**from** skimage.io **import** imread  
**from** skimage.filters **import** threshold\_otsu  
**import** matplotlib.pyplot **as** plt  
  
car\_image = imread(**"C:\\Users\\lenovo\\Desktop\\car2.jfif"**, as\_grey=**True**)  
*# it should be a 2 dimensional array*print(car\_image.shape)  
  
*# the next line is not compulsory however, a grey scale pixel  
# in skimage ranges between 0 & 1. multiplying it with 255  
# will make it range between 0 & 255 (something we can relate better with*gray\_car\_image = car\_image \* 255  
fig, (ax1, ax2) = plt.subplots(1, 2)  
ax1.imshow(gray\_car\_image, cmap=**"gray"**)  
threshold\_value = threshold\_otsu(gray\_car\_image)  
binary\_car\_image = gray\_car\_image > threshold\_value  
ax2.imshow(binary\_car\_image, cmap=**"gray"**)  
plt.show()  
  
*#character segmentation***from** skimage **import** measure  
**from** skimage.measure **import** regionprops  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.patches **as** patches  
**import** localization  
  
*# this gets all the connected regions and groups them together*label\_image = measure.label(localization.binary\_car\_image)  
fig, (ax1) = plt.subplots(1)  
ax1.imshow(localization.gray\_car\_image, cmap=**"gray"**);  
  
*# regionprops creates a list of properties of all the labelled regions***for** region **in** regionprops(label\_image):  
 **if** region.area < 50:  
 *#if the region is so small then it's likely not a license plate* **continue** *# the bounding box coordinates* minRow, minCol, maxRow, maxCol = region.bbox  
 rectBorder = patches.Rectangle((minCol, minRow), maxCol-minCol, maxRow-minRow, edgecolor=**"red"**, linewidth=2, fill=**False**)  
 ax1.add\_patch(rectBorder)  
 *# let's draw a red rectangle over those regions*plt.show()  
*#3***from** skimage **import** measure  
**from** skimage.measure **import** regionprops  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.patches **as** patches  
**import** localization  
  
  
*# this gets all the connected regions and groups them together*label\_image = measure.label(localization.binary\_car\_image)  
  
*# getting the maximum width, height and minimum width and height that a license plate can be*plate\_dimensions = (0.08\*label\_image.shape[0], 0.2\*label\_image.shape[0], 0.15\*label\_image.shape[1], 0.4\*label\_image.shape[1])  
min\_height, max\_height, min\_width, max\_width = plate\_dimensions  
plate\_objects\_cordinates = []  
plate\_like\_objects = []  
fig, (ax1) = plt.subplots(1)  
ax1.imshow(localization.gray\_car\_image, cmap=**"gray"**);  
  
*# regionprops creates a list of properties of all the labelled regions***for** region **in** regionprops(label\_image):  
 **if** region.area < 50:  
 *#if the region is so small then it's likely not a license plate* **continue** *# the bounding box coordinates* min\_row, min\_col, max\_row, max\_col = region.bbox  
 region\_height = max\_row - min\_row  
 region\_width = max\_col - min\_col  
 *# ensuring that the region identified satisfies the condition of a typical license plate* **if** region\_height >= min\_height **and** region\_height <= max\_height **and** region\_width >= min\_width **and** region\_width <= max\_width **and** region\_width > region\_height:  
 plate\_like\_objects.append(localization.binary\_car\_image[min\_row:max\_row,  
 min\_col:max\_col])  
 plate\_objects\_cordinates.append((min\_row, min\_col,  
 max\_row, max\_col))  
 rectBorder = patches.Rectangle((min\_col, min\_row), max\_col-min\_col, max\_row-min\_row, edgecolor=**"red"**, linewidth=2, fill=**False**)  
 ax1.add\_patch(rectBorder)  
 *# let's draw a red rectangle over those regions*plt.show()  
  
*#4character segmentation***import** numpy **as** np  
**from** skimage.transform **import** resize  
**from** skimage **import** measure  
**from** skimage.measure **import** regionprops  
**import** matplotlib.patches **as** patches  
**import** matplotlib.pyplot **as** plt  
**import** cca2  
  
*# on the image I'm using, the headlamps were categorized as a license plate  
# because their shapes were similar  
# for now I'll just use the plate\_like\_objects[2] since I know that's the  
# license plate. We'll fix this later  
  
# The invert was done so as to convert the black pixel to white pixel and vice versa*license\_plate = np.invert(cca2.plate\_like\_objects[2])  
  
labelled\_plate = measure.label(license\_plate)  
  
fig, ax1 = plt.subplots(1)  
ax1.imshow(license\_plate, cmap=**"gray"**)  
*# the next two lines is based on the assumptions that the width of  
# a license plate should be between 5% and 15% of the license plate,  
# and height should be between 35% and 60%  
# this will eliminate some*character\_dimensions = (0.35\*license\_plate.shape[0], 0.60\*license\_plate.shape[0], 0.05\*license\_plate.shape[1], 0.15\*license\_plate.shape[1])  
min\_height, max\_height, min\_width, max\_width = character\_dimensions  
  
characters = []  
counter=0  
column\_list = []  
**for** regions **in** regionprops(labelled\_plate):  
 y0, x0, y1, x1 = regions.bbox  
 region\_height = y1 - y0  
 region\_width = x1 - x0  
  
 **if** region\_height > min\_height **and** region\_height < max\_height **and** region\_width > min\_width **and** region\_width < max\_width:  
 roi = license\_plate[y0:y1, x0:x1]  
  
 *# draw a red bordered rectangle over the character.* rect\_border = patches.Rectangle((x0, y0), x1 - x0, y1 - y0, edgecolor=**"red"**,  
 linewidth=2, fill=**False**)  
 ax1.add\_patch(rect\_border)  
  
 *# resize the characters to 20X20 and then append each character into the characters list* resized\_char = resize(roi, (20, 20))  
 characters.append(resized\_char)  
  
 *# this is just to keep track of the arrangement of the characters* column\_list.append(x0)  
  
plt.show()  
*#5***import** os  
**import** numpy **as** np  
**from** sklearn.svm **import** SVC  
**from** sklearn.model\_selection **import** cross\_val\_score  
**from** sklearn.externals **import** joblib  
**from** skimage.io **import** imread  
**from** skimage.filters **import** threshold\_otsu  
  
letters = [  
 **'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'** ]  
  
**def** read\_training\_data(training\_directory):  
 image\_data = []  
 target\_data = []  
 **for** each\_letter **in** letters:  
 **for** each **in** range(10):  
 image\_path = os.path.join(training\_directory, each\_letter, each\_letter + **'\_'** + str(each) + **'.jpg'**)  
 *# read each image of each character* img\_details = imread(image\_path, as\_grey=**True**)  
 *# converts each character image to binary image* binary\_image = img\_details < threshold\_otsu(img\_details)  
 *# the 2D array of each image is flattened because the machine learning  
 # classifier requires that each sample is a 1D array  
 # therefore the 20\*20 image becomes 1\*400  
 # in machine learning terms that's 400 features with each pixel  
 # representing a feature* flat\_bin\_image = binary\_image.reshape(-1)  
 image\_data.append(flat\_bin\_image)  
 target\_data.append(each\_letter)  
  
 **return** (np.array(image\_data), np.array(target\_data))  
  
**def** cross\_validation(model, num\_of\_fold, train\_data, train\_label):  
 *# this uses the concept of cross validation to measure the accuracy  
 # of a model, the num\_of\_fold determines the type of validation  
 # e.g if num\_of\_fold is 4, then we are performing a 4-fold cross validation  
 # it will divide the dataset into 4 and use 1/4 of it for testing  
 # and the remaining 3/4 for the training* accuracy\_result = cross\_val\_score(model, train\_data, train\_label,  
 cv=num\_of\_fold)  
 print(**"Cross Validation Result for "**, str(num\_of\_fold), **" -fold"**)  
  
 print(accuracy\_result \* 100)  
  
  
current\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))  
  
training\_dataset\_dir = os.path.join(current\_dir, **'train'**)  
  
image\_data, target\_data = read\_training\_data(training\_dataset\_dir)  
  
*# the kernel can be 'linear', 'poly' or 'rbf'  
# the probability was set to True so as to show  
# how sure the model is of it's prediction*svc\_model = SVC(kernel=**'linear'**, probability=**True**)  
  
cross\_validation(svc\_model, 4, image\_data, target\_data)  
  
*# let's train the model with all the input data*svc\_model.fit(image\_data, target\_data)  
  
*# we will use the joblib module to persist the model  
# into files. This means that the next time we need to  
# predict, we don't need to train the model again*save\_directory = os.path.join(current\_dir, **'models/svc/'**)  
**if not** os.path.exists(save\_directory):  
 os.makedirs(save\_directory)  
joblib.dump(svc\_model, save\_directory+**'/svc.pkl'**)  
  
*#6***import** os  
**from** cython **import** segmentation  
**from** sklearn.externals **import** joblib  
  
*# load the model*current\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))  
model\_dir = os.path.join(current\_dir, **'models/svc/svc.pkl'**)  
model = joblib.load(model\_dir)  
  
classification\_result = []  
**for** each\_character **in** segmentation.characters:  
 *# converts it to a 1D array* each\_character = each\_character.reshape(1, -1);  
 result = model.predict(each\_character)  
 classification\_result.append(result)  
  
print(classification\_result)  
  
plate\_string = **''  
for** eachPredict **in** classification\_result:  
 plate\_string += eachPredict[0]  
  
print(plate\_string)  
  
*# it's possible the characters are wrongly arranged  
# since that's a possibility, the column\_list will be  
# used to sort the letters in the right order*column\_list\_copy = segmentation.column\_list[:]  
segmentation.column\_list.sort()  
rightplate\_string = **''  
for** each **in** segmentation.column\_list:  
 rightplate\_string += plate\_string[column\_list\_copy.index(each)]  
  
print(rightplate\_string)