**Using Convolutional Neural Network to the Recognition of Handwritten Mathematical Equations**

COSI 101A Introduction to Artificial Intelligence

Term Project Report

Si Chen

[chensi@brandeis.edu](mailto:chensi@brandeis.edu)

Fan Di

[fdi@brandeis.edu](mailto:fdi@brandeis.edu)

Jiaming Xu

[jmxuzju@brandeis.edu](mailto:jmxuzju@brandeis.edu)

05/05/2017

# Introduction

## Intro to Recognition of Handwritten Equations

Recognition of mathematical symbols and expressions is a tough problem in the fields of artificial intelligence and computer science in general. Part of the reason is enormous amounts of mathematical symbols, combinations which constitute equations, and variety of different hand writing styles. Such complex structures and varieties present enormous challenges in how to use programming and artificial intelligence to segment and recognize individual components of a mathematical expression and then re-construct mathematical equations.

## 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 These papers are generously considered as outdated by us.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 for our project. Most papers suggest methods of partitioning an expression into individual mathematical symbols, and then use trained convolutional neural network model to recognize individual symbols, and then use algorithm to re-construct equations.

## Our Approach

Out approach can be divided into 4 steps:

1. Data set preparation and formatting;
2. Symbols segmentation;
3. Symbols recognition;
4. Re-construct equation.

We will discuss the steps of our approach by chapters.

# Data Sets Preparation and Formatting

## Source of data sets

Our main training set comprises of two 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.

1. Online sources

<https://www.kaggle.com/xainano/handwrittenmathsymbols>

This resource provides over 30,000 training images.

1. 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 use it in later phase.

## Images preparation

**Resizing:**

Images from course provided training set are of irregular sizes. Images from online source are bigger than 28x28. In order to convert these raw images into trainable images, we need to prepare and 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 some 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 compute 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 the lines.

## Formatting data

### JIAMING

## 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.

This initial attempt shows the accuracy rate of segmentation and combination is 93%, the accuracy rate of recognition based on segmentation is 87.7%.

# Symbols Segmentation

## 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 calculate the entity’s upper left corner and bottom right’s coordinates, and finally return these two coordinates as a bounding box.

The challenge 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 interested in two approaches proposed in these papers:

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.

Analysis about **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 on 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, not all letters or operators work in this way. In a lot of occasions, 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 minimum-spanning tree based on different types of distances to overcome this problem, but then in some sense this simple problem is magnified.

1. It is against 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.

## 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 on 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, not all letters or operators work in this way. In a lot of occasions, 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 minimum-spanning tree based on different types of distances to overcome this problem, but then in some sense this simple problem is magnified.

1. It is against 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.

## 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.

1. It is more flexible to choose different types of distances as parameters to decide combination.

This approach is not bounded to only use distances between centroid.

Disadvantages:

1. Need to handle specific cases.

For example, the criterial to combine individual bounding boxes into “i", division mark, and ellipsis are 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 specific way, thus the expected accuracy is higher than using a minimum-spanning tree.

1. 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.

## 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 combine them if necessary.

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 box.

For example:

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

area = (yh - y) \* (xw - x)

return area < 200 and 0.5 < (xw - x)/(yh - y) < 2 and abs(xw - x) < 20 and abs(yh - 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 complex. 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 is satisfactory:



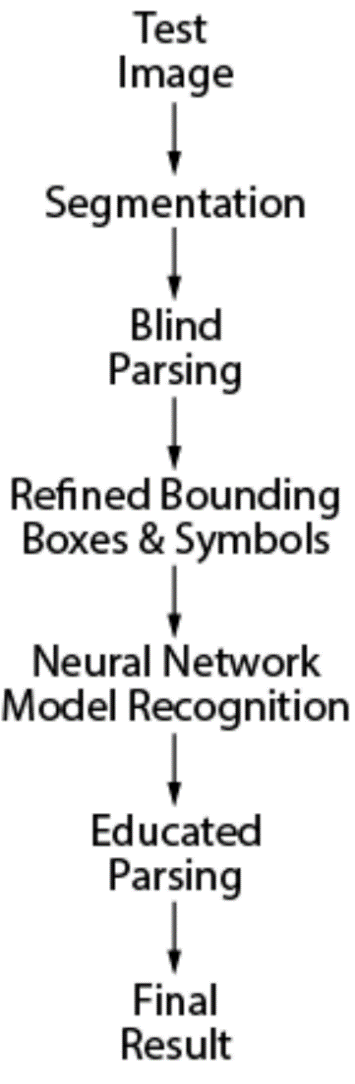


We ran all of our 70 testing images, and the accuracy rate of segmentation and symbol combination is 93%.

## 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 style in the training set, the result 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. 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 necessary, but in a conservative way;
2. Feed the model with the combined symbols;
3. Run a second parsing (educated) after recognition, and combine missed elements from blind parsing.

# Symbols Recognition

### JIAMING

### DI FAN

# Reconstruct Equations

### CHEN SI

# Statics