Convolutional Neural Networks in Galaxy Zoo Challenge

Maxim Milakov
maxim.milakov@gmail.com

April 2014

Abstract

Convolutional neural networks have been winning nearly all image classification competitions last several years. Galaxy Zoo challenge asked participants to solve regression problem instead, applied to images as well. Top 3 winners used convnets confirming that feature extraction capabilities of these models could be applied to regression problems too.

1 Introduction

Convolutional neural networks ([1]) were quite successful recently in images classification tasks: GTSRB ([2]), ImageNet 2012 ([3]), ICPR 2012 mitosis detection competition ([4]).

At the same time there are fewer reports on using convolutional nets for regression tasks, probably due to fewer regression tasks at all out there in the wild. While there seems to be nothing preventing convnets to utilize their great automatic feature extraction capabilities and excel in these tasks it would be interesting to see it in practice.

Galaxy Zoo project[1] keeps a large collection of galaxy images. It asks its visitors to answer a number of questions about images, which then averaged into probabilities of getting specific answer for each question for each image.

Galaxy Zoo challenge[2] started on December 20, 2013. It provided participants with training data set consisting of 61,578 images along with probabilities for 37 answers for each image. Contest organizers asked contestants to provide predictions for probabilities of these answers by April 4, 2014 for another 79,975 images from the test set.

While Galaxy Zoo project's participants solved classification task - give an 1-out-of-N answer for a set of questions - competitors of the challenge had to solve the regression problem - give a probability the Galaxy Zoo project's participant giving the specific answer. Below is the description of the approach I took to solve this problem using convolutional neural networks.

[1]http://www.galaxyzoo.org
[2]https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge

2 Methods

2.1 Data preparation

Original galaxy images are 424 pixels in width and 424 pixels in height, 3 channel RGB. They were resized to 130 x 130 pixels and then the central 100 x 100 pixels crop was extracted from it.

I distorted each 100 x 100 image during training and testing (more on this in 2.3.1 and 2.4) and took the central 68 x 68 image from it. This 68 x 68 x 3 channels image was used as an input to the model.

2.2 Network schema

I used rather simple schema of convolutional neural network, consisting of 7 logical blocks:

1. Convolutional layer, 3 input x 128 output feature maps, 5x5 window + Maxout layer, 2x subsampling (in feature map dimension), see [5] + Max subsampling layer, 2x2 subsampling (spatial)
2. Convolutional layer, 64 input x 192 output feature maps, 5x5 window + Maxout layer, 2x subsampling + Max subsampling layer, 2x2 subsampling
3. Convolutional layer, 96 input x 256 output feature maps, 5x5 window + Maxout layer, 2x subsampling + Max subsampling layer, 2x2 subsampling
4. Convolutional layer, 128 input x 384 output feature maps, 1x1 window + Maxout layer, 2x subsampling
5. Convolutional layer, 192 input x 256 output feature maps, 1x1 window + Maxout layer, 2x subsampling
6. Convolutional layer, 128 input x 256 output feature maps, 1x1 window + Maxout layer, 2x subsampling
7. Convolutional layer, 128 input x 37 output feature maps, 1x1 window

This model has about 2.1 millions of parameters.

The schema converts input data of size 64 x 64 x 3 feature maps to the output data of size 1 x 1 x 37, the size we have labels for, see Figure 1. Note that the last 4 convolutional layers become fully connected ones in this case.

Replacing Maxout layers with rectified linear units (see [6]), tweaking feature map counts to keep the number of parameters the same, showed the same performance as the original model with Maxout layers.

I tried adding more non-linearity by putting hyperbolic tangent layer on top of each convolutional layer - and didn't get any improvement with it.

Increasing the feature map count in the first several layers increased the model performance but also increased time for training and testing. The schema above was the maximum I was able to run to fit into the competition's schedule.

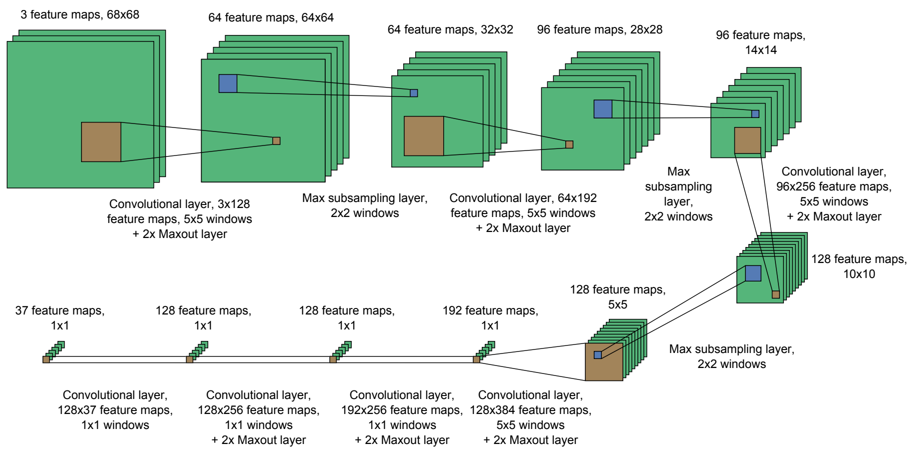
3 feature maps, 68x68
64 feature maps, 64x64
64 feature maps, 32x32
96 feature maps, 28x28
96 feature maps, 14x14
Convolutional layer, 3x128 feature maps, 5x5 windows + 2x Maxout layer
Max subsampling layer, 2x2 windows
Convolutional layer, 64x192 feature maps, 5x5 windows + 2x Maxout layer
Max subsampling layer, 2x2 windows
Convolutional layer, 96x256 feature maps, 5x5 windows + 2x Maxout layer
128 feature maps, 10x10
128 feature maps, 5x5
Max subsampling layer, 2x2 windows
37 feature maps, 1x1
128 feature maps, 1x1
128 feature maps, 1x1
192 feature maps, 1x1
Convolutional layer, 128x37 feature maps, 1x1 windows
Convolutional layer, 128x256 feature maps, 1x1 windows + 2x Maxout layer
Convolutional layer, 192x256 feature maps, 1x1 windows + 2x Maxout layer
Convolutional layer, 128x384 feature maps, 5x5 windows + 2x Maxout layer

Figure 1: Convolutional neural network schema applied to a single sample

2.3 Training

2.3.1 Distortions

Training data set contained 61,578 images, the model overfit these data alone noticeably. Yet even shallow knowledge of the problem domain allowed creating new data samples easily by distorting original ones. The galaxy image was likely to be classified by humans the same way when we applied limited random distortions:

Distortion type	Domain
Mirror	Yes, No
Rotation	$-180° \,..\, 180°$
Scale	$\frac{1}{1.1} \,..\, 1.1$
Shift	$-2.0 \,..\, 2.0$ pixels in x and y directions

Each image taken from randomized training data set was randomly distorted this way right before being fed into the neural network.

I tried one more type of distortion, which seemed relevant to this problem - stretching - but it didn't improve the generalization capabilities of the model.

2.3.2 Error function

Contest organizers set RMSE as the evaluation metric:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2}$$

Where N is the number of galaxies times the total number of responses, p_i is predicted value, a_i is the actual value.

The task of optimizing this RMSE is equivalent to the task of optimizing MSE:

$$MSE = \frac{1}{2N} \sum_{i=1}^{K} \sum_{i=1}^{M} (p_{ij} - a_{ij})^2$$

Where K is the number of samples (galaxies), L is the number of output neurons (responses), p_{ij} is predicted value, a_{ij} is the actual value.

I optimized MSE during training, although all reported error values (scores) in the article are RMSE.

2.3.3 Training method

I used a slightly modified version of Stochatic Diagonal Levenberg-Marquardt method described in [7], [1], and [8]. It computes estimates of the diagonal terms of the Hessian via a backpropagation algorithm similar to the error back-propagation. The original method computes individual learning rates according to:

$$\epsilon_k = \frac{\eta}{\mu + h_{kk}}$$

Where ϵ_k is individual error rate, η is global learning rate, h_{kk} is estimate of the diagonal term of the Hessian, and μ is a constant preventing individual learning rates from becoming too large when the curvature h_{kk} becomes too small.

I found out that for this particular problem it was beneficial to have μ to be layer-specific and set it to the average of the hessian term for parameters belonging to this layer:

$$\mu_L = avg_{k \in L} h_{kk}$$

Per-layer learning rate was introduced which depended on global one:

$$\eta_L = \mu_L \eta$$

And, finally, individual learning rates depended on per-layer learning rates, per-layer μ and the corresponding Hessian term:

$$\epsilon_k = \frac{\eta_L}{\mu_L + h_{kk}} = \frac{\eta}{1 + \frac{h_{kk}}{avg_{k \in L} h_{kk}}}$$

Basically, I applied original Stochastic Diagonal Levenberg-Marquardt method in a layer-wise fashion - taking into account differences in curvature of the error function for different parameters within the same layer, but not doing it for parameters across different layers. This proved to be useful (in comparison with original method) for this particular problem - the training was more robust, showed higher convergence rate. The very basic stochastic gradient method would likely do well here as well.

Neither dropout ([9]) nor limit on L2-norm of incoming weights ([9]) helped improve the generalization capabilities for this model.

2.3.4 Hyper-parameters training

Contest organizers split the whole testing data set into two parts: 25% were used for reporting scores in Public leaderboard[3], and other 75% were held out for the Private leaderboard[4], which was revealed after the competition finished and used for the final standings.

I set aside 10% of the original training data set as validating data set and used it to train hyper-parameters: schema of the network, global learning rate η, number of epochs, and learning rate decay schedule.

Score on this validating data set was a good predictor for the score on Public testing data set - the difference was within 0.001 for all 11 submissions I did. I didn't overfit Public testing data set either: Scores on Private testing data set happened to be quite close to the score on Public one - the difference was about 0.0001 on all my 11 submissions. And the standings didn't change much when the leaderboard changed switched from Public data set to Private one.

I ended up with running training for 100 epochs, global learning rate $\eta = 0.01$ and decaying it by a factor of 0.8 during last 20 epochs. It took about 6 hours to train each network on NVIDIA GeForce GTX Titan.

2.4 Running test data

For the final submission I trained 15 networks on full training data set, with validation data set merged into the main training one. 48 deterministic distortions were run through each of the networks for each testing sample - 24 rotations x 2 mirroring. Output neuron values for all 720 samples were averaged. Both training multiple neural networks and testing multiple distorted images were significant alone, although with one present the other didn't add much value:

Configuration	Samples	Error
Base	1	0.08138
15 networks	15	0.07826
Distortions: 24 rotations & mirroring	48	0.07838
15 networks x 48 distortions	720	0.07753

There was a certain structure in questions and answers - decision tree[5] - imposing some constraints on the probabilities. I didn't use this structure; the only rectification I applied was cropping the final numbers to [0;1].

Running all 79,975 test images x 720 samples took about 10 hours on GeForce Titan.

3 Results

329 teams - each consisting of one or more Kaggle members - participated in the challenge. Rather simple setup of convolutional neural networks along with

[3]`https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/leaderboard/public`

[4]`https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/leaderboard/private`

[5]`https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/details/the-galaxy-zoo-decision-tree`

problem specific distortions allowed me to take the 2nd place:

Place #	Team Name	Score
1	sedielem (Sander Dieleman)	0.07492
2	Maxim Milakov	0.07753
3	6789 (tund, Truyen Tran)	0.07869

Sander Dieleman, the winner, won with a large margin. He used convolutional neural networks too, although his approach was more sophisticated.[6] Moreover, *tund* reported using convolutional networks as well.[7]. Thus all prize winners used convnets.

All the source code of my solution is available as part of the nnForge[8] library, serving as an example of library usage and satisfying the requirements of contest organizers.

4 Discussion

Convolutional neural networks certainly excelled in solving regression task for natural images in competitive environment. All top 3 places were occupied by the teams using convnets.

Could I have done better? Probably, yes. Learning *rotation* invariant features was a relatively difficult task for the convolutional neural network. I was thinking about converting images to log-polar coordinates and thus making the network to learn *cyclic shift in one direction* invariant features, which seemed to be an easier task for the convnets. Yet I didn't do it, mostly due to the tight competition schedule.

Convolutional neural networks are computational intensive models. High computational power of GPUs in recent years and availability of Open Source packages running these models enabled high rate of their adoption.

Data Analysts using convnets are winning various competitions on dense spatially-connected data classification, and now regression. Yet creativity in data pre-processing and network design plays a key role in achieving the highest results.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[2] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[6] `http://benanne.github.io/2014/04/05/galaxy-zoo.html`
[7] `http://bit.ly/1oENoNE`
[8] `http://nnforge.org`

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In P. Bartlett, F.c.n. Pereira, C.j.c. Burges, L. Bottou, and K.q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1106–1114. 2012.

[4] Dan Claudiu Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *MICCAI*, volume 2, pages 411–418, 2013.

[5] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *ICML*, 2013.

[6] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

[7] S. Becker and Y. LeCun. Improving the convergence of back-propagation learning with second-order methods. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 29–37, San Mateo, 1989. Morgan Kaufman.

[8] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade.* Springer, 1998.

[9] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing coadaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.