Visualizing and Understanding Convolutional Networks

Richeek Das

April 27, 2020

1 Introduction

Here, I am going to summarize a paper by **Matthew D. Zeiler and Rob Fergus from the Dept. of Computer Science, New York University, USA**. Its a quite recent paper of 2014 titled "Visualizing and Understanding Convolutional Networks" and it aims to understand the behind-the-scenes of Convolutional Neural Networks and help us extract specific features and exploit the concept of **CNN** even further.

As the abstract of the paper says, Large Convolutional Network models have recently demonstrated impressive classification performance. However, there is no clear understanding of why they perform so well, or how they might be improved.

In this summary of the mentioned paper, we will be talking mainly about a novel vizualization technique that gives insight into the functioning of intermediate feature layers and the operation of the classifier.

2 Basic Model Overview

Standard convnet models were used in the entire paper. These models map a 2D input image, via a series of layers, to a probability vector, over a number of different classes.

2.1 Layer Description

Each layer consists of :

1. Convolution of the previous layer output with a set of learned filters.
2. Passing the responses through a rectified linear function ($relu(x) = max(x, 0)$)
3. Max Pooling over local neighbourhoods(optional)
4. A local contrast operation that normalizes the reponses across feature maps.(optional)

Additionally, the top few layers are conventional fully-connected networks and the final layers is a *softmax classifier*.

2.2 Basic Training Details

The models were trained using a large set of labeled images, where *label* is a discrete variable indicating the **true class**. **Cross-Entropy** was used as a *loss function*, and the parameters of the network (filters in the convolutional layers, weight matrices in the fully connected layers and biases) are trained by *back-propagating* the derivative of the *loss with respect to the parameters throughout the network*, and updating the parameters via **Stochastic Gradient Descent**.

3 Visualization

Understanding the operation of a convnet requires interpreting the feature activity in intermediate layers. Here we will talk about a good way to map these activities back to the **input pixel space**, showing what input pattern originally caused a given activation in the feature maps.

3.1 Visualization with a DeConvolutional Network(deconvnet)

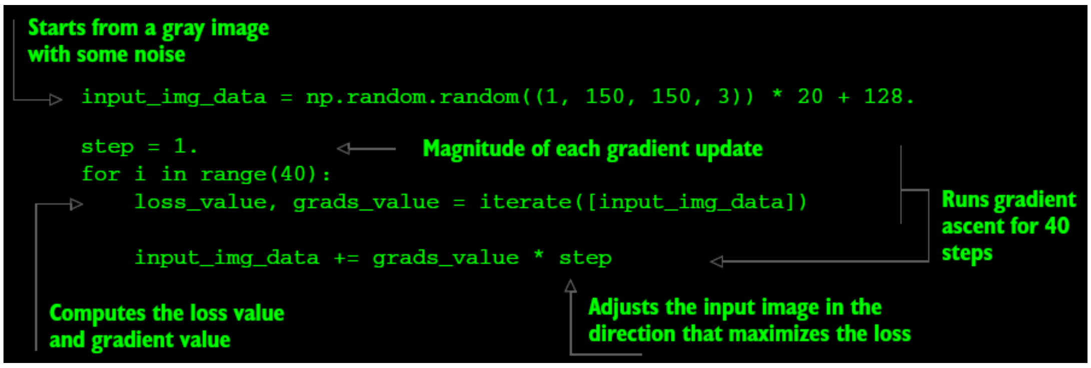
A deconvnet can be thought of as a convnet model that uses the same components but in reverse. To visualize a convnet, a *deconvnet* is attached to each of its layers, providing a perfect path back to its **input pixel space**.

To examine a given convnet activation, we successively **(i)** unpool, **(ii)** rectify, and **(iii)** filter to reconstruct the layer beneath that gave rise to the activation which the neural net chose. This process is repeated until *input pixel space* is reached.

Let's explore each step further :

3.1.1 Unpooling

In the convnet, the max pooling operation is non-invertible. But, we can obtain an approximate inverse by recording the locations of the maxima within each pooling region. We can do so, by using the (opposite of Stochastic Gradient Descent), Stochastic Gradient Ascent for a few steps until we reach satisfactorily close to our unpooled convnet.


Starts from a gray image with some noise
Magnitude of each gradient update
Runs gradient ascent for 40 steps
Computes the loss value and gradient value
Adjusts the input image in the direction that maximizes the loss

```
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.

step = 1.
for i in range(40):
    loss_value, grads_value = iterate([input_img_data])

    input_img_data += grads_value * step
```

This is just an example code snippet from Python performing **Gradient Ascent** and **iterate** is a **Keras** backend function to compute the value of loss tensor and the gradient tensor, given a NumPy Tensor.

3.1.2 Rectification

The convnet uses **relu non-linearities**, which rectify the feature maps thus ensuring the feature maps are always positive. To obtain valid feature reconstructions at each layer, we pass the reconstructed signal through a **relu non-linearity**.

3.1.3 Filtering

The convnet uses learned filters to convolute the feature maps from the previous layer. To approximately invert this, the deconvnet uses transposed versions of the same filters, which means flipping each filter vertically and horizontally.

3.2 Convnet Visualization

We can show the practical use of deconvnet to visualize the feature activations, on the ImageNet validation set.

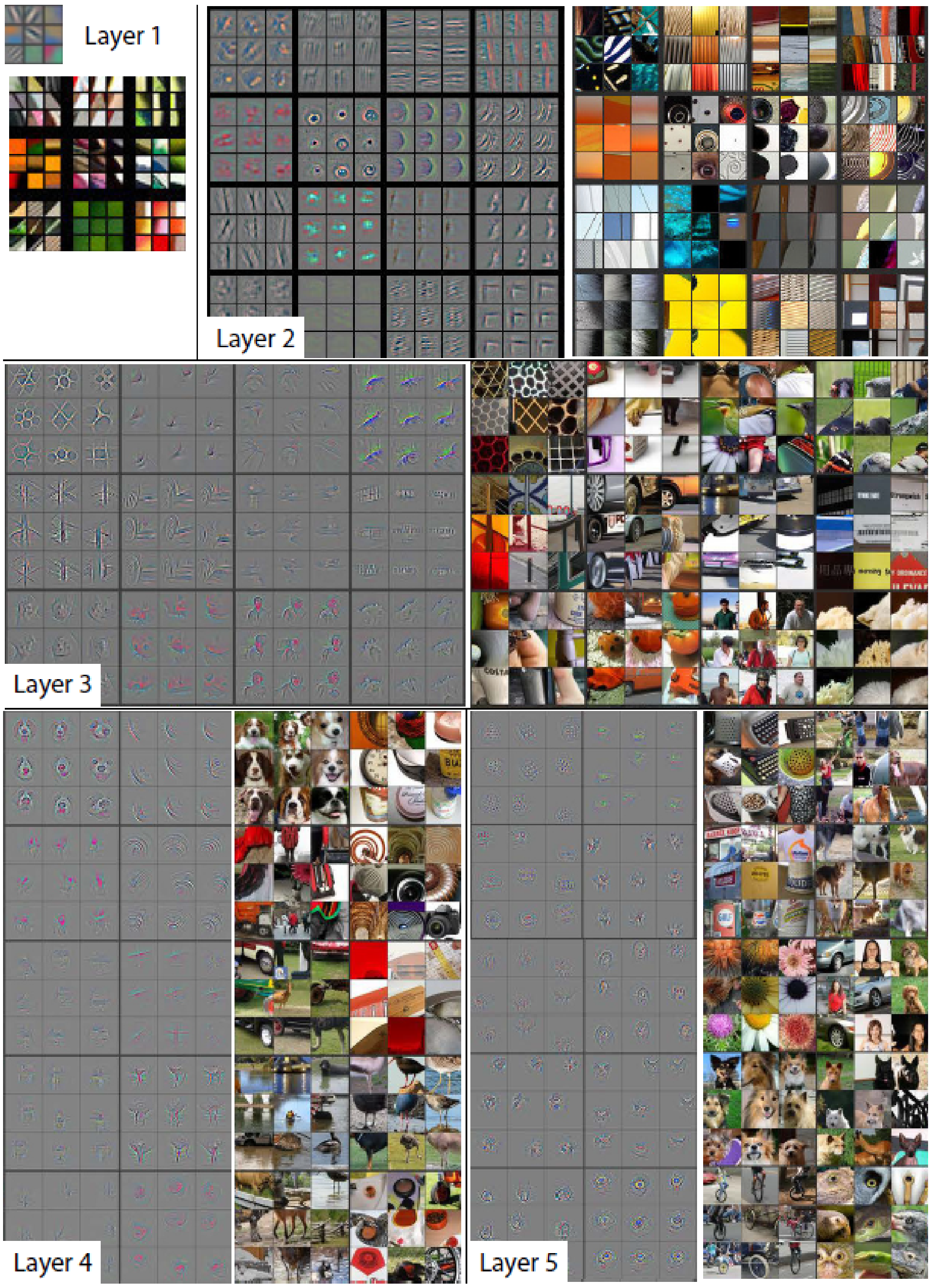
3.2.1 Feature Visualization :

In the figure below, the top 9 activations, each projected separately down to its input pixel space, have been shown, also showing its invariance to input deformations.

Quoting the original excerpt from the paper :

> The projections from each layer show the hierarchical nature of the features in the network. Layer 2 responds to corners and other edge/color conjunctions. Layer 3 has more complex invariances, capturing similar textures (e.g. mesh patterns (Row 1, Col 1); text (R2,C4)). Layer 4 shows significant variation, and is more class-specific: dog faces (R1,C1); bird's

legs (R4,C2). Layer 5 shows entire objects with significant pose variation, e.g. keyboards (R1,C11) and dogs (R4).



A natural question that comes up is if the model truly identifies the location of the object in the image or if it just uses the surrounding context to get the picture. To check this, the author systematically occludes different portions of the input image using a grey square. The outputs of the classifier model, clearly showed that the model was localizing the objects within the scene and the probability of the prediction drops significantly on occlusion.

Another surprising observation was that removing two of the middle convolutional layers made a relatively small difference to the error rate. However, removing both the middle convolution layers and the fully connected layer yielded a model with only 4 layers whose performance was dramatically worse. This would suggest that the overall depth of the model is important for obtaining good performance, but changing the size of the fully connected layers made little difference to performance, however, increasing the size of the middle convolution layers gave a useful gain in performance. But increasing these, while also enlarging the fully connected layers results in over-fitting.

4 Conclusion

This paper explored **large convolutional neural network models**, trained for image classification.

Firstly, the paper presented a novel way to visualize the activity within the model. This revealed the features to be far from random, uninterpretable patterns. Rather, they showed many intuitively desirable properties.

Secondly, it also showed how these visualizations can be used to identify problems within the model and so obtain better results.

Thirdly, it further explained that having a minimum depth to the network, rather than any individual section, is vital to the model's performance.

But above all, the key takeaway from this paper is the concept of De-Convolutional Network, and its implementation.

5 References

- The Original Paper
- Deep Learning with Python by **François Chollet**