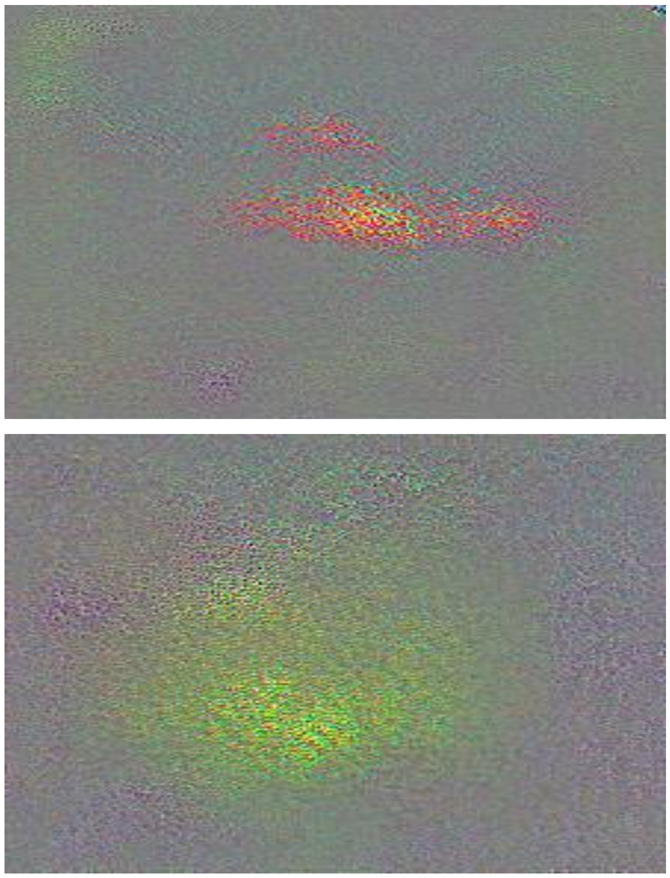

Fig4.2 Saliency map result for self-building model (upper: goldfish; lower: lemon)

Saliency maps with random initialization method do show certain color and outline features of original image class, which indicates certain generalization ability of the method. However, much more texture and other detailed information is needed for us to recognize the true class of each picture precisely.

Part Five: Discussion

We attempt to explain how those convolutional neural network models work by visualizing the output of different layers. While some patterns are far from random and kind of interpretable, a large part of the inside mechanism remains to be explored.

5.1 Visualization of convolutional layers

From the visualization of the convolutional layers, we're able to make the following observations.

(1) For simple images and network architecture, we plot the output directly. It's shown that the features generated by each layer are mainly colors, overall shapes and significant patterns. These functionalities may be similar to the way human eye works.

(2) With the complication of images and network architecture, it becomes more difficult for us to implement the visualization. Using the gradient saliency map, we are able to interpret some of those patterns. For example, kind of like a routine, the first layers always capture the color information. As the layer goes deeper, more complex features such as invariant patterns are generalized.

(3) As a matter of fact, some of the filters are quite similar to each other, only with different colors or rotated by certain degree. This observation indicates that some neurons have limited contribution and we could potentially compress the number of filters. For example, by finding a way to make the convolution filters rotation-invariant. However, since we can never fully understand the effect of each channel, this compression may potentially impair the model performance, which means we need to be extremely cautious.

(4) Some of the invariant features are certain part of objects which network is trained on (feathers, eyes etc.) while others are difficult to recognize. Which is to say, although some rules can be found, the whole mechanism is still quite implicit for us to understand.

5.2 Visualization of final fully convolutional layer

Based on the visualization of the final fully convolutional layer, our findings are as below.

(1) Implemented on the VGG model, saliency maps do show some important features of the original image class. With random initialization captures more texture patterns while mean initialization reserves more color information. Due to the random characteristic of the initialization, the result can be quite unstable. However, the pattern is kind of consistent with different initialization.

(2) As the object becomes complex, more noise may be introduced and thus make the visualization less interpretable. Although some pattern may actually relate to the object class (e.g. bridge appears in the picture of dog class may due to the fact that dog frequently stand on the bridge in the training pictures), others may simply be a result of noise. Thus, we only select five classes from the original dataset to avoid unnecessary noise and interaction.

(3) To achieve better performance, we can possibly optimize the regularization method. For example, instead of only penalizing on the final conventional layer, penalization on each convolutional layer can be implemented. Also, the best choice of regularization formula requires further exploration.

(4) A large part of the visualized feature remains unexplainable for us. This is possibly due to the limitation of our visualization methods. Maybe it loses some important information those neural networks captured or how each layer works or interacts at a higher level. Or more likely, those neural networks may actually see the world in a much different way from us humans. After all, human perception is fundamentally sequential and active, highly integrated while neural networks are relatively static and passive. Either way, deeper understanding of the whole mechanism remained to be seen.

Reference

[1] Agrawal, Pulkit, Ross Girshick, and Jitendra Malik. "Analyzing the performance of multilayer neural networks for object recognition." Computer Vision–ECCV 2014. Springer International Publishing, 2014. 329-344.

[2] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. 2012.

[3] D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In Proc. CVPR, pages 3642–3649, 2012.

[4] Erhan, D., Bengio, Y., Courville, A., Vincent, P.: Visualizing higher-layer features of a deep network. Technical report, University of Montreal (2009)

[5] Le, Q.V., Ngiam, J., Chen, Z., Chia, D., Koh, P., Ng, A.Y.: Tiled convolutional neural networks. In: NIPS (2010)

[6] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524 (2014)

[7] Hinton, G.E., Osindero, S., Teh, Y.: A fast learning algorithm for deep belief nets. Neural Computation 18, 1527–1554 (2006)

[8] https://keras.io/datasets/#mnist-database-of-handwritten-digits

[9] Simonyan, K., Vedaldi, A., Zisserman, A.: Deep Inside Convolutional Networks: Visualizing Image Classification Models and Saliency Maps. arXiv: 1312.6034v2 (2014)

[10] https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html