Southern Illinois University Carbondale

# Handwritten Equation Recognizer

ECE572 Final Project Report

Ye Wu and Zichang Zhang

# TABLE OF CONTENTS

1 Image preprocessIng ..... 1

2 Image/Symbol Segmentation ..... 3

3 Symbol Recognition ..... 3

3.1 Neural Network Building ..... 3

3.2 Training ..... 5

4 Result ..... 5

5 DepenDencies ..... 6

References ..... 7

## 1 IMAGE PREPROCESSING

For uniformity, all equation pictures downloaded online or written by ourselves are first trunked to the same width. We use the width of the entire equation for 1696. Then transfer the images to grayscale and set the background to 0 (black) and the other part as 255 (white). The above transformation makes our standard input. We use Matlab implements this part, any picture other than the above format can't be recognized by our program

For the standard input, we crop each ever symbol and uniform them to the same size. Here we use 32*32. Explicitly, for each symbol, we shrink the longer side to 32 then shrink the shorter side according to the original ratio. Thus, each symbol picture is less or equal than 32*32.

Optimally, for some special case, if the source picture has a too thin line, after resizing, it would be invisible. Then an extra dilation should be applied to the picture. Simply use the dilation() function in skimage package.

The training sets and testing set are annotated as paten

".*eq\d*_(.*)_(\d*)_(\d*)_(\d*)_(\d*)\.png", the first group of this paten ("(.*)") carriers the information of what's the accurate name, i.e., if the picture is "x" the "(.*)" is x if the. The training sets are built by math the image with the symbol using the shelf. The training information was stored in the shelve format as flowers:

```
[array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]), 'a']
```

The array is the resized image and the "a" is the symbol of the picture refers to, which consider the label of the training set. For simplicity, we only trained of 40 symbols including:

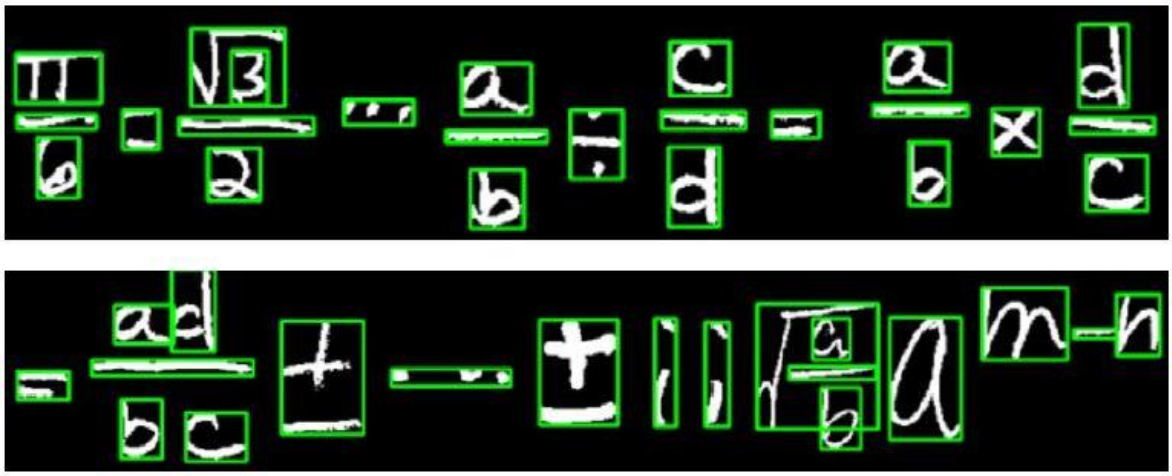
' [', ']', '(', ')',dots', 'tan', '+', '-', 'sqrt', '1', '0', '3', '2', '4', '6', 'mul', 'pi', '=', 'sin', 'pm', 'A','frac', 'cos', 'delta', 'a', 'c', 'b', 'bar', 'd', 'f', 'i', 'h', 'k', 'm', 'o', 'n', 'p', 's', 't', 'y', 'x', 'div'

## 2 IMAGE/SYMBOL SEGMENTATION

Our idea is to use parsing to combine important symbols to prepare a 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 four steps are:

1. divide the equation into different strokes using connected components algorithm
2. establish a minimum spanning tree of strokes
3. find the best partition of strokes
4. classify the image according to the recognized symbols



## 3 SYMBOL RECOGNITION

### 3.1 Neural Network Building

We build the neural network model based on the existing convolutional neural network CNN on digit recognizing code. We used its normal layer as our normal layer function and add noise function as our network base and pended to 6 layers as followers:

1. First layer of convolution: 3 X 3 window, input:1, output:32 (cause our picture size is 32*32 so the second layer we chose for 32 neurons)
2. Second layer of convolution: 3 X 3 window, input:32, output:32
3. The third layer of convolution: 3 X 3 window, input:32, output: 64
4. Fourth layer of convolution: 3 X 3 window, input:64, output:64
5. Fifth layer of convolution: 3 X 3 window, input:64, output: 128
6. Sixth layer of convolution: 3 X 3 window, input:128, output: 128
7. Densely Connected layer: 1 X 1 window, input:128, output: 1024
8. readout layer: input:1024, output: 40 (we have recognized 40 symbols in total)

With the built normal layer, we can establish as many layers as we want, but we found six layers already gave an acceptable result. After training of 3483 set of data with 4900 steps, the accuracy reaches 96.4 %.

```
step 4900, training accuracy 0.964 loss 0.108773 max 0.997109 0.992883 lr 0.0002
```

To improve the efficiency, according to the Deep Residual Network, a skip layer can be added[1].

Specifically, we defined the window for each layer as a filter ([3,3,1,32]), and use tensorflow 2-D convolution to convert the input according to the window then puss through the converted data to the normal layer. The normal layer gives output as the input of the next layer. The next layer is just going through the same procedure just

with the new defined window to pass to the tensorflow 2-D convolution. With the dealt NHWC formate, the 2-D convolution works as follows:

1. Read the input tensor shape `[batch, in_height, in_width, in_channels]`
2. Read the a filter / kernel tensor with shape `[filter_height, filter_width, in_channels, out_channels]`
3. Flattens the filter to a 2-D matrix with shape `[filter_height * filter_width * in_channels, output_channels]`
4. Extracts image patches from the input tensor to form a *virtual* tensor of shape `[batch, out_height, out_width, filter_height * filter_width * in_channels]`.
5. For each patch, right-multiplies the filter matrix and the image patch vector.

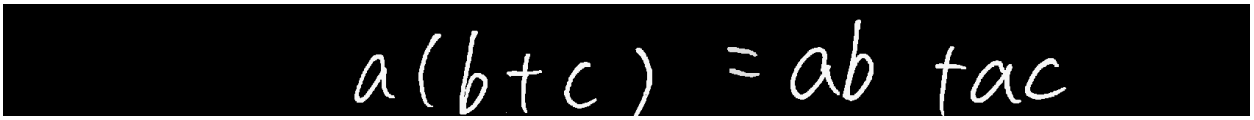
### 3.2 Training

The training loss was evaluated by the tensorflow (tf) softmax_ cross_ entropy_ with_ logits to compare the label (y_) and the calculated y. The Adam algorithm[2] (AdamOptimizer) was used to minimize the training error calculated from the loss.

First, we use tf. Sections to run the initialization of the global variables, then read the annotated data store the label to the "y" and image to the "x", then calculate the accuracy, recall and et al. using Session.run().

## 4 RESULT

Our code can handle some basic level and average level equation with 93% accuracy. However, for the complicated equation, there is still some problem. The obstacle is hard to distinguish the relationship between symbols. As we can see the perdition result, it's very good for linear equations but has a drawback for another type.




```
SKMBT_36317040717363_eq9.png    13      a(b+c)=ab+ac
a       36      30      107     133
1       64      49      71      69
(       120     20      160     126
b       179     22      227     132
+       237     23      296     131
c       311     45      373     125
)       343     34      387     168
a       561     48      627     119
b       646     23      702     132
+       714     37      786     110
a       808     47      883     121
c       889     63      969     132
=       464     63      533     107
~
~
~
```

## 5 DEPENDENCIES

| Package Name | Version |
|---|---|
| **PIL** | 5.0.0 |
| **numpy** | 1.14.6 |
| **cv2** | 3.4.1 |
| **scipy** | 1.1.0 |
| **scikit-image** | 0.13.1 |
| **tensorflow** | 1.1.0 |
| **pickle** | |

**REFERENCES**


[1] N. E. Matsakis, "Recognition of handwritten mathematical expressions," Massachusetts Institute of Technology, 1999.

[2] D. P. Kingma and L. J. Ba, "Kingma, D.P., Ba, L.J.," *International Conference on Learning Representations (ICLR),* 2015.