Satyaki Chakraborty

November 30, 2015

5.2. Semantic Labeling

The report of this section has been divided into the following sub-sections:

- Abstract
- Proposed architecture
- Rationale
- Results
- Analysis
- Source code
- References

Abstract

Over the past few years, deep neural architectures have been used to train in an end-to-end manner (from raw pixels to labels) for semantic labeling. The advantages of these methods are
(i) No need for computing hand engineered features
(ii) No need for processing input data (like generating super pixels or segmentation).
(iii) Unlike traditional CRFS, MRFs the computational cost during test time is much less.

In this report, I have tried to show how a recurrent neural network (which learns on a sequence of patches of different scales surrounding a pixel) can be used for the purpose of semantic labeling. The architecture is as follows.

Architecture

Semantic Labeling using Convolutional Neural Networks(Grangier et al. [1]) is done by learning on patches surrounding each pixel. The input of the neural net is a patch surrounding a pixel and the output is the corresponding label of the pixel. Here, at first CNNs are used to learn on patches of three different scales (4x4, 10x10, 14x14). The hyperparmameters of the convolution & pooling layers are chosen such that the input to the hidden layer of the MLP always has 90 features (same for all the different patch sizes).

Let, the representation layer (of 90 features) of patch size 4x4, 10x10 & 14x14 of pixel at (i, j) be denoted by f^{ij}_{4x4}, f^{ij}_{10x10}, f^{ij}_{14x14} respectively. For each such pixel at (i, j) a sequence of learned representations, $f^{ij} = (f^{ij}_{4x4}, f^{ij}_{10x10}, f^{ij}_{14x14})$ is created. The recurrent neural network learns on these sequence of convolved features to predict the label y. At each step of the RNN, the output (against which loss is computed is the one-hot vector corresponding to the label y). A somewhat similar idea was used by Pinheiro et al. [2] which uses the prediction of the previous layer and a scaled version of the raw RGB data as input for the next layer. But this implementation differs from theirs as instead of using scaled raw RGB data at every time step of the RNN, the learned features from the CNN and the representation (not prediction) of the previous layer are used as input for the next layer. The reason for using such an architecture is explained in the next section.

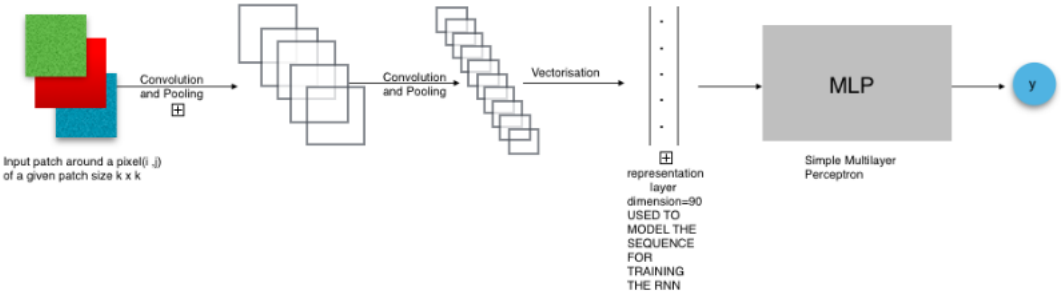

Fig: Convolutional Neural Network for scene labeling; y is the label's corresponding one-hot vector form; input is a patch surrounding a pixel.

The representation layers are generated for 3 different patch sizes (4x4, 10x10, 14x14) for every pixel and the sequence is fed to the RNN. (shown below)

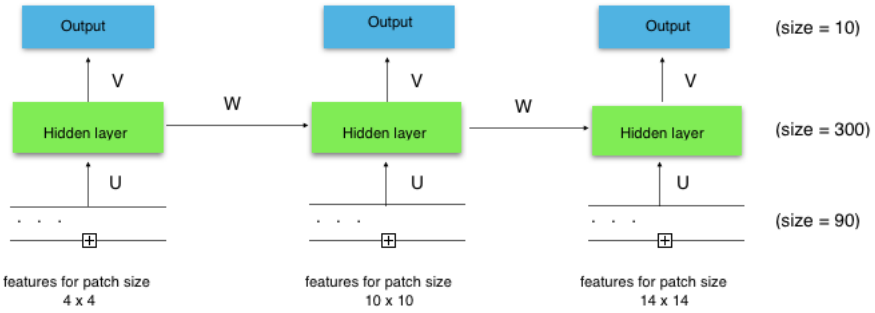

Fig: Recurrent Neural Network learning on the features of different patch sizes

As for predicting the label of a pixel, at first all three different patch sizes are generated. Feature vectors corresponding to these three patches are obtained from the convolution-pooling layers. This sequence of feature vectors is then fed to the RNN. The output of the RNN is the predicted value of y (the label).

Rationale

The reason for using CNNs to train on patches surrounding a pixel is that the CNN learns to predict the label based on a context of the pixel. It has been shown earlier that in order to get a good visual coherence, the model needs to capture long range pixel dependencies(Farabet et al. [3]). But using a large context with large-size convolution and pooling kernels introduces too much translational invariance, which introduces error for classifying accurately. This problem is less severe for smaller context sizes with smaller convolution-pooling kernels. But smaller context fails to capture long range pixel dependencies as mentioned earlier.

This is why the RNN is used to learn on the representations of patches of 3 different scales. The idea is that the shared parameters of the RNN will thus be able to learn on varying context sizes and will be able to address the trade-off mentioned earlier.

Also one might think that since the RNN learns on the features which are learnt from the CNN, there is some error propagation from the CNN (in modelling the representations) to the RNN. But this idea is not entirely true, because even with randomly initialised weights, convolution and pooling layers can generate good features[4]. Thus a supervised pre-training should theoretically only model the representations better. The architecture is based on this assumption.

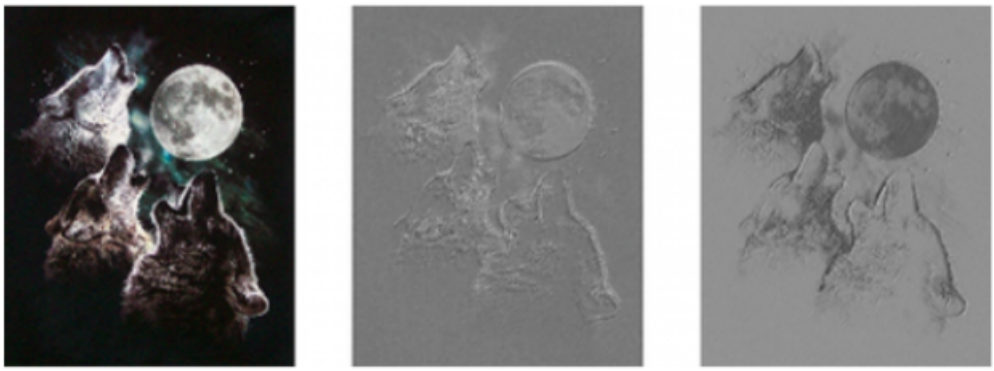

Fig: a randomly initialized filter acting as an edge detector
Courtesy, Deep Learning Tutorials Release 0.1, pg - 56

Also, in the RNN, the prediction of the previous layer is not fed into the next layer. Instead the hidden layer is fed. The reason for such is that the former architecture relies on a somewhat stronger assumption that the prediction of the previous layer is both accurate enough and rich enough in features to represent the current state of the RNN (Y. Bengio, Deep Learning, Ch - 10)[5]

Results

Results obtained after performing 5-fold validation on the randomised data is :

Fold	Accuracy(%)
Fold-1	62.14%
Fold-2	**66.87**%
Fold-3	59.53%
Fold-4	63.48%
Fold-5	62.61%

Testing on the entire dataset was done in a machine with 2.4 Ghz Octacore Intel Xeon CPU with 48 GiB memory. Note that generating patches takes a lot of time compared to training. Hence the input images were scaled by a factor of 0.25 in height and width.

Analysis

For analysis, I have used four labeled images from the dataset(due to time and resource constraints). Note that these 4 images are very similar in terms of semantics and structure. Hence, the accuracy of labeling is relatively high. However this section of the report is just meant to verify whether the architecture is actually learning what it is supposed to learn (as mentioned in the 'Rationale' section).

Arch.	Test error(%)	Accuracy(%)
CNN_{4x4}	19.7%	80.3%
CNN_{10x10}	14.35%	85.65%
CNN_{14x14}	15.2%	84.8%
$RCNN$	11.81%	88.19%

Note that here also the input images were scaled. Moreover, as seen from the results, the accuracy of RCNN is more than that of each of individual CNN architectures. This somewhat supports the very idea behind this. The terminal output is made available at https://goo.gl/TDC45D. The testing for analysis was done in a machine with 1.4 GHz Intel Ci5 CPU and 4GiB memory.

Source code

The source code is made available at https://github.com/shady-cs15/rcnn. It is written in python on top of Theano library. It is hereby stated that no part of the source code uses any 'off-the-shelf' code from any of the references mentioned. However some basic blocks of code (like logistic regression) have been re-used from the Theano deep learning tutorials.

References

[1] Grangier, D., Bottou L., Collobert, R. Deep convolutional networks for scene parsing. In *International Conference on Machine Learning(ICML) Deep Learning Workshop*, 2009.
[2] Pinheiro P., Collobertm R. Recurrent Convolutional Neural Networks for scene labeling. In *International Conference on Machine Learning(ICML) Deep Learning Workshop*, 2014.
[3] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. Learn- ing hierarchical features for scene labeling. *IEEE Trans- actions on Pattern Analysis and Machine Intelligence*, 2013.
[4] MILA lab, Deep Learning Tutorials Release 0.1, pg - 56
[5] Y. Bengio, Deep Learning, Ch - 10, pg - 334