Handwritten mathematical equation recognition and calculate

Xiaoqi Lin
Illinois Institute of Technology
China
xlin31@hawk.iit.edu

Changjun Hou*
Illinois Institute of Technology
China
chou4@hawk.iit.edu

Abstract

Most of the current calculator software is through the button or keyboard input number and symbol for calculation, which is inefficient for combinatorial arithmetic input. We wanted to make a handwritten calculator to compute the results more humanly. For example, users can write directly on the system's writing pad to generate a picture. We will send the handwritten calculation picture to the system, for example, the picture may contain: 3 + 46 ÷ 2 - 5.9 =, the system needs to identify the specific content of the calculation equation from the handwritten picture and judge whether the calculation equation is valid. If the equation is valid, then automatically output results.

Keywords: datasets, handwritten, recognition, caluculate

1 Introduction

In the current machine learning algorithms, the recognition of a single number is a very mature technology, which can be trained directly by using dataset like MNIST. Handwritten symbols can also be used to collect relevant data sets for training. But the main difference in the project is that

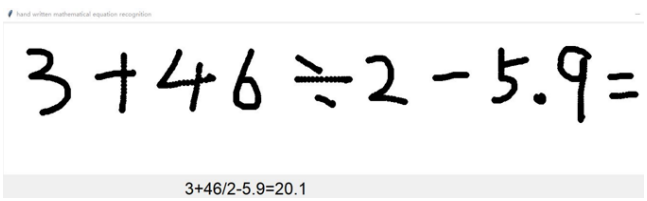

Figure 1. The sample of handwritten recognition

the handwritten picture will contain an lot of numbers and symbols. We need to split the contents in the image into individual images of numbers or symbols of different lengths and widths. For example, in the picture: 3 + 46 ÷ 2 - 5.9 =, we should identify: 3, +, 4, 6, ., ÷, 2, -, 5, ., 9, = , and other small images. Then combine each small picture in the content to identify, and form a complete equation.

The goal is to make the machine automatically recognize mathematical formulas and output the final result, like a primary school student. In order to achieve this goal, we have to analyze at least the following three major problems to solve:

- 1. Numbers and mathematical symbol recognition.
- 2. Handwriting image input, segmentation.
- 3. Combination and calculation of image recognition results.

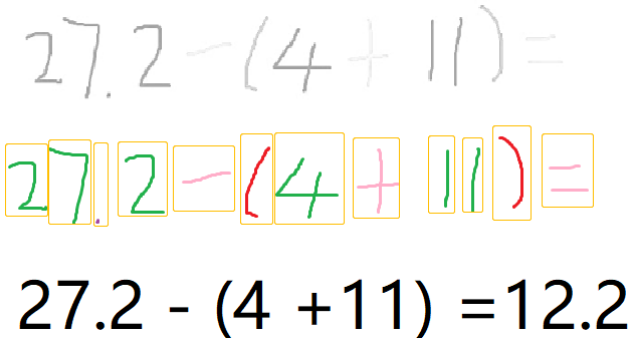

Figure 2. Handwritten expression recognition process

2 The dataset used in the project

In order to recognize numbers and mathematical symbols, we must first have a suitable data set. At the beginning, we wanted to use MNIST data set directly, and soon implemented pure number recognition through Python and keras. But this dataset does not include mathematical symbols, and MNIST is not a standard image format, making it difficult to expand the additional symbol dataset we need.

Later, we found a new data set from kaggle(https://www.kaggle.com/clarencezhao/handwritten-math-symbol-dataset), which contains 10 numbers and 6 kinds of mathematical symbols, a total of 8570 pictures. The size of each image is 135 * 155, which is more HD than MNIST. This dataset was generated on the iPad by multiple people using a stylus and saved in JPG format.

Figure 3. The mix dataset of Kaggle

*Both authors contributed equally to this research.

Since there are no open and close parentheses in the dataset, we generated several images by hand, and then generated the relevant datasets automatically through Keras.
Images are divided into two folders: Train and Eval, and each folder is divided into sub folders based on numbers or mathematical symbols. For example, each image of the number 6 is placed in the "six" folder; Each picture with the plus sign is placed in the "plus" folder. So the name of the folder is the label of the dataset.

3 Model of training single character recognition

We use python, keras, opencv, sklearn, tensorflow and other libraries to train this model.
First of all, we use the OS Library of Python to read the image of the data set, and label the data set by the directory name when reading.Since the training data set and the test data set are already in different folders, we naturally make the distinction when we read the image file.
Because Keras divides training data set and validation data set into training data set, it is divided and then randomized. However, when we read the training set, we read the files in the folder in order, which is easy to cause the situation that part of the numbers are all divided into the validation set and do not exist in the training set. Therefore, we had to randomize the data before handing it to Keras for split.
Since our UI program for this issue is entering monochromatic strokes, we have converted all image datasets to grayscale images and scaled them to a 100 * 100 size. This can also accelerate the speed of training, certain procedures to avoid overfitting.
Since Sigmoid has the problem of gradient disappearance, we choose ReLU as the activation function.The output of ReLU is stable, because its Z >0 interval is a linear function, and there is no problem that the gradient of Sigmoid disappears. And ReLU is very fast. Softmax can show the results of multiple classifications in the form of probability, which is very suitable for our project scene this time and classifies the recognition results into one of the 16 classifications.
The loss function of this model uses the categorical crossentropy. The cross entropy is used to evaluate the difference between the probability distribution obtained by training and the real distribution. It describes the distance between the actual output (probability) and the expected output (probability), that is, the smaller the value of cross entropy, the closer the two probability distributions are.
The optimizer method we use is ADAM, whose name is derived from Adaptive Moment Estimation, which is also a variant of gradient descent algorithm. However, the learning rate of each iteration parameter has a certain range, and the learning rate (step size) will not become too large due to the large gradient. The value of the parameter is relatively stable. The ADAM algorithm adjusts the learning rate of each parameter dynamically by using the first and second moment estimation of gradient.
Next, we use maxpooling on the models and then use dropout

```
# using sequential model for training
model = Sequential()

# each grayscale image is 100x100x1
model.add(Conv2D(32, (3, 3), padding='same', input_shape=(100, 100, 1), activation="relu"))
model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same', activation="relu"))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(18, activation="softmax"))

model.compile(
    loss='categorical_crossentropy',
    optimizer="adam",
    metrics=['accuracy']
`
```

Figure 4. the keras code of Model

to prevent overfitting.Dropout can effectively alleviate the occurrence of overfitting and achieve the regularization effect to a certain extent. Dropout can be used as a trick to train a deep neural network. In each training batch, overfitting can be significantly reduced by omitting a portion of the characteristic detectors (allowing a portion of the hidden layer node value to be 0). This approach can reduce the interaction between characteristic detectors (hidden layer nodes), where some detectors depend on other detectors to function. To put it simply, Dropout says that when we propagate forward, we let the activation value of a neuron stop working with a certain probability p, which makes the model more generic because it doesn't rely too much on some feature.
Currently Dropout is heavily used on fully connected networks and is generally considered to be set to 0.5 or 0.2. Dropout is a super parameter that needs to be experimented with for specific networks and application domains. In this project, the number of data sets is not very large, and part of the data is generated by the machine, so we finally used dropout three times, which were set to 0.25, 0.25 and 0.5 respectively.
Finally, Softmax is used in the output layer of the model. Softmax is used in the process of multi-classification. It maps the output of multiple neurons to the interval (0,1), which can be understood as a probability, so as to carry out multi-classification. Our project is a multi-classification problem. We need to identify the input picture and output it as one of 18 categories of numbers or symbols.
When the model construction was complete, we began to perform the training. On a computer with an integrated graphics card, each epoch takes about one minute to execute. We found that we only needed to execute 50 epochs to get good results, so the total training time was about 50-60 minutes.

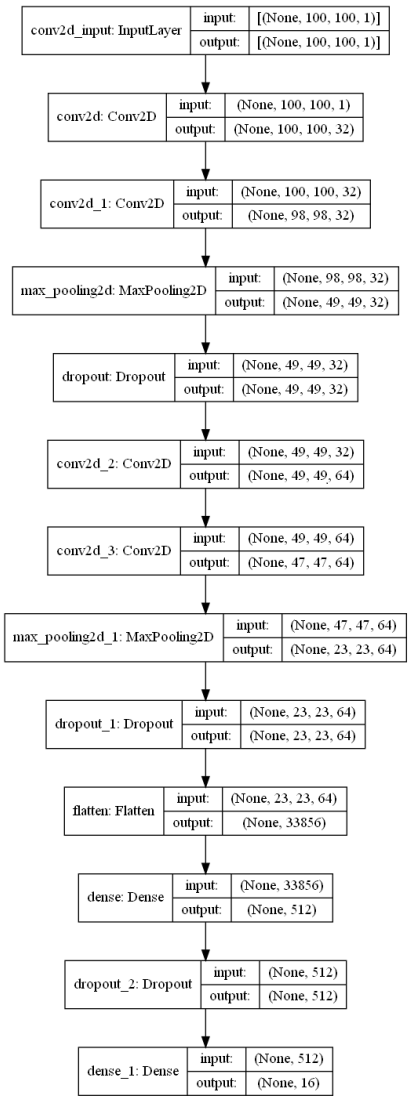

Figure 5. Model Summary

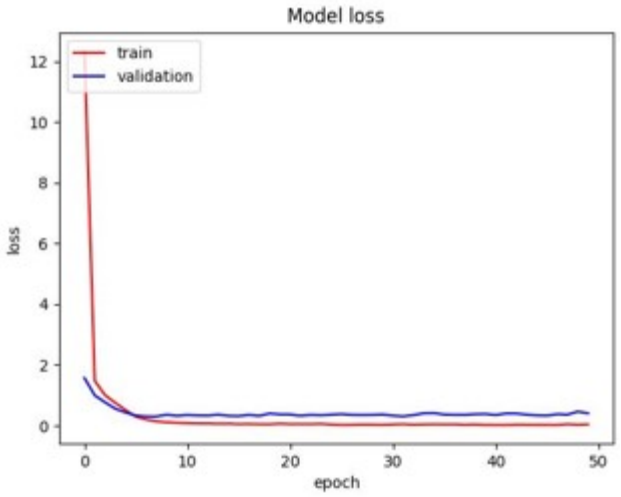

Figure 6. The loss trend of the model

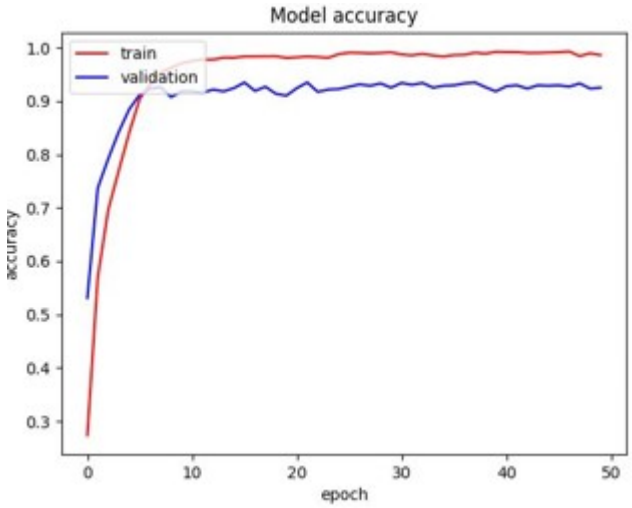

Figure 7. The accuracy trend of the model

In order that the results of the training can be used repeatedly, instead of having to retrain for a long time each time, we saved the trained model into a local file through Keras. Finally, we plotted the loss and accuracy images in the result process through Matplotlib.

4 Results of training single character recognition

At the end of the training, let's look at the final results. In the picturet, we can see that with the increase of training epochs, loss is decreasing, and tends to 0. And the accuracy rate is rising.

In the test results in the figure, we can see that the prediction results are very accurate. For example, in the third picture in the first line of figure 7, this is an incomplete number 8. There is a large gap in the upper left corner, but after our training, the machine can still recognize it correctly.

The following line is just as accurate. We guarantee the accuracy of a single character, so that it can be used in the following multiple character recognition process.

The final accuracy of 50 epochs is 92%. The accuracy of 300 epochs can reach 98%. This is a good result for us.

In the following practical test, we will use the mouse to simulate handwritten numbers and mathematical symbols. Its handwriting is slightly different from that of a stylus, but the difference in recognition accuracy is not big.

In the middle of this project, we use Tkinter to develop a UI program that can recognize a single character, which can accurately recognize a single number or mathematical symbol. Then, we can try the case that multiple characters appear at the same time. The most basic example is the case that two numbers appear at a time, for example: 42. Not surprisingly,

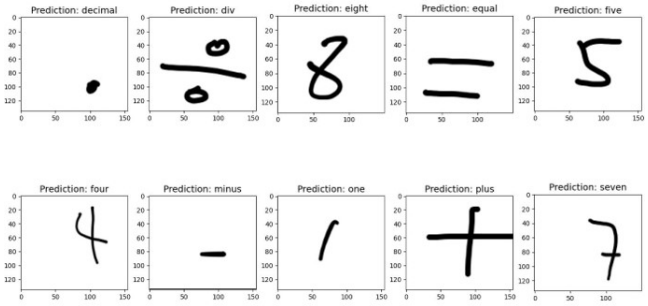

Figure 8. The validation sample.

this program can't accurately identify 42, up to 4 or 2. The reason is that as the core of our artificial intelligence application, the model itself does not have the ability to identify multiple numbers. As the source of the model, the training data set only covers a single handwritten digit.
In the case of writing multiple numbers, we are actually "forcing" the AI model to do reasoning beyond its adaptive range. This is a misuse of AI models. Naturally, the results are not satisfactory. So, in order to enhance the usability of the application, can we improve the application to handle common mathematical expressions? This requires our application to recognize not only numbers and symbols, but also multiple characters appearing at the same time. For the case of multiple characters, we naturally think that since the model can recognize a single character well, we just need to separate multiple numbers and let MNIST model recognize one by one. In this way, we introduce a new subproblem, namely "split of multiple handwritten characters".

5 Split of multiple handwritten characters

The most critical part is how to split each characters from the image.
There are many techniques to do object detection in image, like sliding window and YOLO. But as the nature of the number characters, when we are trying to use sliding window, it is very easy to cut part of the number and produce many unexpected results. The following figure is a test result of sliding window.

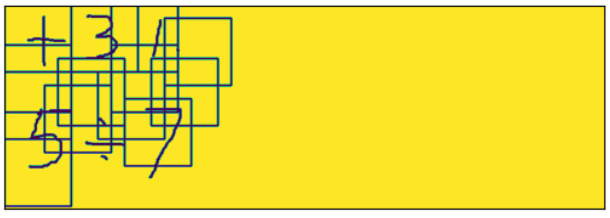

Figure 9. The test of sliding window

So we decided to use our own method to do the split. Firstly by checking the values from the image, we can see that the image is a real Binary Image.

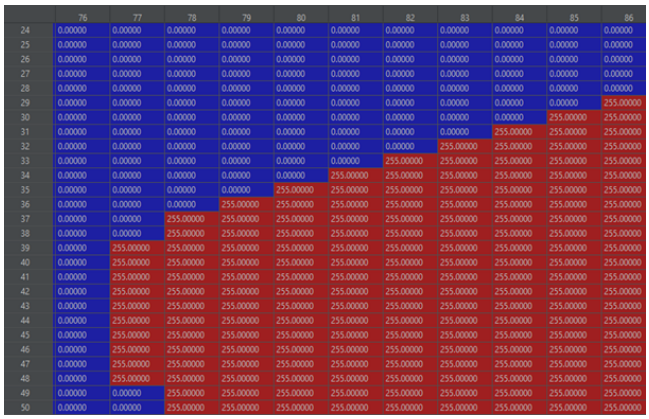

Figure 10. pixel values in a Binary Image

The most intuitive way of splitting each characters is by drawing vertical and horizontal lines on the image, if the line can go through the white space and cut out the characters, then we know that the line is an effective cut line.

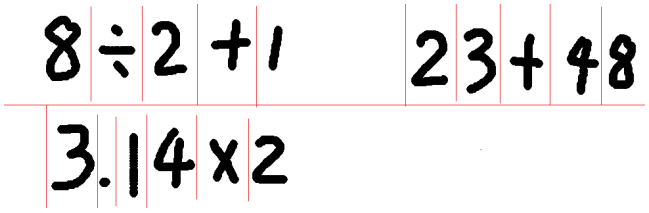

Figure 11. The most intuitive split method

By actual programming, we sum the pixel values by each row first, then we got the histogram like the left side in following figure, and we can split the rows of the expression by the lowest value points.
After this we know there are 2 rows of expressions in the image.

Then by each rows of the expression we do the same operation, but this time we sum the pixel values vertically and so get the diagram like the left side in following figure, and by the diagram each lowest value point will be a separation between each character.

After previous steps we know how to cut out the characters, we have one problem is that many of the characters are not exactly center aligned, see this number "8" as example, it is aligned on the top, and it has a large margin on the bottom.

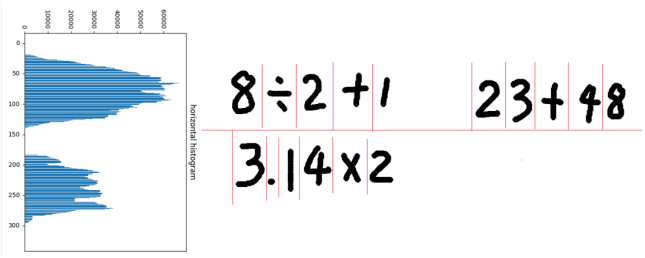

Figure 12. The horizontal split by the histogram of the sum of pixel values

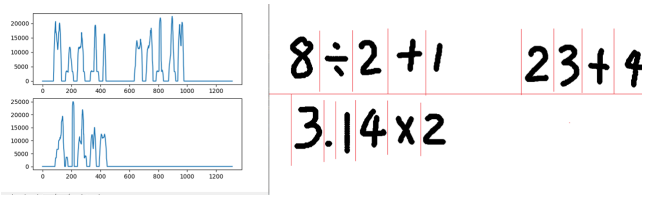

Figure 13. The vertical split by the histogram of the sum of pixel values of each row

We can directly give this picture to our trained model, but the accuracy will drop down dramatically sometimes.
So we do following operations for the image.

1. Cut out the image by the histogram split.
2. Shrink the white space around the character.
3. Add 20 percent height white space to the top and bottom side of the character.
4. Then add white space on left and right side to make it square (make the character center aligned with proper white background).
5. Scale the squared image to 100x100 pixel size that our model expects, so that it will not be distorted.

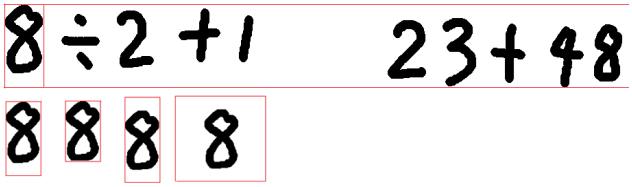

Figure 14. The detailed steps of how to split one character out of the image

Another issue is how to split each expression from one row, for example, the above row has 2 separate expressions. We do this by dynamically calculate average margin between each characters, and if this margin is larger than a threshold like 2 times larger than average, then we take it as a separation between the expressions.

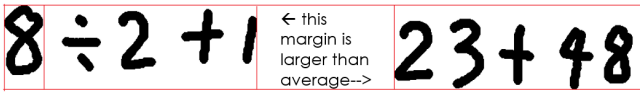

Figure 15. How to split expressions in one row

6 Create additional dataset of Parentheses

The dataset we found on Kaggle does not include parentheses, so we tried to use Keras ImageDataGenerator to generate the required training and evaluation datasets.
As parentheses and numbers like "1" are similar sometimes, the variation should not be too large.
Also we don't need flip, otherwise it will be hard to distinguish left and right side of the parentheses.
By following parameters, we can make the whole training validation accuracy near to 0.98:
rotation_range=1,
width_shift_range=0.1,
height_shift_range=0.1,
shear_range=0.05,
zoom_range=0.05,
horizontal_flip=False
In the following figure, the left side is what I wrote and the right side is the what the generator generates.

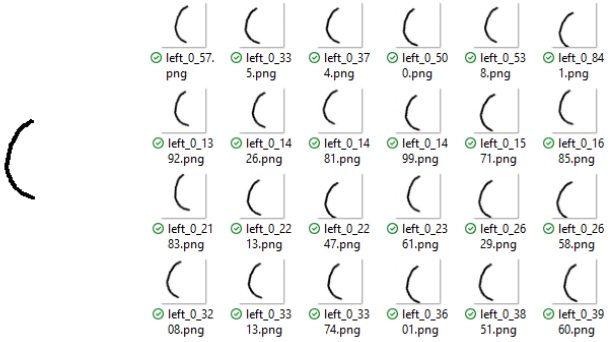

Figure 16. How to generate parentheses

By adding 2 parentheses dataset, we still get similar train and evaluation accuracy.
And we can use the trained model to predict the parentheses correctly.

7 Calculation of formula

First of all, we will recognize the characters (whether numbers or mathematical symbols) in order to form a string.Then, we use regular expressions to identify the operation symbols in the string, which can be divided into two categories: multiplication and division, addition and subtraction.Although

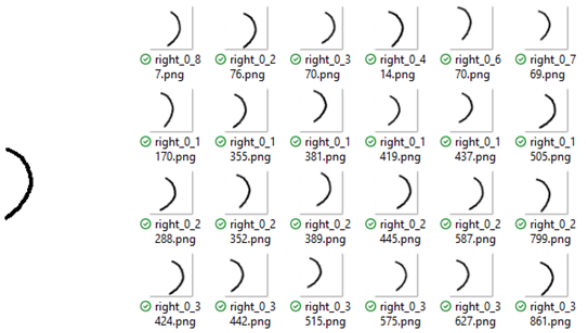

Figure 17. How to generate parentheses

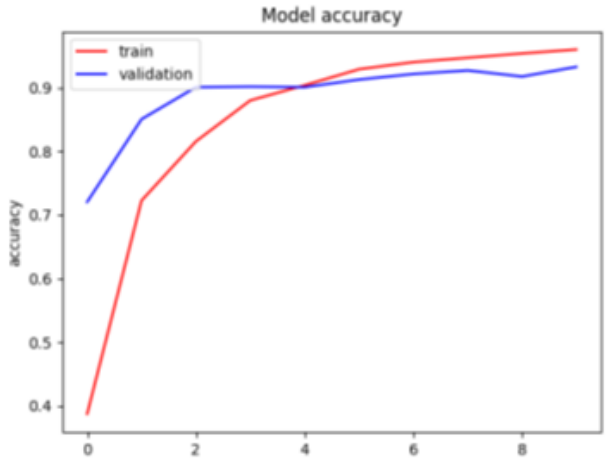

Figure 18. The model accuracy after adding parentheses

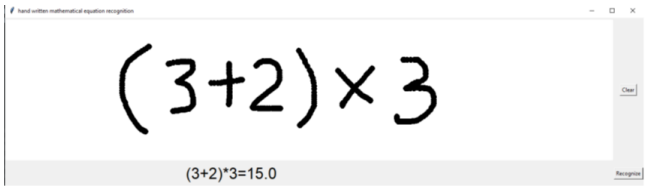

Figure 19. The ui that recognizes parentheses

regular expression writing is more complex, it can achieve accurate matching and fast operation. So we have adopted this scheme, and the actual implementation is good.

We recursively carry out the calculation in the order of multiplication and division before addition and subtraction, and carry out special treatment for the negative sign to get the final result.

```
import re
md_r=re.compile("\d+\.*\d*[\*\/]+[\-]?\d+\.*\d*")
am_r=re.compile("[\-]?\d+\.*\d*[\+\-]{1}\d+\.*\d*")
p_r=re.compile(r"\(([^()]+)\)")
```

Figure 20. The regular expression

When the priority of the operation is determined, the following procedure is simple. It simply judges the operation symbol, calculates the left and right side of the character, and adds the result to the recursive operation again until all the operation results are complete.

8 UI program

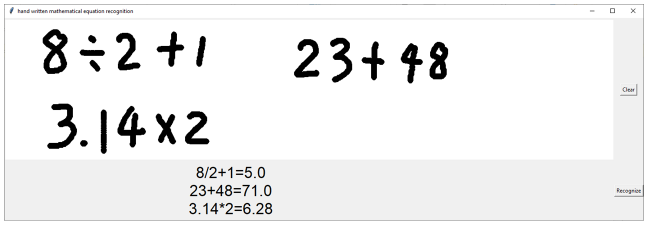

Figure 21. The ui program

This project requires a graphical interface for hand-written input, rather than a command line, as other Python-based machine learning projects do. Considering the universality of PC and cost, we designed to use dragging mouse to achieve handwriting input in the project, instead of using the combination of stylus and iPad.

According to the above goals, we choose the Tkinter library for development.Tkinter is a window design module that uses Python. The Tkinter module (" TK Interface ") is the interface to Python's standard TK GUI toolkit. As a Python specific GUI interface, it is an image window. Tkinter is Python's own GUI interface, you can edit the GUI interface, we can use GUI to achieve a lot of intuitive functions. Programs written with it can display and run normally on Windows, MacOS and Linux.

In this project, in addition to the basic Button and Label components, we mainly use Canvas component. Canvas is a general-purpose component that is typically used to display and edit graphics. You can use it to draw lines, circles, polygons, and even other components. To better simulate handwriting, we use the generated handwriting of canvas.create-oval () when the mouse moves to draw lines.Note that objects added to the Canvas will always stay straight. If you want to modify them, click the "Clear" button to Clear them.

We set the UI background as all white, and the written as all black, so this simplifies our problem, the image we captured from the program is a Binary Image. In tkinker, each button can be bound with an event. For example, if you click the "clear" button, the program will call the clear event to clear all the contents on canvas. When the "recognize" button is pressed, the program will call the recognition event and output the recognition result through the trained model.

9 Conclusions

- We successfully trained a convolutional neural network using Keras.
- We used convolution layer, max pooling layer, and relu activation function for hidden layer, and softmax activation function for the output layer.
- We used dropout to improve the performance and prevent from overfitting.
- We create a UI program to capture written to Binary Image.
- We use our own character segmentation method to split each characters.
- We use Keras ImageDataGenerator to generate parentheses dataset that original dataset doesn't have, and it works.
- We use regular expression to calculate final result of each predicted expressions.

To sum up, a good data set is the foundation of training. When the dataset is incomplete, we can augment the dataset, or even generate the dataset through a generator. Keras simplifies our program, but the core model definition is still up to the developer. Image content cutting is a difficult point, different ideas and algorithms will produce different results, we currently implemented one of the ideas. We can continue to optimize later.

10 Future work

However there are still several special cases that we haven't dealt with

The first special case is that one expression overlaps with other 2 expression vertically, please see the example from following figure.

The possible solution is that we can detect vertically and horizontally at the same time, and remove the detected characters and continue with the left part of the image and so on.

So in the possible solution we start to use a line to cut the image from the upleft side of the image, and move on the cut line simultaneously, and until the vertical and horizontal line cross, then we detected one character.

In the case that the following figure shows, we detect number 8 by vertical and horizontal lines firstly, and then devide character secondly, and so on.

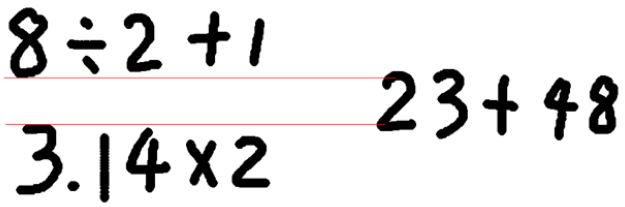

Figure 22. Future work case that row overlaps

Another case that may happen frequently in written numbers is that 2 numbers connect with each other.

The possible solution maybe we can solve this by detecting the unusual width of the character and try to split from above or below side.

As showing in following figure, if we detect 2 and 3 as one characters, then we compare the width of this one character with others in the image, if this character's width is unusually larger than a threshold of the average of others, then we try to cut the character by histogram like we used before, but this time the lowest value point may not be zero.

Figure 23. Future work case that 2 numbers are connected with each other

Another special case is that the decimal is hidden under a number, please see the example from below figure.

The possible solution maybe we can check the connected area of the detected character, as in this example if we detect that this detected characters has 2 onnected areas, then we try separate the decimal from right bottom side of the image using horizontal and vertical lines.

Figure 24. Future work case that decimal is hidden by a number

The last case that we think is that, if we want to put our project into real use scenarios, the image should not be Binary Image anymore, it should be a common RGB image with various backgrounds.

So in this case we have to change the camera image into a

Binary Image first, then we can use our current project to do the detection.

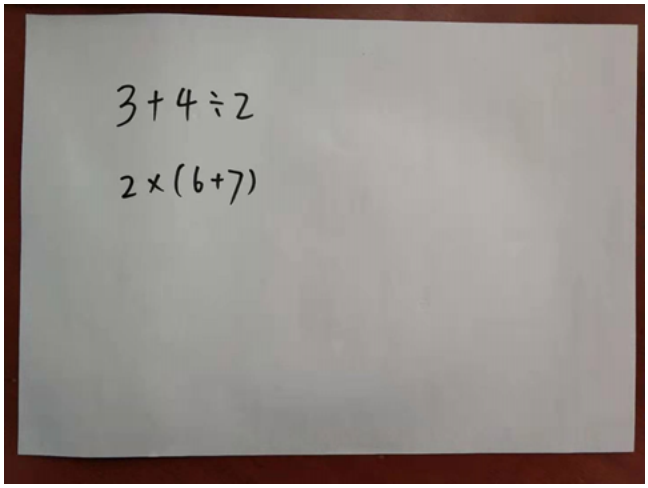

Figure 25. Future work case that detect from camera image

11 Citations and Bibliographies

References

[1] Emmanuele Grosicki and Haikal El Abed. 2009. Icdar 2009 hand-writing recognition competition. *In 2009 10th International Conference on Document Analysis and Recognition IEEE* (May 2009).

[2] Minesh Mathew Kartik Dutta, Praveen Krishnan and CV Jawahar. 2018. Improving cnn-rnn hybrid networks for handwriting recognition. *In 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)* (2018), 2672–2680.

[3] Thomas Deselaers Jonathan Baccash R. Reeve Ingle, Yasuhisa Fujii and Ashok C. Popat. 2019. A scalable handwritten text recognition system. abs/1904.09150 (2019).