MLND: Capstone Project

Ming-Cheng Yen June 22, 2018

I. Definition

Project Overview

As shoppers move online, it'd be a dream come true to have products in photos classified automatically. But, automatic product recognition is challenging because for the same product, a picture can be taken in different lighting, angles, backgrounds, and levels of occlusion. Meanwhile different fine-grained categories may look very similar, for example, ball chair vs egg chair for furniture, or dutch oven vs french oven for cookware. Many of today's general-purpose recognition machines simply can't perceive such subtle differences between photos, yet these differences could be important for shopping decisions.[1]

Problem Statement

This is a competition of automatic image classification from Kaggle, for this competition we have a dataset of furniture images, each image has one ground truth label, and our goal is classify the furniture correctly, even they are in similarly. For this problem, we will need to build a CNN model to do image classification.

Metrics

For this competition, each image has one ground truth label. An algorithm to be evaluated will produce 1 label per image. And every class is equally important, hence we only care about how many images are classified correctly, if the predicted label is the same as the groundtruth label, then the error for that image is 0, otherwise, it is 1.

The final score is the error averaged across all images, formula sees as below.

$\text{let } F \text{ is the sum of which image is classified wrongly, let } P \text{ is the number of images}$

$Score = \frac{F}{P}$

But if we more care about the correct classification of some specific classes, we can choose something like the F1 score, maybe we can define formula something like as below.

$\text{let } T \text{ is the sum of which image is classified correctly.}$

$\text{let } FN \text{ is the sum of image predicted wrongly and the image actually belongs to some specific classes.}$

$\text{let } FP \text{ is the sum of which image is predicted to some specific classes wrongly .}$

$F1 = \frac{2T}{2T+FN+FP}$

II. Analysis

Data Exploration

Datasets and Inputs

Overview

train.json: training data with image urls and labels
validation.json: validation data with the same format as train.json
test.json: images of which the participants need to generate predictions. Only image URLs are provided.
sample_submission_randomlabel.csv: example submission file with random predictions to illustrate the submission file format

Training Data

The training dataset includes images from 128 furniture and home goods classes with one ground truth label for each image. It includes a total of 194,828 images for training and 6,400 images for validation and 12,800 images for testing. Train and validation sets have the same format as shown below:

```
{
"images" : [image], "annotations" : [annotation], }
image{
"image_id" : int,
"url": [string]
}
annotation{
"image_id" : int,
"label_id" : int
}
```

Testing data and submissions

The testing data only has images as shown below: {

```
"images" : [image],
}
image { "image_id" : int,

"url" : [string], }
```

JSON to Data Frame

Train Data Shape:(194828, 3)

First 5 records of training data

	image_id	label_id	url
0	1	5	https://img13.360buyimg.com/imgzone/jfs/t2857/...
1	2	5	http://www.tengdakeli.cn/350/timg01/uploaded/i...
2	3	5	https://img13.360buyimg.com/imgzone/jfs/t8899/...
3	4	5	http://img4.tbcdn.cn/tfscom/i1/2855447419/TB2S...
4	5	5	http://a.vpimg4.com/upload/merchandise/287883/...

Validation Data Shape:(6400, 3)

First 5 records of validation data

	image_id	label_id	url
0	1	38	http://www.ghs.net/public/images/fb/3d/51/3beb...
1	2	63	https://img.alicdn.com/imgextra/TB2chFei9YH8KJ...
2	3	33	http://static-news.17house.com/web/news/201602..
3	4	126	http://img000.hc360.cn/g6/M07/CB/88/wKhQsFNNVJ...
4	5	18	https://img.alicdn.com/imgextra/T1sLtpFH8aXXXX...

Test Data Shape:(12800, 2)

First 5 records of test data

	image_id	url
0	1	https://img13.360buyimg.com/imgzone/jfs/t13174...
1	2	http://img35.ddimg.cn/79/22/1258168705-1_u.jpg
2	3	https://img.alicdn.com/imgextra/TB19HtjKXXXXXc..
3	4	https://img13.360buyimg.com/imgzone/jfs/t16498...
4	5	http://img4.99114.com/group1/M00/7D/C5/wKgGTFf...

Checking Data

Missing Data in Taining Data set

	Total	Percent
url	0	0
label_id	0	0
iamge_id	0	0

Missing Data in Validation Data set

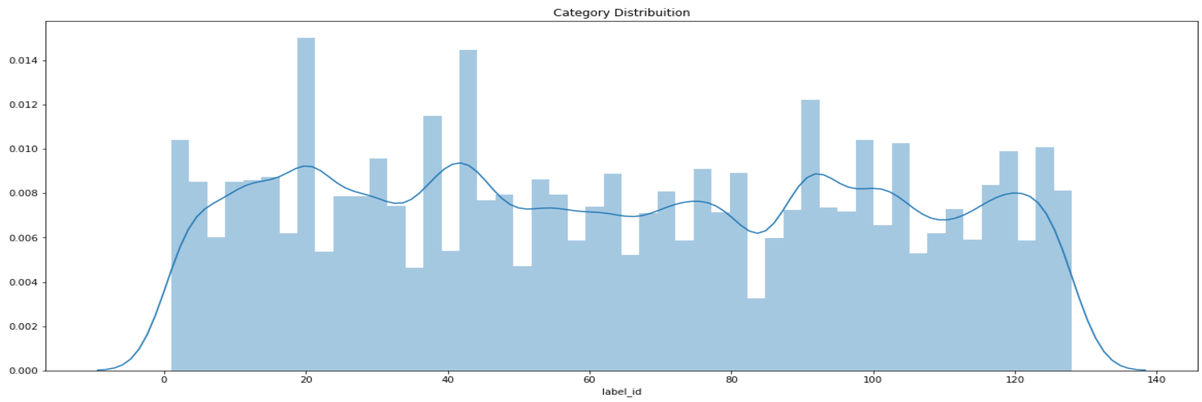
	Total	Percent
url	0	0
label_id	0	0
iamge_id	0	0

Missing Data in Test Data set

	Total	Percent
url	0	0
label_id	0	0

Distribution

Distribution of Training Data



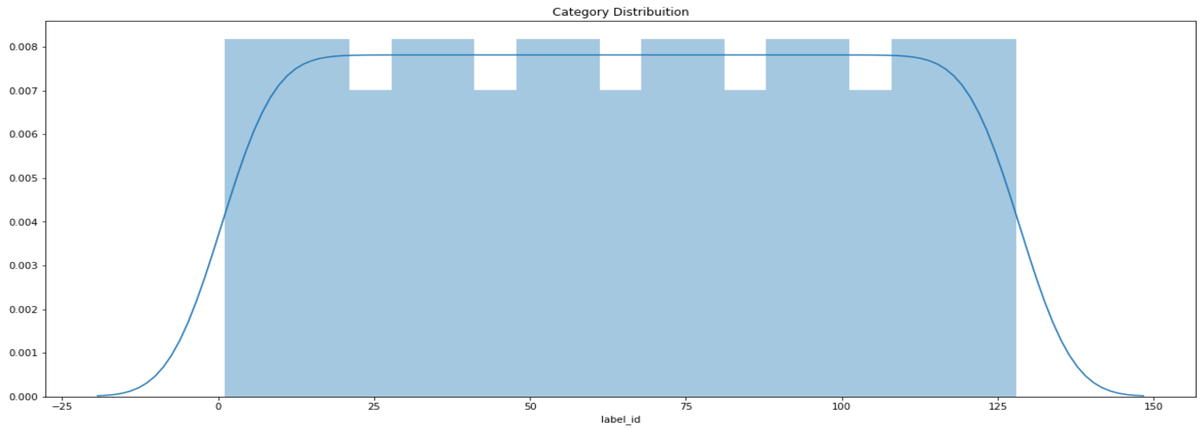
Most frequent of class in training data

	0	1	2	3	4	5	6	7	8	9
label_id	20	42	92	12	125	21	122	3	89	93
count	3996	3973	2666	2609	2598	2577	2462	2368	2353	2350

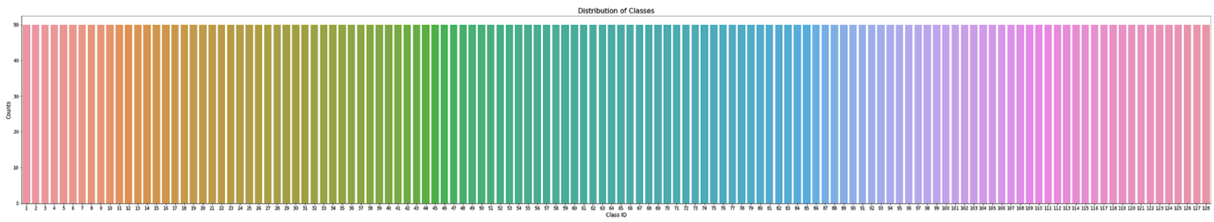
Least frequent of class in training data

	0	1	2	3	4	5	6	7	8	9
label_id	89	66	124	121	9	25	77	85	41	74
count	332	342	415	442	477	527	543	621	625	629

Distribution of validation data



Detail counts of validation data



Probability calibration

We can see as above about the data distribution, in training data some classes have around 4000 examples but others only have around 500 examples. In validation data, for each class have around 50 images.

At first I have tried undersampling, but I guess because of the gap between majority class and minority class is huge, it seems not really helpful to me.

After the competition the winner releases his idea calls probability calibration, he assumes the test data has same amounts of images for each class. And he tries to use Bayesian perspective to solve this problem, shown as below [4]:

Let's start with 2 class problem. From Bayesian perspective our final predicted probability from unbalanced train dataset could be seen as:

$$let\ P(y0|X)\ is\ the\ predicted\ probability\ for\ class\ y0, P(y1|X)\ is\ the\ predicted\ probability\ for\ class\ y1$$
$$let\ Pr(y0)\ is\ the\ prior\ probability\ for\ class\ y0, let\ Pr(y1)\ is\ the\ prior\ probability\ for\ class\ y1$$
$$let\ L(X|y0)\ is\ the\ likelihood\ of\ some\ data\ for\ y0, let\ L(X|y1)\ is\ the\ likelihood\ of\ some\ data\ for\ y1$$
$$(0) - P(y0|X) \propto Pr(y0)L(X|y0),\ \ P(y1|X) \propto Pr(y1)L(X|y1)$$

In words, our predicted probability is multiplication of some prior probability and some function from X.

For different distribution, likelihood should be the same, but because prior is different, we get different probability:

$$let\ P(y0|X)'\ is\ corrected\ probability\ for\ class\ y0,\ \ let\ P(y1|X)'\ is\ corrected\ probability\ for\ class\ y1$$
$$(1)\ -P(y0|X)' \propto Pr(y0)' L(X|y0),\ \ P(y1|X)' \propto Pr(y1)' L(X|y1)$$

And we know desire priors (1/2 for balanced), and because likelihood is the same we can get it from (0):

$$(2) - L(X|y0) \propto \frac{P(y0|X)}{Pr(y0)},\ \ L(X|y1) \propto \frac{P(y1|X)}{Pr(y1)}$$

and insert likelihood in (1) to get corrected probability:

$$(3) - P(y0|X)' \propto Pr(y0)' \frac{P(y0|X)}{Pr(y0)},\ \ P(y0|X)' \propto Pr(y1)' \frac{P(y1|X)}{Pr(y1)}$$

and it's almost our final formula, we just need to normalize probability so they sum to 1.

Changing this formula to multiclass is quite easy. For every class we just represent other 127 classes as one class.
Finally we'll use:

$$P(y1|X) = 1 - P(y0|X)$$
$$Pr(y1) = 1 - Pr(y0)$$
$$Pr(y1)' = 1 - Pr(y0)'$$
$$Pr(y0)' = \frac{1}{128}$$

Algorithms and Techniques

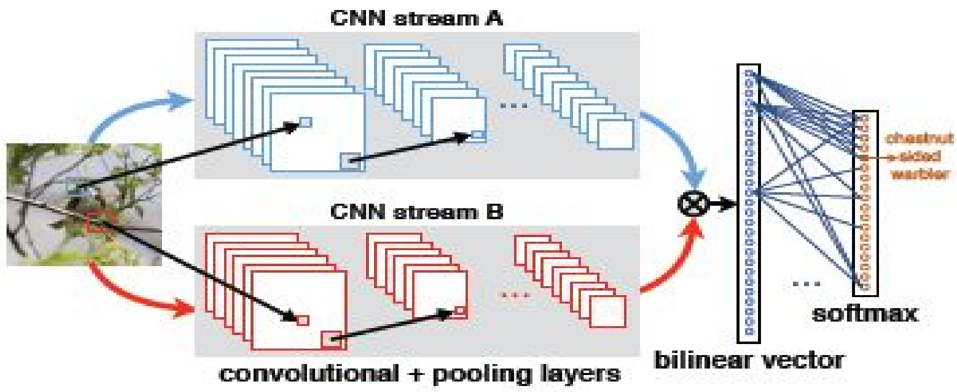
Bilinear CNN namely BCNN for short

Since our Images have no object bounding, we need to build an end to end model, and the key of a good solution is fine-grained image classification, hence I plan to build a bilinear CNN model, which it is determined to be a good solution of the fine-grained visual classification.

What is BCNN?

BCNN is a recignition architecture that consists of two feature extrators whose outputs are multipled using outer product at each location of the image and pooled to obtaine an image descriptor(as cited in http://vis-www.cs.umass.edu/bcnn/docs/bcnn_iccv15.pdf).

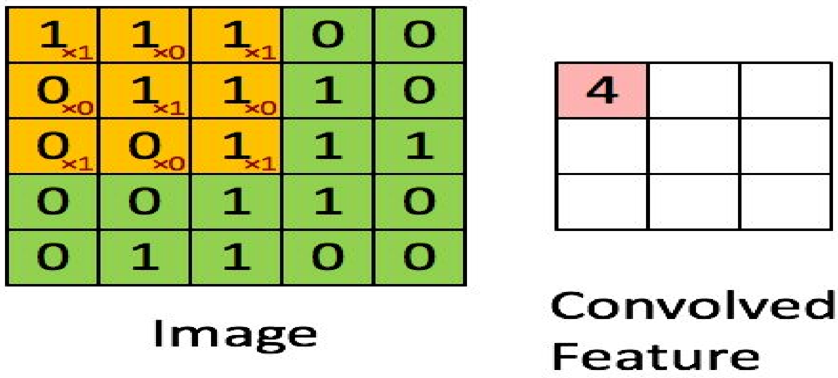
So basically we need to use two feature extractors, namely A and B, and use the outer product of A and B as the input of FC-layer, the whole architecture as below.



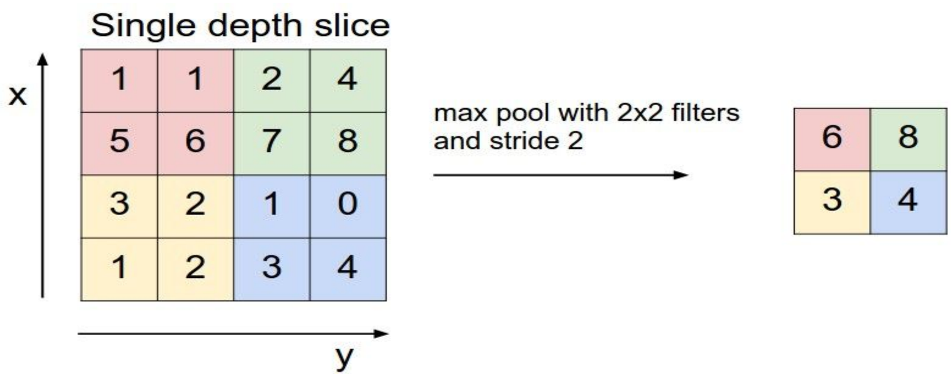
Main composition of CNN

A simple CNN architecture it includes three main types of layers, as below [5]:

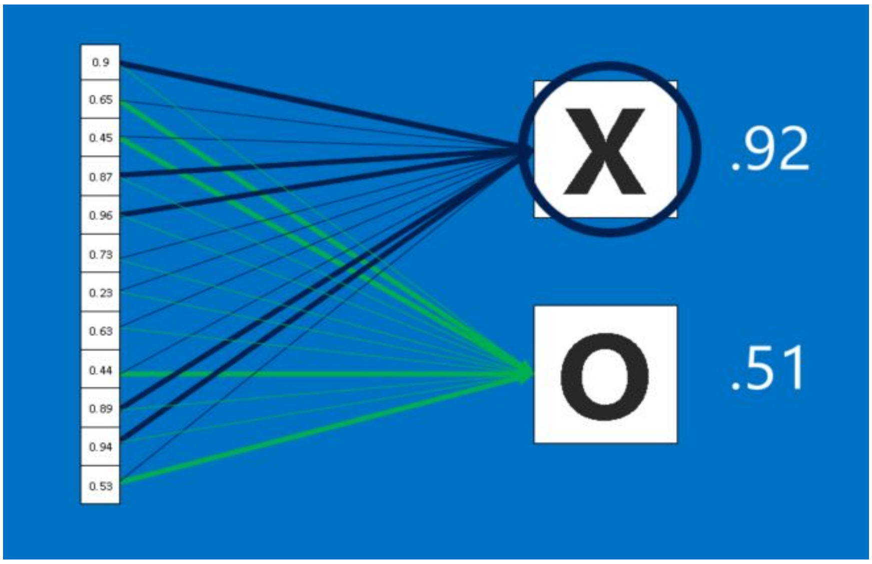
- Convolutional Layer: compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. shown as below:



- Pooling Layer: perform a downsampling operation along the spatial dimensions (width, height), max pooling layer shown as below:



- Fully-Connected Layer :Compute the class scores, if we have N classes, it resulting in volume of size [1x1xN], where each of the N numbers corresponds to a class score, if we have N=2 it shown as below:



Why CNN working good with image data?

To understand why CNN is good working on images, we have to understand what the difference between CNN and DNN.

Imagine that we have an image of an apple, we human being can identify this is an apple no matter it appears at which location of the image, it is because the human can select to focus on the local feature by our experience.

For good to simulate that above, CNN used Convolutional Layer to help to capture local feature, and used Pooling Layer ensures to some extent of invariance, which that is DNN can't achieve since DNN only use FC layer as the hidden layer, that makes DNN always focuses on the whole image.

Benchmark

According Lin and others work[2], their bilinear CNN models include classification of birds and aircrafts and cars, they got over 80% of accuracy roughly, compare with the condition of above, this competition we have 128 classes and 194,828 images for training, for the model that I plan to build, 80% of accuracy it could be reasonable.

III. Methodology

Data Preprocessing

Since I have kinds difference size of images, I have to resize all of the images to the same size as input for one model, hence I want to know which size should I use?

	width size	height size
unique	2054	2091
top	800	800
freq	98467	96811
mean size	761.169280	723.012783

It seems to me use the size of the image around 761x723 would be a good idea, but consider the computing costs, first I would try to resize to 96x96, and then use another size128x128 to determine if the result of bigger size is better.

Step 1.convert training images and validation images to .h5 file with RGB channel, use util.create_train_val_h5_file.

Step 2.convert test images to .h5 file with RGB channel, use util.create_train_val_h5_file.

I would have the files as below:

1.size of 96x96:

- training.h5
- validation.h5
- test.h5

2.size of 128x128:

- training.h5
- validation.h5
- test.h5

Implementation

In this project, I used pretrained model of VGG16. For BCNN it requires tow feature extractors, here I let two feature extractors the same, It can save the costs of training because the weights of two feature extractors are same during the training, hence I can just take the outer product of one feature extractor as the input of FC layer, instead of training two feature extractors.

Here are two model architecture and utils as below:

- Model A: load weights from VGG16 and only last FC layer trainable.
- Model B: load weights from Model A and the entire model trainable.
- Random flip: For each batch images random flip them right to left.

Furthermore to more saving costs of training, I used two-step training as below:

- Step 1: Training Model A with Momentum optimizer (learning rate=0.9,momentum=0.9).
- Step 2: Training Model B with Momentum optimizer (learning rate=0.001,momentum=0.9).

Refinement

In this project, I think Image size is the important parameter, and I try to deal with it.

At first, I resize images to 96x96 and after 62 epochs training, model got overfitting, and I got accuracy around 76%.

And then I resize images to 128x128 and after 30 epochs training, model got overfitting, and I got accuracy around 78%.

IV. Results

Model Evaluation and Validation

The model gets the private score on Kaggle around 0.20, in other words, the accuracy around 80% base on 30% of test data, and this score it's almost the same with local validation score.

The model gets the public score on Kaggle around 0.22, in other words, the accuracy around 78% base on the other 70% of test data.

In summary I think this model is good to generalize with this problem.

All the accuracy as above are evaluated by upload predict result to Kaggle, for the detail sees as below:

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions Late Submission

12 submissions for bobyan Sort by Private Score

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
pred.csv 19 days ago by bobyan full layer 128X128--30 epochs	0.20919	0.22306	☐
pred.csv 23 days ago by bobyan full layer 96X96--62 epochs	0.22815	0.23993	☐

Justification

The final results I got the accuracy is around 78% with image size 128x128.

Compare to the accuracy of benchmark is over 80%, it seems not good enough, but consider the result from image size of 96x96 accuracy is 76%, hence I expected if increase the image size, the accuracy would be improve to over 80%.

benchmark model	model with 96x96 image size	model with 128x128 image size
accuracy over 80%	accuracy 76%	accuracy 78%

V. Conclusion

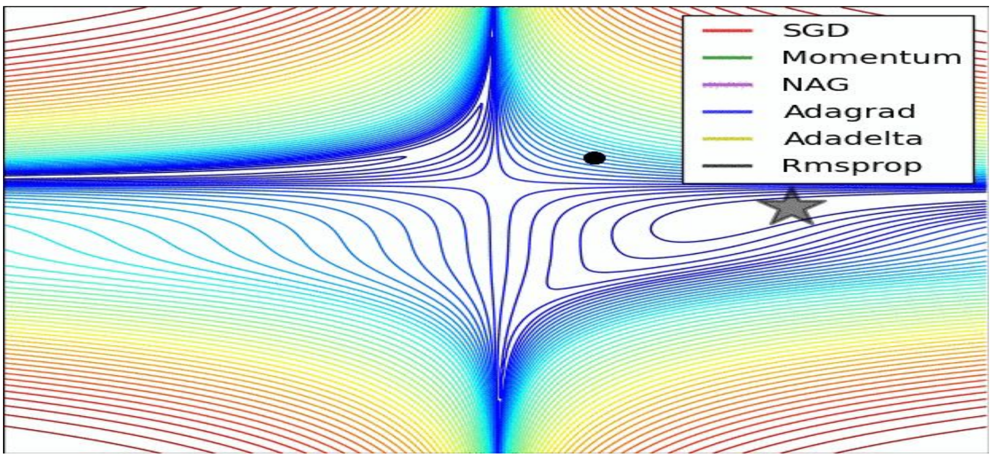
Free-Form Visualization

Optimizer:

In this project consider the amounts of training data and compare to our target categories, I think the input data is not sparse, we should not use adaptive learning-rate methods, such as RMSprop or Adam[3].

Hence I use momentum optimizer to accelerate training, and if we want to more approach the optimal, maybe we could try SGD optimizer but it really takes time than the other optimizer.

See SGD optimization on loss surface contours as belowe[3]:



Reflection

This project is a competition from Kaggle, and all the data we need to download from URLs that Kaggle provided, hence the first we need to write program to download all these images automatically.

After finish download all these images, we have kinds of different size of image, hence the first we need to resize all the image, then convert these images to .h5 file, and implement the BCNN architecture specifically is the architecture of BCNN[D,D] [2], the interesting thing is I can just take the outer product of one feature extractor as the input of FC layer, because the weights of two feature extractors are same during the training.

The most difficult thing I think is saving the costs of computing, for this competition I already spend over 200 dollars on the GCP.

I think at beginning I should use very small portion of images to figure out which size of image is good and affordable to computing.

Improvement

Here are some idea I think that would improve the result:

- Increase Image size: I got the image mean size around 800x800, and I tried two different image size from 96x96 to 128x128 the accuracy is increase, hence I expect increase image size is helpful.
- Shuffling the training data after every epoch: I have tried to do this but unfortunately memory is not affordable.
- Use SGD optimizer: Generally if training data is big I think use SGD optimizer is most likely to approach optimal.

VI. References

[1] https://www.kaggle.com/c/imaterialist-challenge-furniture-2018

[2] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear CNN models for fine-grained visual recognition. In Proceedings of IEEE International Conference on Computer Vision, pages 1449–1457, Sandiago, Chile, Dec. 2015.

[3] http://ruder.io/optimizing-gradient-descent/

[4] https://www.kaggle.com/dowakin/probability-calibration-0-005-to-lb

[5] http://cs231n.github.io/convolutional-networks/