Camera Calculator

CSC420 Report

Yian Wu

Group member:

Quan Zhou
Luyao Yang
Yian Wu

Abstract: The whole project

The whole project is to write a camera calculator. Our program is able to detect and recognize the math digits and symbols on a photo. Then, it is able to do three things. Firstly, solve printed math problem. Secondly, solve hand-written math problem. Thirdly, solve Sudoku problem. Each person in our group will be responsible to one of the following parts mainly at first. For this project, I am responsible to solve the printed digit math problem.

Keywords

Image processing, Features and Matching, and Recognition.

Introduction: Solve printed math problem

The program takes photos of printed math problem as input. Some of the photos are oblique. It is able to recognize the math expression on the photo, and solve it. Then, paste the result back to the photo.

Input data:

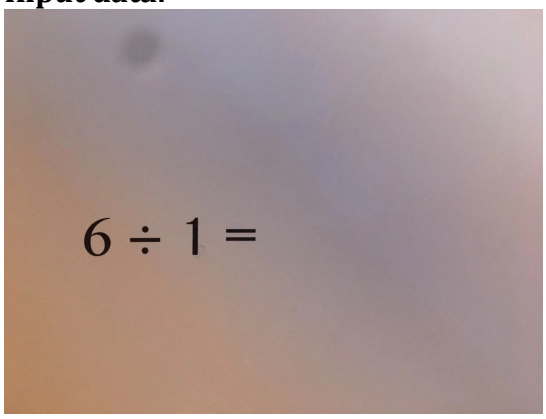


Figure 1

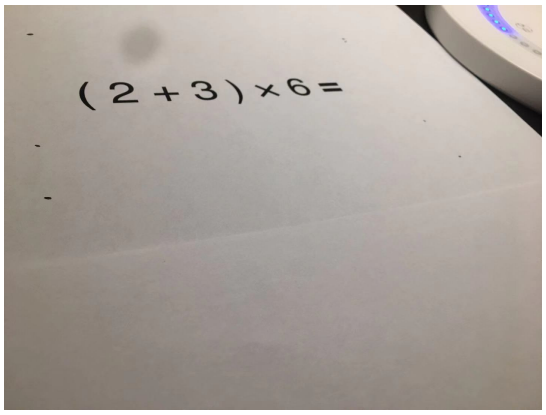


Figure 2

Methods and Steps

1 Image Processing and Element Detection

1.1 Image Processing:

For none oblique image, the image need to be processed as clear white background and

black digits and symbols. This can be down by applying thresholds. This will be continent for recognizing. (reference to the function **preprocessing_img**)
For oblique image, the image should first be processed by applying **homography theory.** (reference to the function **do_homography**) Then using opencv, to produce a clear white background and black digits and symbols.

After apply homography and thresholds:

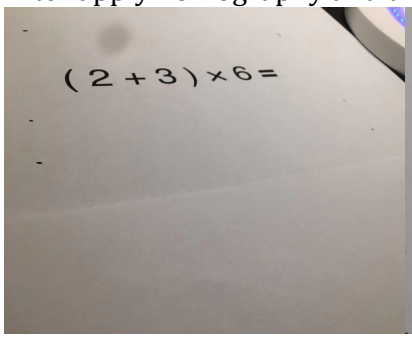


Figure2

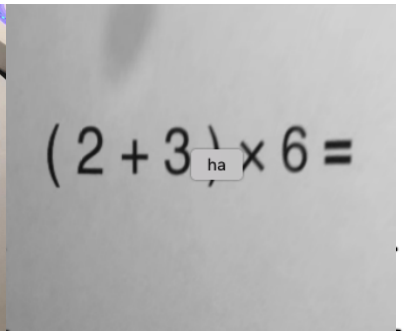


Figure3

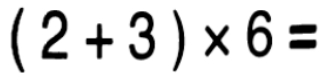


Figure4

1.2 Element Detection:

We need to detect each digits and symbols on the photo. After discussing with group member. I decided to first **finding the contour** of the graph, and then throw those contour which are two small. I use the method instead of general object detection methods as we did in a4 is because, We have already preprocessed the image, the image now is just black and white, so there are no need to use complex method to detect as in a rgb image. In addition, all the element we need are highlighted in black. (reference to the function **detect_element_from_pic(img)**)
Then I cropped each elements in the photos. We also need to save the coordinates of the last "equal"

Cropped image:

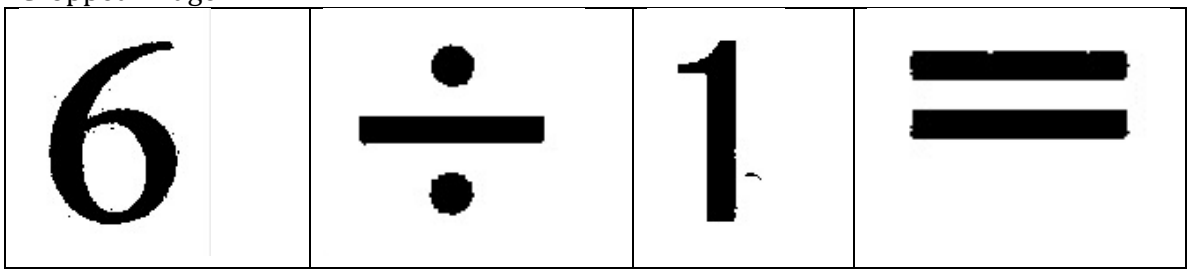


Figure 5

2 Digit and Symbol recognition

2.1 Get Training data:

There are no online sources. So I generate the data by myself using the system font.
Data for digits and symbols:

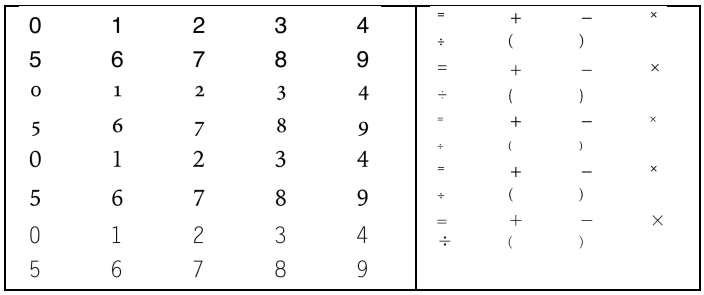


Figure 6

I have 681 images in total. There are 24 images for each digits, and 63 images for each symbols. I used so many data is because, this program require the recognition part to be precise, so then the calculation can be performed.

Then, I split the data into training set and test set (18:6 for digits, 53:10 for symbols). (refer to the function **split_data() , split_data_symbol()**)

For digits set label, I am using 0-9, for symbols set label, I am using 10 – 16.

2.2 Neural network model

Neural network techniques are suitable to do classification. Firstly, I build up the layer with ReLu and softmax. I tried other activation function such as sigmoid and tanh but ReLu performs the best. This might due to avoid vanishing gradient problem. The loss function is sparse cross entropy.

```
keras.layers.Dense(128, activation=tf.nn.relu),
keras.layers.Dense(17, activation=tf.nn.softmax)
```

I chose Adam Optimizer, I also tried SGD, SGD with momentum, RMSprop. Among those, Adam performs the best. SGD with momentum is better than SGD, this is because it accelerates. RMSprop consider AdaGrad method but doesn't contain momentum. Adam combined both momentum and AdaGrad. So Adam can help to jump through dampens.[1] I also use a batch size 40.

The final accuracy for test set reaches 100%

```
551/551 [==============================] - 0s 60us/step - loss: 0.0013 - acc: 1.0000
Epoch 150/150
551/551 [==============================] - 0s 64us/step - loss: 0.0013 - acc: 1.0000
130/130 [==============================] - 0s 375us/step
Test accuracy: 1.0
```

The accuracy for training reaches 100% at 82 epoch, and maintain the same accuracy after.

```
Epoch 82/150
551/551 [==============================] - 0s 63us/step - loss: 0.0073 - acc: 1.0000
Epoch 83/150
551/551 [==============================] - 0s 58us/step - loss: 0.0060 - acc: 1.0000
Epoch 84/150
551/551 [==============================] - 0s 70us/step - loss: 0.0050 - acc: 1.0000
Epoch 85/150
551/551 [==============================] - 0s 74us/step - loss: 0.0040 - acc: 1.0000
```

(refer to the function **classifier_data()**)

2.3 Make prediction

[1] An overview of gradient descent optimization algorithms 19 JANUARY 2016 Sebastian Ruder http://ruder.io/optimizing-gradient-descent/

Then I used the model we have trained above to do prediction, and find the maximum value among all prediction.

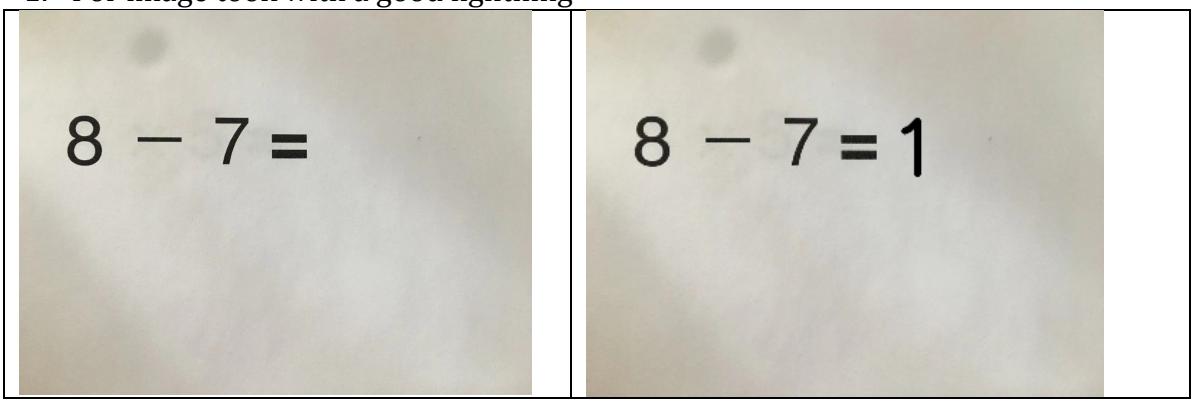
3 perform calculation and paste the result back to original image

After I get the prediction. I combined all the element in to a string, and use eval() to calculate the result.

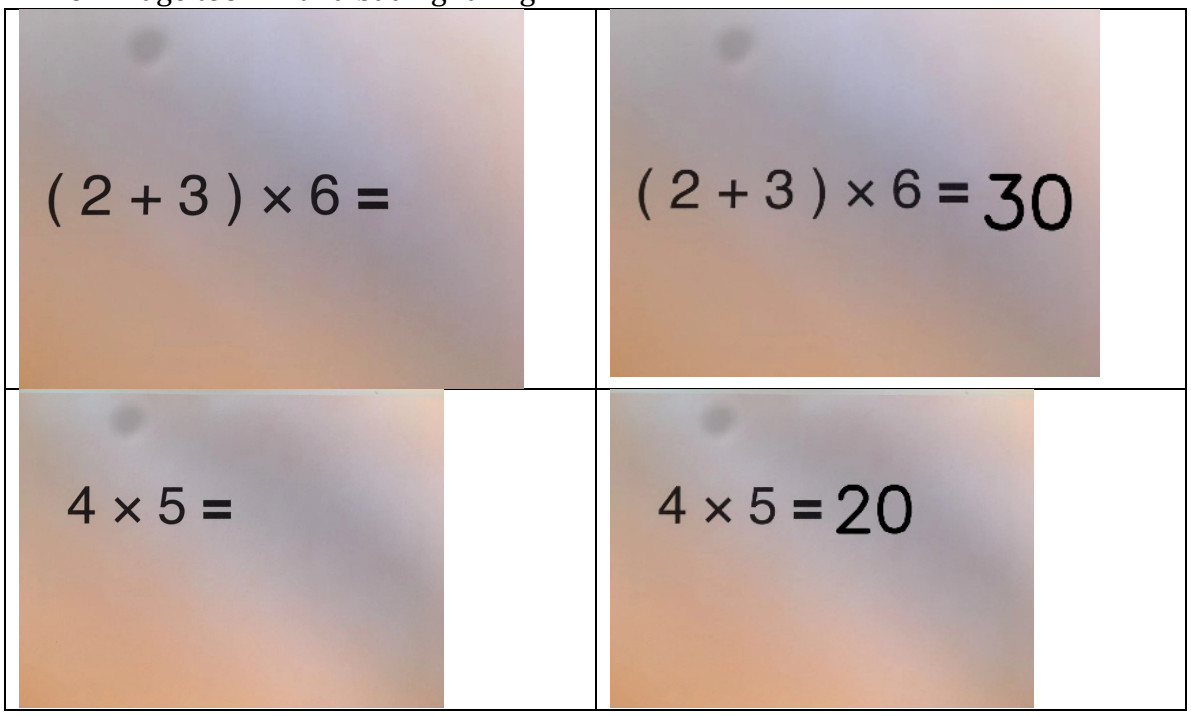
Then, I paste the result back to original image. Since I already get the pixels of the last element (as mentioned above), I paste it back by using cv2.putText. Then adjusting the size of the image, and save it to the directory.

Result

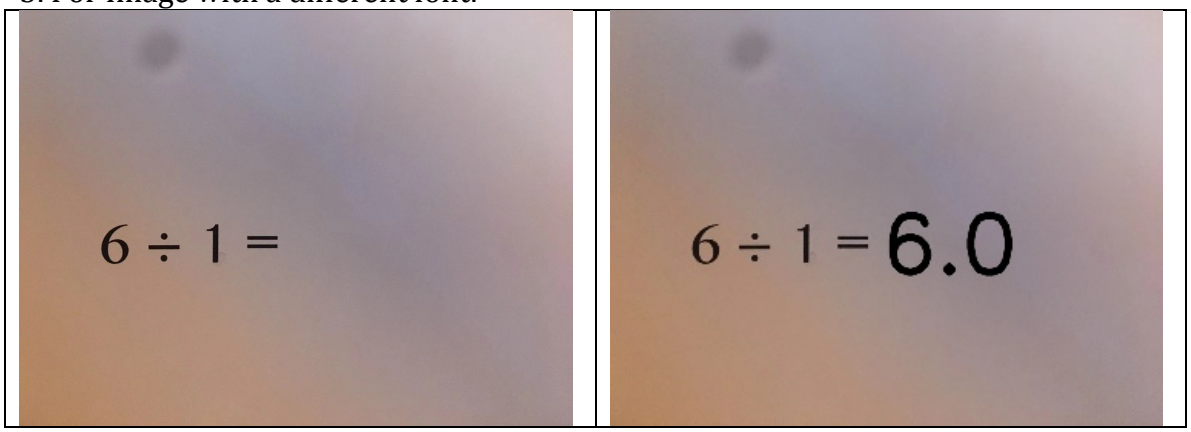
1. For image took with a good lightning



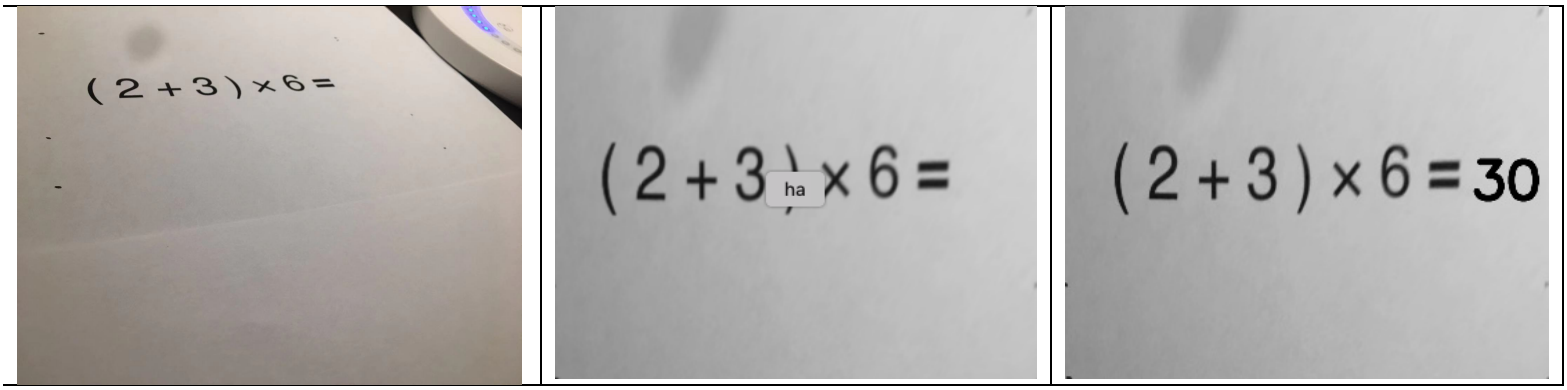
2. For image took with a bad lightning



3. For Image with a different font:



4. For oblique image



Discussion:
My method works when it successfully detect the image, and fails when it does not. I wrote a way of improve this in the conclusion section. However, the model we have now is enough for the test image I took.

In addition, if it does not successfully detect each element it will fail too. So, preprocessing the image is also very important. That's why I contour the original photo, and make transformation.

Main Challenge

To get a nearly perfect model is important in this program. If there is an mistake in prediction, then the math error will raise. At very beginning I tried to use knn to predict, but then I realized that it is not accurate enough, that's why I chose neural network.

At first the test accuracy for symbol is only 40% which is not enough. So I tried to increase the training data, cropped unnecessary white board for the training data, then it rise to 70%. After that, I adjusted the batch size, optimizer, activation function, and also number of epochs. Then it achieve 100% accuracy on test set now. Another challenge is the element detection. I thought about some model like we did

in a4, but I think it is too complex for our already processed photo. Many method required us to have a digit symbol model first. The element detection we are using now are simplest and most suitable for digit and symbol detection.

Conclusion and Improvement

Overall, the results are quite good. And the training result are quite good (it reaches 100% accuracy) In order to receive this result, I adjusted the parameter in my model, and improved my training data. I enlarged my training data size, and also cropped out unnecessary white board for each data.

I also adjusted the threshold for image processing to receive a clean image in order to crop out each element precisely.

When during investigating, I found that ")" "(" and "1" sometimes can be messed up together, so does "2" and "=". If there is a mistake, then there will be a math error, but fortunately we get a 100% accuracy. From my point of view, if the model is not perfect enough, then we can build a try catch part. If an exception raised, then for those elements that can be easily messed up, we do the prediction again, and chose the second best one.

For the oblique part, I use homography theory. However, I think there are other ways to approach it. For example, add those italic in to training data. Because oblique digits and symbols sometimes are just italic fonts.

My program cannot recognize complex math expression now, such as the square root. I think this can be done by detecting the square root first, and then do the calculation to the rectangle below the square root. We can also calculate the power. After cropped out each elements, we will check the smallest one and find out if it is a power or not, if it is, then we can do the calculation after.

We can even and unknown to the expression. We can train all the English and Greek letters, and if we detect these, recognize them as unknowns, and then solve the equation. We can used the structure we already built in the project.

We can even solve word math problem, we just need to recognize some words like "where" and "there exists", "probability".

Absolutely there are things to be improved, but we already get the basic ideas. For those improvement I mentioned above, these can be done based on what we already have.