TEAM_1: FRANK & JACOB

Homework_1: Question 4 Solution

The aim of this solution is to demonstrate character recognition (alphabets and numbers) on Python script using KNN algorithm with Open CV. The steps are as follows:

A. CLASSIFICATION STEPS

Step 1:

i. We generate our training characters from an input image using the command (line 16, in the file GenData.py):

```
imgTrainingNumbers = cv2.imread("training_chars.png")
```

ii. Next, we filter the input image from grayscale to black and white using the command (lines 28-33):

```
imgThresh = cv2.adaptiveThreshold(imgBlurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
```

iii. Then we show threshold image for reference (as in Fig. 4.1) using the command (line 35): cv2.imshow("imgThresh", imgThresh)

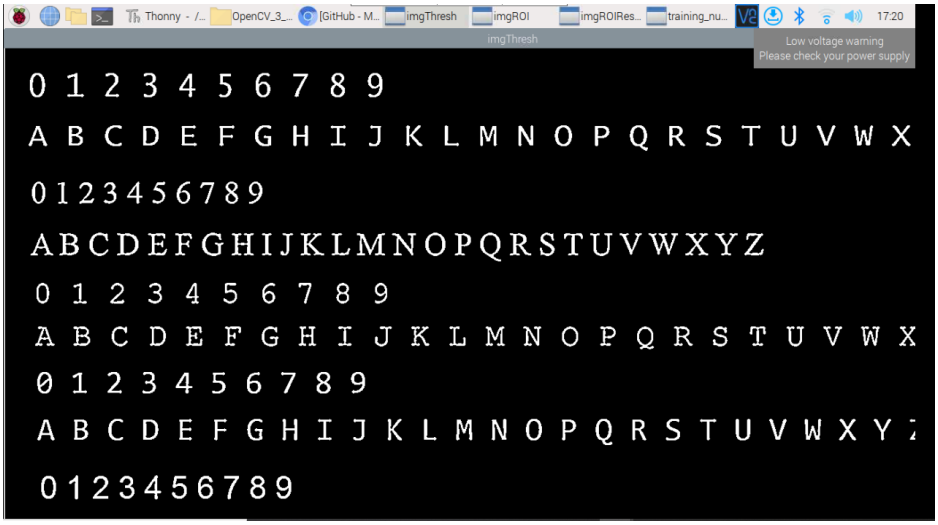


Fig. 4.1. Image threshold

Step 2:

i. We make a copy of the threshold image using the command (line 37):

```
imgThreshCopy = imgThresh.copy()
```

ii. We find contours using Open CV (lines 39-41):
npaContours, npaHierarchy = cv2.findContours(imgThreshCopy, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

iii. We classify our characters by manually inputting the characters from our keyboard, using the command (line 47):

intClassifications = []

iv. Next, the program draws rectangle around each contour as it asks user for input (see Fig. 4.2) – (lines 60 – 64):

cv2.rectangle(imgTrainingNumbers, (intX, intY), (intX+intW,intY+intH), (0, 0, 255), 2)

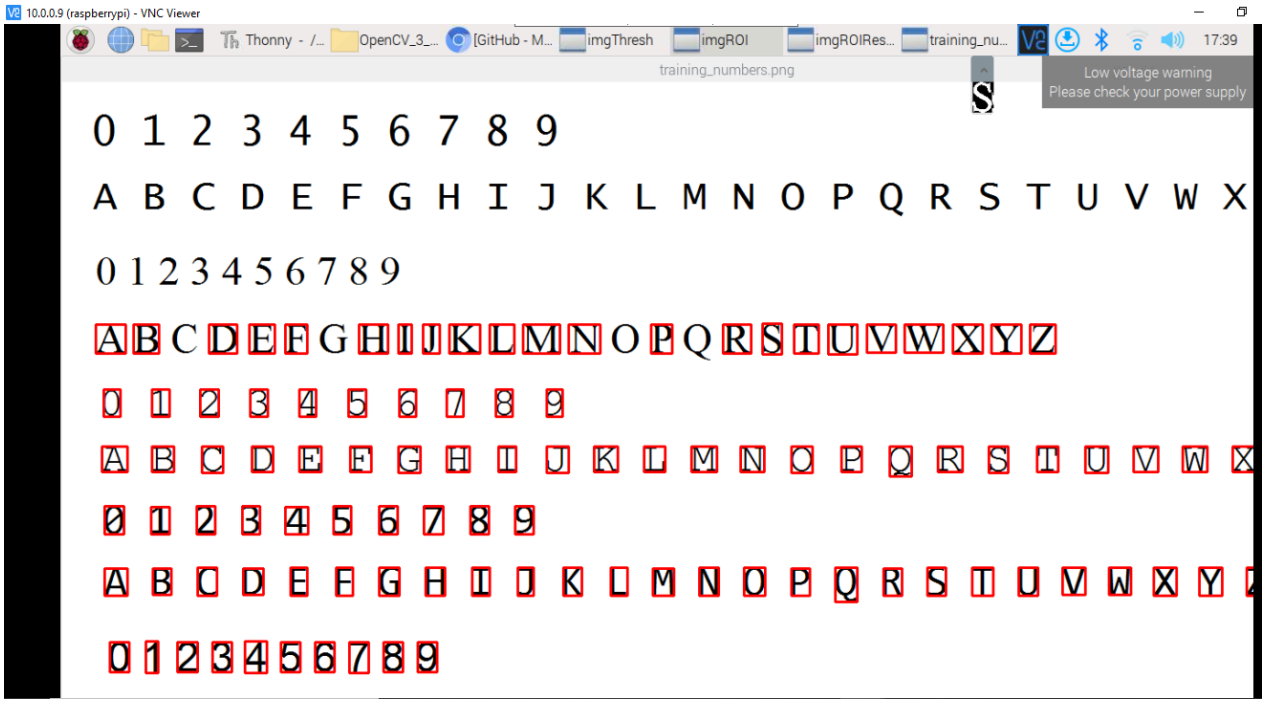


Fig. 4.2. Image classification inputs (with red rectangles)

Step 3:

i. We append classification characters to integer list of characters (we will convert to float later before writing to file – line 79):

intClassifications.append(intChar)

ii. We convert classifications list of integers into numpy array of floats, and write to file (lines 87-94):

```
fltClassifications = np.array(intClassifications, np.float32)
npaClassifications = fltClassifications.reshape((fltClassifications.size, 1))
print ("\n\ntraining complete !!\n")

   np.savetxt("classifications.txt", npaClassifications)
   np.savetxt("flattened_images.txt", npaFlattenedImages)
```

B. TRAINING & TESTING STEPS

The Classification Steps above generate the files "classifications.txt" and "flattened_images.txt" which will be used in the training and testing.

We train and test using the Python file "TrainAndTest.py" as follows:

Step 1: We read the training classifications and flattened images using the command (as in lines 43-55):

```
npaClassifications = np.loadtxt("classifications.txt", np.float32)

npaFlattenedImages = np.loadtxt("flattened_images.txt", np.float32)
```

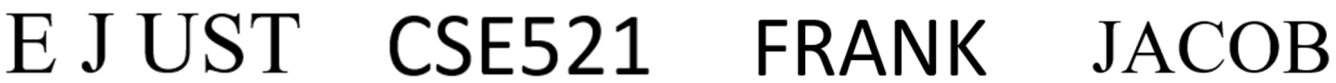
Step 2: We reshape numpy array to one dimension, and initiate the KNN object (lines 58 – 62):

```
npaClassifications = npaClassifications.reshape((npaClassifications.size, 1

kNearest = cv2.ml.KNearest_create()

kNearest.train(npaFlattenedImages, cv2.ml.ROW_SAMPLE, npaClassifications)
```

Step 3:

(i.) We call in our testing data by importing some images containing characters which we would like to recognize (using the command in line 64). Fig. 4.3 shows the testing images.

```
imgTestingNumbers = cv2.imread("test2.png")
```



(a) test1.png (b) test2.png (c) test3.png (d) test4.png

Fig. 4.3. Testing images

(ii.) The image is filtered (lines 76 – 81) and contoured (lines 85 – 106).

(iii). The program draws a green rectangle around each character for each contoured image (lines 108-117), then the image is resized using Open CV for better recognition (lines 119-123) as shown in Fig. 4.4.

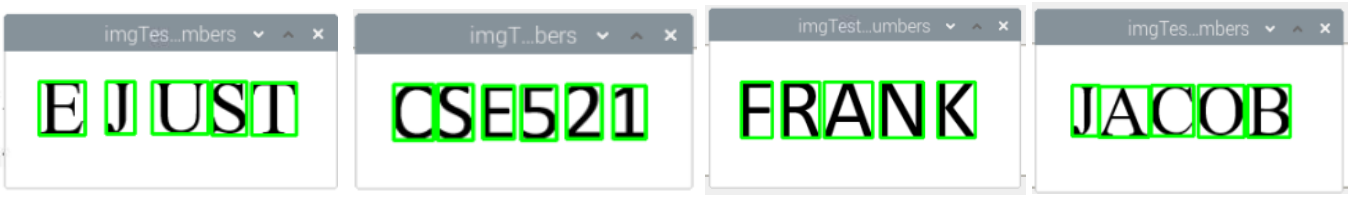


Fig. 4.4. Contoured and resized test images with green rectangles

Step 4:

(i.) We call the KNN function again to find the 1-NN (i.e. k = 1) using the command (line 125):

retval, npaResults, neigh_resp, dists = kNearest.findNearest(npaROIResized, k = 1)

(ii.) The character is then predicted from the results using the commands in lines 127 – 129.

(iii.) We print the predicted character using the command (line 132):

print ("\n" + strFinalString + "\n")

Fig. 4.5 shows the accurately predicted characters.

Fig. 4.5. Predicted characters with KNN algorithm

Conclusion

The solution presented above has been able to demonstrate the use of KNN Algorithm in Python using Open CV library for image recognition (in this case, character recognition). The results show that KNN is a good tool for image recognition tasks. It works best when the test dimension is small (such as 1-d, 2-d) as in this solution.