Sudoku Project Report

518030910360 Lang Weilin

January 5, 2021

1 OpenCV Sudoku Solver and OCR

Each Sudoku puzzle begins with an NxN grid (usually 9×9), with some cells blank and others containing numbers. The purpose is to use knowledge of existing numbers to correctly infer other numbers.

1.1 Introduction

Using OpenCV to create an automatic Sudoku puzzle solver requires six steps:
Provide the input image containing Sudoku puzzle to our system.
Locate where in the input image the puzzle is and extract the board.
Given the board, locate each of the individual cells of the Sudoku board.
Determine whether there is a number in the cell, if so, OCR it.
Apply a Sudoku puzzle solver/checker algorithm to validate the puzzle.
Display the output result to the user.

1.2 Process

1.2.1 SudokuNet

```
# first set of CONV => RELU => POOL layers
model.add(Conv2D(32, (5, 5), padding="same",
    input_shape=inputShape))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

# second set of CONV => RELU => POOL layers
model.add(Conv2D(32, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

# first set of FC => RELU layers
model.add(Flatten())
model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.5))

# second set of FC => RELU layers
model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.5))

# softmax classifier
model.add(Dense(classes))
model.add(Activation("softmax"))
```

Figure 1: network

The network is composed of two sets of CONV => RELU => POOL layers, two sets of FC => RELU layers and a softmax classifier.

The first set of CONV => RELU => POOL layers is composed of 32 convolution kernels with size of 5 * 5, padding method "same". The second set of CONV => RELU => POOL layers is composed of 32 convolution kernels with size of 3 * 3, padding method "same". Activation functions are both "relu".

CONV is convolution layer. POOL is pooling layer. RELU is activation function. FC is fully connected layer.

1.2.2 Model training

Determine appropriate settings for the learning rate, number of training epochs, and batch size, 1×10^{-3}, 10 and 128.

To work with the MNIST digit dataset:

Load the dataset into memory,which is already split into training and testing data.

Add a channel dimension to the digits to indicate that they are grayscale.

Scale data to the range of [0, 1] and one-hot encode labels.

Build and compile the model with the Adam optimizer and categorical cross-entropy loss.

Then with parameters "$--modeloutput/digit\_classifier.h5$", we can train and get a model.

1.2.3 Image search

We need to locate and extract the Sudoku puzzle board from the input image and then examine each cell of the Sudoku puzzle board and extract the digit from the cell.

Determine which of the cnts is the puzzleCnt using the following approach:

Loop over all contours beginning. Determine the perimeter of the contour. Approximate the contour. Check if contour has four vertices, and if so, mark it as the puzzleCnt, and break out of the loop.

Applying a four-point perspective transform effectively deskews our Sudoku puzzle grid, making it much easier for us to determine rows, columns, and cells.

1.3 Result

With parameters "$--modeloutput/digit\_classifier.h5$", train and get a model.

```
469/469 [==============================] - 6s 12ms/step - loss: 0.6577 - accuracy: 0.7857 - val_loss: 0.0927 - val_accuracy: 0.9747
Epoch 2/10
469/469 [==============================] - 4s 8ms/step - loss: 0.2460 - accuracy: 0.9284 - val_loss: 0.0670 - val_accuracy: 0.9807
Epoch 3/10
469/469 [==============================] - 4s 9ms/step - loss: 0.1869 - accuracy: 0.9469 - val_loss: 0.0556 - val_accuracy: 0.9848
Epoch 4/10
469/469 [==============================] - 4s 9ms/step - loss: 0.1542 - accuracy: 0.9550 - val_loss: 0.0420 - val_accuracy: 0.9878
Epoch 5/10
469/469 [==============================] - 4s 9ms/step - loss: 0.1342 - accuracy: 0.9610 - val_loss: 0.0428 - val_accuracy: 0.9884
Epoch 6/10
469/469 [==============================] - 4s 9ms/step - loss: 0.1214 - accuracy: 0.9638 - val_loss: 0.0387 - val_accuracy: 0.9891
Epoch 7/10
469/469 [==============================] - 5s 10ms/step - loss: 0.1078 - accuracy: 0.9672 - val_loss: 0.0348 - val_accuracy: 0.9900
Epoch 8/10
469/469 [==============================] - 4s 8ms/step - loss: 0.1013 - accuracy: 0.9696 - val_loss: 0.0329 - val_accuracy: 0.9902
Epoch 9/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0922 - accuracy: 0.9731 - val_loss: 0.0361 - val_accuracy: 0.9904
Epoch 10/10
469/469 [==============================] - 12s 26ms/step - loss: 0.0913 - accuracy: 0.9718 - val_loss: 0.0398 - val_accuracy: 0.9891
```

Figure 2: model training

```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       980
           1       0.99      1.00      0.99      1135
           2       0.98      1.00      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.98      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       1.00      0.97      0.98       974
           9       0.99      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

[INFO] serializing digit model...

Process finished with exit code 0
```

Figure 3: model training

Set debug=1 to show intermediate steps. With parameters "$--modeloutput/digit\_classifier.h5$ $--imagesudoku\_puzzle.jpg$", we can test the model on a given image.

Figure 4: test result

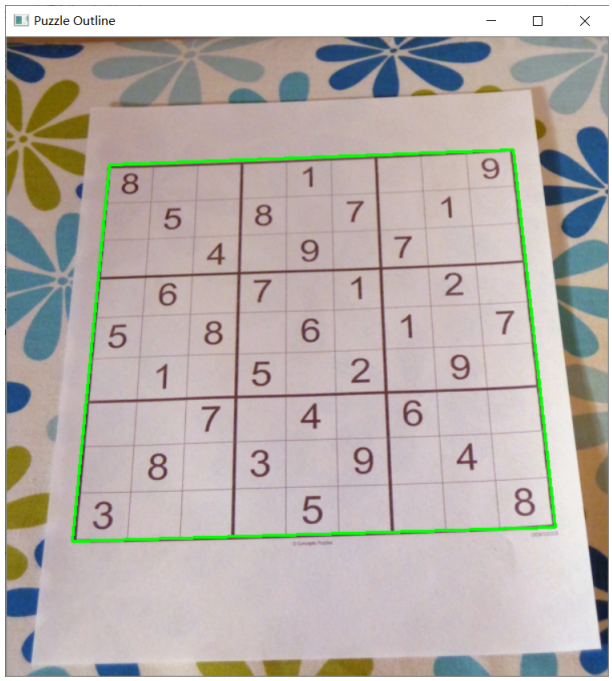


Figure 5: test result

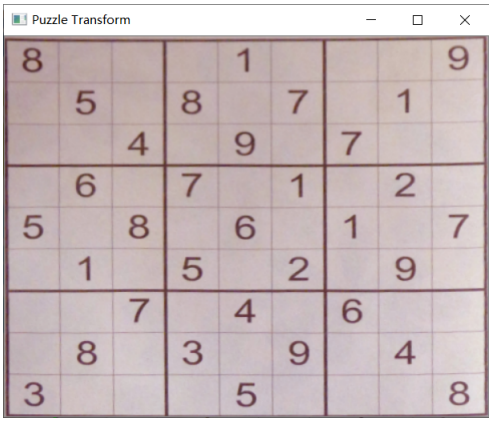


Figure 6: test result

```
[INFO] solving sudoku puzzle...

----------------------------
9x9 (3x3) SUDOKU PUZZLE
Difficulty: SOLVED
----------------------------
+-------+-------+-------+
| 8 7 2 | 4 1 3 | 5 6 9 |
| 9 5 6 | 8 2 7 | 3 1 4 |
| 1 3 4 | 6 9 5 | 7 8 2 |
+-------+-------+-------+
| 4 6 9 | 7 3 1 | 8 2 5 |
| 5 2 8 | 9 6 4 | 1 3 7 |
| 7 1 3 | 5 8 2 | 4 9 6 |
+-------+-------+-------+
| 2 9 7 | 1 4 8 | 6 5 3 |
| 6 8 5 | 3 7 9 | 2 4 1 |
| 3 4 1 | 2 5 6 | 9 7 8 |
+-------+-------+-------+
```

Figure 7: test result

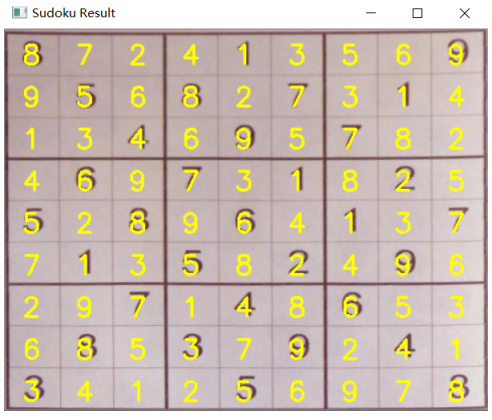


Figure 8: test result

2 CNN and model training

2.1 Convolutional Neural Network

[1]

Convolutional neural networks (CNN) is a kind of feedforward neural networks with convolution computation and depth structure. It is one of the representative algorithms of deep learning. Convolutional neural network has the ability of representation learning, and can shift invariant classify the input information according to its hierarchical structure, so it is also called shift invariant artificial neural networks (SIANN).

2.2 Process

The model we design is like this. For different models, the number of convolution kernels or the size of convolution kernels will be different.

The network is composed of three sets of CONV => RELU => POOL layers, one sets of FC => RELU layers and a softmax classifier.

The first set of CONV => RELU => POOL layers is composed of 6 convolution kernels with size of 5 * 5, padding method "same". The second set of CONV => RELU => POOL layers is composed of 32 convolution kernels with size of 3 * 3, padding method "valid". The third set of CONV => RELU => POOL layers is composed of 120 convolution kernels with size of 2 * 2, padding method "valid". Activation functions are all "relu".

```
# first set of CONV => RELU => POOL layers
model.add(Conv2D(6, (5, 5), padding='same',
    input_shape=inputShape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# second set of CONV => RELU => POOL layers
model.add(Conv2D(32, (3, 3), padding='valid'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# conv layer3
model.add(Conv2D(120, (2, 2), padding='valid'))
model.add(Activation('relu'))

# first set of FC => RELU layers
model.add(Flatten())
model.add(Dense(84))
model.add(Activation('relu'))
# model.add(Dropout(0.5))
```

Figure 9: model

In total, we train 7 models that differ in convolution layers or learning rate.

	learning rate	epochs	batch size	layer2 number	layer3 size
mm1	0.0015	20	128	32	2*2
mm2	0.0015	20	128	32	5*5
mm3	0.002	20	64	16	2*2
mo1	0.001	20	128	32	5*5
mo2	0.002	20	128	32	5*5
mo3	0.001	20	128	32	2*2
mo4	0.0015	20	128	32	2*2

The logs of these models while training are saved in the folder. The we draw curves according to the saved data.

2.3 Curve

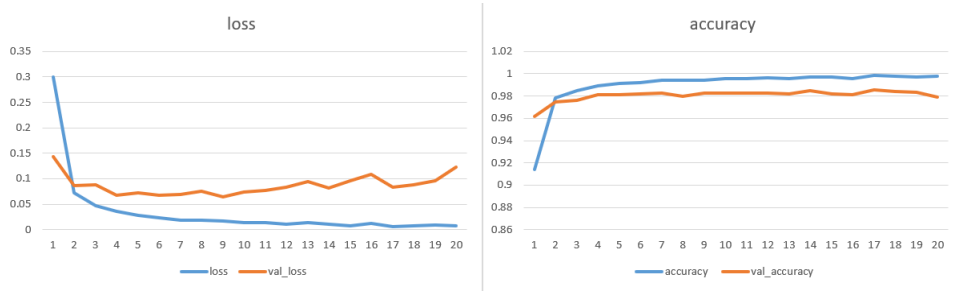


Figure 10: mm1 train curve

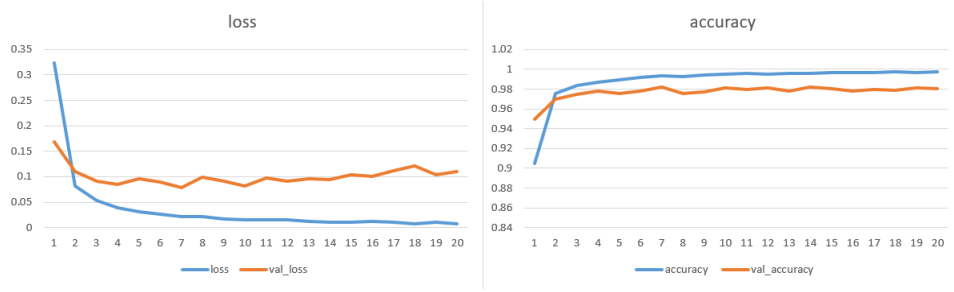


Figure 11: mm2 train curve

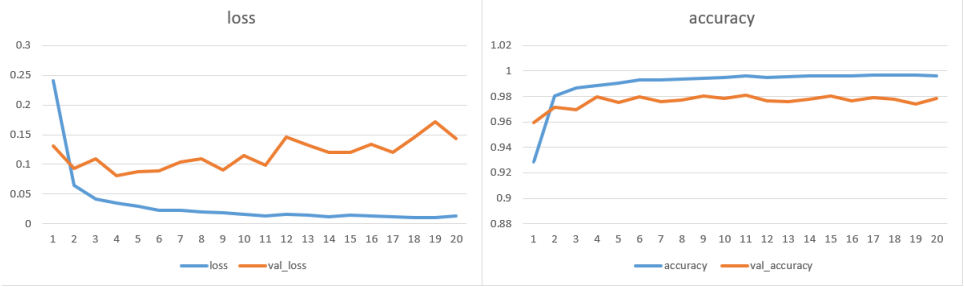


Figure 12: mm3 train curve

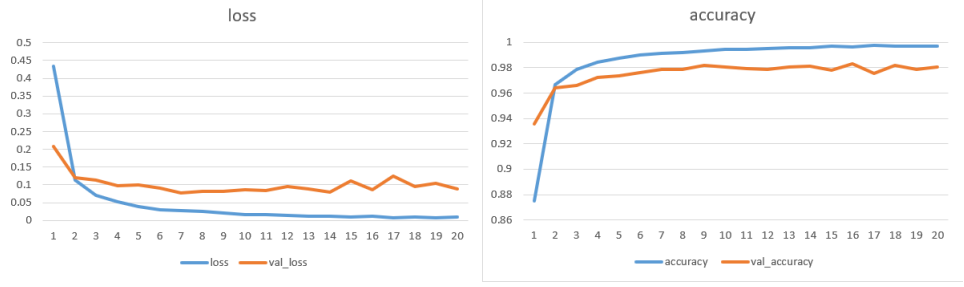


Figure 13: mo1 train curve

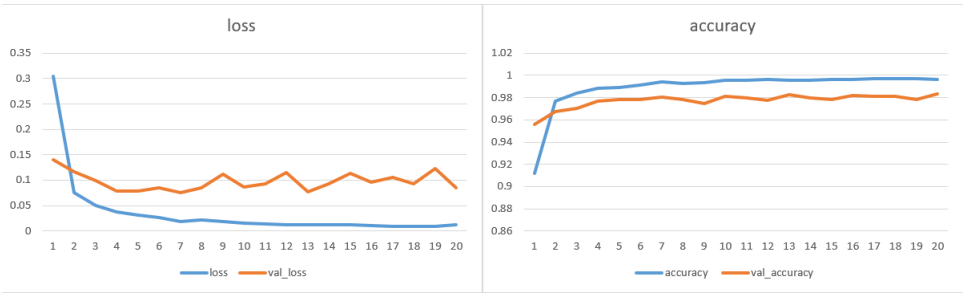


Figure 14: mo2 train curve

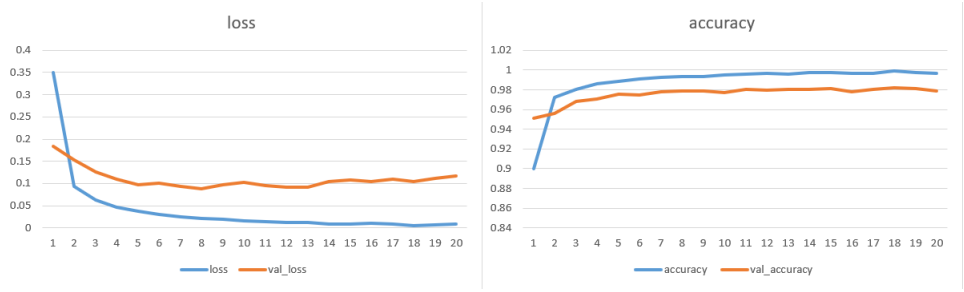


Figure 15: mo3 train curve

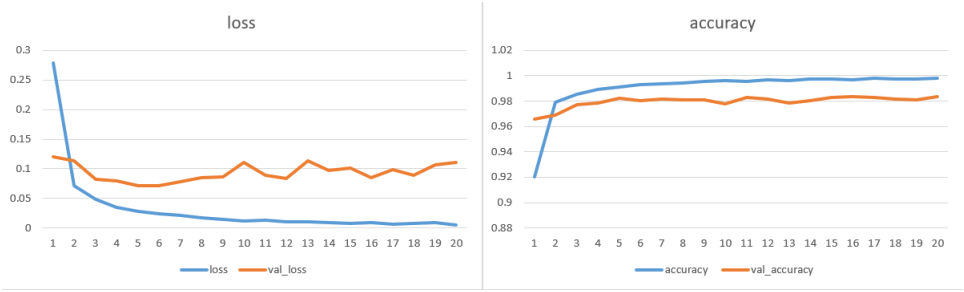


Figure 16: mo4 train curve

All the data and the curves can be checked in the "data_and_curve" folder saved in xlsx files.

2.4 Result

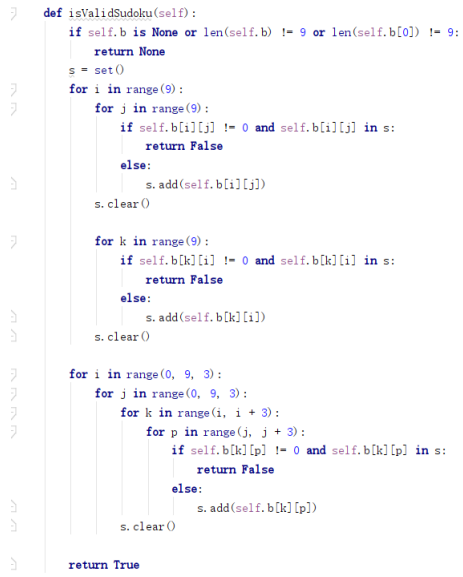
According to the data saved, the train accuracy of these models is over 0.995. The validation accuracy is over 0.97, occasionally over 0.98, which is much lower than the train accuracy. The final recognition accuracy is 0.97 or 0.98.

The selected models have been adjusted, and the results are not very different from each other. In the final test, one of the models will be selected as a representative.

3 Sudoku Solver

3.1 Solve Algorithm

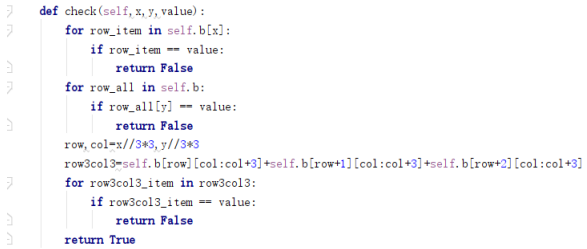
To solve the sudoku, we choose to use backtracking algorithm.



```
def isValidSudoku(self):
    if self.b is None or len(self.b) != 9 or len(self.b[0]) != 9:
        return None
    s = set()
    for i in range(9):
        for j in range(9):
            if self.b[i][j] != 0 and self.b[i][j] in s:
                return False
            else:
                s.add(self.b[i][j])
        s.clear()

        for k in range(9):
            if self.b[k][i] != 0 and self.b[k][i] in s:
                return False
            else:
                s.add(self.b[k][i])
        s.clear()

    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            for k in range(i, i + 3):
                for p in range(j, j + 3):
                    if self.b[k][p] != 0 and self.b[k][p] in s:
                        return False
                    else:
                        s.add(self.b[k][p])
            s.clear()

    return True
```

Figure 17: valid

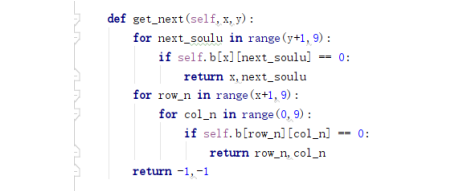
When a sudoku is input, the function isValidSudoku will determine whether the sudoku is valid.



```
def check(self, x, y, value):
    for row_item in self.b[x]:
        if row_item == value:
            return False
    for row_all in self.b:
        if row_all[y] == value:
            return False
    row, col=x//3*3, y//3*3
    row3col3=self.b[row][col:col+3]+self.b[row+1][col:col+3]+self.b[row+2][col:col+3]
    for row3col3_item in row3col3:
        if row3col3_item == value:
            return False
    return True
```

Figure 18: check

The function check is to check whether there are the same items in each row, column and block when the program is filling in a digit.



```
def get_next(self, x, y):
    for next_soulu in range(y+1, 9):
        if self.b[x][next_soulu] == 0:
            return x, next_soulu
    for row_n in range(x+1, 9):
        for col_n in range(0, 9):
            if self.b[row_n][col_n] == 0:
                return row_n, col_n
    return -1, -1
```

Figure 19: get next

The function get_next is to get the next open item.

```
def try_it(self,x,y):
    if self.b[x][y] == 0:
        for i in range(1,10):
            self.t+=1
            if self.check(x,y,i):
                self.b[x][y]=i
                next_x,next_y=self.get_next(x,y)
                if next_x == -1:
                    return True
                else:
                    end=self.try_it(next_x,next_y)
                    if not end:
                        self.b[x][y] = 0
                    else:
                        return True
```

Figure 20: try_it

The function try_it is the main function. Try from 1 to 9, fill in the 0 cell with the qualified ones. Get the next 0 cell, if no 0 cell, return True. If there is a next 0 cell, recursively judge the next 0 cell until the Sudoku is filled. In the process of recursion, if there are any unqualified ones, we can backtrack to the previous level and continue.

3.2 Solve with model

```
def extract_digit(cell, debug=True):
    if debug:
        cv2.imshow("Original", cell)
        cv2.waitKey(0)
    # apply automatic thresholding to the cell and then clear any
    # connected borders that touch the border of the cell
    cell=cv2.resize(cell,(28,28))
    #thresh = cv2.threshold(cell, 160, 255,cv2.THRESH_BINARY_INV )[1]
    thresh = cv2.threshold(cell, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
```

Figure 21: extract_digit

```
    if debug:
        cv2.imshow("Cleared", thresh)
        cv2.waitKey(0)

    kernel = np.ones((2, 2), np.uint8)
    thresh = cv2.dilate(thresh, kernel=kernel)
    if debug:
        cv2.imshow("Dilated", thresh)
        cv2.waitKey(0)

    (h, w) = thresh.shape
    percentFilled = cv2.countNonZero(thresh) / float(w * h)
    if percentFilled < 0.001:
        return None

    # check to see if we should visualize the masking step
    if debug:
        cv2.imshow("Digit", thresh)
        cv2.waitKey(0)

    return thresh
```

Figure 22: extract_digit

In order to recognize Chinese numbers better, the function extract_digit needs to be improved.

```
print("[INFO] solving sudoku puzzle...")
puz=board.tolist()
prob=copy.deepcopy(puz)
pu=solution(puz)
sol=pu.start()
print(sol)
```

Figure 23: call function

When solving the sudoku, we call our solve function like this.

4 Test

```
image_path="2-1.jpg"
model_path="output/mm2.h5"
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=False,
    help="path to trained digit classifier")
ap.add_argument("-i", "--image", required=False,
    help="path to input sudoku puzzle image")
ap.add_argument("-d", "--debug", type=int, default=1,
    help="whether or not we are visualizing each step of the pipeline")
args = vars(ap.parse_args())

# load the digit classifier from disk
print("[INFO] loading digit classifier...")
#model = load_model(args["model"])
model = load_model(model_path)

# load the input image from disk and resize it
print("[INFO] processing image...")
#image = cv2.imread(args["image"])
image = cv2.imread(image_path)
image = imutils.resize(image, width=600)
```

Figure 24: debug

When testing, we set debug=1, choose the model mm2.h5 and record the test results.

4.1 Result

4.1.1 Arabic numerals

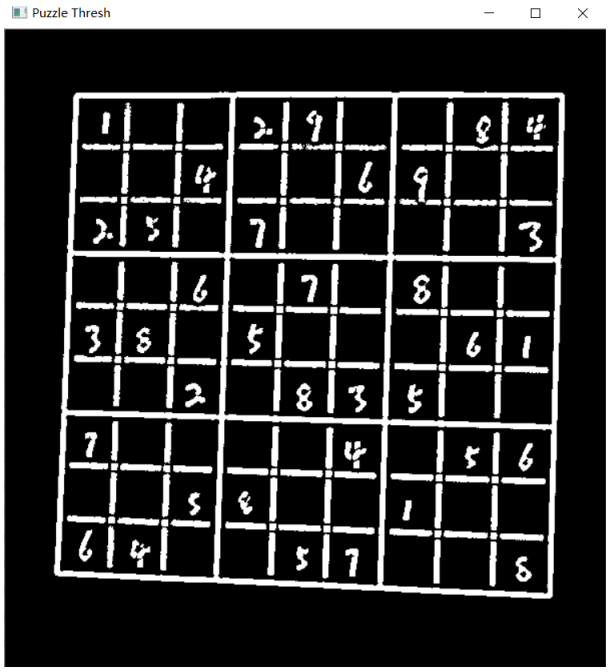


Figure 25: 1-1

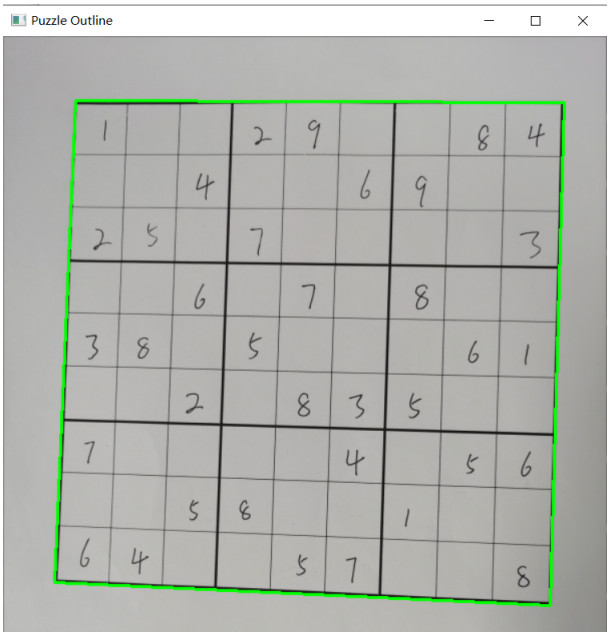


Figure 26: 1-1

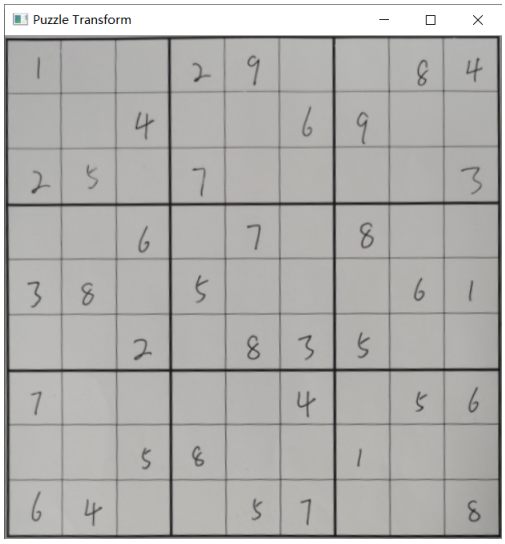


Figure 27: 1-1

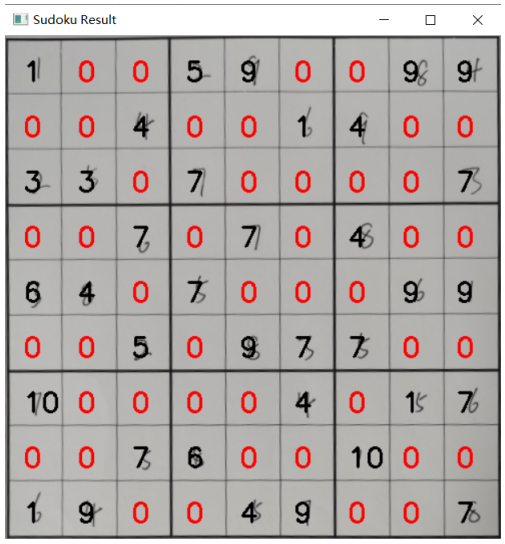


Figure 28: 1-1

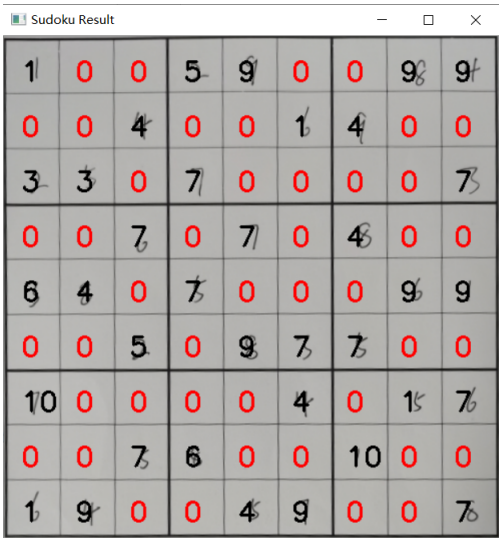


Figure 29: 1-1

Figure 30: 1-2

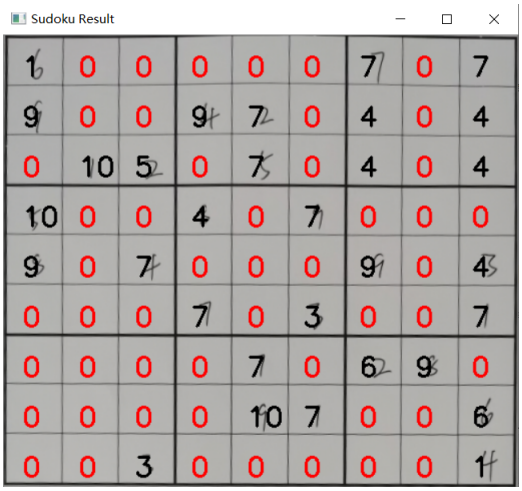


Figure 31: 1-3

Figure 32: 1-4

Figure 33: 1-5

We can see that the test result for Arabic numerals with our model is not good. The recognition accuracy is very low. And a prominent problem is that the model prefers to recognize a Arabic numeral as a Chinese numeral ten.

All the results can be checked in the "result" folder.

4.1.2 Chinese numerals

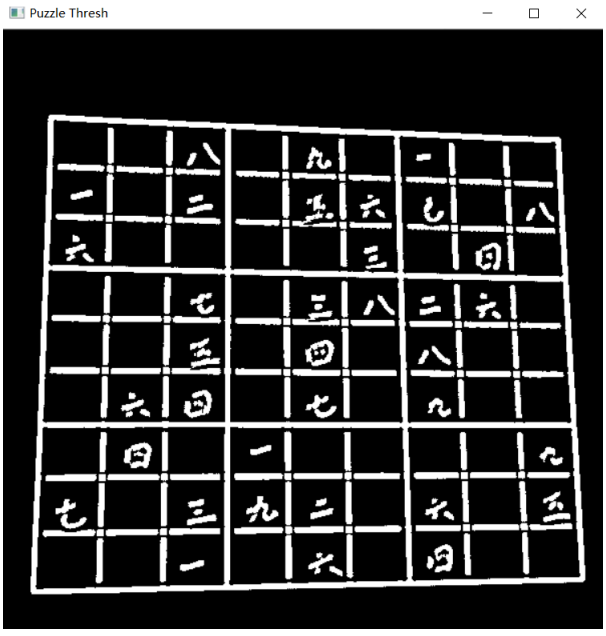


Figure 34: 2-1

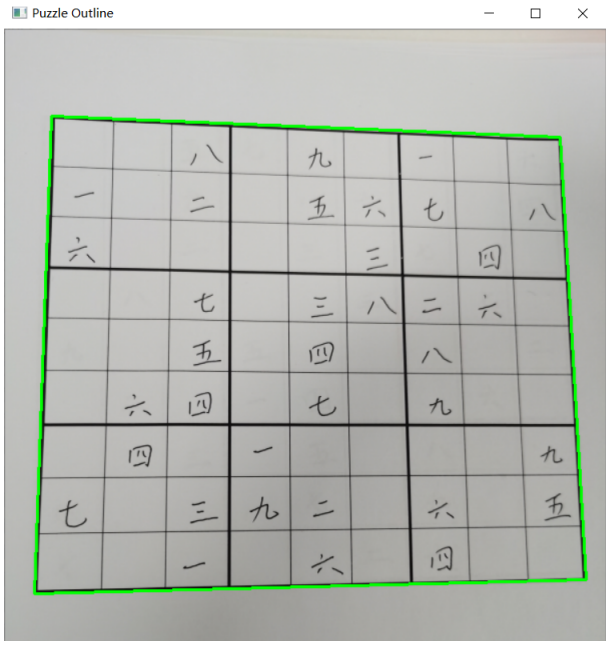


Figure 35: 2-1

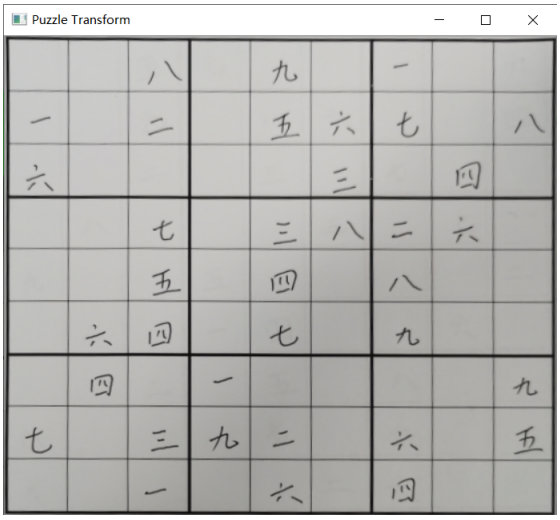


Figure 36: 2-1

Figure 37: 2-1

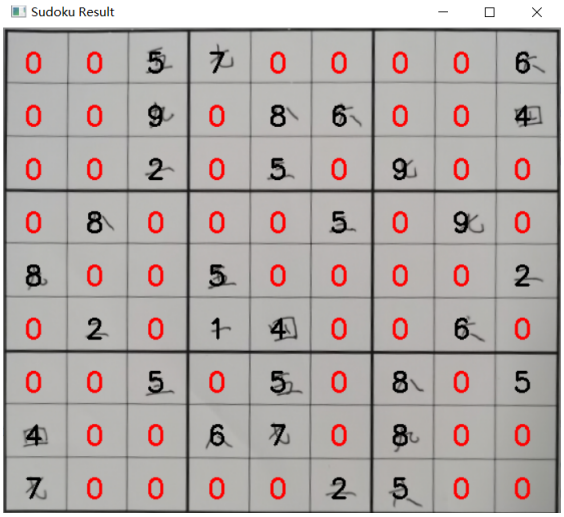


Figure 38: 2-2

Figure 39: 2-3

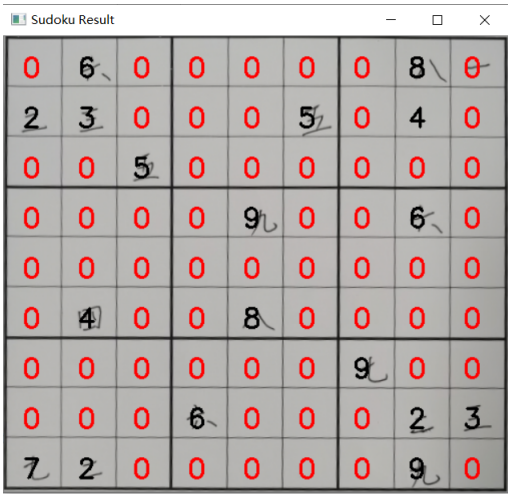


Figure 40: 2-4

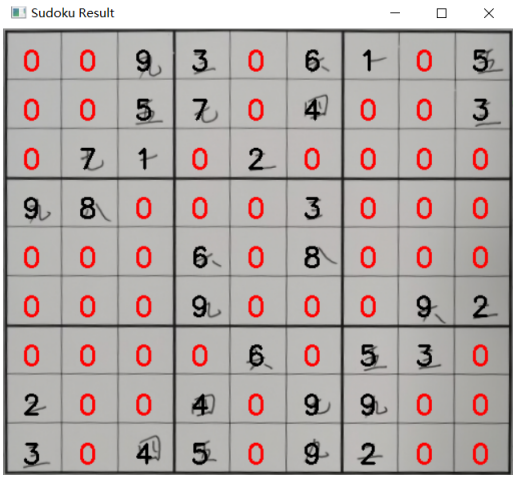


Figure 41: 2-5

For image 2-4, we set the threshold of percentFilled to 0.05(default 0.001). The shadow of different images is different. So change this value may improve the result. By testing, 0.001 is appropriate enough for other images. Image 2-4 is a little special.

Compared with Arabic numerals, The results of Chinese numerals are very delightful. There are only one or two mistakes for the recognition result of each image. And we find that the mistakes are mainly concentrated in regarding Chinese seven as Chinese nine. This is the problem of this model. It often causes repetitive results of nine. So the program can not solve the sudoku correctly.

Luckily, for image 2-1, it just ignores a Chinese numeral one in the upper left corner block. Other numerals are all correct. So the program solve the sudoku successfully. This proves that our sudoku solving algorithm is correct.

4.2 Improvement

```python
def extract_digit(cell, debug=True):
    if debug:
        cv2.namedWindow('Original', cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO)
        cv2.imshow("Original", cell)
        cv2.waitKey(0)
    # apply automatic thresholding to the cell and then clear any
    # connected borders that touch the border of the cell
    cell=cv2.resize(cell,(28,28))
    blurred = cv2.GaussianBlur(cell, (3, 3),0,0)
    # apply adaptive thresholding and then invert the threshold map
    thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
    thresh = cv2.bitwise_not(thresh)
```

Figure 42: extract_digit

```python
    # check to see if we are visualizing the cell thresholding step
    if debug:
        cv2.namedWindow('Cleared', cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO)
        cv2.imshow("Cleared", thresh)
        cv2.waitKey(0)

    (h, w) = thresh.shape
    percentFilled = cv2.countNonZero(thresh) / float(w * h)
    if percentFilled<=0.03:
        return None

    # check to see if we should visualize the masking step
    if debug:
        cv2.namedWindow('Digit', cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO)
        cv2.imshow("Digit", thresh)
        cv2.waitKey(0)

    return thresh
```

Figure 43: extract_digit

To improve the result, we use a different extract_digit function. This will make the results of Chinese numerals a little better.

4.2.1 Result

Figure 44: 2-1

Figure 45: 2-2

Figure 46: 2-3

Figure 47: 2-4

Figure 48: 2-5

Happily, for image 2-1, our recognition and solving result can be absolutely right.

We can see that, this new extract_digit function can determine whether there exist a digit better than before. However, since our model is not improved, it still regard Chinese seven as Chinese nine easily. This causes that the program can not solve the sudoku. For image 2-4, it recognizes a different sudoku and solve that sudoku successfully.

Nevertheless, this is exactly an improvement.

5 Summary

As far as this project is concerned, the model obtained by mixing Arabic numerals and Chinese numerals for training is relatively poor in recognizing Arabic numerals, which may be affected by various factors.

In fact, our goal of recognizing Chinese numbers has basically been achieved.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.