



Galaxy Zoo - The Galaxy Challenge

Classify the morphologies of distant galaxies in our Universe

\$16,000 · 326 teams · 3 years ago



George Oblapenko

So, what were your approaches?

posted in [Galaxy Zoo - The Galaxy Challenge](#) 3 years ago



With the competition nearing to an end, why don't we share our approaches? It might be interesting regardless of what place they received.

I'll start first, since there's no way I'll manage to do anything in the next 33 hours (last time my classifiers trained for 4 days).

I had a team, but I was the only member who did any work, and I started only by the end of February, so I didn't have a lot of time to try different stuff.

I used Python, scikit-image and Mahotas for image processing, and the GradientBoostingClassifier from scikit-learn for the multi-class classification (increased the `max_depth` parameter to 5 and had 150 estimators on my most successful attempt).

My features can be split in 2 groups: general and local.

The general features included values of the Otsu threshold parameter for each colour channel (and the grey image, too), mean and std for the whole grey image (and for each of the colour channels too), colour percentages, a 10-bin histogram (how many pixels in each bin), and some similar stuff (ratio of red to green pixels, etc.), Zernike moments, average distance between local maxima, amount of separate regions.

The local features were features I computed for the largest region (after doing some small eroding): ratio of galaxy size to bounding box size, ratio of convex hull size to galaxy size, ratio of the axes of the best approximation ellipse, offset of the brightest spot from the "centre of mass", Hu moments, perimeter divided by equivalent diameter, perimeter divided by the amount of pixels in the skeletonized version of the region.

Last thing I did was a PCA on the central part of the image, and it gave a few points, so I did a bigger PCA (which gave a big feature vector which took 4 days to analyse), but that actually made the result worse.

concerned Hu and Zernike moments).

In the end, I didn't have too much time to work on this challenge, I started too late, I hardly know anything about image classification, and alone I lacked the computing power to perhaps try some bigger things (some robust feature extractors, for example). Sadly, the team was kind of non-existent.

And btw, I started by creating a Python library to simplify similar tasks (extracting/storing features from a large amount of files): <https://pypi.python.org/pypi/mldatalib>

So, what did features did you use?

P.S. Our team is at 155th place currently, so the features I used are not exactly great.

Options

Comments (38)

Sort by Hotness

Please [sign in](#) to leave a comment.



sedielem • (1st in this Competition) • 3 years ago • Options

^ 34 v

A detailed overview of my solution can be found here: <http://benanne.github.io/2014/04/05/galaxy-zoo.html>

tl;dr: I used a convolutional neural network (surprise surprise) with a modified architecture to exploit the rotation invariance of the images and increase parameter sharing. I divided each image into a bunch of overlapping parts, rotated them so they all had the same orientation, and applied the same set of convolutions to each one. The resulting features were aggregated in the dense part of the network. I also incorporated the decision tree constraints in the output layer of the network.

My best single model had 7 layers in total: 4 convolutional layers and 3 dense layers. I used dropout for regularisation, as well as maxout in the dense layers. I also made extensive use of data augmentation (both during training and to generate test set predictions).

My final submission was a blend of 17 models.

I used Python, NumPy and Theano to implement everything, as well as the Theano wrappers for the cuda-convnet convolution implementation that come with pylearn2.

Thanks to Kaggle and the organisers for a very interesting competition!



Maxim Milakov • (2nd in this Competition) • 3 years ago • Options

^ 12 v

Hi all,

I used a convolutional neural network, although simpler one than Soumith did. Decision tree constraints were not used. I will prepare the report soon, meanwhile you could check the solution (source code + instructions), I published it as one of the examples on using nnForge library: https://github.com/milakov/nnForge/tree/master/examples/galaxy_zoo.

Best regards,

Max.



Soumith Chin... • (9th in this Competition) • 3 years ago • Options



Congrats to Sander on his excellent work regularizing the symmetries, it was definitely novel.

We used a standard deep convnet that runs on the GPU, input jittered by rotation, translation, hflip, vflip, scaling. Testing is averaged on a deterministic output of 256 jitters for each input image.

Each of the decision tree question outputs were a probability distribution using a SoftMax layer, and the cost function normalized the outputs by their question weightings (exactly as how the decision tree is structured)

We did a simple averaging of 4 models trained on two Titan GPUs.

We started training the models less than 7 days before the competition, so might have been a little short on time.

The code is in Torch-7 (Lua based language) and is available here:

<https://github.com/soumith/galaxyzoo>

Hope this helps, and if you want further documentation for the code, please feel free to open an issue on the github repository itself.



pjsugi • (38th in this Competition) • 3 years ago • Options



This being my first Kaggle competition, I approached this competition as a learning experience, choosing to hand-craft my features. I was also curious to see how well a feature processing approach would get to black-box neural network algorithms. I was initially disappointed to place outside the top-10%, but since it seems most/all of the top finishers used CNNs, I am happy to place so highly! I am curious to know the highest placed finish that hand-crafted their features.

For software, I used a python stack: OpenCV for image processing and scikit-learn for machine learning models. I was able to create my features in less than an hour using full-sized images. The types of feature classes I tried were related to:

1. contours (ellipticity, areas, aspect ratio, solidarity, location)
2. color
3. rotational asymmetry (b&w and color)

6. intensity concentration metrics (bulge-disk ratio, %light within a given radius, gini coefficients, etc.).

7. spirality metrics. e.g. number of peaks, slope of peaks (at a given radius, plot the intensity as a function of angle, calculate peak-trough intensity difference and how the peaks move as a function of radius)

Of these features, aspect ratio, color, gini coefficient, bulge-disk ratio, and rotational asymmetry turned out to be the most important, but none was dominant.

I tried a variety of models, linear models (ridge, lasso, glm), random forests, and GBMs. I found that GBMs performed the best, giving me RMSE of ~.1 on the public leaderboard compared to .105 or more with random forests and linear models. I believe this is because all of my features were relatively weak and with a reasonably sized dataset, GBMs were able to combine the weak learners into a stronger prediction.

I obtained my final score of .0977 by using a linear ensemble of GBMs.

I did not use a neural network which I felt I would not learn as much from since neural networks are more of a black box algorithm. However, Sedielem's excellent writeup shows that it is not merely the faster computer that won, rather there is room for subtlety and technique. If I enter another image processing competition, I will certainly give deep neural networks a try.



tund • (3rd in this Competition) • 3 years ago • Options



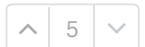
Hi,

Here are the [\[report\]](#) and [\[code\]](#) of our team.

Sorry, I'm a little bit late because of family moving house.



npetitclerc • (75th in this Competition) • 3 years ago • Options



Hi,

I'm one of those who didn't used CNN, but more features engineering. Clearly the winning technique for this competition was CNN, but I though I would share my work. By comparing which features worked best maybe we can learn something new.

My best result was: 0.10564

I got this doing a fit on every class individually and using either SVM with rbf kernel, ExtraTrees or GradientBoosting from Scikit-learn, which ever worked best for a given class.

I developed 48 features, some based on published methods and some of my own. I mostly focused on the central 210x210 pixels images converted to grey scale. I made all the features scale free by measuring ratios.

Here's a list of my 48 features:

skimage.feature.greycoprops texture properties:

0-ent, 1-std, 2-grad, 3-contrast, 4-energy, 5-homogeneity, 6-dissimilarity, 7-correlation, 8-ASM

Then I fitted a 2D ellipse to the galaxy to get the global shape of the galaxy as well as the exact location of the centre of the galaxy.

9-Amplitude, 10-Eccentricity,

I measured the maximum brightness of the bulge (average brightness of the central 10x10px), the disk average and median brightness, as well as their ratio.

11-bmax, 12-dmed, 13-davg, 14-bmax/davg, 15-bmax/dmed, 16-davg/dmed,

Of course I had to measure the color differences, I measure the average over the central 100px.

17-R-G, 18-R-B, 19-G-B

I thresholded the image at 20% and counted the number of individual contours I got.

20-ncont

I calculated the radial profile of the galaxy, then I calculated the averaged and max standard deviation of all the radial bins, and the standard deviation of the radial bins standard deviation.

21-rstd, 22-mrstd, 23-rstdstd

I fitted a Sersic profile to the radial profile

24-n, 25-bn,

Then I calculated the radius containing 10, 20, 40 and 60 % of the light, and calculated their ratios.

26-1020, 27-1040, 28-1060, 29-2040, 30-2060, 31-4060

I thresholded the image at 45% of the maximum and measured the Compactness and Distance of the contour. Then I fitted an ellipse to that contour and kept only the difference between the two and found the contours of those small differences. I measured the area of the biggest contour divided by the total area of difference and the biggest contour divided by the total area of the ellipse. Then I measured the average contour area and the area per contour.

32-compactness, 33-distance, 34-big_diff, 35-big_prop, 36-avca, 37-aperc

Then I looked at the image intensity histogram, I found there is generally a linear regime between bins at 40 and 120 (pixel intensity was between 0 and 255). But some galaxies had a significant bump in that region. So I fitted a straight line between those points and measured the standard deviation and the maximum difference to the straight line.

38-std2, 39-max2,

Finally, I measured the ellipticity of the galaxy as well as the number of contours at different thresholds.

40-ell75, 41-ell50, 42-ell35, 43-ell25, 44-nc75, 45-nc50, 46-nc35, 47-nc25

So now the question is **which one of those features helped the prediction the most.**

9.28 % by feature: 19 - G-B

9.21 % by feature: 40 - ell75

7.74 % by feature: 42 - ell35

6.67 % by feature: 18 - R-B

Most important features for class 1

6.46 % by feature: 28 - 1060

5.66 % by feature: 30 - 2060

4.85 % by feature: 27 - 1040

3.97 % by feature: 19 - G-B

3.12 % by feature: 38 - std2

Most important features for class 2

6.47 % by feature: 28 - 1060

4.77 % by feature: 30 - 2060

3.78 % by feature: 27 - 1040

3.42 % by feature: 19 - G-B

2.64 % by feature: 29 - 2040

Most important features for class 3

3.89 % by feature: 30 - 2060

3.53 % by feature: 28 - 1060

3.12 % by feature: 27 - 1040

3.06 % by feature: 29 - 2040

2.65 % by feature: 18 - R-B

Most important features for class 4

3.9 % by feature: 38 - std2

3.71 % by feature: 19 - G-B

3.17 % by feature: 28 - 1060

2.91 % by feature: 30 - 2060

2.13 % by feature: 40 - ell75

Most important features for class 5

13.48 % by feature: 34 - big_diff

12.55 % by feature: 38 - std2

9.24 % by feature: 6 - dissimilarity

7.61 % by feature: 33 - distance

6.49 % by feature: 39 - max2

Most important features for class 6

5.69 % by feature: 42 - ell35

5.39 % by feature: 21 - rstd

4.98 % by feature: 41 - ell50

4.43 % by feature: 43 - ell25

3.34 % by feature: 22 - mrstd

Most important features for class 7

1.86 % by feature: 22 - mrstd

1.71 % by feature: 38 - std2

1.56 % by feature: 18 - R-B

Most important features for class 8

2.38 % by feature: 21 - rstd

1.26 % by feature: 43 - ell25

1.19 % by feature: 23 - rstdstd

1.02 % by feature: 22 - mrstd

0.97 % by feature: 41 - ell50

Most important features for class 9

2.38 % by feature: 34 - big_diff

2.26 % by feature: 28 - 1060

2.08 % by feature: 19 - G-B

1.78 % by feature: 27 - 1040

1.73 % by feature: 30 - 2060

Most important features for class 10

2.24 % by feature: 28 - 1060

2.17 % by feature: 34 - big_diff

2.08 % by feature: 19 - G-B

1.64 % by feature: 18 - R-B

1.62 % by feature: 30 - 2060

Most important features overall

3.12 % by feature: 34 - big_diff

2.89 % by feature: 19 - G-B

2.58 % by feature: 38 - std2

2.54 % by feature: 28 - 1060

2.19 % by feature: 30 - 2060

2.1 % by feature: 41 - ell50

2.07 % by feature: 18 - R-B

1.9 % by feature: 21 - rstd

1.77 % by feature: 27 - 1040

1.76 % by feature: 6 - dissimilarity

1.64 % by feature: 42 - ell35

1.63 % by feature: 23 - rstdstd

1.58 % by feature: 40 - ell75

1.55 % by feature: 22 - mrstd

1.49 % by feature: 33 - distance

1.32 % by feature: 29 - 2040

1.21 % by feature: 1 - std

1.08 % by feature: 39 - max2

1.06 % by feature: 43 - ell25

0.96 % by feature: 31 - 4060

0.79 % by feature: 3 - contrast

0.78 % by feature: 32 - compactness

0.7 % by feature: 44 - nc75
 0.63 % by feature: 26 - 1020
 0.47 % by feature: 7 - correlation
 0.43 % by feature: 35 - big_prop
 0.4 % by feature: 9 - Amplitude
 0.39 % by feature: 11 - bmax
 0.34 % by feature: 10 - Eccentricity
 0.32 % by feature: 45 - nc50
 0.27 % by feature: 36 - avca
 0.25 % by feature: 4 - energy
 0.24 % by feature: 8 - ASM
 0.24 % by feature: 37 - aperc
 0.23 % by feature: 14 - bmax/davg
 0.22 % by feature: 17 - R-G
 0.21 % by feature: 5 - homogeneity
 0.17 % by feature: 25 - bn
 0.16 % by feature: 15 - bmax/dmed
 0.16 % by feature: 46 - nc35
 0.14 % by feature: 13 - davg
 0.14 % by feature: 0 - ent
 0.14 % by feature: 20 - ncont
 0.12 % by feature: 16 - davg/dmed
 0.06 % by feature: 47 - nc25
 0.04 % by feature: 24 - n

I'm a bit surprised big_diff turned out to be the most important features, I didn't expect that. The colour difference feature is no surprise there. Also, the linear regime in the intensity histogram seems to be a key feature as well.

Which features did you use?

[C0_features_imp.png \(25.17 KB\)](#)

[A_features_imp.png \(31.2 KB\)](#)



tund • (3rd in this Competition) • 3 years ago • Options



Hi all,

Many thanks to Kaggle, the organizers and all competitors for a very interesting challenge!

Congrats sedilem! You won with a large margin.

My team also used Convolutional Neural Network with the code developed from [cuda-convnet](#) of Alex Krizhevsky -- great thanks to your very fast implementation!

Our architecture is a smaller and slightly modified version of the OverFeat. We did almost tricks which are similar to sedilem's. We also blended the outputs of several models. The decision tree, however, did not help

We will write things up soon.

Best,

Tu



X • (13th in this Competition) • 3 years ago • Options

^ 5 v

A 9-layer deep neural net trained in 2 days (shame I entered the competition too late) on raw RGB data using simple square loss gave me the current rank. I am at 14th now and I hope I can stay top 20 eventually...

The interesting thing is that, although there is a complicated 'decision tree' structure, ignoring it still gets something reasonably good.



X • (13th in this Competition) • 3 years ago • Options

^ 6 v

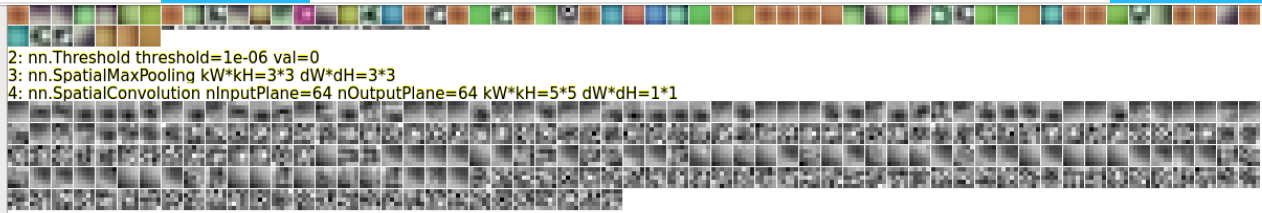
mlearn wrote

I'd be keen to see more about your approach. Ignoring computational cost, your approach sounds simple (e.g. no pre-processing, ensembling etc.) and it did very well. Thanks.

The neural network I was referring to is actually a convolutional neural network. Same as Soumith, I also use Torch 7 (in Lua programming language). The code can be accessed here:

<https://github.com/zhangxiangxiao/GalaxyZoo>

It has a nice visualization of the network using qlua, as the picture below. It is the final model I used to submit my results, although I had only 2 days to train it on one NVidia GTX Titan.



```

2: nn.Threshold threshold=1e-06 val=0
3: nn.SpatialMaxPooling kW*kH=3*3 dW*dH=3*3
4: nn.SpatialConvolution nInputPlane=64 nOutputPlane=64 kW*kH=5*5 dW*dH=1*1

```

```

5: nn.Threshold threshold=1e-06 val=0
6: nn.SpatialZeroPadding pad l=1 pad r=1 pad t=1 pad b=1
7: nn.SpatialMaxPooling kW*kH=3*3 dW*dH=3*3
8: nn.SpatialConvolution nInputPlane=64 nOutputPlane=128 kW*kH=3*3 dW*dH=1*1

```

```

9: nn.Threshold threshold=1e-06 val=0
10: nn.SpatialZeroPadding pad l=1 pad r=1 pad t=1 pad b=1
11: nn.SpatialConvolution nInputPlane=128 nOutputPlane=256 kW*kH=3*3 dW*dH=1*1

```

```

12: nn.Threshold threshold=1e-06 val=0
13: nn.SpatialConvolution nInputPlane=256 nOutputPlane=512 kW*kH=3*3 dW*dH=1*1

```

```

14: nn.Threshold threshold=1e-06 val=0
15: nn.SpatialMaxPooling kW*kH=3*3 dW*dH=3*3
16: nn.SpatialConvolution nInputPlane=512 nOutputPlane=1024 kW*kH=3*3 dW*dH=1*1

```

```

17: nn.Threshold threshold=1e-06 val=0
18: nn.Reshape
19: nn.Linear inputSize=1024 outputSize=2048
20: nn.Threshold threshold=1e-06 val=0
21: nn.Dropout p=0.5
22: nn.Linear inputSize=2048 outputSize=2048
23: nn.Threshold threshold=1e-06 val=0
24: nn.Dropout p=0.5
25: nn.Linear inputSize=2048 outputSize=37

```



mymo • (12th in this Competition) • 3 years ago • Options



I also use CNN for this task. Perhaps I will just add a few minor things that I found useful for this challenge.

1. I use polar coordinates, which gives improvement over Cartesian coordinates based on some early networks that I trained.
2. The networks are trained as regression models to minimize the RMS loss, (i.e., the outputs of the networks are the 37 labels). However, feeding the outputs of the next-to-the-last layer to a RidgeCV regressor of sklearn, then use its prediction instead of the raw outputs of the networks still gives a significant improvement.
3. To exploit the relations between labels, I feed the predictions and their cross products to a ridge regressor. This gives some improvement on the results (~0.003 points). I tried to incorporate decision tree constraints by regressing the normalized scores, but it does not seem to make much difference over the results that I obtained from using cross products.



Maxim Milakov • (2nd in this Competition) • 3 years ago • Options



Hi,

Here is the report I promised to

publish: https://github.com/milakov/nnForge/blob/master/examples/galaxy_zoo/galaxy_zoo.pdf?raw=true

Julian de Wit • (5th in this Competition) • 3 years ago • Options

^ 4 v

Not much to add.

I used this competition to get some more mileage with 'practical' NN problems. It was all pretty much trial and error.

Sedilem's approach was much more subtle so I think he's a deserving winner.

Cropped to around 160x160 with zoom variations. Then rescaled to 96x96.

With small translations and rotations this resulted in an almost infinite set of training images.

The sweetspot for me was 96 filters in first layer and 11x11 kernel.

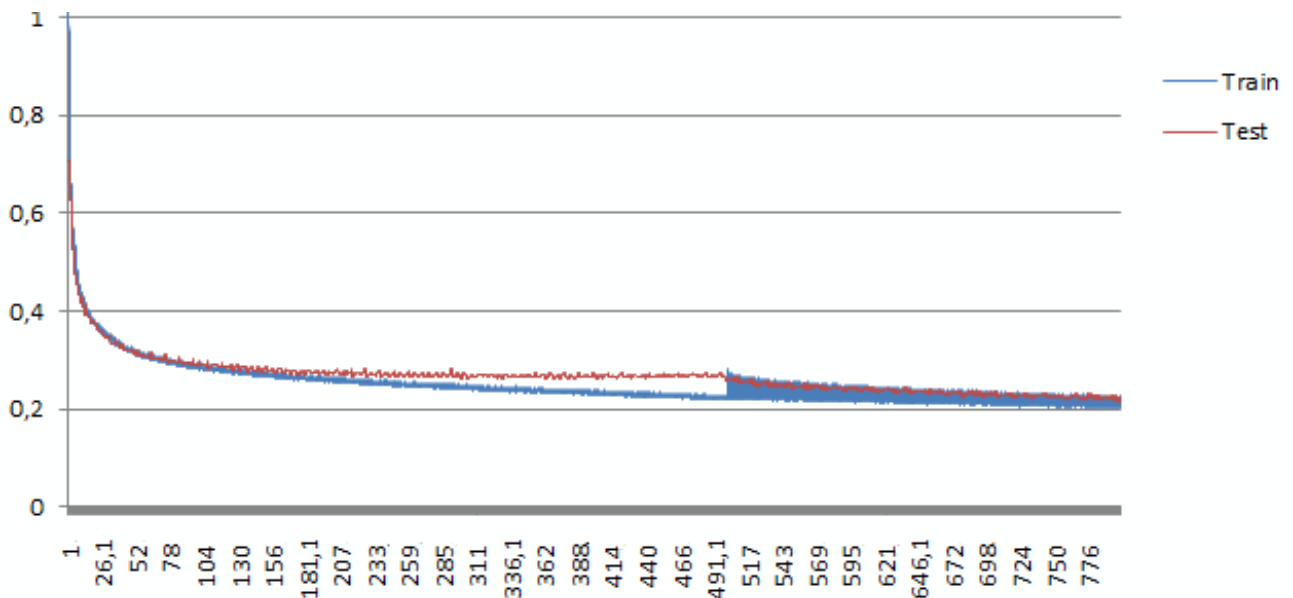
Then two extra convolutions and one big fully connected with dropout.

Softmax didn't help me (even for Q1) so it was one big final layer with 37 values.

A faulty GPU and limited time and resources barred me from seriously ensembling..

Predictions were averages of many variations of the test images. I also did small rectifications on the final results. (>1 and <0).

I'm really fascinated by the filters and I've been looking at learning curves for so long that I think it's nice to post them. I'm interested if someone has completely different figures.. $Y = \text{sum of squares}$ and must be read as $\text{Sqrt}(Y/37)$..





I had a question about the filters.. Many seem dead.. However, lowering the amount gave worse results..



Beck Chen • (185th in this Competition) • 3 years ago • Options



sugi wrote

This being my first Kaggle competition, I approached this competition as a learning experience, choosing to hand-craft my features. I was also curious to see how well a feature processing approach would get to black-box neural network algorithms. I was initially disappointed to place outside the top-10%, but since it seems most/all of the top finishers used CNNs, I am happy to place so highly! I am curious to know the highest placed finish that hand-crafted their features.

For software, I used a python stack: OpenCV for image processing and scikit-learn for machine learning models. I was able to create my features in less than an hour using full-sized images. The types of feature classes I tried were related to:

1. contours (ellipticity, areas, aspect ratio, solidarity, location)
2. color
3. rotational asymmetry (b&w and color)
4. smoothness
5. exponential fit to the minor/major axis
6. intensity concentration metrics (bulge-disk ratio, %light within a given radius, gini coefficients, etc.).
7. spirality metrics. e.g. number of peaks, slope of peaks (at a given radius, plot the intensity as a function of angle, calculate peak-trough intensity difference and how the peaks move as a function of radius)

I tried a variety of models, linear models (ridge, lasso, glm), random forests, and GBMs. I found that GBMs performed the best, giving me RMSE of ~ 1 on the public leaderboard compared to $.105$ or more with random forests and linear models. I believe this is because all of my features were relatively weak and with a reasonably sized dataset, GBMs were able to combine the weak learners into a stronger prediction.

I obtained my final score of $.0977$ by using a linear ensemble of GBMs.

I did not use a neural network which I felt I would not learn as much from since neural networks are more of a black box algorithm. However, Sedielem's excellent writeup shows that it is not merely the faster computer that won, rather there is room for subtlety and technique. If I enter another image processing competition, I will certainly give deep neural networks a try.

Hi sugi, you are probably the highest ranked person who used handcrafted features. Do you mind posting your detailed feature extraction code? I would be very interested in learning about it.

We used a Python stack and gradient boosting too, and the final RMSE we got was 0.1237 . The features we used were contour area, eccentricity, orientation, solidity, avg and std of pixel intensity between 40 and 160, color differences (b-g, g-r, r-b), and radial profiles.



tund • (3rd in this Competition) • 3 years ago • Options



Rafael wrote

tund wrote

Hi,

Here are the [\[report\]](#) and [\[code\]](#) of our team.

Sorry, I'm a little bit late because of family moving house.

Hi and thank you for publishing code. Convnet for windows64 is difficult to compile. Would it be possible for you since you kindly made your code available to include the necessary files to run your code.

.....

Initialized neuron layer 'fc8_neuron', producing 2048 outputs

Initialized neuron layer 'fc37_neuron', producing 37 outputs

=====

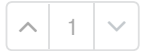
Importing pyconvnet C++ module

ImportError: No module named pyconvnet

Clearly some files from pyconv are missing. Because their compiling is so difficult and obviously you have done it could you please make it available in your git account?



hxu • (57th in this Competition) • 3 years ago • Options



Congrats to the winners, and thanks to the organizers for putting together a challenging competition.

You can find the code and more details about our final model here: <https://github.com/hxu/galaxy-zoo>

This was our first real project working with image data, and we had little prior experience with neural networks, much less deep learning. We spent a lot of time reading the literature in this area and settled on an approach outlined in [this paper](#) from [Adam Coates](#) and Andrew Ng of Stanford, which is a k-means based feature learner. We ended up using this model instead of a convnet because it was simpler and easier for us to understand -- we just didn't think that we could implement an effective deep learning network given the time and our resources (no GPU machines). The model actually worked quite well and we were pleased with the results, especially given the comparatively quick training time (less than 2 hours end to end) using only Python/Sklearn.

The main challenge that we encountered was in tuning. Because of our inexperience, it was hard to know what parameters might improve the model, and the long training time made it impractical to try lots of parameters at random. Since tuning yielded little improvement, we also had nowhere to go after the initial model was implemented. We tried playing with multi-level k-means feature generators late in the competition (as outlined in the paper), but couldn't get the code to work correctly.

Thanks to all of the other competitors for sharing their code -- we'll definitely be studying it closely to learn more about convnets!



Alec Radford • 3 years ago • Options





sedielem • (1st in this Competition) • 3 years ago • Options



X wrote

Ah huh, convnet must be the best for winning this!

Did the 'decision tree' output structure play a role?

When I introduced it into the network I went from about 0.0835 on my own validation set to 0.0831 iirc, this is not a huge jump but significant, I think. After that I didn't test without it again, so I don't know how much of a difference it would make now.

As mentioned in the post I linked earlier, it only helped if I applied rectification and then divisive normalisation - using a softmax function per question didn't help.



Chenglong Ch... • (58th in this Competition) • 3 years ago • Options



I entered this after the loan default competition. I use a CNN+MLP with square error cost function implemented in theano. The network configurations are: 2 layer CNN each followed by max-pooling + 2 layer MLP + 1 logistic regression output layer (not very deep); all the activation functions are ReLU's to speed up the training. I compared the implementation with and without those probability constraints illustrated in the Galaxy Zoo decision tree, but did not see a significant difference. (Those constraints are encoded in the cost function, and thanks to theano, I don't have to bother the derivation of the gradient.)

Since my laptop doesn't have enough memory to hold all the training data, I first cropped and resized each image to the same 44x44x3, and read in a few mini-batches for training each time. I also make predictions for the testing images every 5~10 epoches since I don't have GPU either and the whole training process (for 2000 epoches) will take a week or something like that. So, I have to make sure I can make submission in case the deadline comes before the training ends...(Next time, I might better prepare more memory and a GPU!!)

My NN currently doesn't seem to break the 0.10 limit, but I hope I can stay 25% in the private leaderboard...

Er, my approach is not very promising, but I would like to share in case it helps anyone. You can find it here: https://github.com/ChenglongChen/Kaggle_Galaxy_Zoo

Congrats to everyone and for sharing your thought here!! Thank you sedielem for posting the winning solution!! I am going to spend some time on it and to learn more about theano/pylearn2.



X • (13th in this Competition) • 3 years ago • Options



sedielem wrote

A detailed overview of my solution can be found here: <http://benanne.github.io/2014/04/05/galaxy-zoo.html>

tl;dr: I used a convolutional neural network (surprise surprise) with a modified architecture to exploit the rotation invariance of the images and increase parameter sharing. I divided each image into a bunch of overlapping parts, rotated them so they all had the same orientation, and applied the same set of convolutions to each one. The resulting features were aggregated in the dense part of the network. I also incorporated the decision tree constraints in the output layer of the network.

My best single model had 7 layers in total: 4 convolutional layers and 3 dense layers. I used dropout for regularisation, as well as maxout in the dense layers. I also made extensive use of data augmentation (both during training and to generate test set predictions).

My final submission was a blend of 17 models.

I used Python, NumPy and Theano to implement everything, as well as the Theano wrappers for the cuda-convnet convolution implementation that come with pylearn2.

Thanks to Kaggle and the organisers for a very interesting competition!

Did the decision tree output structure play a role?



zero zero • (234th in this Competition) • 3 years ago • Options



hey, back in January (I think it was January), I started this comp by using opencv to extract out the region of interest. I did this by using some thresholding, then looking for the contour that contained the middle pixel, then blacked out some areas outside that region and then thresholded again. With the final thresholding, I picked the central contour and got the best fit ellipse. With the data for the ellipse (angle, major/minor axes), I rotated the image to make the ellipse horizontal (and stored the aspect ratio) and clipped the rectangular area using the bounding box. With a quick calculation, I resized to 64 by 64 without stretching (started with 64x64 black and resize the x axis of extracted patch to 64 and proportionally scale the y). This yielded a 64x64 version of the original photos with (mostly) only the feature in the pic. It actually worked very well and I was excited to use the much smaller images in some NN and CNNs mixed with my aspect ratios which did a reasonable job telling the 'cigariness', etc.. I don't mind giving the opencv code away after the comp if anyone thinks they can improve their score with better/smaller images.

Then I got a job and didn't spent any time until March-madness was over....so I sadly have not been able to capitalize on my nice little images! I was also planning to experiment with another method of resizing the images....which is to take the ellipse/bbox and turn it into a square, whereby turning the disks into circles to make the spiral searching not have to learn anything about flatness, and with added rotations, a whole lot of data could be produced.

My final results are composed of using my aspect ratios and some very hastily thrown together NNs tuned with some of the obvious conditional probs for a few of the tree branches. shucks....sometimes work gets in the way of fun hobbies such as this.



mymo • (12th in this Competition) • 3 years ago • Options



For labels 5 and 6 ('bar' question), after around 70 epoches, the square loss on a validation set for Cartesian is about 0.055 and 0.04 for polar coordinates (although I expect them to get closer after more epoches). For labels 7 and 8 ('Is there spiral arm' question), after about 200 epoches, Cartesian and polar reaches about 0.05 and 0.045 respectively. I only started training all 37 labels together at a later stage, so I have not compared the results for other labels. Also the nets I used are rather small so the effect of preprocessing is likely to be more prominent. If you are interested, I have a python script for generating the polar coordinate images (requires opencv), just call the 'create_polar' function in an interpreter.

[polar_coord.py \(3.78 KB\)](#)



Kevin Keraudr... • (56th in this Competition) • 3 years ago • Options



Hi,

My approach, detailed in this [blog post](#), consists in training SVM classifiers on the most characteristic galaxies, and putting the probabilistic output of these classifiers in a regression Random Forest trained on the whole dataset.

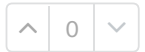
The code is on [github](#).

I am glad I took part in this challenge!

Kevin



utdiscant • (28th in this Competition) • 3 years ago • Options



I trained a convolutional neural network with the following architecture:

8C9-S2-16C5-S2-32C5-S2-64C4-F37

where 8C9 means a convolutional layer with 8 kernels of size 9x9 pixels. S2 means a sub-sampling layer with a scaling factor of 2 (I used average pooling). F37 means a fully-connected layer with 37 output neurons. I used the MATLAB implementation by Rasmus given here (<https://github.com/rasmusbergpalm/DeepLearnToolbox>).

I preprocessed the images slightly (to get to 64x64 pixels) before inputting them into the network, used three input images (one for each color), and I simply modelled the problem as a regression problem with 37 outputs.

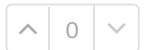
For training I used 55,000 images each rotated 4 times (giving 220,000 images in total). When predicting on the test set I inputted each image with 4 rotations and made a flat average. I finally post-processed the predictions to match the 11 constraints specified by the decision tree.

The biggest issue was training time, which reached 2 weeks on a university cluster. The implementation we used ran only on CPU and was written in MATLAB. I used quite a bit of time trying to optimize a more clever cost-function by using 11 softmax units and then deriving correct gradients, but this did not give anything useful. I also tried ensembling together multiple network architectures - this did not help significantly either.

Edit: I gave a more thorough description of my approach here: <http://www.davidwind.dk/?p=49>



mlearn • 3 years ago • Options



X wrote

A 9-layer deep neural net trained in 2 days (shame I entered the competition too late) on raw RGB data using simple square loss gave me the current rank. I am at 14th now and I hope I can stay top 20 eventually...

The interesting thing is that, although there is a complicated 'decision tree' structure, ignoring it still gets something reasonably good.

(e.g. no pre-processing, ensembling etc.) and it did very well. Thanks.



tund • (3rd in this Competition) • 3 years ago • Options

^ 0 v

Hi Rafael, I've contacted you via private message. We can discuss the problems and then post the final reason in the forum later.

Best,



tund • (3rd in this Competition) • 3 years ago • Options

^ 0 v

I'm also interested in some extracted features such as: SIFT (BOW and HOG), GIST. Actually, I did use [LabelMe-ToolBox](#) to generate SIFT features. However it took me about 2 weeks.

Please tell me I was wrong or not? And please suggest me (faster) tools to extract such types of features?

Thanks,



Maxim Milakov • (2nd in this Competition) • 3 years ago • Options

^ 0 v

mymo wrote

I use polar coordinates, which gives improvement over Cartesian coordinates based on some early networks that I trained.

May I ask how much was the improvement?



Maxim Milakov • (2nd in this Competition) • 3 years ago • Options

^ 0 v

mymo wrote

For labels 5 and 6 ('bar' question), after around 70 epoches, the square loss on a validation set for Cartesian is about 0.055 and 0.04 for polar coordinates (although I expect them to get closer after more epoches). For labels 7 and 8 ('Is there spiral arm' question), after about 200 epoches, Cartesian and polar reaches about 0.05 and 0.045 respectively.

It is a big improvement indeed. Thanks a lot for sharing these data!

cipolla • (283rd in this Competition) • 3 years ago • Options

^ 0 v

benchmark file. Then I just tried the sparse autoencoder method described in <http://cs229.stanford.edu/proj2010/Broxtan-AutomatedClassificationWithGalaxyZoolImages.pdf>, which should be quite promising I felt. But I always fall into saturation when I trained the model. I tried some different ways to avoid that but no luck.

I don't have more time to spend on it. So I really would like to find someone used the same method and obtained good results. Even if you didn't choose this method, you can still try it out. Time is still enough.

Good luck to every competitor!



Rafael • 3 years ago • Options

^ 0 v

tund wrote

Hi,

Here are the [\[report\]](#) and [\[code\]](#) of our team.

Sorry, I'm a little bit late because of family moving house.

Hi and thank you for publishing code. Convnet for windows64 is difficult to compile. Would it be possible for you since you kindly made your code available to include the necessary files to run your code.

.....

Initialized neuron layer 'fc8_neuron', producing 2048 outputs

Initialized neuron layer 'fc37_neuron', producing 37 outputs

=====

Importing pyconvnet C++ module

ImportError: No module named pyconvnet

Clearly some files from pyconv are missing. Because their compiling is so difficult and obviously you have done it could you please make it available in your git account?



Rafael • 3 years ago • Options

^ 0 v

tund wrote

I already added the binary file: "pyconvnet.pyd" to git repo. Actually, I missed that file by accident (.gitignore includes that). Could you please clone and run again? Sorry about that!

Dear tund

There is still problem in running the code and I don't want to overuse the forum. We must be close to finding what the problem is and I was wondering if you could help me into running your code smoothly. After

```
run ./cuda_convnet/convnet.py --data-path ./KON/data/ --save-path ./KON/model/ --test-range 00-01
train-range=1-59 --layer-def=./model_config/gz.cfg --layer-params=./model_config/gz_param.cfg --data-
provider=kaggle-galaxy-zoo-128-cropped-x90rot-zoom-memory --test-freq=590 --test-one=0 --crop-
border=4 --epochs=50 --max-filesize=100000
```

And we get:

```
=====
Loading the whole dataset into memory...
Loading batch #60
Loading batch #61
=====
Loading the whole dataset into memory...
Loading batch #1
Loading batch #2
....
Loading batch #59
Initialized data layer 'data', producing 43200 outputs
Initialized data layer 'labels', producing 37 outputs
Initialized convolutional layer 'conv1', producing 116x116 48-channel output
Initialized max-pooling layer 'pool1', producing 39x39 48-channel output
.....
Initialized neuron layer 'fc37_neuron', producing 37 outputs
=====
Importing pyconvnet C++ module
-----
ImportError Traceback (most recent call last)
C:\Anaconda\lib\site-packages\IPython\utils\py3compat.pyc in execfile(fname, glob, loc)
195 else:
196 filename = fname
--> 197 exec compile(scripttext, filename, 'exec') in glob, loc
198 else:
199 def execfile(fname, *where):

C:\Users\midas\Desktop\KAGGLE\GalaxyZoo\kaggle-galaxy-zoo-master\cuda_convnet\convnet.py in
<module>()
230
231 op, load_dic = IGPUModel.parse_options(op)
--> 232 model = ConvNet(op, load_dic)
233 model.start()

C:\Users\midas\Desktop\KAGGLE\GalaxyZoo\kaggle-galaxy-zoo-master\cuda_convnet\convnet.py in
__init__(self, op, load_dic, dp_params)
40 dp_params['multiview_test'] = op.get_value('multiview_test')
41 dp_params['crop_border'] = op.get_value('crop_border')
--> 42 IGPUModel.__init__(self, "ConvNet", op, load_dic, filename_options, dp_params=dp_params)
43
44 def import_model(self):
```

```
86 setattr(self, var, val)
87
---> 88 self.import_model()
89 self.init_model_lib()
90

C:\Users\midas\Desktop\KAGGLE\GalaxyZoo\kaggle-galaxy-zoo-master\cuda_convnet\convnet.py in
import_model(self)
46 print "=====
47 print "Importing %s C++ module" % lib_name
---> 48 self.libmodel = __import__(lib_name)
49
50 def init_model_lib(self):
```

ImportError: DLL load failed: The specified module could not be found.

What am I doing wrong??

My best



George Oblapov • 3 years ago • Options

^ 0 v

X wrote

A 9-layer deep neural net trained in 2 days (shame I entered the competition too late) on raw RGB data using simple square loss gave me the current rank. I am at 14th now and I hope I can stay top 20 eventually...

The interesting thing is that, although there is a complicated 'decision tree' structure, ignoring it still gets something reasonably good.

What language did you use? I never tried neural networks in Python.

Had I had more time, I would've probably tried using scikit-learns RBM implementation to extract features.



Frank Inklaar • (243rd in this Competition) • 3 years ago • Options

^ 0 v

I tried this competition because I participated as volunteer in GalaxyZoo creating this data and I also have a (long time ago) astronomical background. But I don't have much experience in image classification. I started off with neural networks only to discover that Deep Learning was more complicated than I anticipated. The result was mostly overfitted where my model was pretty good in recognizing a particular image but not very good at handling the ones it didn't see before. After some attempts where I couldn't beat a simple predictor

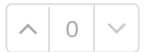
For preprocessing: for memory reasons I limited myself to the central 200x200 part in black and white. I then converted this to a polar coordinate representation where I shifted the x-position reference to the line with the maximum total brightness. This should make the analysis rotation insensitive. I also tried to add some extra features (Fourier, some color data from the originals) but the effect doesn't seem to be very large.

Finally I trained a bunch (20 to 50) SVM models on samples of 3000 each. Looks like I won't have enough left time to train and predict for all the questions.

It wasn't very successful but still fun and I did learn a few things.



Chenglong Ch... • (58th in this Competition) • 3 years ago • Options



I entered this after the loan default competition. I use a CNN+MLP with square error cost function implemented in theano. The network configurations are: 2 layer CNN each followed by max-pooling + 2 layer MLP + 1 logistic regression output layer (not very deep); all the activation functions are ReLU's to speed up the training. I compared the implementation with and without those probability constraints illustrated in the Galaxy Zoo decision tree, but did not see a significant difference. (Those constraints are encoded in the cost function, and thanks to theano, I don't have to bother with the derivation of the gradient.)

Since my laptop doesn't have enough memory to hold all the training data, I first cropped and resized each image to the same 44x44x3, and read in a few mini-batches for training each time. I also make predictions for the testing images every 5-10 epochs since I don't have GPU either and the whole training process (for 2000 epochs) will take a week or something like that. So, I have to make sure I can make submission in case the deadline comes before the training ends...(Next time, I might better prepare more memory and a GPU!!)

My NN currently doesn't seem to break the 0.10 limit, but I hope I can stay 25% in the private leaderboard...



Johannes Amt... • (89th in this Competition) • 3 years ago • Options



X wrote

A 9-layer deep neural net trained in 2 days (shame I entered the competition too late) on raw RGB data using simple square loss gave me the current rank. I am at 14th now and I hope I can stay top 20 eventually...

The interesting thing is that, although there is a complicated 'decision tree' structure, ignoring it still gets something reasonably good.

Does that mean you trained one single network using regression?

Edit: Also, what were your parameters for each layer?

Also, in the same vein as X, do you know if your technique was performing particularly well on a few of the questions and then approximating conditional means for the others or was it learning uniformly well on all the questions?



Kyle Willett • 3 years ago • Options

^ 0 v

I'll second that - while most of the highest-ranking solutions used convnets, we're very interested in analyzing other techniques and looking at their strengths and weaknesses. Posting detailed code and descriptions would be extremely valuable for the science and Kaggle teams, if you're willing to do so.



Isaac Flath • (182nd in this Competition) • 3 years ago • Options

^ 0 v

We began by taking Coursera's introductory course Machine Learning. (Three of us earned certificates).

We started off using the OpenCV library to run regression techniques and neural networks. Our initial solution was a very simple one layer neural net. Instead of feeding every pixel into the net we did a Principal Component Analysis (PCA), calculating 30 eigenvectors to serve as the input features. We split the training set into training(60%), validation (20%), and test (20%) sets. We trained neural nets from 5 - 60 features and found that 25 features tended to give the best results. This one layer network with 25 hidden features gave us our lowest RMSE value.

After that we tried several different ideas to improve the score. Using the classification trained neural net we increased the number of hidden layers and attempted the same analysis. This was unsuccessful. The best scores for 2 hidden layers was comparable to the average value for the 25 feature 1 layer network but was unable to match our best score. These were also considerably more time intensive to train. We think the higher layer networks were getting stuck in local optima.

The next method we tried was support vector machines using the scikit library. A very basic SVM taking as input the same PCA analysis was very easy to set up but did not improve our score. Our intent was to combine the predictions from the SVM and neural networks but when we compared the individual errors for the 37 requested values we discovered that the SVM had done worse on every question. Interestingly all predictions seemed to be worse by about the same amount.

We tried to improve performance by increasing the number of features per galaxy. The plan was to average 5x5 blocks of nearby pixel values to take advantage of the spatial arrangement of the pixels in the image. Unfortunately, it took about 4 days to do this on 500 images so the scheme didn't work. Since this is a performance issue one of the things we intend to do is try this part in C or C++ to see if this is a python performance issue or if this is just extremely resource intensive.

We are now looking into utilizing a couple of deep learning libraries (pylearn2 and torch7) so we can utilize

tutorial competitions.