Car Plate Recognition

Jinsheng Pan Steve Zhu Xuecheng Liu

Project Idea

Car plate recognition, identifying the plate number from car images, is a useful application of computer vision in many real life scenarios. In this project, we plan to construct a model that takes a car image with a plate on it as the input and output the corresponding characters one by one on that plate.

Method Used

Plate Detection:

1. Turn the image into binary and then use Canny to detect edges.
2. Use morphological opening and closing operations to remove small regions/noise, and there would be several rectangle regions left in the image although we can't remove small regions/noise completely.
3. Find contours and use features of plate (i.e. ratio of height and width, color) to rule out unqualified contours.
4. Use the corner points of the contour left to find the location of the plate.

Letter Splitting:

1. Capture the main Area of the plate (contains letters and numbers) out and turn the image into binary.
2. Blur and setting gray-scale value threshold to remove the noise and background.
3. Among x the direction, count the number of white pixels (assume background is black and letters are white). If the number is larger than a certain value, we regard this column is inside of the letter, otherwise, the column is outside of the letter.

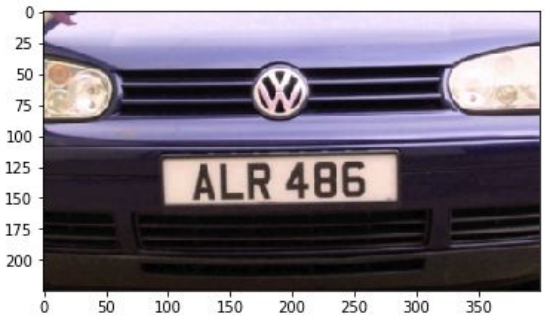
Letter Recognition:

1. Load and adjust the training data for KNN (In theorem, HoG and PCA could capture features of training and testing images and give us a good result, but in practice they don't and we need to figure out the reason).
2. Adjust the test image then predict

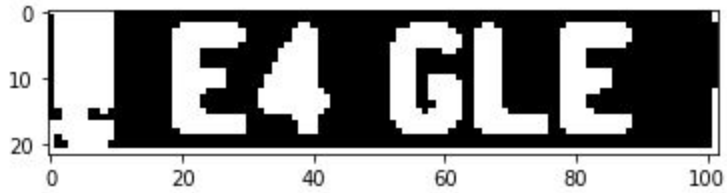
Problems Encountered

Different images need different hyperparameters, difficult to find a universal hyperparameter applied to all images. (unsolved)

The size of the training set is so small that when doing recognition, there some letters like 'B' will be regarded as '8'. (unsolved)

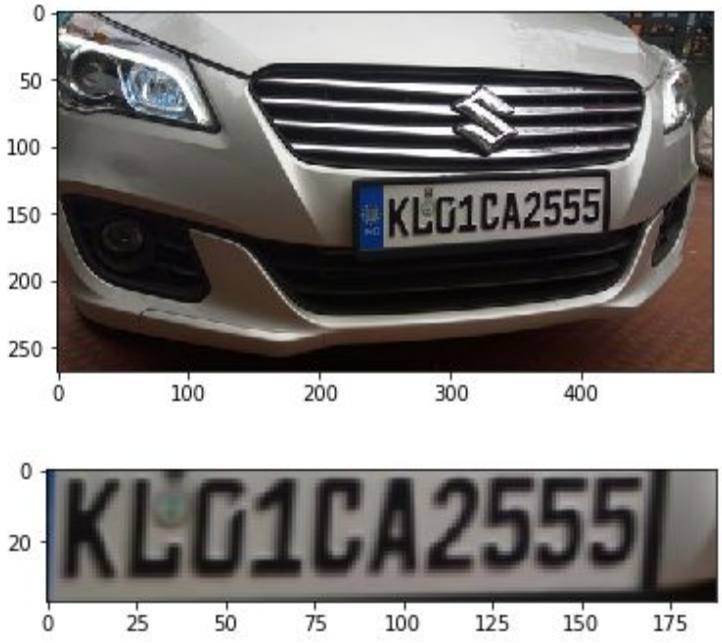


Difficult to remove noises, and in the end it results in an undesirable outcome. (unsolved)



It's difficult to clean the rectangle left side after closing & opening operation. Probably we should come up with an idea to better crop out the main area of letters and numbers.

Camera angles could cause problems for recognition but we solve it by getting the coordinate of corner points of the rectangle and transfer it into the coordinate we want. (solved)



Outcomes

Even if we could recognize most of the characters on the plate, there is a misclassified character in the outcome. In order to overcome this error, we might need to find the solution to some unsolved problems above.

Lessons Learned

Preparing a good dataset is important but sometimes it's hard to find a proper one due to many reasons (i.e. privacy, copyright).

A good camera is crucial for image processing because it could help reduce noises and support a high quality image.

We could use different algorithms to solve different problems based on our knowledge. For example, HoG is doing well in feature extraction, we could use it to letter recognition instead of splitting letters.

Work Allocation

Plate Detection: Jinsheng Pan

Letter Splitting: Steve Zhu

Letter Recognition: Xuecheng Liu

Future Work

1.Prepare a better and larger dataset for training.

2. Use CNN for plate detection because it has a better result of capturing the feature of images, which could be helpful when removing candidates rectangle of plate or removing noise.

3. When splitting letters, we could calculate the gradient of the image among x direction and then find the peak and trough. The letters exist between a peak and its corresponding trough.

4. For recognition, we could try several better classifiers such as svm and logistic regression because they could support higher accuracy than KNN.

Code Attached

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from imutils import perspective
from skimage import measure
import os
import operator


class Plate:
    def __init__(self):
        self.block = []
        self.ker1 = np.ones((5,19),np.uint8)
        self.ker2 = np.ones((11,5),np.uint8)
        self.maxWeight = 0
        self.maxIndex = -1
        self.edge = None
        self.contour = None
        self.img = None
        self.lower = np.array([0,0,150])
        self.upper = np.array([30,60,255])
        self.gray = None
        self.shift = True

    def getEdge(self,img):

        self.img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.img = cv2.GaussianBlur(self.img, (3,3), 0)
        self.gray = cv2.cvtColor(self.img, cv2.COLOR_RGB2GRAY)
        ret, img_thresh = cv2.threshold(self.gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
        self.edge = cv2.Canny(img_thresh, 128, 200)



    def getContour(self):
```

```
        c1_img= cv2.morphologyEx(self.edge, cv2.MORPH_CLOSE,self.ker1)
        o1_img = cv2.morphologyEx(c1_img, cv2.MORPH_OPEN,self.ker1)
        o2_img = cv2.morphologyEx(o1_img, cv2.MORPH_OPEN, self.ker2)
        _,cnts,_ = cv2.findContours(o2_img,
cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        self.contour = cnts

    def getCandidate(self):

        for cnts in self.contour:
            y,x = [], []
            for c in cnts:
                y.append(c[0][0])
                x.append(c[0][1])

            rect = cv2.minAreaRect(cnts)
            box = cv2.boxPoints(rect)
            box = np.int0(box)



            angle = rect[2]
            h,w = rect[1]
            if h > w:
                h,w = w, h
            r = w / h

            if r > 2 and r < 8.5:
                self.block.append([[[min(y), min(x)], [max(y), max(x)]],box,angle])

    def adjustCandidate(self):

        for i in range(len(self.block)):
            start = self.block[i][0][0]
            end = self.block[i][0][1]
            candidate = self.img[start[1]:end[1], start[0]:end[0]]

            hsv = cv2.cvtColor(candidate, cv2.COLOR_RGB2HSV)
```

```
            mask = cv2.inRange(hsv, self.lower, self.upper)

            w1 = 0
            for m in mask:
                w1 += m / 255.0

            w2 = 0
            for wi in w1:
                w2 += wi

            if w2 > self.maxWeight:
                self.maxIndex = i
                self.maxWeight = w2
        def getPlate(self):

            angle = self.block[self.maxIndex][2]
            if np.abs(angle) == 0 or (np.abs(angle) > 0 and np.abs(angle) < 1):
                self.shift = False

            if self.shift:
                box = self.block[self.maxIndex][1].astype('float32')
                plate = perspective.four_point_transform(self.img, box)
            else:
                loc = self.block[self.maxIndex]
                start = (loc[0][0][0], loc[0][0][1])
                end = (loc[0][1][0], loc[0][1][1])
                plate = self.img[start[1]:end[1],start[0]:end[0],:]
            plt.imsave('plate.png',plate)


class SegLetter:

    def __init__(self):
        self.thresh = None
        self.gray = None
        self.flag = False
        self.flagList = []
```

```
    self.imList = []
    self.char = []
    self.thelta = 2

def crop(self):
    labels = measure.label(self.thresh, connectivity=2, background=0)
    mask = np.zeros(self.thresh.shape, dtype='uint8')

    for y in range(len(labels)):
        for x in range(len(labels[y])):
            if labels[y][x] > 0:
                mask[y][x] = 1

    for i in range(labels.shape[1]):

        count = np.count_nonzero(labels[:,i])
        if count > self.thelta:
            self.flag = True
        else:
            self.flag = False
        self.flagList.append(self.flag)

    for i in range(len(self.flagList)):
        if self.flagList[i]:
            self.char.append(mask[:,i])
        else:
            if self.char:
                if len(self.char) > 2:
                    self.imList.append(self.char)
                    self.char = []
    if self.char:
        if len(self.char) > 2:
            self.imList.append(self.char)
            self.char = []

    num = 1
    if not os.path.exists('characters'):
```

```
            os.makedirs('characters')

        for ele in self.imList:
            root = 'characters/'
            if num > 9:
                root = 'characters/a'
            ele = np.transpose(ele)
            plt.imsave(root + str(num) + '.png', ele,cmap='gray')
            num += 1


    def preProcess(self, img):
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
        cnts = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(c)
        mainArea = img[y:y + h, x:x + w, :].copy()
        self.gray = cv2.cvtColor(mainArea, cv2.COLOR_BGR2GRAY)

        _, self.thresh = cv2.threshold(self.gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)


class KNN:

    def __init__(self):
        self.trainData = np.zeros([28,28,36,29])
        self.trainLabels = np.zeros([36,29])
        self.maps = {}

    def loadData(self,path):

        trainFile = os.listdir(path)
```

```
        idx = 0

        for files in trainFile:
            self.maps[idx] = files
            pth = path + '/'+ str(files)
            loc = 0

            for file in os.listdir(pth):
                im_path = pth + "/" + file
                image = cv2.imread(im_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                ret,image = cv2.threshold(image, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)
                image = cv2.resize(image, (28,28))
                for i in range(image.shape[0]):
                    for j in range(image.shape[1]):
                        if image[i,j] < 255:
                            image[i,j] = 0

                self.trainData[:,:,idx,loc] = image
                self.trainLabels[idx,loc] = idx
                loc +=1
            idx += 1
        self.trainLabels = self.trainLabels.reshape(-1,1)
        self.trainData = self.trainData.reshape(28*28, 36*29).T

    def predict(self,testData, k = 3):

        size = self.trainData.shape[0]
        classCount = {}
        diff = np.abs(np.tile(testData, (size,1)) - self.trainData)

        distance = np.sum(diff, axis = 1)
        distIndex = np.argsort(distance)

        for i in range(k):
            label = self.trainLabels[distIndex[i]][0]
```

```
            classCount[label] = classCount.get(label,0) + 1
        classCount = sorted(classCount.items(), key = operator.itemgetter(1),
reverse=True)

        return self.maps[classCount[0][0]]


if __name__ == '__main__':
    path = 'images/Cars1.png'
    img1 = cv2.imread(path)
    plate = Plate()
    plate.getEdge(img1)
    img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
    plt.imshow(img1)
    plate.getContour()
    plate.getCandidate()
    plate.adjustCandidate()
    plate.getPlate()

    seg = SegLetter()
    img2 = cv2.imread('plate.png')
    seg.preProcess(img2)
    seg.crop()

    knn = KNN()
    path2 = 'data/Training'
    knn.loadData(path2)

    lst = os.listdir('characters')
    lst.sort()
    prediction = []
    for file in lst:
        pth = 'characters/' + file

        if pth.endswith('.png'):
            test = cv2.imread(pth)
            test = cv2.cvtColor(test, cv2.COLOR_BGR2GRAY)
```

```
        ret,test = cv2.threshold(test, 100, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        test = cv2.resize(test, (28,28), interpolation=cv2.INTER_NEAREST)
        test = test.reshape(-1,1).T
        predict = knn.predict(test, 5)
        prediction.append(predict)
   print(prediction)
```