Galaxy zoo

Chen Zhang (cz1389)
Guang Yang (gy552)

Abstract

The Galaxy Challenge was an Kaggle competition held 4 years ago. At that time, deep convolution neuron networks has just start gaining the interest of the field. Four years has past, during which, comes a lot of sophisticated models, such as AlexNet, VGG, ResNet, and very recently DenseNet and NasNet. In this experiments, we seek to revisit the problem using state-of-the-art ConvNet models. We adjust the PyTorch built-in models (customized for ImageNet) to cater to our input. We also tried different data augmentation methods.

1 Introduction

Galaxies have various shapes, colors and sizes. These properties indicate their age, formation conditions and histories. Understanding the distribution, location and types of galaxies as a function of shape, color and size are critical parts for us to learn about the universe.

Studies of galaxies normally use morphology to probe formation process of them, but these studies need the observation of large numbers of galaxies and accurate classification of them. So in the kaggle Galaxy Challenge, researchers are required to use machine learning methods to automatically tag those galaxies to speed up the classification progress. And in nowadays, with the development of deep learning, more and more powerful learning networks are introduced, which largely raised the accuracy of the conventional classification tasks.

In this paper, we propose some modified machine learning networks from recently deep learning networks including ResNet and DenseNet. We fit these networks into our Galaxy regression task, and we also use some techniques such as augmentation and stacking to help increasing the invariance and generality.

The rest of this paper is structured as follows: we present our models and methods in section 3, and in section 4, we carry our some experiments using our models to train the galaxy classifier. Finally, we draw some interesting conclusions in section 5.

2 Related works

In the original Kaggle competition, the winning solution [1] shaped the problem as a regression problem, where the output is an array of 37 decimal numbers. Several committee of 7-layer-networks were used, each consisting of 4 convolutional layers and 3 fully connected layers. Dropout was used for regularization, as well as maxout in the dense layers. Extensive data augmentation is also used. The decision tree constraints was incorporated in the output layer of the network. The final solution was a blend of 17 models. They achieved test score of 0.07491.

The second place solution, with test score of 0.07752. used a simpler ConvNets. [4] The third place solution also used ConvNets. They used 37 nets, one for each label. Data augmentation is also used. Their final solution was generated by averaging results from 4 experiments. [5] They achieved test score of 0.07869.

Most of the solutions used some sort of data pre-processing and augmentation. They most of them directly center cropped the image, while some did further pre-processing, adjusting the translation and rotation, then do the cropping. Then the images were rescaled to fit into their different implementations of ConvNets. Popular data augmentation methods include rotation(in multiples of 90 degrees), flipping, scaling, and random crop.

In the Galaxy Zoo project, the decision tree workflow is an important guideline. However, in most of the ConvNets approaches, this constraint was ignored.

All known top 20 solutions have used ConvNets.[4] The competition showed that ConvNets have great advantage over traditional machine learning methods with hand-crafted feature extraction not only in classification tasks but also regression tasks in practice.

3 Method

The Galaxy challenge are using nearly the same scheme as the Galaxy Zoo, which is an interactive selection derived from a decision tree to determine which class the user thought this galaxy to fall into. There are total 37 answers for the whole 11 questions, so there are 37 classes in the Galaxy challenge tasks. We are required to give a probability distribution on these 37 classes, and loss is calculated by root mean squared error (RMSE), so this task is actually an regression problem.

Normally, computer vision problems using ConvNets are classification problems. In order to concur the regression problem, we used sigmoid instead of softmax in the output layer.

The training set contains $61,578$ images with size 424×424. We sampled $6,134$ images to be the validation set. We first take the 224 by 224 pixels in the center and scale it based on the model.

We tried based our experiments on ResNet and DenseNet. As a benchmark, we first used pure resnet152 with naive data cropping and scaling. Then we explored data augmentation, image crops stacking, and ensembling.

In the baseline experiments, we used input size of 64. We then tried the RandomCrop, RandomHorizontalFlip, and RandomVerticalFlip for data augmentation. [8]

In then we tried more aggressive data methods, i.e., image stacking and ensembling. Since we wanted to fit all the data into memory, we have to sacrifice the input image size. In both experiments, we take the 32×32 FiveCrops (the 4 corners and the center crop) out of the image scaled to 45×45. For image crop staking, stack the 5 crops together into a $15 \times 32 \times 32$ tensor, and modify the input convolution layer to have 15 input channels. While for the ensembling method, we defined 5 ResNets and feed the 5 crops into one of the nets respectively.

4 Experiments

We used AdaDelta [9, 7] as optimizer in all our experiments. The training data was shuffled. In most cases, the batch size is set to 64. We did quite a few experiments on the Prince HPC cluster. For each job, we asked for a 24-hour quota. However, not all results are meaningfully to present here, mostly because the training is too low. In some cases, the training only moved forward 40 epochs in 24 hours (probably on K80). We will present these results in appendix.

4.1 ResNet152

As an naïve attempt, we first adopt the PyTorch built-in ResNet152 implementation.[6] The orginal implementation is customized for ImageNet, where the popular input size is 224×224. However, the nature of our task makes it hard to read the image from disk on the fly. We need to fit our data into memory. In this experiment, we took the 224 by 242 square in the center, and scale it to 64 by 64. We therefore change the output average pooling layer from 7 by 7 to 2 by 2. Fig. 1 shows the training and validation loss.

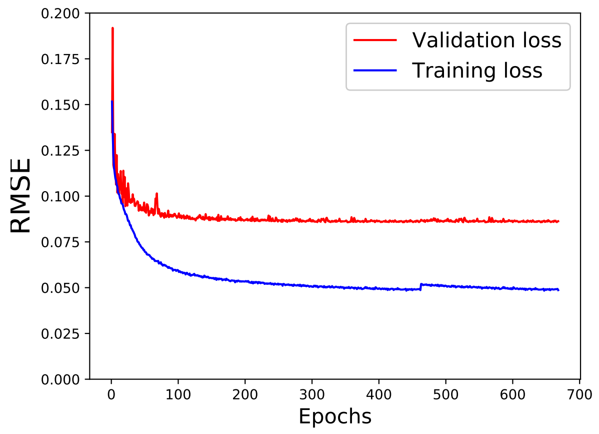

Figure 1: Loss using modified ResNet152

The training loss went down steadly, but the validation loss plateaued after 200 epochs. The best validation loss is 0.0857. There is a glitch in training loss. That is because after 24 hours of training, the job times out, we then start a new job to continue. AdaDelta needs some time find the perfect setup. We already see a sense of overfiting in that the training loss is much larger than the validation loss. But the validation loss and the training loss plateaus almost spontaneously.

4.2 DenseNet121

We then adopt the PyTorch built-in DenseNet121 implementation.[6] The orginal implementation is also customized for 224×224 input. In this experiment, we took the 224 by 242 square in the center, and scale it to 64 by 64.. We therefore change the output average pooling layer from 7 by 7 to 2 by 2.

DenseNet[3] is very similar to ResNet[2], but has denser and more aggressive connections among convolution layer. Therefore with less layers, DenseNet is believed to achieve similar or even better performence than ResNet. In this experiment, we set input batch size to 32. Fig. 2 shows the training and validation loss.

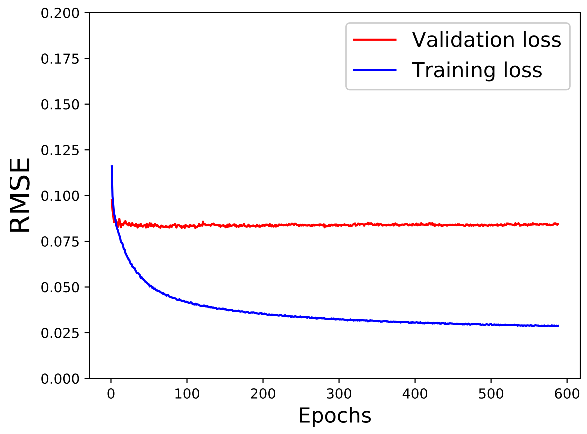

Figure 2: Loss using DenseNet121

Compared to Fig. 1, the training loss is much smaller. The validation loss is also improved (plateaued around 0.0822). However, the validation loss plateaued in less than 100 epochs, but the training loss was still going down. That is strong indication of overfitting.

4.3 Image Crop Stacking

In this experiment, we introduce data augmentation and stacking. We first took the 224 by 242 square in the center, and scale it to 45 by 45. We then applied the PyTorch transforms RandomHorizontalFlip, and RandomVerticalFlip[8], then we take the five 32 by 32 crops: four corners and the center crop, from the 45 by 45 image, and stacked them and normalized into a $15 \times 32 \times 32$ tensor. Although this image size may loss a lot of information due to down sampling, we think it worth trying, since more input channels contain more information. Fig. 3 shows the training and validation loss.

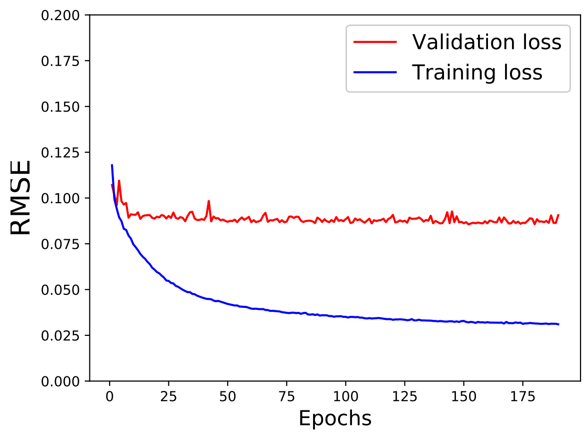

Figure 3: Loss using image crop stacking with ResNet152

Compared with Fig. 1, Fig. 3 achieves much lower training loss, but the validation loss is merely improved (around 0.0856). The validation loss stopped decreasing in less than 50 epochs, but the training loss kept decreasing. That is strong indication of overfitting.

5 Discussion

The test score of our models discussed above are shown in Tbl. 1.

Model	Public Score	Private Score
ResNet152	0.08860	0.08893
DenseNet121	0.09066	0.09063
StackResNet152	0.09394	0.09433

Table 1: RMSE loss on test set

In this experiment, we revisited the Kaggle Galaxy Zoo challenge held in 2014. Almost 4 years has passed, during which a lot of fancy and sophisticated network structures have been invented, such as the ResNet and DenseNet. However, using these state-of-the-art models, we failed to get as good performance as the winners 4-year-old record. Our best submission merely ranked top 20.

The most serious problem we are facing is overfitting. We think this is due to the models we are using. Theses models are prefect fit for large and tough tasks, such as ImageNet, an 1000-class classification problem, but may not fit in this task, which can be shaped as a 37-class regression problem. Thanks to ImageNet, we are having more and more sophisticated and advanced deep models, but not all tasks are as tough as ImageNet. In the Galaxy Zoo competition, the winning solutions are using ConvNets that are "very shallow", at least in today's viewpoint. Shallow as they were, they managed to capture much more moderate yet reasonable amount of features. This raised a very profound question: given the problem size, what is the proper scale of the ConvNet to use?

We also figured out that data normalization is very very important. With out normalization, it is still possible to get a good validation loss, but when testing, the result could be surprising. When stacking the images, we forgot to normalize the other channels than the first 3. The test score went as high as 0.29, worse then the all-zero benchmark.

Anyways, the most important lesson we learned from the experiments is that powerful as deep learning is, it is not suitable for all tasks. For a small problem like ours, a smaller model may be more robust and reasonable.

References

[1] S. Dieleman, K. W. Willett, and J. Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[4] T. D. Nguyen and T. Tran. Galaxy zoo - the galaxy challenge - discussion - so, what were your approaches? *https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/discussion/7599*, 2014.

[5] T. D. Nguyen and T. Tran. Galaxy zoo challenge with convolutional neural networks. *http://docs.nvidia.com/cuda/maxwell-tuning-guide/index.htmll1-cache*, 2014.

[6] PyTorch. pytorch/vision. *https://github.com/pytorch/vision/tree/master/torchvision/models*, 2017.

[7] TorchContributors. torch.optim — pytorch master documentation. *http://pytorch.org/docs/master/optim.htmltorch.optim.Adadelta*, 2017.

[8] TorchContributors. torchvision.transforms — pytorch master documentation. *http://pytorch.org/docs/master/torchvision/transforms.html*, 2017.

[9] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.