

Using Convolutional Neural Network to the Recognition of Handwritten Mathematical Equations

COSI 101A Introduction to Artificial Intelligence

Term Project Report

Fan Di

05/05/2017

CONTENTS

1. INTRODUCTION	3
1.1 INTRO TO RECOGNITION OF HANDWRITTEN EQUATIONS	3
1.2 BACKGROUND LITERATURE	3
1.3 OUR APPROACH	3
2. DATA SETS PREPARATION AND FORMATTING	5
2.1 SOURCE OF DATA SETS	5
2.2 IMAGES PREPARATION	5
2.3 FORMATTING DATA	6
2.4 TEST SET	7
3. SYMBOLS SEGMENTATION	8
3.1 SEGMENTATION METHODS OVERVIEW	8
3.2 MINIMUM SPANNING TREE APPROACH	8
3.3 PARSING APPROACH	9
3.4 BLIND PARSING	10
3.5 EDUCATED PARSING	11
4. SYMBOLS RECOGNITION	13
4.1 SETTING UP VARIABLES	13
4.2 TEST OF DIFFERENT ARCHITECTURES	14
4.3 FINAL ARCHITECTURE	16
5. RECONSTRUCT EQUATIONS	18
5.1 UNDERSTAND INDIVIDUAL SYMBOLS IN CONTEXT	18
5.2 RELATIONSHIPS BETWEEN SYMBOLS IN CONTEXT	19
5.3 SIMPLE SYMBOLS	20
6. RESULT	21
7. DEPENDENCIES	22

1. Introduction

1.1 Intro to Recognition of Handwritten Equations

Recognition of mathematical symbols and expressions is a tough problem in the fields of computer science. Part of the reason is enormous amounts of mathematical symbols, combination rules, and different hand writing styles. Such complex structures and varieties present enormous challenges in how to use programming and artificial intelligence to segment, recognize individual symbols, and then reconstruct mathematical expressions.

1.2 Background Literature

The Digit Recognizer Competition on kaggle is a competition on recognizing handwritten mathematical symbols that has attracted participants, and generated a lot of solutions to recognize mathematical symbols. Most of the submissions used tensorflow to construct convolutional neural networks.

We also read some papers on the recognition of handwritten mathematical symbols. Some early papers used more traditional ways to segment and recognize mathematical symbols. For examples, some computed centroid and used clustering to segment data, some used geometric features for recognitions. These papers are generously considered as outdated by us.

Due to the release and application of tensorflow, papers after 2015 are more relevant. Most papers suggest methods of partitioning an expression into individual mathematical symbols, then use trained convolutional neural network model to recognize individual symbols, and finally use algorithm to reconstruct equations.

1.3 Our Approach

Our approach can be divided into 4 steps:

1. Data set preparation and formatting;
2. Symbols segmentation;
3. Symbols recognition;

4. Reconstruct equation.

We will discuss our approach by steps in following chapters.

2. Data Sets Preparation and Formatting

2.1 Source of data sets

Our main training set comprises of three sources:

1. Course provided training set
These training images are scanned handwritten symbols and equations from our students.
There are 3,800 images from this training set.
Problems with this training set is that it lacks special symbols like division mark, and lowercase letter i.
2. Online sources
<https://www.kaggle.com/xainano/handwrittenmathsymbols>
This resource provides over 30,000 training images. However, the overall quality is low.
3. Standard computer fonts training set
In the fear of the diverse variation of different hand writing styles, we also created a set of standard computer fonts for training, but did not eventually use them.

2.2 Images preparation

Resizing:

Images from course provided training set are of irregular sizes. Images from online source are bigger than 28*28px. In order to convert these raw images into standard sizes 28*28px, we need to resize the raw images.

The tool we used is a Python library called Pillow.

The basic steps are:

1. Measure the two dimensions of the input image;
2. Resize the longer side into 28px, and calculate the resize ratio;
3. Resize the shorter side using the calculated ratio;
4. Paste the resized image on black canvas;
5. Return the resized image.

Dilation:

Lines in images from the online source are too thin, after resizing, some lines even become invisible.

We used OpenCV to dilate thin lines in raw images. The idea behind it consists of convoluting an image A with some kernel B. The kernel B has a defined anchor point, usually being the center of the kernel. As the kernel B scans through the image A, OpenCV computes the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value. During the process, the maximizing operation causes bright regions within an image to “grow”, therefore thickens thin lines.



2.3 Formatting data

We use Pickle data structure to store our train and test images' data. The format is as below:

Set of train symbol (40):

Symbol = ['dots', 'tan', ']', '(', '+', '-', 'sqrt', '1', '0', '3', '2', '4', '6', 'mul', 'pi', '=', 'sin', 'pm', 'A', 'frac', 'cos', 'delta', 'a', 'c', 'b', 'bar', 'd', 'f', 'i', 'h', 'k', 'm', 'o', 'n', 'p', 's', 't', 'y', 'x', 'div']

Train and Predict Pickle File Format:

```
{ "images": {  
    "filename0" : array of image data [784]  
    "filename1" : array of image data [784]  
    .....  
}  
"labels": {
```

```

    "filename0" : label (list of size 40) [0,0,1,0,0,...,0]
    "filename1" : label (list of size 40) [1,0,0,0,0,...,0]
    .....
}
}

```

Created training pickle file can be found here:

<https://drive.google.com/open?id=0B0NomvTvnSbMXdxWDIKcDZIZ3M>

2.4 Test set

The course provided 389 images for 35 equations, we reserved 70 images as our test set (2 images per equation), and used the rest 319 images as our training set.

Also, in order to improve generalization of our model, we added 700 ~ training images for each individual symbol from online source in our training data set.

3. Symbols Segmentation

3.1 Segmentation methods overview

It is relatively easy segment individual symbols from an image of equation. We used a library called OpenCV to do this work. The basic idea behind the code is a depth-first-search algorithm on a 2D array – when the program reads a valid pixel (has a color different than canvas), it continues to read all neighboring pixels as an entity, and record the entity's upper left corner and bottom right's coordinates, and finally return these two coordinates as a bounding box.

The challenging part for segmentation lies in how to combine individual symbols. For example, a division mark comprises of two dots and one horizontal line.

There are some papers discussing equation segmentations and symbol combination, and we are particularly interested in two approaches:

1. Building a minimum-spanning tree, and then do combination based on distances between bounding boxes;
2. Parse returned bounding box list from left to right, and do combinations as necessary.

We compared and tried both method, and decided to use the parsing methods, reasons are as bellow.

3.2 Minimum spanning tree approach

Analysis of minimum-spanning tree:

Advantages:

1. it is more universal – the code of building a minimum spanning tree can be applied on any equation images;
2. This approach provides a more general analysis of an equation image, rather than parsing individual bounding boxes in a fixed sequence.

Disadvantages:

1. The parameter used to construct the minimum-spanning tree is flawed.

The centroid is a good indicator to represent distances between two bounding boxes, but in real life, few letters or operators are constructed in this way. More often, we use the distances between two bounding boxes' edges rather than their centroids. For example, the distance between the "dot" and "vertical line" of a lowercase "i" should be measured by the distances between the two bounding boxes, but not the distances between their centroids.

Of course, we can build several different minimum-spanning trees based on different types of distances to overcome this problem, but then in some sense we are complicating the combination.

2. It is not following human reading / writing convention.
Minimum-spanning tree treats all bounding boxes equally, purely based on their distances, not matter of their sequence. This is against the way how mathematical equations are written/read. Normally, we read from left to right, and when we read continuously three "dot", we know it is an ellipsis.

3.3 Parsing approach

Analysis of parsing in sequence:

Advantages:

1. it follows human writing convention.
We parse the bounding boxes in sorted order (based on their x coordinate), and combines when necessary.
2. It is more flexible to choose different types of distances as parameters to calculate combination.
This approach is not bounded to the distance between centroids.

Disadvantages:

1. Need to handle specific cases.
For example, the criterial to combine individual bounding boxes into "i", division mark, and ellipsis are all different.

The way to overcome it is to write different methods to handle different cases. And actually this becomes an advantage – since each case is handled in a specific way, thus the expected accuracy is higher than using a general minimum-spanning tree.

2. Number of different cases to handle.

This could be a disadvantage for the parsing approach, however, there are only limited cases need to be handled when combining symbols – combine to division mark, combine to letter i, combine to equation mark, combine to ellipsis, and combine to plus-minus.

Therefore, after comparing minimum-spanning tree approach and parsing approach, we decided to go with parsing approach.

3.4 Blind parsing

Our initial idea is to use parsing to combine necessary symbols to prepare training set. Thus the approach is more like a “blind parsing” – at the stage before individual symbols are recognized, we can only use bounding boxes to guess the content of the symbol, and then make combination decisions.

The two steps are:

1. Guess individual symbols' contents based on bounding boxes;
2. Make necessary combinations.

There are several symbols must be detected before combination, they are: dot, vertical line, horizontal line, square

Possible combination results are:

division mark, letter i, equation mark, fraction, and ellipsis.

There are several methods to executes these evaluations, the basic rule is to use the dimension and relative position of bounding boxes.

For example:

For single dot, we can use its length, and area to make the evaluation:

$area = (y_h - y) * (x_w - x)$

return $area < 200$ and $0.5 < (x_w - x)/(y_h - y) < 2$ and $abs(x_w - x) < 20$ and $abs(y_h - y) < 20$

For vertical and horizontal lines, the ratio of their width / length can be used as criterial:

return (xw - x) / (yh - y) > 2

return (yh - y) / (xw - x) > 2

To combine symbols into division mark, we must have 2 dots, and 1 horizontal line, and their positions should meet criterial:

cenY1 = y1 + (yh1 - y1) / 2

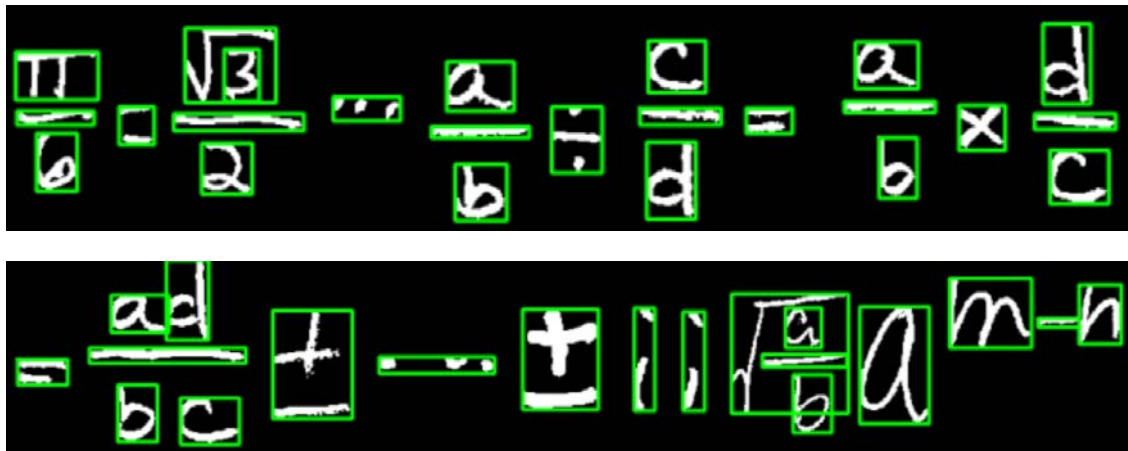
cenY2 = y2 + (yh2 - y2) / 2

return (isHorizontalBar(boundingBox) and isDot(boundingBox1) and

isDot(boundingBox2) and x < x1 < x2 < xw and max(y1, y2) > y and min(y1, y2) < y and max(y1, y2) - min(y1, y2) < 1.2 * abs(xw - x))

There are more evaluation methods in our submission, and the criterial are diverse. We spent quite a lot of time on tuning these evaluation methods. The conclusion we got is to use relative position relations as criterial, and avoid absolute relations and arbitrary numbers as much as possible.

The final result of our segmentation is satisfactory:



We ran all of our 70 testing images, and the accuracy rate of segmentation and symbol combination in our approach is 93%.

3.5 Educated parsing

The blind parsing seems working well on segmenting and combing individual symbols, as mentioned above, the accuracy rate reached 93%. Considering the diverse writing styles in the training set, the rate is satisfactory.

However, later during our test phase, we sadly found that even if complex symbol like division mark is successfully detected and combined, our model has difficulty recognizing it. The division mark is more likely to be recognized as a plus mark, the accuracy rate of recognizing a division mark is merely 28%.

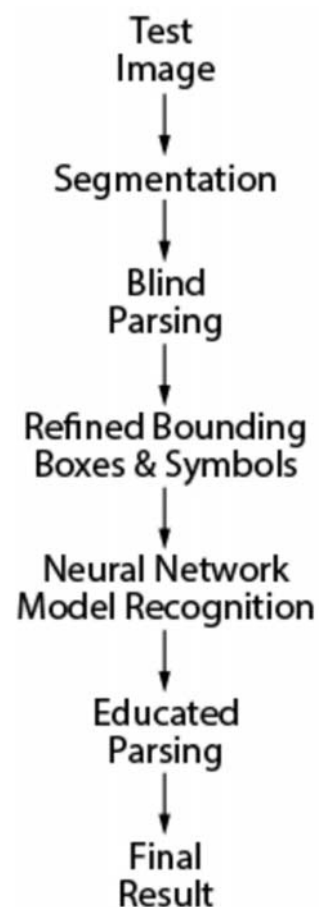
So we decided to try educated parsing after blind parsing, which is to test the model with un-combined raw symbols, and the combinations happen after recognition.

In this approach, we do not need to use bounding boxes to guess content, but rather rely on the result of recognition. The algorithms remain the same. The risk of this approach is to solely rely on the model to recognize individual simple symbols.

However during our practice, we found that the educated parsing sometimes will fail on simple certain cases.

With the enlargement of our training set, our final solution to symbols segmentation is as shown in the diagram on the right:

1. Use blind parsing to combine symbols as much as possible, but in a conservative way;
2. Feed the model with the combined symbols;
3. Run a second parsing (educated) after recognition, collect and combine missed elements from blind parsing.



4. Symbols Recognition

We used the architecture from our HW4 as the training model for term project. The model architecture is tested and tuned using the HW4 training set. Procedures are as below:

4.1 Setting up variables

```
28 # variables
29 steps = 20000
30 batchSize = 100
31 convolution = (1, 1)
32 kennelSize = (2, 2)
33 maxPoll = (2, 2)
34 v #Optimizer = AdamOptimizer
35 #https://www.tensorflow.org
36 learningRate = 0.001
37 layer1Feature = 16
38 layer1Patch = 5, 5
39 layer2Feature = 32
40 layer2Patch = 5, 5
41 hiddenLayer = 100 # the m
42 dropoffRate = 0.5 # reduc
43 layer3Feature = 64
44 layer3Patch = 5, 5
```

These are the variables which set up the architecture of our model, and control the training of our model, and will eventually determine the specs of our model.

I did not list all of the variables which influence our model. For a more complete list of affecting factors and explanations, read our course slides and visit tensorflow official guide

4.2 Test of different architectures

The complete specs and test outcomes of models I set up and tested can be found in the Excel File [Model specs & logs.xlsx](#)

1. First, I tried the original setting of the model guided by tensorflow official tutorial. Its specs are:

	modelDefault.ckpt	
Steps	3000	
BatchSize	50	
Convlution	1,1	
KennelSize	2,2	
MaxPool	2,2	
Optimizer	AdamOptimizer	
Learning Rate	0.0001	
Layer 1 feature	32	
Layer 1 patch	5	
Layer 2 feature	64	
Layer 2 patch	5	
Hidden	1024	
Dropoffrate	0.5	
Layer 3 feature	NA	
Layer 3 patch	NA	

Although after 3000 steps' training, the accuracy rate of the model reached 98% on the online test sample, its accuracy rate on the test sample created by our classmates is only 49%.

2. Second, I tried GradientDescentOptimizer which was used in the beginner's model on the official tutorial.

The outcome is bad, with accuracy on our own test sample of 6%.

3. The third major model I tested is to micro tune BatchSize, Learning Rate, and the hidden layer number. Specs are:

	modelHiddenLayer.ckpt		
Steps	3000		
BatchSize	100		
Convolution	1,1		
KennelSize	2,2		
MaxPool	2,2		
Optimizer	AdamOptimizer		
Learning Rate	0.0010		
Layer 1 feature	32		
Layer 1 patch	5		
Layer 2 feature	64		
Layer 2 patch	5		
Hidden	100		
Dropoffrate	0.5		
Layer 3 features			
Layer 3 patch			

Although the change is not major, its performance is much better than the original one.

After 3000 steps' training, its accuracy on the online test sample reaches 99.21%, and its accuracy on our own created test sample reaches **72.82%**.

The result is satisfactory.

The major factor which distinguishes this model from the original one is its hidden layer numbers, which is reduced from 1024 to 100.

Reason behind it is that: our own created test sample is very different from the online training sample and test sample. Thus by reducing the hidden layer numbers, the model's ability to generalize is enhanced, thus perform much better on irregular hand writing digits by our students.

- Before reaching our final architecture of the model, I also tried many other architectures, which can be found in the excel file. (I also tried change patch sizes, max_pool, batchSize, etc, but did not include these in the excel file, because the outcome of these changes are not positive). One conclusion is that the training steps influence on the model's performance is not linear. The worse the architecture is, the bigger the training steps impact on the model is. In other words, if the model's architecture is already well designed, then the difference between

training the model for 3000 steps and 20000 steps is not as big as I expected.

4.3 Final architecture

After reaching the conclusion that:

1. our sample is different from online test sample
2. To improve the model's accuracy on our own test sample, we need to improve the model's ability to generalize

The goal of the rest of my work is much clearer, which is to improve the model's ability to generalize.

The approach I adopted based on the last model I mentioned above, is to make the model longer and skinnier, which is to –

1. Add a third convolutional layer
2. Do not further increase the features on all 3 convolutional layers.
I kept the features of the final layer to be maintained as 64
3. Keep the hidden layer number low as 100

A tricky part is to choose proper **max_Pool** values for 3 convolutional layers.

Because the size of our image are 28x28, so actually the choices are limited. The original model chooses **max_pool_2x2** for both layer 1 and layer 2, the final result image size becomes 7x7. Given the fact, we can only add a **max pool 1x1** for one of the three layers, then the problem becomes to test to apply the max pool 1x1 on which layer can we get the optimal result.

The best option is to apply the max_pool_1x1 on the third convolutional layer.

The final architecture and specs of our model is:

	model3Layer.ckpt	
Steps	3000	
BatchSize	100	
Convlution	1,1	
KennelSize	2,2	
MaxPool	2,2	
Optimizer	AdamOptimizer	
Learning Rate	0.0010	
Layer 1 feature	16	
Layer 1 patch	5	
Layer 2 feature	32	
Layer 2 patch	5	
Hidden	100	
Dropoffrate	0.5	
Layer 3 features	64	
Layer 3 patch	5	
	layer3_maxPool = 1,1	

After training for only 3000 steps, its test accuracy on the online test sample is which is probably better than most models I found online. Its accuracy on recognize our own hand writing digits reaches **89.74%**, consider it was only trained for 3000 steps, the outcome is satisfactory.

Feel free to find more training details in our submitted Model_tune_log.xlsx

5. Reconstruct Equations

The next step after symbol recognition is that construct equation based on the value and position of each segmented symbol from equation. In this application, we implemented this construct equation function in **toLatex** method.

The features used to analyze of each symbol for constructing corresponding equation are the predicted value and position information of symbol. In this project, we need deal with several patterns as below:

5.1 Understand individual symbols in context

As the classification rule treats several symbols as the same groups, the first step of construct equation is that determine the value of symbol. The examples of classification rule are as below:

mul, $x \rightarrow x$

frac, bar, $- \rightarrow -$

O $\rightarrow 0$

we convert the symbol to which it should be by analyzing the context. Here I raised the example of the method how we classify fraction and bar and rewrite the prediction result of “-”. From the graph below, we can learn that the context before and after “-” and the relative position of the context with “-” is the key feature to determine the value of “-”.

1. no other symbol over and under “-”, determine as “-” minus
2. only one symbol under “-” and no symbol over “-”, determine as “bar”
3. there exists at least one symbol over and under “-”, determine as “fraction”



(1) “-” minus

(2) bar

(3) fraction

Based on the determined result, we generate each symbol into the expression as Latex.

5.2 Relationships between symbols in context

Even though recognition of single symbol, construct an equation should analyze the relationship between symbols and to determine the expression of equation. There are four types of determining relative position relationship that we implemented in our application.

1. superscript

Some horizontal arrayed symbols need to determine the superscript in these two symbols. Firstly, we need to find the center of two symbols and if one symbol A's center is higher than symbol B's center plus half of A's height, they are not vertically same, we can determine A is superscript of B.

$A's\ center < B's\ center - A's\ height/2$

and $A's\ x > B's\ center's\ position\ in\ x\ axis$



2. subscript

Some horizontal arrayed symbols need to determine the subscript in these two symbols. The same as superscript, we find the center of two symbols and if one symbol A's center is lower than symbol B's center minus half of A's height, and they are not vertically same, we can determine A is subscript of B.

$A's\ center < B's\ center + A's\ height/2$

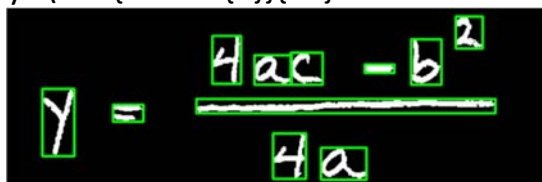
and $A's\ x > B's\ center's\ position\ in\ x\ axis$



3. fraction

The method to determine the parts around a fraction is to find the symbols over and under fraction.

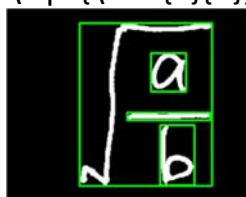
We analyze the x-value to find the symbols belongs in the range of a fraction and use the y-value to determine over or under fraction. At last we output the expression of each part using toLatex function recursively. The Latex expression of the image as below is shown as:
 $y = \frac{4ac - b^2}{4a}$



4. square

For dealing with square part, we take use of x-value and y-value to determine the list of symbols belong to one square, and process the list of symbols using toLatex method recursively to output completed equation expression. The output result of equation is as:

$\sqrt{\frac{a}{b}}$



5.3 Simple symbols

Besides cases discussed above, other symbols can be translated into Latex straightforwardly with no need to analyze relative position.

The final step is to output mathematical equations in Latex format according to name rule of Latex. For example:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

6. Result

After setting up our code, we started training and testing our model. We used another model to compare the performance of our final model.

The result are as below:

Final Model				
5k training steps	70 test set	389 test set	After debug	20k steps
Course training set	88.87%	88.13%		
Course + external	86.20%	78.85%	92.85%	94.65%

Comparison Model				
5k training steps	70 test set	389 test set		
Course training set	82.94%	81.24%		
Course + external	74.18%	67.76%		

The training logs of these two models are in our submitted:
Train_log.xlsx

A new finding is that after using online training set, the accuracy rate actually dropped. We examined the online training set, and found that some training images are incorrect and may be misleading for the models. However, due to enormous amount of images in the online training set, we were not able to do a thorough sorting.

7. Dependencies

package name	version
pickle	
numpy	1.12.1
opencv-python	3.2.0.7
PIL	1.1.7
Pillow	4.0.0
scipy	0.19.0
scikit-image	0.12.3
tensorflow	1.0.0