COSI-101 Term Project

Equation Recognition

# Description

A project to recognize isolated symbols and classify hand-written equations.

# Team member & contribution

Generally, we have nearly equal responsibility for the project.

* [Jiadong Yan](https://github.com/FrankYan93) : preprocessing.py, recognize.py, recognizeFromShelf.py, partition.py, predict.py
* [Xinyi Jiang](https://github.com/xyjiang94) : segmentation.py, MinimumSpanningTree.py, classifyEq.py
* [Zhengyang Zhou](https://github.com/zhengyjo) : DataWrapperFinal.py, MER\_NN.py, readDB.py

# Dependencies

* tensorflow
* scipy
* skimage
* shelve
* imghdr
* PIL
* numpy

# Run Instructions

python predict.py [img folder path]

# Build Instructions

1. python preprocessing.py Generate shelf of training data.(The annotated folder should be at same directory.)
2. python MER\_NN.py train [the path to save model]

# Introduction of the frame of project

The project has two main part. The first part is the neural networks that is trained to recognize isolated symbols. The other part is the algorithm to partition the image correctly and recognize the equation. The second part is inspired by the paper written by NE Matsakis. [1]

## Part one: Neural Network

At the very first beginning, we decided to use convolutional neural network to recognize the equations directly without segmentation, but we failed as we saw the model could cause more confusions when predicting. Since the time is limited, we switch our way to deal with this project in the current way.

For the neural network, we construct the training model based on Zhihao's CNN on digit-recognizing [2]. To make the model more efficient, we apply a lot of techniques such as size wrapping, deformation, drop-out rate adjustment and so on, which we will elaborate in Some Tips and Interesting findings. Generally, we find that in order to recognize 38 different symbols, we need to include more features on the input of the readout layer. In addition, in case the F(X) is too small, we also have some skip layers to mitigate the effect of vanishing gradient for bottom layer like the following:

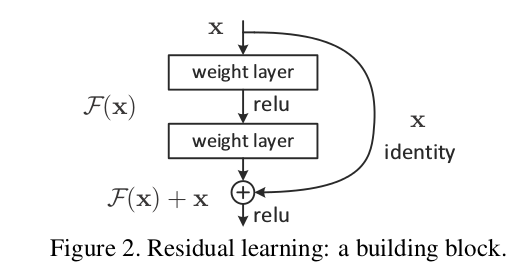
[](https://ooo.0o0.ooo/2017/05/07/590e299547c07.png)

Figure 1. Residual learning: a building block

Inspired by the Deep Residual Neural Network, we come up with the following structure with 1024 features in the final input of readout layers (input and output are recorded as number of features):

1. First layer of convolution: 3 X 3 window, input:1, output:32
2. a. Second layer of convolution: 3 X 3 window, input:32, output:32

b. Skip layer: 3 X 3 window, input:32, output:64

1. Third layer of convolution: 3 X 3 window, input:32, output: 64
2. a. Forth layer of convolution: 3 X 3 window, input:64, output:64

b. Skip layer: 3 X 3 window, input:64, output:128

1. Fifth layer of convolution: 3 X 3 window, input:64, output: 128
2. Sixth layer of convolution: 3 X 3 window, input:128, output: 128
3. Densely Connected layer: 1 X 1 window, input:128, output: 1024
4. readout layer: input:1024, output: 38

## Part two: partition algorithm

The basic idea of the partition algorithm is to divide the equation into different strokes, and find the best way to combine the strokes into symbols. There are 4 steps:

### 1. divide the equation into different strokes using connected components algorithm

We use connected components algorithm to partition the equation into isolated strokes. In the algorithm, two adjacent pixels with values that are larger than 50 are considered to be connected. When we use the algorithm as first step, we use the hidden assumption that each stroke belongs to a single symbol, and only stroke from one symbol could be connected. This assumption is not always true for the test data, which may cause errors in recognizing.

### 2. establish a minimum spanning tree (MST) of strokes

Assume that we have n isolated strokes, then the possible ways to partition those strokes will be a big number. Let f(N) be the number of stroke sets that are examined as possible symbols, then f(N) = 2N. N is the number of strokes.

To reducing this problem to a manageable size, we establish a minimum spanning tree (MST), and only consider the partition of its subtree. The vertices of MST are the strokes, the weight of the edges between two strokes are the Euclidean distance between the centroids of their bounding boxes. Therefore, only strokes that are near each other will be combined to test if it will be a valid symbol. This reduce the complexity to f(M) = 2M, where M is the number of edges.

### 3. find the best partition of strokes

The basic idea is to try different way of partition the MST, in other words, different ways of combining strokes, to find the best one. Every combined stroke will be recognized by the trained neural networks and will return a list of likelihoods. The likelihood of a partition is the sum of the likelihood of every symbol that belongs to it. The best partition will have the highest overall likelihood.

However, this is still a complex algorithm, we simplify it further, which will be discussed in detail in introduction of partition.py

### 4. classify the image according to the recognized symbols

Step 3 gives us a list of symbols and their bounding box. we can use a simple classification algorithm to map the list to one of the 35 equations. Read introduction of classifyEq.py for more details

# Some Tips and Interesting findings

## 1. Binary or Gray Scale; Segmentation VS training set

* We have a choice weather or not to use binary mode of images, which means we assign the value of a pixel to be 255 if it exceeds some threshold, otherwise assign the value to be 0.
* We should use the same/unified format in both the training set and the segmentation. For example, if we use gray scale in training set, then we should also use gray scale when processing the equations pictures that we want to predict.
* After a series of comparisons, we find the gray scale can bring us higher accuracy. So in our training and processing, we use gray scale, instead of the binary mode.

## 2. Picture deformation

Sometimes people may write a symbol in a sloppy way. Then it will be useful to apply the techniques of picture deformation to do the adjustments, such as affine transformation.

## 3. Dropout Rate

In our model, the dropout rate is 0.5, which means in every step we training, we only randomly keep 50% of our batch size to do the learning. This is to mitigate the overfitting.

## 4. Appropriate attributes/features to represent the labels

At the very beginning, we had only 64 features for the input of our readout layer and we found that 64 was not enough to represent 38 target symbols. After the symbol complexity analysis and testing, we found 1024 was appropriate.

## 5. How to classify the equations in the extra part

* If we assume that the input equation is not limited to the 35 ones in training data, then it becomes a parsing problem. One possible solution is to analyze the semantics of the symbols base on their relative position.
* However, in this case we assume that the input equation is guaranteed to be one of the given equations. Then the problem becomes a classification problem. We compare the similarity between the input image and those equation, and chose the equation with the largest likelihood.

## 6. Batch size adjustment

Originally the batch size was 50, but the efficiency was low. Then after some testing, we found 100 was much better since it could cover more cases in the training.

## 7. Input picture size adjustments

Similar with the gray scale problems, we should keep the size of the training pictures and the segmented pictures in the same size. If not, the model will not match with the segmented pictures, which will cause prediction errors.

## 8. The range of each element of picture numpy array

Similar with previous one, we should keep the range of the elements in numpy arrays that the training pictures are transformed to, and the of the segmented pictures the same. For example, if each element in a numpy array of training picture is in [0,1], then the element in the array of segmented pictures should also be in [0,1]. Otherwise, we will have an extremely low accuracy.

## 9. Recognize ".", "-", "x"

* ".": If there are 2 dots and there is a "-" between them. I would combine the three elements and create new segment called "div" and delete old 3 elements. If there are 3 dots consequently, I would transform them to a single segment called "dots"
* "-": If there are two "-" whose x1, x2 has some range in common, then turn them into a "=". If there is a "+" at the up side of "-", combine them and turn them into "pm". If Detected upper element and lower element of "-”, then turn "-" into "frac". If only lower element was detected, turn "-" into "bar".
* If left and right of "x" are variables or frac then turn "x" into "mul".

# Python Files Details

## preprocessing.py

1. Used regular expression to recognize segmented piece of image.
2. Transformed the image to standard 32\*32 np array.
3. Stored data to a db file as training data to avoid waste of preprocessing time when training.

## MER\_NN.py

The training model file, which contains the whole structure of the neural network. The input of this file is from DataWrapperFianl.py

## DataWrapperFinal.py

1. Take the numpy arrays of training pictures and the corresponding labels as input to produce training set
2. get\_valid method is to use the first 500 records as cross validation sets
3. next\_batch method is to produce the batch of training set in each step

## readDB.py

1. Read the training shelf from preprocessing
2. Produce the numpy arrays of training pictures and the corresponding labels. Then we can feed them into DataWrapperFinal.py

## segmentation.py

1. This class takes an image path to initiate. It using connected components algorithm to label the different strokes in the image, and calculates the bounding box of each strokes
2. It can return the image of a stroke cut from the original image. It is also capable of return the image of several combined strokes.
3. It has two modes, the grey mode which cut image of stroke from the original image and the binary mode.
4. Some of the pictures has little white dots that takes few pixels. These noises would be recognized as isolated strokes, which causes the program to recognize more strokes than the actual number. Therefore, we drop the strokes that has less than 50 pixels.

## MinimumSpanningTree.py

* This class takes a list of bounding boxes of strokes to initiate. It calculates the minimum spanning tree of those strokes using Kruskal’s algorithm.
* The weight between any two strokes is the Euclidean distance between the centroids of their bounding boxes
* Its output is a dictionary, whose key = vertex, value = a list of tuples; format of the tuple: [connected vertex, weight]

## partition.py

The Partition class can be initialized with a MinimumSpanningTree ‘s dictionary output, a Segmentation instance that is related to a specific image(of png format), a session and a shared SymbolRecognition instance that is initialized with model\_path.

There will be an error if we create a session and a SymbolRecognition instance inside the Partition class, this is because we would create multiple partition class to recognize multiple equations.

#### The algorithm:

We tried to use probability of the result to find segments with low probability and put them into combine list. However, this is can be very slow since each symbol need to have an additional calculation of softmax and argmax. Therefore, we eventually decided to clear up all the probability related part and rewrite the partition algorithm.

Here is the simplified algorithm:

1. Visit the mst once and recognize all the segmented image and store results to self.ls. Format of results: [symbol, y1, y2, x1, x2, list of labels]
2. Sort self.lst by x1. The order will be left to right.
3. Deal with ".": If there are 2 dots and there is a "-" between them, combine the three elements and create new segment called "div" and delete those 3 elements. If there are 3 dots consequently, transform them to a single segment called "dots"
4. Deal with "-": If there are two "-" whose x1, x2 has some range in common, then recognize them as a "=". If there is a "+" at top of "-", combine them and recognize them as “pm". If there are elements that both on both side of “-”, then recognize it as "frac". If only lower element was detected, recognize it as "bar".
5. If left and right of "x" are variables or frac then turn "x" into "mul".

## recognize.py

Mainly written for debug. Recognize image directly and calculate the accuracy and output results.

## recognizeFromShelf.py

- Mainly written for debug. Recognize image from preprocessed db and calculate the accuracy and output results.

## classifyEq.py

* This class takes the output of `partition.py` to initiate, and returns the number and latex expression of the most likely equation
* To measure the similarity between input image and the 35 equations, we construct a vector space where each dimension represents a symbol. Every equation is a vector, the value of the vector in one dimension is the number of times the symbol appears in the equation.
* Cosine similarity is not appropriate because we want to take the absolute value into consideration. Because an image where x, y both appear once is different from one where x, y appears 3 times.
* Therefore, we use Euclidean distance to measure similarity. The equation with the least distance to the input image has the largest similarity
* We test all the equations in annotated folder, and get 325 correct results out of 389 equations. The accuracy rate is 83.5%

## predict.py

Combine all modules we made together and generate the result txt file.

# Result Analysis

We run this program on every equation pictures in annotated, and recognized 347 equations out of 389 pictures, the accuracy rate is 89%. To analyze the results, we save the results of incorrectly classified equations in lib/err.json.

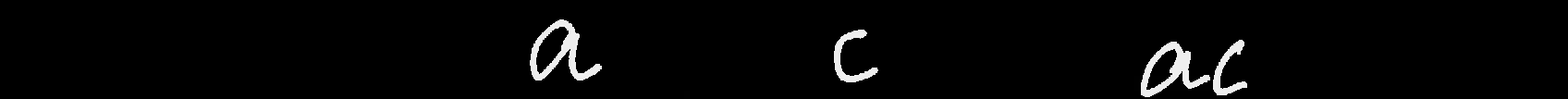
The first type of mistakes is caused by the incomplete equation pictures such as figure 2.

Figure 2. incomplete equation

The second type of mistakes is caused by strokes that is not connected as expected. As figure 3 shown, the square root symbol is supposed to be connected, however, it is written as two parts. In the partition algorithm, we use enumeration to combine strokes such as division, therefore, the algorithm will fail when unexpected situation appears.

This problem can be solved if we use the previous time-consuming partition algorithm, where we examine the maximum probability of the symbols, keeping the strokes with high probability while try to combine the strokes with low probability. Therefore, the separated stroked of square root may have low probability because the neural network is not trained to recognize them, which will lead to combining the strokes. However, there comes the new problem, only neighbors in mst would have the chance to combine. To make the separated strokes of square root be neighbors, the weight of the edge between them must be small. We may have to adjust he way we calculate weight, for example, taking the minimum distance of bounding box into consideration.

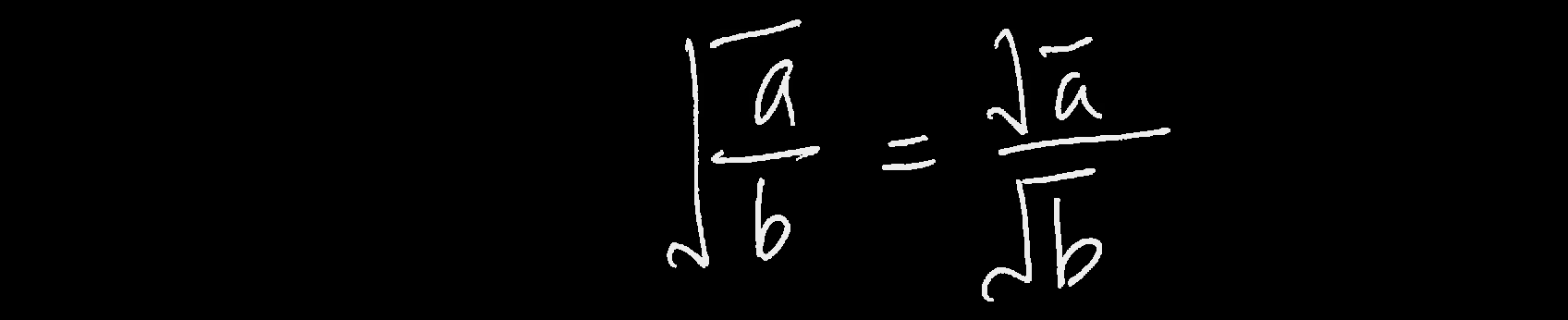


Figure 3. stroke that is not connected

The third type of mistakes is caused by that strokes of different symbols are connected, such as figure 4. The “n” of “tan” is connected to “6” unexpectedly. This problem is unsolvable for current frame of projects, because the algorithm assumes that there’s no stroke that belongs to two symbols at the same time.

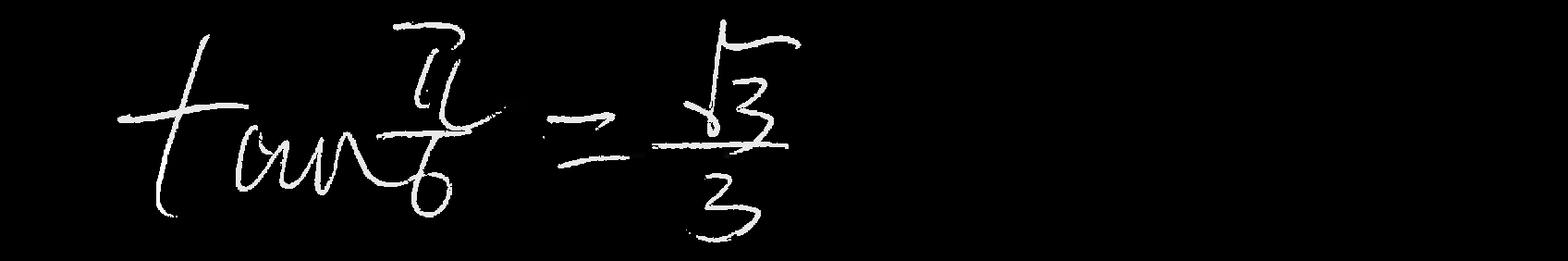


Figure 4. strokes of different symbols are connected

To solve this problem, the only solution is to use different project frame. For example, a neural network that takes the whole equation as input, and use a window to scan through the picture. This causes new issue: First, we have to choose appropriate window size, second, the size of symbols affects recognizing, it may have trouble recognize 2 as square.

# Reference

1. Matsakis, Nicholas E. Recognition of handwritten mathematical expressions. Diss. Massachusetts Institute of Technology, 1999.
2. Zheng, Zhihao NN code for Digits Recognition