Report 5 - Chinese plate recognition problem

Jack 1930026143

Basic working flow

1. Load the image data

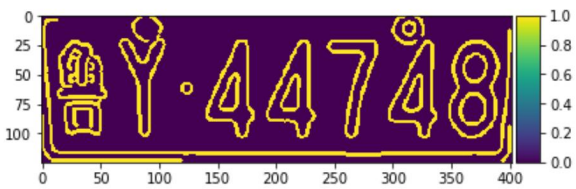
```
In [3]: #img = io.imread('./carplate/粤B5PQ23.png')  #ok
        #img = io.imread('./carplate/粤OT9048.png')  #ok
        #img = io.imread('./carplate/青AE8636.png')  #error
        img = io.imread('./carplate/鲁Y44748.png') #error

In [4]: io.imshow(img)

Out[4]: <matplotlib.image.AxesImage at 0x2204cd63220>
```

2. Transform to Grey Image, we can use the machine learning package $skimage.color$ and transform directly. And we can find that the shape of the image is from 3 dimension (length * width * RGB channel) to 2 dimension.

```
In [11]: img2 = color.rgb2gray(img)
         img2.shapepe

<ipython-input-11-cf7792eadf57>:1: FutureWarning: Non RGB image conversion is now deprecated. For RGBA images, please use rgb2g
ray(rgba2rgb(rgb)) instead. In version 0.19, a ValueError will be raised if input image last dimension length is not 3.
  img2 = color.rgb2gray(img)

Out[11]: (125, 402)

In [12]: io.imshow(img2)

Out[12]: <matplotlib.image.AxesImage at 0x2205142bc40>
```

3. Apply Canny Edge Detection and Dilation. Firstly, we use the $canny()$ method to find the edge and use an useful package method $\boldsymbol{morphology.dilation()}$ to make the edge more obvious.

```
In [17]: img4 = morphology.dilation(img3)

In [18]: io.imshow(img4)

Out[18]: <matplotlib.image.AxesImage at 0x22051212dc0>
```



4. Label and Region Proposal. Firstly find the labeled connected regions of an array by the $label()$ method of $skimage.measure$. Then in order to get the information and the properties of this region, we should use the $regionprops()$. Print the length of the regions we know that the image has 22 labeled regions.

```
In [19]: # Label connected regions of an integer array.
         label_img = measure.label(img4)
         # Measure properties of labeled image regions.
         regions = measure.regionprops(label_img)
```

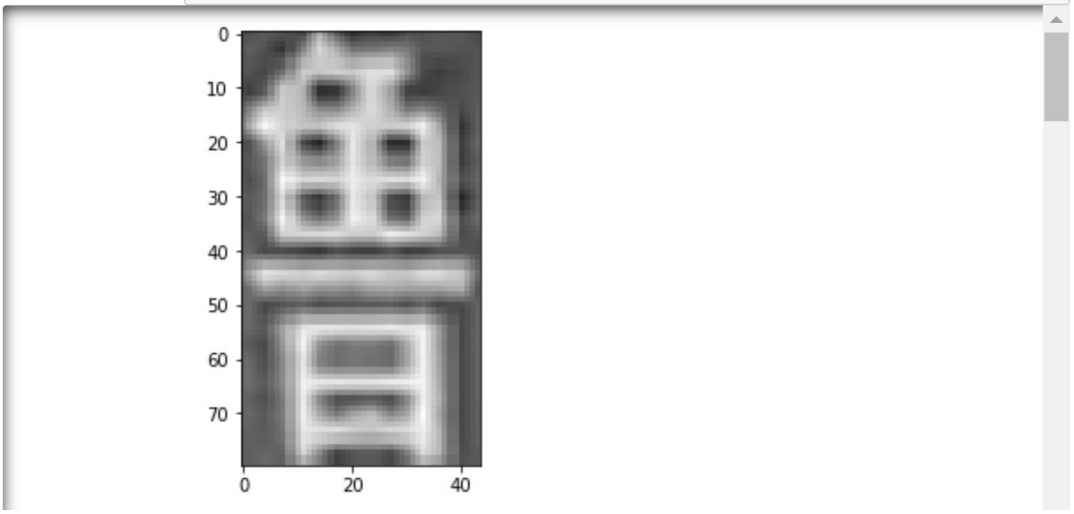
```
In [25]: len(regions)  # There are 22 labeled areas
```

```
Out[25]: 22
```

Then, we should filtrate these regions. For each labeled image area, if it is out of the limitation the scope of the segmented region, or it go beyond the border, we should remover the region. In addition, There are two circular identification knobs on the license plate which may be recognized into the region, we should also remove it. After that, the left regions are the features we want to obtain.

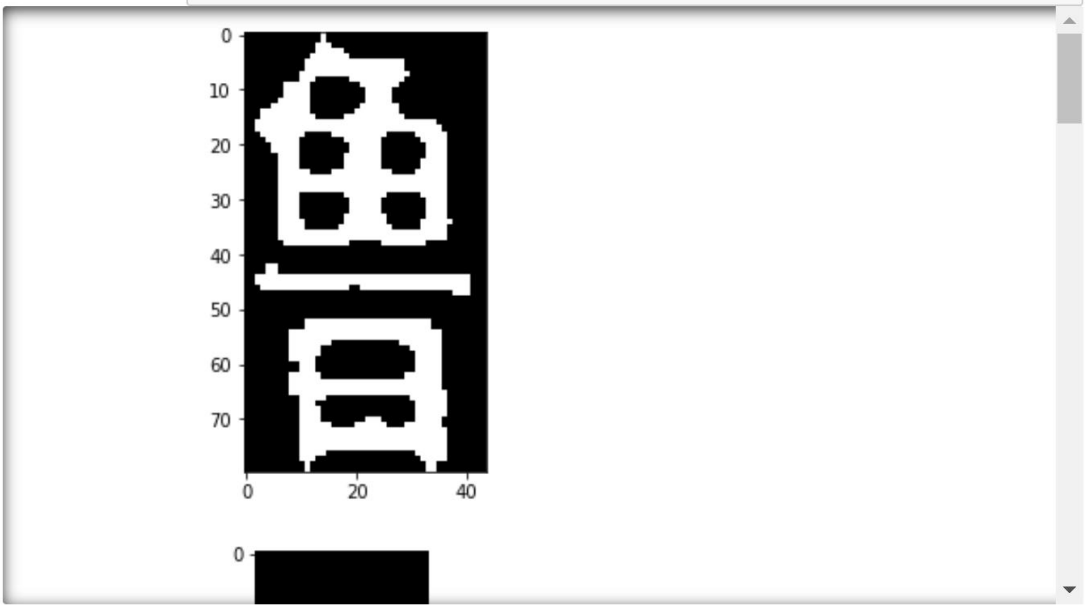
5. Retrieve Each Character Image. By the coordinate of the four corners of filtered area, we can get each image.

```
In [28]: chars = []
         for bbox in bboxes:
             minr, minc, maxr, maxc = bbox
             ch = img2[minr:maxr, minc:maxc]
             chars.append(ch)
             io.imshow(ch)
             plt.show()
```

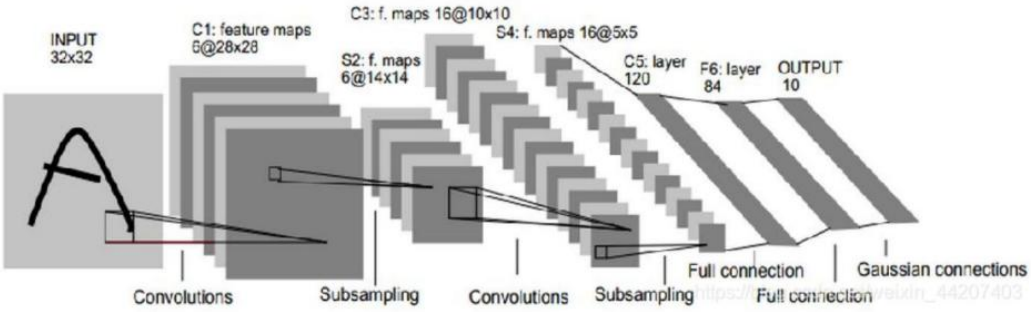


6. Image Enhancement. In order to improve the accuracy and convergence speed of the model, we can input the image data of the model for enhancement, which means that make the features of the image more prominent. We use the cv2 package to normalize and scalarize the data to implement.

```
In [38]: chars2 = []
         for bbox in bboxes:
             minr, minc, maxr, maxc = bbox
             ch = rs[minr:maxr, minc:maxc]
             chars2.append(ch)
             io.imshow(ch)
             plt.show()
```



7. Character Recognition. Firstly, we should build the model. We choose the Convolution Neuro Network (cnn) model. it is very suitable to deal with the images data and it is very similar to the $LeNet$ model.



We can find that it has two convolution-pooling layers and some full connection layers. After training the model we can get the possibility of each label, which means that we can finish the images classification. After deploying the model by the $Sequential()$ method in $tensorflow$, we just compile it and get the initial parameters. Here we do not train by this model again and just load the parameter weights by from the local disk.

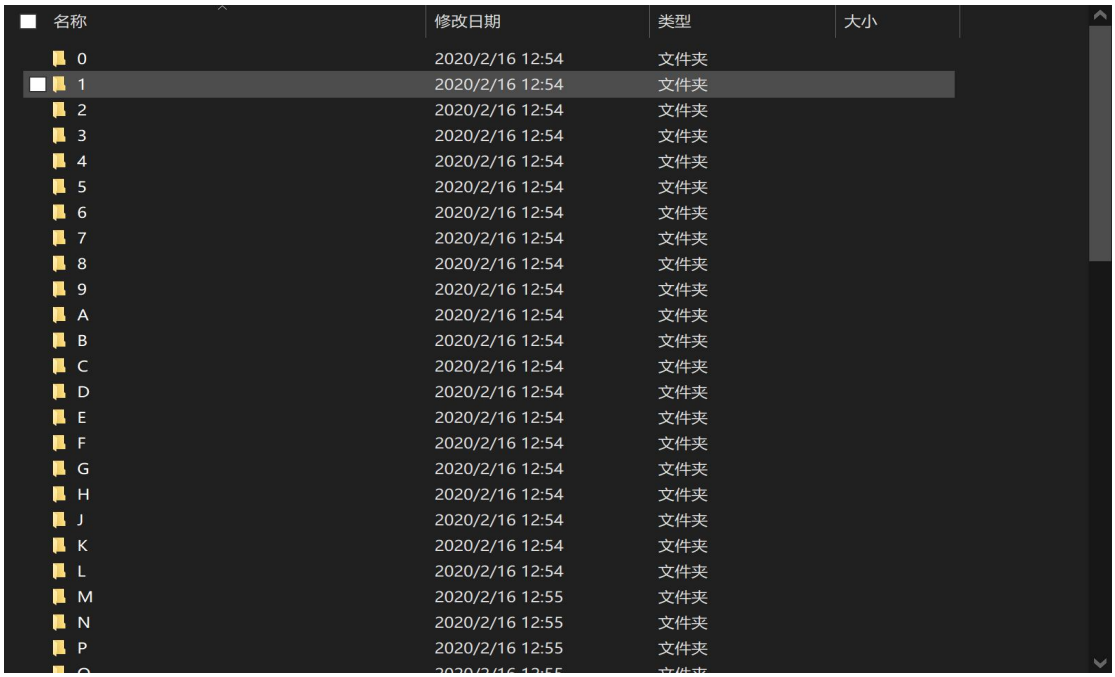
After that, we can task the model and print out the predicted result.

In [44]:
```
ys = np.unique(classes)

#p_test = model_char.predict_classes(extend_channel(chars2))
predict_x=model_char.predict(extend_channel(chars2))
p_test=np.argmax(predict_x,axis=1)

print(' '.join([ys[p_test[i]] for i in range(len(p_test))]))
```

```
鲁 9 4 4 7 4 8
```

Get the result by train the model.

1. Load character data from the dataset subfolder. Under the dataset path, there are many subfolders which includes the images dataset of the folds name. The image is the X in the model and the name of the folder is label of the image.



名称	修改日期	类型	大小
0	2020/2/16 12:54	文件夹	
1	2020/2/16 12:54	文件夹	
2	2020/2/16 12:54	文件夹	
3	2020/2/16 12:54	文件夹	
4	2020/2/16 12:54	文件夹	
5	2020/2/16 12:54	文件夹	
6	2020/2/16 12:54	文件夹	
7	2020/2/16 12:54	文件夹	
8	2020/2/16 12:54	文件夹	
9	2020/2/16 12:54	文件夹	
A	2020/2/16 12:54	文件夹	
B	2020/2/16 12:54	文件夹	
C	2020/2/16 12:54	文件夹	
D	2020/2/16 12:54	文件夹	
E	2020/2/16 12:54	文件夹	
F	2020/2/16 12:54	文件夹	
G	2020/2/16 12:54	文件夹	
H	2020/2/16 12:54	文件夹	
J	2020/2/16 12:54	文件夹	
K	2020/2/16 12:54	文件夹	
L	2020/2/16 12:54	文件夹	
M	2020/2/16 12:55	文件夹	
N	2020/2/16 12:55	文件夹	
P	2020/2/16 12:55	文件夹	

```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C',
'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R',
'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '云', '京', '冀', '吉',
'宁', '川', '新', '晋', '桂', '沪', '津', '浙', '渝', '湘', '琼',
'甘', '皖', '粤', '苏', '蒙', '藏', '豫', '贵', '赣', '辽', '鄂',
'闽', '陕', '青', '鲁', '黑']
```

2. After processing the image just same as the above (GRB to gray, Edge Detection and Dilation) , we should divide the data to the training set and the validation set.

3. Decide a few parameters for training and extend the data channels to 4 dimensions

```
In [79]: # you decide those parameters below
         batch_size = 128
         num_classes = len(dic)
         epochs = 10

         ## input image dimensions
         img_rows, img_cols = 20, 20
```

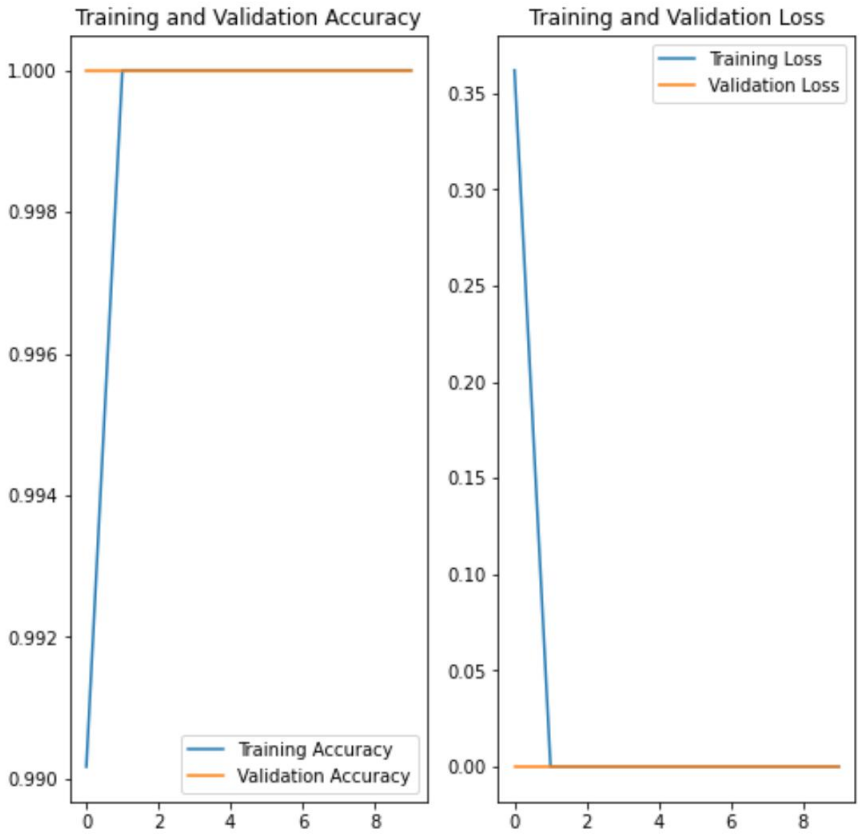
4. Before training the model, we should do the data normalization to help our model to convergence. The model is the CNN model which is introduced before. Compile the model and print the detail by the $summary()$ method.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 18, 18, 32)        320

 conv2d_13 (Conv2D)          (None, 16, 16, 64)        18496

 max_pooling2d_6 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_12 (Dropout)        (None, 8, 8, 64)          0

 flatten_6 (Flatten)         (None, 4096)              0

 dense_12 (Dense)            (None, 128)               524416

 dropout_13 (Dropout)        (None, 128)               0

 dense_13 (Dense)            (None, 65)                8385

=================================================================
Total params: 551,617
Trainable params: 551,617
Non-trainable params: 0
_________________________________________________________________
```

5. Train the model by using the test data and training data. We can find that at the second stage, the loss has gone down to zero

```
In [109]: idx = int(0.8 * len(x_train))
          train_data = x_train[:idx, :, :]
          train_label = y_train[:idx]
          test_data = x_train[idx:, :, :]
          test_label = y_train[idx:]
          pred = model.fit(x_train, y_train, batch_size=batch_size, epochs=ep

          Epoch 1/10
          101/101 [==============================] - 13s 116ms/step - loss:
          0.3646 - accuracy: 0.9903 - val_loss: 0.0000e+00 - val_accuracy:
          1.0000
          Epoch 2/10
          101/101 [==============================] - 12s 119ms/step - loss:
          0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accurac
          y: 1.0000
          Epoch 3/10
          101/101 [==============================] - 11s 112ms/step - loss:
          0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accurac
```

6. Visualize training results, we can see the accuracy and loss of the process of training:



7. Evaluate the model with test dataset. The model is good enough to be 100 percent accurate for the test set. So we don't have to fine-tune the model or deal with the overfitting.

```
In [103]: score = model.evaluate(test_data, test_label)
          print('Test loss:', score[0])
          print('Test accuracy:', score[1])
```

```
101/101 [==============================] - 1s 11ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Test loss: 0.0
Test accuracy: 1.0
```

Finally, just save the weight of the trained model so that next time to use.

```
In [104]: # when you finish training, you should save your model in a file
          # for carplate_recognition.ipynb
          # since it will use this model to recognize car plates
          model.save_weights('char_cnn.h5')
```

Difficulties Encountered

The biggest problem is to fit the model. By plotting the training history, I found that both loss and accuracy on validation set and training set are presented as fluctuations are high (From 92% to 98%). I modified some hyperparameters, however, after many

tweaks, the result is not changed obviously. Then I do the dilation operation and encounter the training set. In addition, I check the label definition and find that the test set form. Then after dealing with that, after only two iterations, the accuracy can reach 99% at least.