Galaxy Zoo Challenge with Convolutional Neural Networks

Team 6789
Members: Tu Dinh Nguyen, Truyen Tran
{nguyendinhtu,tranthetruyen}@gmail.com

Abstract

Convolutional neural networks – ConvNets – have won recent visual recognition challenges and beaten the state-of-the-arts on various vision datasets. We describe an implementation which landed us the third position in the Galaxy Zoo challenge. The method includes a ConvNet in the feature extraction phase, followed by a standard Neural Net. The final result was achieved using blending of several variations of the Neural Net.

1 Introduction

Galaxy Zoo project recruits a number of volunteers to describe the morphology of galaxies by analyzing their images. More specifically, the volunteers step-by-step answer several questions about the galaxies. Their next questions are determined basing on their previous answers. All answers for an image are then aggregated and mapped into a set of probabilities. Galaxy Zoo challenge aims to predict such probabilities for new galaxy images. The predictive performance is evaluated by Root Mean Squared Error (RMSE) metric.

Convolutional neural networks [3] (ConvNets) have recently been demonstrating their superior performances in numerous image classification challenges and datasets. In this challenge, the input is still image-based but the target is regression rather than classification. We, however, keep believing that the ConvNets with excellent capabilities of automatic feature self-learning are the first priority candidatures for this problem. Therefore we begin and end up with using ConvNets. In the following sections, we describe our approaches, implementations and results.

2 Methods

2.1 Data preprocessing and data augmentation

Cropping and downsampling images The resolution of original images is 424×424 with 3 color channels – RGB. Except several images consisting of more than one galaxies which are far apart, all of the images contains galaxies at the center. Thus we crop all images at the center using 200×200 windows. After that we resize the images to 128×128 to fit the input size of our network architecture.

Exploiting spatial invariances The multilayer ConvNets belongs to deep architecture family. The more data they see in the training phase, the better performance they can achieve in the testing phase. In this dataset, the galaxy images are rotation, scale and translation invariant. These are good points to explore the data augmentation idea – creating more data by perturbing original ones. Besides, augmenting more data avoids overfitting and increases the generalization of the network. We use following data augmentation techniques:

- **rotation**: a random angle in the set of $\{\pm 90^\circ; 180^\circ\}$;

- **flipping:** horizontal reflection;
- **zooming**: random zoom-in scale factor between 2.0 and 3.0;
- **translation**: random cropped patches.

Although we use max-pooling in subsampling layers which can obtain translation invariance, the above translation helps to supplement more training data. We extract a random crop of size 120×120 for each image. Such cropped patch is then randomly kept unchanged or perturbed using the above rotation, horizontal reflection or zoom-in. Then it is fed into the ConvNets.

2.2 Network architecture

We adopt the convolutional schema of OverFeat model [5] with half the numbers of channels and several modifications. Table 1 describes our architecture with 9 layers. We apply dropout techniques [1] to the fully connected layers 7 and 8. The network has about 19.5 million parameters. All layers contains rectified linear units [4]. The output of the last layer is used as predicted values.

Layer	1	2	3	4	5	6	7	8	9
Stage	conv+max	conv+max	conv	conv	conv	conv+max	full	full	full
# channels	48	96	192	192	384	384	2048	2048	37
Filter size	5×5	5×5	3×3	3×3	3×3	3×3	-	-	-
Conv. stride	1×1	1×1	1×1	1×1	1×1	1×1	-	-	-
Pooling size	3×3	2×2	-	-	-	3×3	-	-	-
Pooling stride	3×3	2×2	-	-	-	3×3	-	-	-
Zero-padding size	-	-	-	-	-	-	-	-	-
Spatial input size	120×120	39×39	18×18	16×16	14×14	12×12	4×4	1×1	1×1

Table 1: Network architecture in detail.

2.3 Training

Objective function The competition uses Root Mean Squared Error (RMSE) as the evaluation metric:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(p_n - a_n)^2}$$

where N is the number of galaxies times the total number of responses, p_n is the predicted value and a_n is the actual value. Minimizing this metric is equivalent to minimizing the following Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}(p_{nk} - a_{nk})^2$$

in which N now is the number of galaxies and K is the number of responses. Note that $K = 37$. We optimize MSE during training and report RMSE for the final results.

Hyperparameters setting We randomly shuffle the training data and take the first 98% for training and the rest for validation. The 2% validation data seem small yet adequate to match the public and private scores. We use validation data to test network architecture, data preprocessing, augmentation techniques and to specify hyperparameters including learning rates, weight initializations, weight decays, number of epochs and learning rate decays schedule. We end up with the following hyperparameters setting for all layers:

- The number of epochs is 182.

- The weights are randomly initialized from small normally distributed numbers, i.e. Gaussian $\mathcal{N}(0;0.01)$.
- The biases are set to zeros. Setting to a small nonzero value (e.g., 0.1) can accelerate the early stages of few learning steps.
- The momentums are fixed to 0.9.
- The weight decays – ℓ_2-norm regularization – are 0.0005.
- Dropout rates are 0.5.
- The learning rates of weights and biases are initialized at 0.01 and then reduced by a factor of 0.1 after $(140,166,174)$ epochs.

2.4 Testing

For each image at testing phase, we extract five 120×120 patches, i.e., four corners patches and center patch, as well as their horizontal reflections, 90°, 180° and 270° rotations. Thus there are 25 patches in total. We then average the predictions computed by the last layer on the 25 patches to take the final result. Interestingly, the final result satisfies all the constraints of decision tree. Given sufficient data and parameters, the ConvNets indeed are powerful enough to encode all constraints and information. Therefore we only crop the predictions into $[0,1]$ range, an obvious constraint.

2.5 ConvNets as Feature Extractor

Once the ConvNet had been trained, we used its last feature layer as the input for a standard neural net. The reason is that the last layer of the ConvNet is linear, and thus can benefit from a more flexible classifier. The strategy worked well and we found it consistently improved over the ConvNet.

The neural net has one hidden layer and an output layer of 37 elements. We normalized the training outcomes to have zeros mean and unit standard deviation, and this was to create the uniform scale among all outcomes. The net is regularized by using three methods: the ℓ_1-norm, the ℓ_2-norm (or weight decay), and the norm constraint to weights of connection between inputs and each hidden unit. In particular, the norm constraint is regularized using the following barrier function:

$$B(W) = \beta \sum_k \mathbb{I}\left[\sum_i W_{ik}^2 > \tau\right] \left(\sum_i W_{ik}^2 - \tau\right)^2$$

where W_{ik} is the mapping weight from input to hidden units, $\tau > 0$ is adjustable threshold, and $\beta > 0$ is the regularizing factor. A small τ in the range $(0,1)$ allows a large number of hidden units without overfitting. Typically we set $\beta = 1$ to keep it simple, but it can be adjusted by the training algorithm, as described below.

2.5.1 NNet training

The nets are trained with conjugate gradients and a simple continuation method to reach better local minima. The continuation method works as follows:

1. The NNet is trained until a local minimum is reached.
2. For $c = 1,2,..C$
   (a) The ℓ_2-norm regularizing factor and the norm constraint regularizing factor β are randomly scaled up by a number in the range $(0,200)$. This is to create a slightly different model so that a new local minimum can be found.
   (b) Reduce the regularizing factors to original settings, and find a new local minimum.
   (c) Break the loop if convergence has been reached.

Not all training data was used. We picked the first 2,000 training data, and 500 validation data to train the NNet, the rest of the validation data is used to tune hyperparameters and decide which models to retain.

2.5.2 Models blending

As the NNet is flexible, it could overfit easily. Second, it is very hard to search for the best model. Third, features learnt by ConvNet are highly redundant. These suggest models blending. We learn multiple NNets:

1. ℓ_1-norm factor: 10^{-4}, ℓ_1-norm factor: 10^{-7}, hidden size: 30, $C = 5$, $\tau = 1$, $\beta = 1$.
2. ℓ_1-norm factor: 0, ℓ_1-norm factor: 10^{-7}, hidden size: 50, $C = 20$, $\tau = 1$, $\beta = 1$.
3. Same as (2), but use only the positive indicators of features
4. Same as (2), use $\sqrt{data}$;
5. Same as (2), use $(data)^2$
6. Same as (2), add 10% noise to data.
7. Same as (2), use first 50% features.
8. Same as (2), use first 50% data.

The outputs are then blended using 37 NNets, one per outcome. We found that many outcomes are very easy to learn (outcome no. 1, 2, 4, 5, 6, 8, 14-19, 24, 26, 28, and 33), and thus we did not train them but performed simple averaging. Again, for NNets, the norm constraint is important to combat overfitting.

2.6 Final blending

Our final submission is generated by averaging four results. The results are produced by four models: ConvNet, NNet and two of models blending.

3 Implementation

We use python, numpy and matlab to materialize our solution. To implement convolutional networks, we utilize the cuda-convnet[1] package [2], a fast GPU-based convolutional network library. Many thanks to Alex Krizhevsky, it would be much harder to meet the challenge's timeline without your package. We use PIL-image[2] and scipy[3] to perform preprocessing and data augmentation. Our source codes are free under BSD-3 license and can be found here.

Workstations with one NVIDIA Tesla M2070 GPU card each are employed to train and test our networks. The training and testing phases take about 31.5 hours and 2.26 hours respectively.

4 Results

Our approach allows us to land the third position. Our private score (0.07869) is close to the validation score and worse only 0.0001 than public score (0.07859). It is funny that our team's name is ***6789*** and our final result ends with "7869". Top 10 on the leaderboard are shown on Table 2.

Sander Dieleman, the winner, won with a large margin. His convolutional architecture is smaller yet similar to ours. His preprocessing and data augmentation tricks are almost the same. The second place, Maxim Milakov, is an experienced master of ConvNets. He have been achieving high rankings in a number of challenges using ConvNets. In this competition, his architecture is simple yet rather very efficient. Not all top 10 participants have shown their approaches. However, so far we have known that at least the half have used ConvNets.

[1] https://code.google.com/p/cuda-convnet/

[2] http://www.pythonware.com/products/pil/

[3] http://www.scipy.org/

#	Team Name	Method	Score
1	sedielem (Sander Dieleman)	convnet	0.07492
2	Maxim Milakov	convnet	0.07753
3	**6789 (tund, Truyen Tran)**	**convnet**	**0.07869**
4	simon	-	0.07951
5	Julian de Wit	convnet	0.07953
6	2numbers 2many	-	0.07964
7	Ryan Keisler	-	0.08072
8	Voyager	-	0.08083
9	SuperDeep (Soumith Chintala, Pierre Sermanet)	convnet	0.08246
10	Owen	-	0.08304

Table 2: The final standings of Galaxy Zoo challenge. The "blank" methods means they have not been exposed so far. Our team, named as **6789**, landed the third place.

5 Things have been tested

Here are several things that we have tried but are not used in the end because they did not improve the results:

- Preprocessing data: several cropping windows such as 256×256 and 150×150; performing a pipeline of techniques – adjusting intensity, auto-rotating and auto-cropping.
- Network architecture: other designs for various image resolutions, i.e., 32×32, 48×48, 64×64 and 256×256 as well as the model of Krizhevsky *et al* [2]; network with several softmax layers to incorporate the decision tree.
- Training: using full training data without validation.
- Data augmentation: zoom-out; rotations with $\{\pm 45^\circ; 90^\circ\}$;

6 Conclusion

Convolutional Neural Networks – ConvNets – once again have demonstrated their outstanding performances on not only classification but also regression tasks in practice. Top 3 prized winners have used ConvNets and lots of ConvNets are in top 10. From now on, ConvNets indeed have to be considered as the first candidate in any visual recognition tasks.

References

[1] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS)*, pages 1106–1114, 2012.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[5] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.