Comparing two Convolution Neural Networks for Image Classification

Shikhar Vashishth
M.Tech CSA
IISc Bangalore
shikhar.vashishth@csa

Abstract

The project compares two popular Convolution Neural Network (CNN) architectures: AlexNet and ZFNet which have won ImageNet LSVRC challenge in the year 2012 and 2013 by a considerable margin. Both the architectures have been a breakthrough in the field of deep learning. The goal of this project is to explore these successful architectures and look into the working of their each individual layer using the technique called Devolution and explore the evolution of ZFNet from AlexNet.

1 Introduction

The concept of Neural Networks has been around for several decades but in the year 2012 the real potential of this model was actually admitted and acknowledged by the entire world. The Convolutional Neural Network architecture, **AlexNet**, proposed by Alex Krizhevsky et al [1] was the first breakthrough architecture which won the ImageNet LSVRC-2012 challenge with a huge margin. It won the challenge with an error rate of **15.3 %** while the second-best contest entry achieved an error rate of **26.2 %** on classification task of 1.2 million images into 1000 different categories. Thus the model was the state-of-the-art at that time. AlexNet although was the best among all the known models but still the reason for its performance remained unknown. In the year 2013, Mattew D. Zeiler et al came up with a way to understand the working of CNN architectures by giving a method for visualizing the activations of convolution layers in the network. The method was called as **Deconvolution** because it allows to project the output of convolution layers back to the pixel domain. Based on the analysis of AlexNet using their new technique, they discovered certain loopholes in AlexNet which they rectified and proposed their own CNN architecture, ZFNet, which won the ImageNet LSVRC challenge in 2013.

1.1 Style

Papers to be submitted to NIPS 2015 must be prepared according to the instructions presented here. Papers may be only up to eight pages long, including figures. Since 2009 an additional ninth page *containing only cited references* is allowed. Papers that exceed nine pages will not be reviewed, or in any other way considered for presentation at the conference.

Please note that this year we have introduced automatic line number generation into the style file (for LaTeX 2_ε and Word versions). This is to help reviewers refer to specific lines of the paper when they make their comments. Please do NOT refer to these line numbers in your paper as they will be removed from the style file for the final version of accepted papers.

The margins in 2015 are the same as since 2007, which allow for $\approx 15\%$ more words in the paper compared to earlier years. We are also again using double-blind reviewing. Both of these require the use of new style files.

Authors are required to use the NIPS LaTeX style files obtainable at the NIPS website as indicated below. Please make sure you use the current files and not previous versions. Tweaking the style files may be grounds for rejection.

1.2 Double-blind reviewing

1.3 Retrieval of style files

The style files for NIPS and other conference information are available on the World Wide Web at

`http://www.nips.cc/`

The file `nips2015.pdf` contains these instructions and illustrates the various formatting requirements your NIPS paper must satisfy. LaTeX users can choose between two style files: `nips15submit_09.sty` (to be used with LaTeX version 2.09) and `nips15submit_e.sty` (to be used with LaTeX2e). The file `nips2015.tex` may be used as a "shell" for writing your paper. All you have to do is replace the author, title, abstract, and text of the paper with your own. The file `nips2015.rtf` is provided as a shell for MS Word users.

The formatting instructions contained in these style files are summarized in sections **??**, 4, and **??** below.

2 Convolutional Neural networks

Convolutional Neural Networks are just an extension of basic neural networks, which are designed to process data in the form of multi-dimensional arrays. Similar to Neural Nets, ConvNets also transform inputs through a series of hidden layers, but they do it in a way which is more efficient to implement and with fewer parameters. ConvNets take advantage of the spatial structure of the data. Their architecture is composed of a sequence of layers, which can be of three types: convolution layer, pooling layer, and fully-connected layer.

Convolution layers have several filters which are convolved with every part of the input. These filters look for a specific feature in the input and make ConvNets translation invariant. Filters are also responsible for drastic parameter reduction as they are shared across the entire layer. The convolution layer accepts an input of size $W_1 \times H_1 \times D_1$ and gives an output of size $W_2 \times H_2 \times D_2$ which is determined by the input size and the other hyperparameters of the layer. Hyperparameters of the layers include number of filters K, filter size F, stride with which filters were applied S, and the zeros padding border width used while convolving.

$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$
$$D_2 = K$$

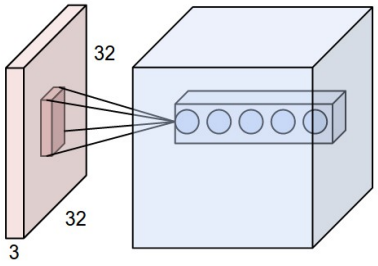

Figure 1: Convolution Layer

Pooling layers are generally kept after the convolution layers for downsampling the convolution output. They downsample the input spatially, making network lose locality information but keeping the local information of the features. Most commonly, max-pooling is used as the pooling layer.

Pooling layer takes an input of size $W_1 \times H_1 \times D_1$. The hyperparameters of the layer include spatial extent F and the stride S. Based on the input size and the hyperparameters the size of the output, $W_2 \times H_2 \times D_2$ is defined as:

$$W_2 = (W_1 - F)/S + 1$$

$$H_2 = (H_1 - F)/S + 1$$

$$D_2 = D_1$$

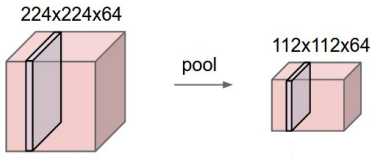

Figure 2: Pooling Layer

In **fully connected layers**, just like in a regular neural network each pixel is considered as a separate neuron. The number of neurons in final fully connected layer is dependent on the desired output from the network. These layers constitute the end layers of ConvNets.

3 Word Representations

The most common and the brute-force way of representing words is **one-hot encoding**. In this each word is represented as a $\{0, 1\}^{|V|}$ vector ($|V|$ is the size of the vocabulary), with all $0s$ and 1 at the index of the word in the sorted list of the entire vocabulary. The drawback of the representation is that each word is represented as an independent entity which is not true in reality. Moreover, the memory requirement of the representation is also quite huge, $O(|V|^2)$ bits.

Another way of representing words is **Window based cooccurence matrix**. In this method, we keep track of the frequency of occurrence of each word inside a window of a given size around the word of interest. The final representation is a matrix $R^{|V| \times |V|}$, whose dimension can be reduced by SVD. The obvious drawback of the representation is its space requirement. In situations where the vocabulary size is huge (13 million), the generation of matrix will be computationally expensive and moreover, if the size of the vocabulary changes then the whole matrix needs to be updated which is quite frequent in the real world.

To overcome all the above mentioned difficulties the idea of **word2vec** was proposed which involves predicting the neighboring words of every word, instead of computing the cooccurrence counts directly. Word2vec allows us to add new words and documents to our current representation efficiently. For predicting the neighboring words in a m length window around a word the following objective function is maximized:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t)$$

where, θ represents all the variables which need to be optimized. To predict any surrounding word the following measure can be used:

$$P(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$

where, o and c refers to outside and center word and v_c and u_o are "center" and "outside" vectors of c and o. For each word two vectors are maintained.

Table 1: Label mapping

Label desription	numerical mapping value
Dendrite	Input terminal
Axon	Output terminal
Soma	Cell body (contains cell nucleus)

4 Data-set Description

The dataset we use is Stanford Sentiment Treebank[1].

The Data set contains 10,605 processed snippets from the original pool of Rotten Tomatoes HTML files. Some snippet may contain multiple sentences. The whole data set is splitted into 3 sets, namely training/validation/test splits.Each split contain data and corresponding labels. Data contains phrases and their corresponding phrase_id and Labels contains phrase_ids and the corresponding sentiment labels, both separated by a vertical line. Differents labels are very positive, positive, neutral, negative, very negative.

structure of the phrases has been encoded in a parent pointer format. E.g. Offers that rare combination of entertainment and education is encoded as 16—14—13—13—12—10—10—11—17—11—12—15—14—15—16—17—0

Label is mapped as

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D. S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609-616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer-Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[]

[4] http://nlp.stanford.edu/sentiment/index.html