ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

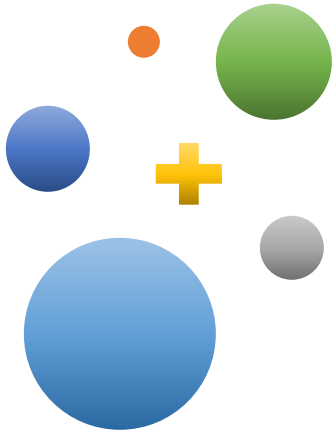
TRƯỜNG ĐẠI HỌC KHOA HOC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Báo cáo đồ án

(Cải tiến và bổ sung)

Automatic license plate recognition (ALPR)



GVHD: Võ Hoài Việt

Mục lục

A Thành viên nhóm:

Tên nhóm: Fusion

STT	MSSV	Họ tên	Email
1	1612174	Phùng Tiến Hào	tienhaophung@gmail.com
2	1612269	Võ Quốc Huy	voquochuy304@gmail.com
3	1612272	Trần Nhật Huy	nhathuy13598@gmail.com

B Mức độ hoàn thành:

STT	Nội dung	Mức độ hoàn thành (%)
1	Phát hiện biển số (Plate detection)	100
2	Kiểm tra biển số (Validation)	100
3	Nhận dạng kí tự trong biển số (Character recognition)	100
Tổng cộng:		**100**

C Báo cáo:

I. Tổng quan cách cài đặt:

Các bước thực hiện hiện:

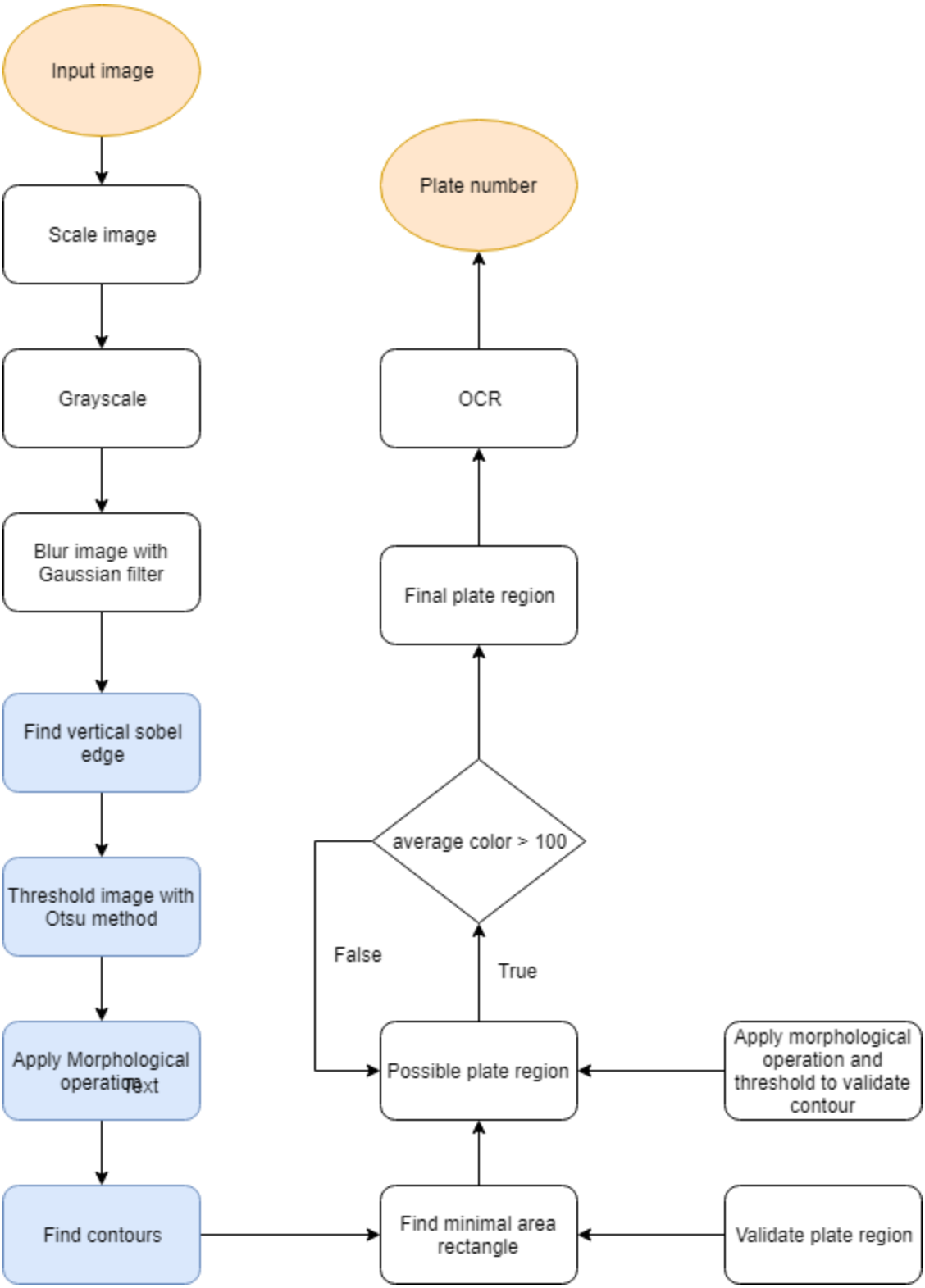


Figure 1 Lưu đồ hoạt động của hệ thống

Chi tiết hơn:

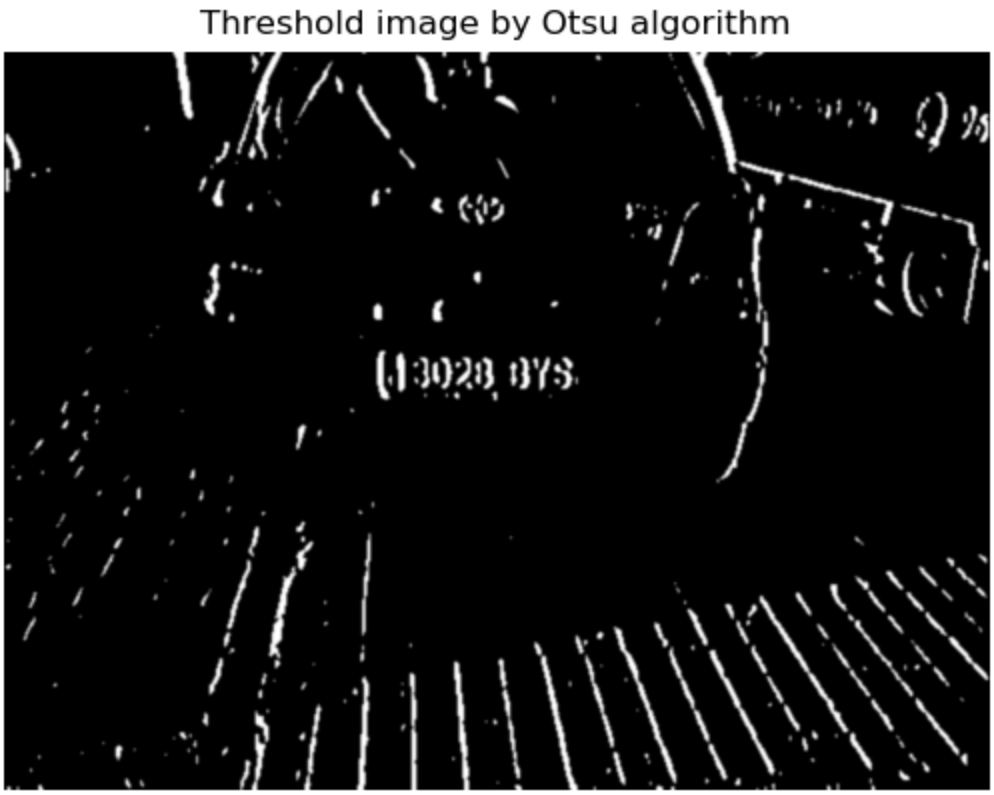
1) Scale ảnh về một kích thước chung: có width = 600 và height = (600 * original_height)/original_width

2) Gray-scale image: để thực hiện tìm biên cạnh và phục vụ cho các bước sau

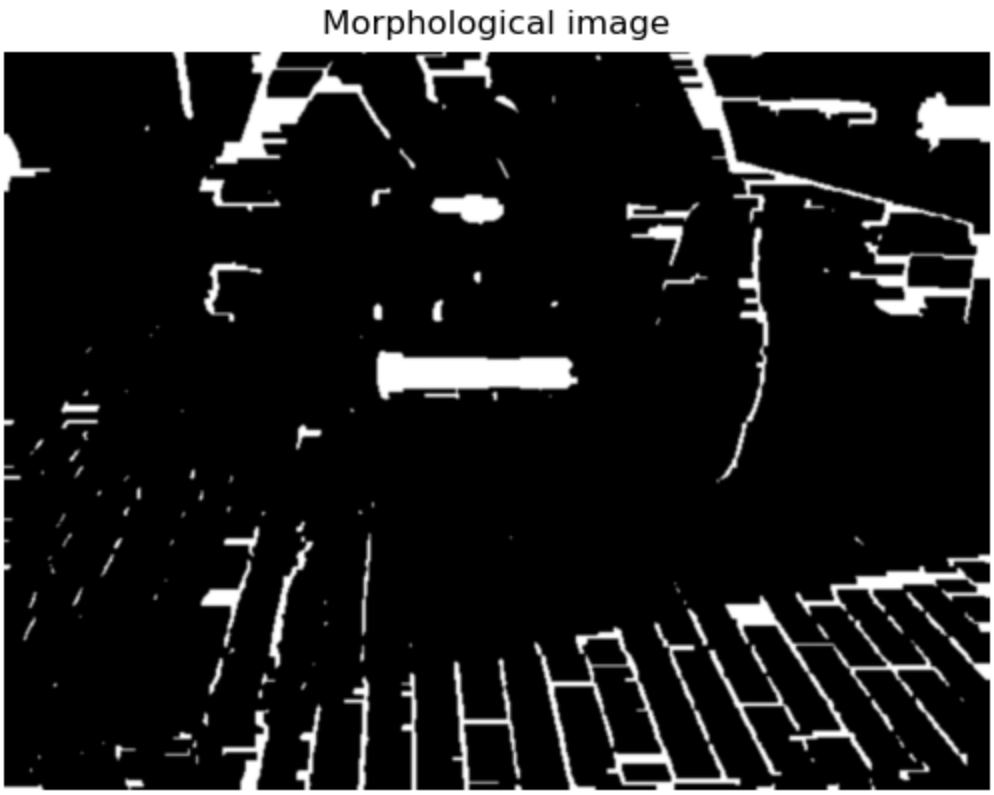
3) Làm mờ ảnh với Gaussian filter để giảm bớt nhiễu

4) Tìm biên cạnh dọc bằng Sobel filter

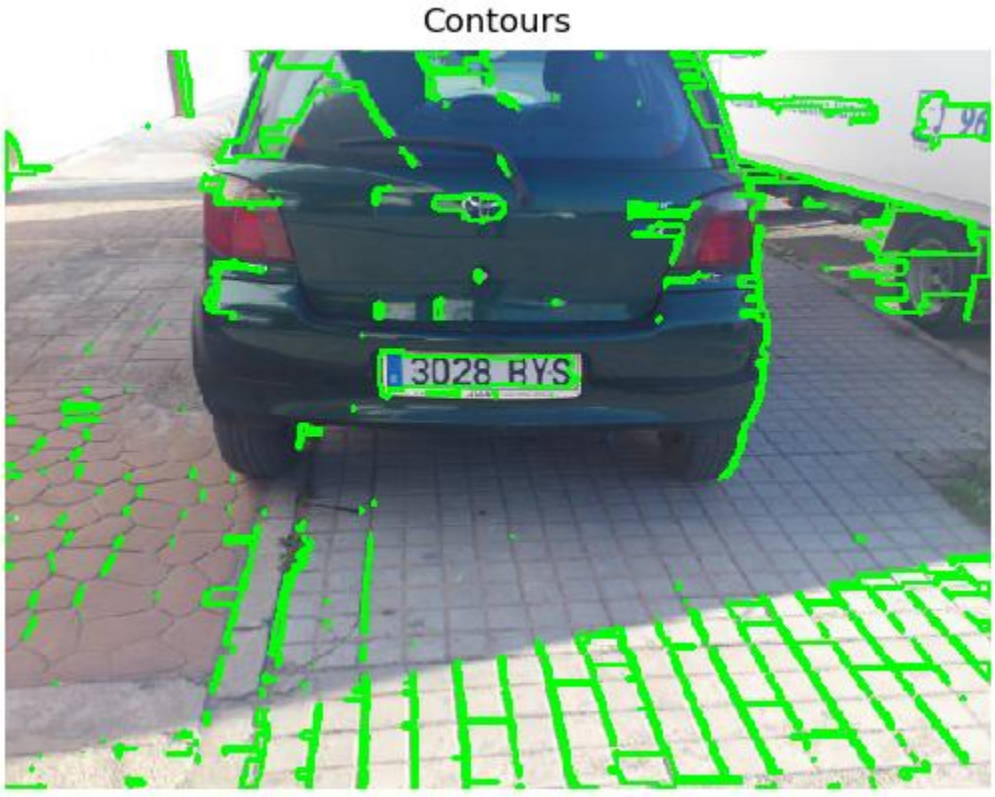
5) Threshold ảnh bằng phương pháp Otsu để nó tự tìm một ngưỡng tối ưu



6) Áp dụng morphological operation, cụ thể là phép “Clossing”. Tức là trước tiên sẽ làm giản nỡ (dilation) ảnh sau đó làm xói mòn (erosion) ảnh để giảm nhiễu và xóa các lỗ chấm đen trên đối tượng. Với kích thước kernel = 19x3 (đây là kích thước thực nghiệm) chỉ áp dụng để nhận diện biển số xe ô tô với tối đa 8 kí tự.



7) Tìm contours để phân vùng ảnh (segmentation)



8) Tìm các bounding box hình chữ nhật (có tính góc nghiêng không quá 30 độ)

9) Kiểm tra và xác thực các bounding box:
Các công viên liên quan:
- Kiểm ra màu trắng có chiếm đáng kể trong vùng đó không
- Kiểm tra tỉ lệ khung hình cho phần vùng biển số:
  - Kích thước biển ô tô của ngước ngoài (cụ thể châu Tây Ban Nha hoặc Châu Âu): chiều rộng 52 cm, chiều cao 11 cm.
  - Do đó, tỉ lệ khung hình: 52/11 = 4.7272 với độ lỗi chấp nhận là 40%.
  - Dao động của tỉ lệ khung hình là min = 4.7272*(1 – 0.4), max = 4.7272*(1 + 0.4)
- Kiểm tra diện tích vùng trong khoảng từ: min = 15*aspect_ratio*15, max = 125*aspect_ratio*125
- Kiêm tra góc quay: Nếu bounding box xoay quá 30 độ thì bị loại.

10) Tận dụng phần nền biển số là màu trắng, tính cường độ màu trung bình của phần ảnh biển số đó xem có gần trắng hay không (> 100)

11) Nhận dạng kí tự trong biển số bằng thư viện mở Tesseract – được phát triển bởi Google

II. Yêu cầu của hệ thống:

Để đảm bảo hệ thống hoạt động tốt và chính xác thì chất lượng ảnh đầu vào đóng vai trò rất quan trọng. Cụ thể:

- Ảnh phải được chụp trực diện và cách xa khoảng 1.5-2 (m).
- Điều kiện sáng đầy đủ
- Ảnh bị nhiễu ít
- Góc nghiêng vừa phải

Ngôn ngữ sử dụng: Python

Các thư viện cần thiết để chạy chương trình:

- Thư viện OpenCV
- Thư viện PIL
- Thư viện Matplotlib
- Thư viện Numpy
- Thư viện Pytesseract

III. Kết quả thực nghiệm:

Mẫu dữ liệu test gồm 66 ảnh trong nhiều điều kiện khác nhau như: độ sáng, nhiễu, ảnh chụp góc nghiêng,…

Công việc	Số lượng mẫu bị thất bại	Độ chính xác (%)
Phát hiện biển số	9	86.36
Nhận dạng kí tư	45	30

Một số ảnh phát hiện và nhận dạng kí tự đúng:

Plate number recognition

Correct plates

Plate number recognition

Correct plates

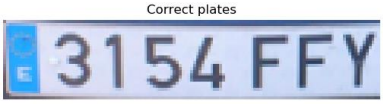


Plate number recognition

Correct plates

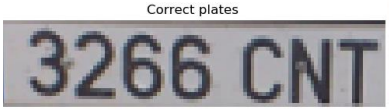
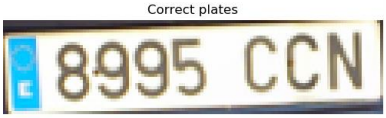


Plate number recognition

Correct plates



Một số ảnh phát hiện sai hoặc nhận dạng sai (hoặc không ra kết quả: None)

Plate number recognition

Correct plates

Plate number recognition

Correct plates

Nhận xét:

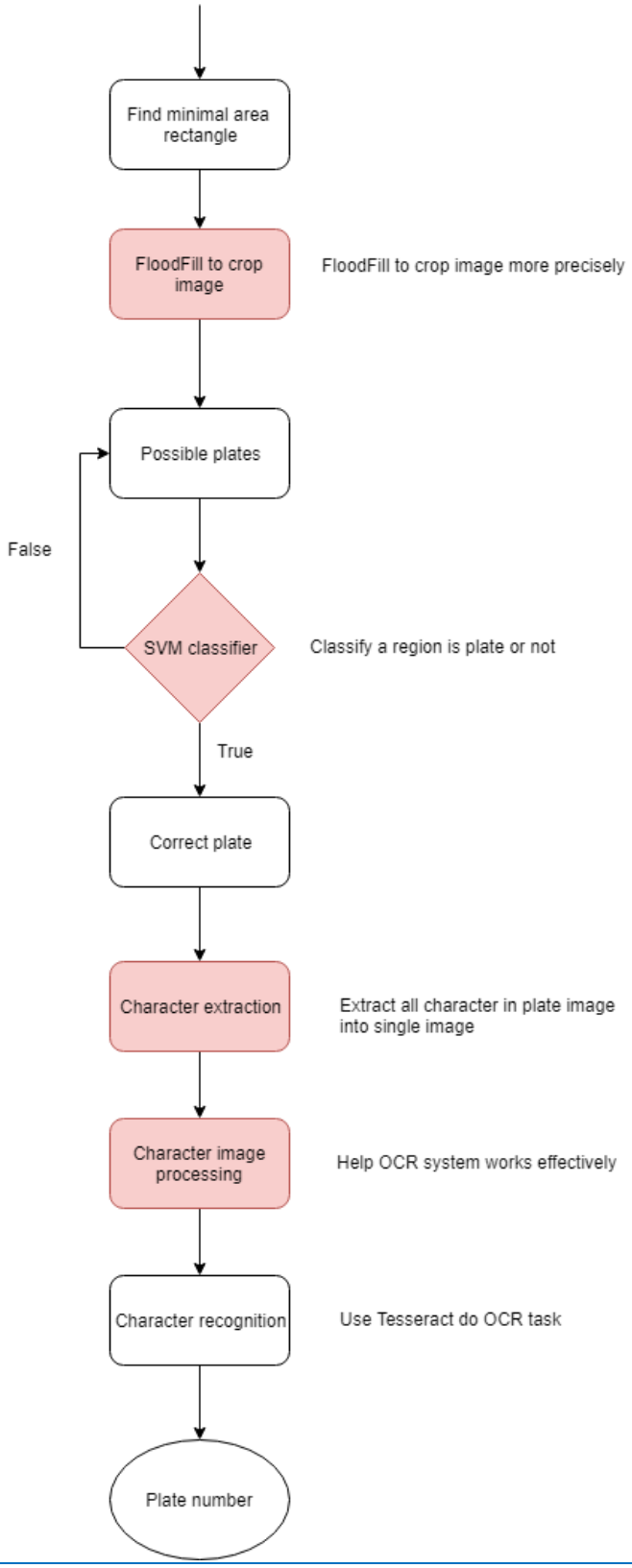
- Kết quả phát hiện biển số vẫn khá thi nhưng chưa thực sự chính xác vì có trường hợp đã phát hiện ra vùng ảnh là biển số nhưng khi qua bước kiểm thực lần nữa thì đã bị loại. Điều này nhóm sẽ cải tiến bằng mô hình SVM để phân lớp và xác định biển số.
- Đa số ảnh bị nhận dạng kí tự sai hoặc không thể nhận dạng kí tự được.
- Bên cạnh đó, nhóm sẽ cố gắng thay thế phương pháp truyền thống bằng việc dùng mô hình mạng deep learning YOLO để giúp phát hiện biển số tốt hơn và chạy với thời gian thực.
- Hiện tại, công việc nhận dạng kí tự được thực hiện bởi thư viên Tesseract của google. Nhóm không hiểu ví sao kết quả nhận dạng lại quá thấp. Điều này trong đợt cải tiến sắp tới nhóm sẽ cố gắng cải thiện.

D Cải tiến và bổ sung:

Ở đây, chúng tôi có hai phiên bản:

I. Phiên bản 1: Bản cải tiến và bổ sung của đợt nộp lần đầu

Qui trình bổ sung và cải tiến



Những phần bổ sung và cải tiến được tô màu đỏ nhạt.

Chi tiết hơn như sau:

FloodFill: với mục đích là giúp cắt ảnh và trích xuất ra vùng ảnh biển số chính xác hơn. Nó giúp loại bỏ các phần thừa không phải biển số trong vùng ảnh đó.

- Ý tưởng: ta sẽ gieo một số hạt giống (seed) ở xung quanh tâm của rotated rectangle (bounding box có tính góc nghiêng, được tạo bởi contours) và sẽ floodFill để fill các vùng đã đặt hạt giống và loang ra. Tân dụng background của biển số là màu trắng, do đó ta sẽ tìm được bounding box sẽ ôm sát biển số hơn.
- Thêm nữa, ta còn thực hiện kiểm định sơ bộ vùng ảnh đó có phải biển số (dựa vào tỉ lệ khung hình) và thực hiện phép biến đổi AFFINE như xoay vùng ảnh đó lại cho nó song song với trục hoành. Mục đích: giúp cân chỉnh ảnh cho việc phân lớp đối tương và nhận dạng kí tự.
- Cuối cùng, ta thực hiện xén ảnh. Tuy nhiên, mỗi ảnh biển số thì có kích thước khác nhau, điều kiện sáng khác nhau. Do đó, ta cần làm thêm bước resize kích thước ảnh về (Width = 144, Height = 33) và cân bằng sáng cho ảnh.

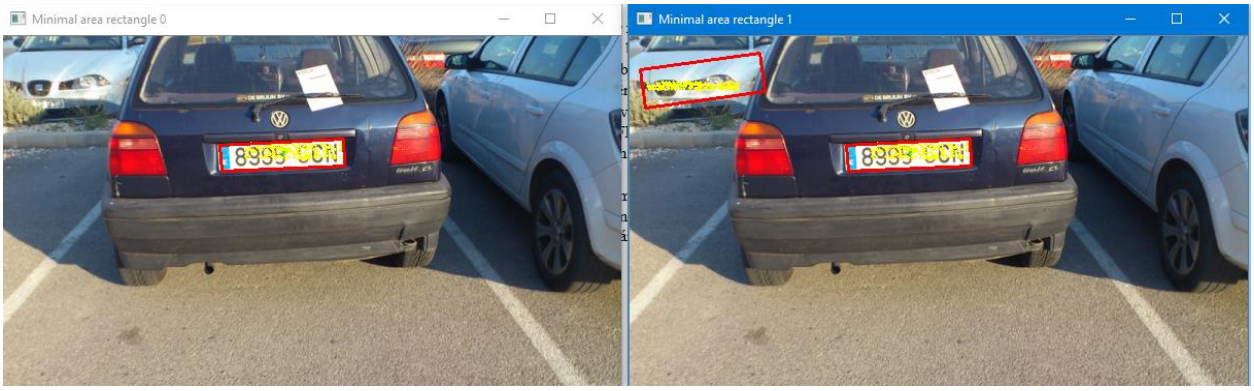


Figure 2: Đặt seed cho vùng ảnh có thể là biển số (các chấm màu vàng)

Figure 3: Mask thu được từ floodfill của 2 ảnh trên

Figure 4: Cân bằng sáng và đã xoay ảnh

Figure 5: Crop ảnh

Bộ phân lớp SVM: dùng để phân lớp xem vùng ảnh có là biển số hay không (0: không phải, 1: biển số).

- Ở đây, nhóm sử dụng pretrained model của tác giá cuốn sách Mastering OpenCV with Practical Computer Vision Projects (6th ed., pp. 161-188). Tác giả đã huấn luyện model này trên tập dataset gồm 75 ảnh dương và 35 ảnh âm với kích thước 144x33 pixels.
- Dữ liệu tuy không nhiều nhưng vẫn đủ cho việc phân lớp và họ sử dụng pixel trong ảnh để làm đặc trưng huấn luyện.

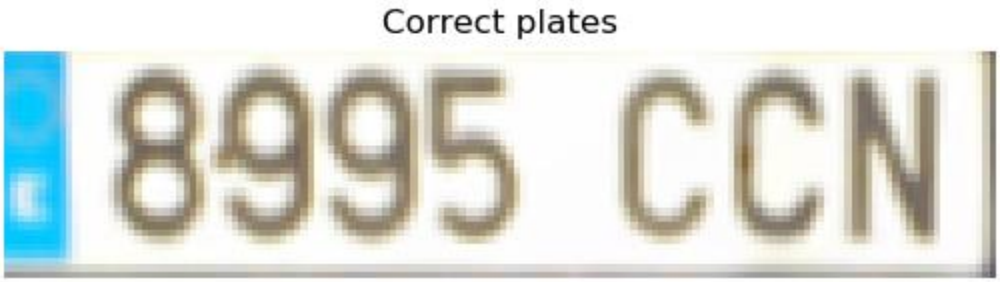


Figure 6: Ảnh biển số đã qua phân lớp

Character extraction: bước này đóng vai trò khá quan trọng trong khâu nhận dạng kí tự. Để tăng độ chính xác của nhận dạng kí tự, ta cần phải thực hiện tách các kí tự trong ảnh ra thành từng ảnh kí tự riêng lẻ và đem vào Tesseract để nhận dạng.

Cách thức: Tìm các contour và chọn lọc lại các contour thích hợp dựa vào diện tích. Sau đó, đóng bounding box cho các contour ấy rồi trích xuất vùng ảnh tương ứng.

Figure 7: Bước 1: Gray-scale, Bước 2: Threshold ảnh, Bước 3: Dùng phép biến đổi hình học Opening và Bước 4: Tìm contours

Figure 8: Trích xuất các kí tự trong ảnh

Character image processing: Thêm vào đó, ta cần phải làm thêm các bước xử lý như: threshold ảnh, dùng phép biến đổi hình thái như Opening (Erosion followed Dilation) để giảm nhiễu và tăng độ chính xác cho nhận dạng kí tự.

Tìm hiểu về Tesseract

1) Giới thiệu:

Tesseract là một OCR engine rất thông dụng, được phát triển ban đầu bởi tập đoàn Hewlett - Packard vào thập niên 80 và trở thành open source vào năm 2005. Google tiếp tục phát triển dự án này vào 2006.

Tesseract v4 có hỗ trợ nhận diện ký tự bằng deep learning.

Nền tảng bên dưới của Tesseract có sử dụng Long Short-Term Memory (LSTM) network, là một loại Recurrent Neural Network (RNN).

Tesseract có hỗ trợ nhận diện hơn 100 ngôn ngữ.

Để có kết quả tốt, cần preprocess ảnh input trước khi đưa vào tesseract.

2) Các flag cần quan tâm khi sử dụng thư viện Tesseract:

- -l: ngôn ngữ sử dụng.
- --oem: OCR engine mode, xác định loại thuật toán sử dụng. Có 4 loại:

```
OCR Engine modes:
  0    Legacy engine only.
  1    Neural nets LSTM engine only.
  2    Legacy + LSTM engines.
  3    Default, based on what is available.
```

- --psm: Page Segmentation Mode, xác định loại phân đoạn tran. Có 14 loại:

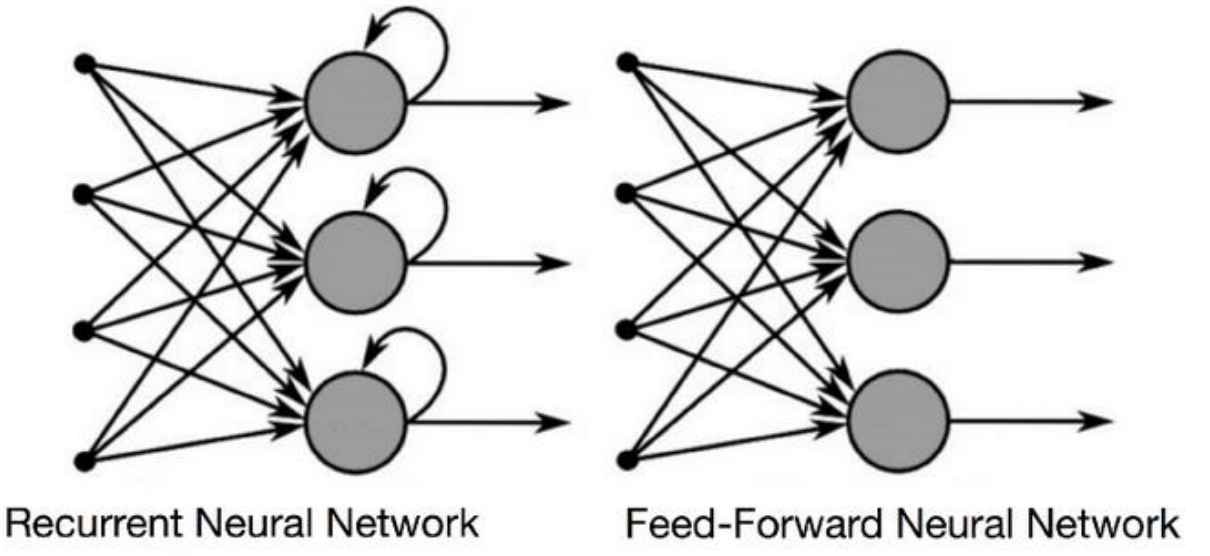
```
Page segmentation modes:
  0    Orientation and script detection (OSD) only.
  1    Automatic page segmentation with OSD.
  2    Automatic page segmentation, but no OSD, or OCR.
  3    Fully automatic page segmentation, but no OSD. (Default)
  4    Assume a single column of text of variable sizes.
  5    Assume a single uniform block of vertically aligned text.
  6    Assume a single uniform block of text.
  7    Treat the image as a single text line.
  8    Treat the image as a single word.
  9    Treat the image as a single word in a circle.
 10    Treat the image as a single character.
 11    Sparse text. Find as much text as possible in no particular order.
 12    Sparse text with OSD.
 13    Raw line. Treat the image as a single text line,
       bypassing hacks that are Tesseract-specific.
```

3) Recurrent Neural Network (RNN):

RNN là một loại neural network rất “mạnh”.

RNN thích hợp cho dữ liệu tuần tự, nối tiếp nhau (sequential data) như một câu văn, một lời nói, ... Nó được gọi như vậy là do nhớ được các input và tận dụng các input này để predict trong các lần tiếp theo.

RNN được phân biệt với feedforward network là do nó có feedback loop liên kết trở lại các node. Các thông tin khi predict sẽ được lưu lại. Có thể nghĩ rằng RNN có memory.



Do có memory, RNN có khả năng nhớ những thứ quan trọng về input nó nhận được, giúp nó dự đoán chính xác trong những lần tiếp theo.

4) Long Short-Term Memory (LSTM) Network:

Tuy RNN có thể nhớ được nhưng trí nhớ của nó là "ngắn hạn".

LSTM là một biến thể của RNN, nó giúp "mở rộng" memory. Như tên gọi của nó (tạm dịch: nhiều trí nhớ ngắn hạn), LTSM lưu trữ các thông tin trong lần predict trước và theo một cơ chế để sử dụng các thông tin này trong các lần predict sau. LSTM có thể đọc, ghi và xóa thông tin trong memory.

Mạng này bao gồm các "cells" có nhiệm vụ trong việc quyết định nên "nhớ" những thông tin gì, và thông tin nào là cần thiết được sử dụng trong mỗi lần dự đoán. Các việc quyết định này được thực hiện thông qua các "gates". Các "gates" này có thể là hàm sigmoid, tanh, hay là elemenwise. multiplication,...

Các kết quả thực nghiệm:

Figure 9: Kết quả nhận diện còn thiếu kí tự

Figure 10: Kết quả nhận diện còn thiếu kí tự do việc tìm contour của một số chữ không được

Figure 11: Tương tự vẫn còn thiếu sót kí tự

Bên cạnh đó, thì vẫn còn có trường hợp biển số bị phân lớp sai (nghĩa là biển số lại bị phân lớp nhầm là không phải) hoặc kích thước vùng biển số không thỏa do khoảng cách chụp quá gần. Dẫn đến, trả về kết quả "None".

Figure 12: Ảnh này đã tìm được contour của biển số, nhưng do kích thước biển số không thỏa nên bị loại

Figure 13: Trường hợp này thì phát hiện được biển số, nhưng nhận dạng không cho ra kết quả

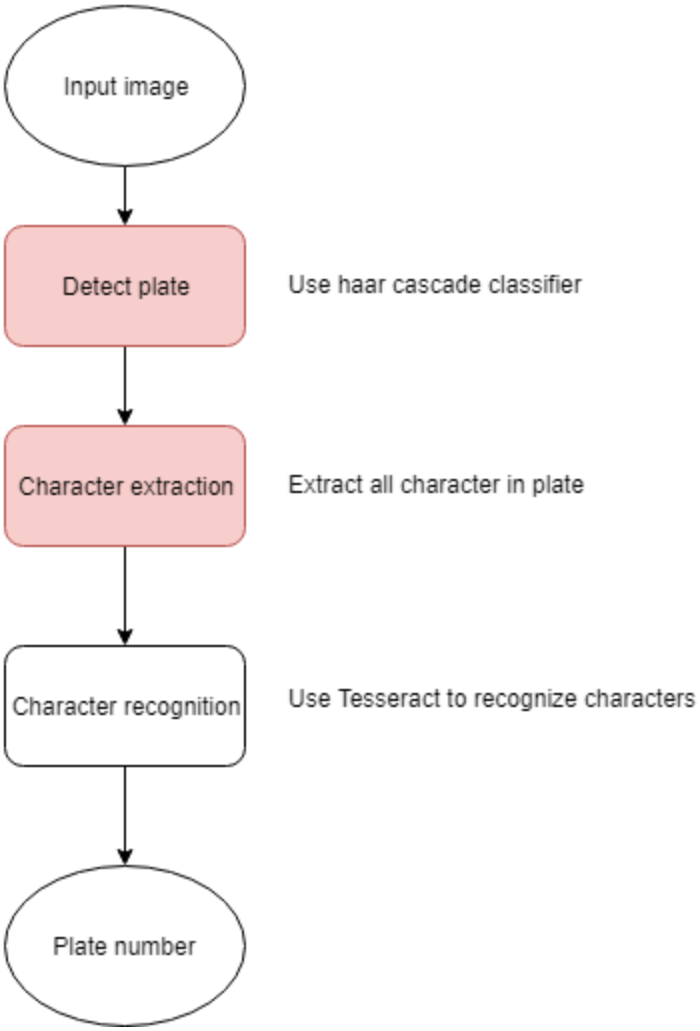
Nhận xét:

- Hệ thống vẫn chưa đạt mức chính xác tốt ở khâu nhận dạng.
- Kích thước biển số nhóm thực hiện dựa vào tập dữ liệu xe của Tây Ban Nha và châu Âu với kích thước biển 52x11. Phải thỏa kích thước này thì hệ thống mới nhận diện chính xác được.
- Trong dataset nhóm thu thập thì nó còn lẫn nhiều xe ô tô của các nước khác như: Hàn Quốc, Mỹ và Úc. Do đó, có một số ảnh sẽ không phát hiện được biển số vì do model SVM được tác giả train trên tập dữ liệu xe ô tô của Tây Ban Nha.
- Việc cải tiến này đã giúp cho hệ thống bắt được các trường hợp fail ở phiên bản cũ và nhận diện OCR phần nào đã cải thiện tốt hơn trước.
- Và nhóm cũng cho phép người dùng chọn là có sử dụng SVM để phân lớp hay không.

II. Phiên bản 2: Bản nhận dạng biển số cho xe moto ở Việt Nam bằng mạng CNN, cụ thể hơn là dùng bộ phân lớp dựa vào dặc trưng Haar.

Qui trình xử lý của hệ thống nhận dạng biển số cho xe máy:

Qui trình nhận dạng biển số xe máy bằng Haar Cascade



1. Giới thiệu về Haar cascade:

Haar Cascade là thuật toán học máy để phát hiện đối tượng trong ảnh hoặc video, được đề suất bởi Paul Viola và Michael Jones trong bài báo "Rapid Object Detection using a Boosted Cascade of Simple Features" năm 2001.

Phương pháp này dựa vào việc huấn luyện trên rất nhiều ảnh âm và ảnh dương. Ảnh âm là không có chứa biển số và ảnh dương là có chứa biển số. Người ta thường chọn tỉ lệ 3:1 vì đa phần trong ảnh có rất ít vùng đối tượng cần tìm.

Thuật toán gồm 4 giai đoạn:

Bước 1: Trích chọn đặc trưng Haar

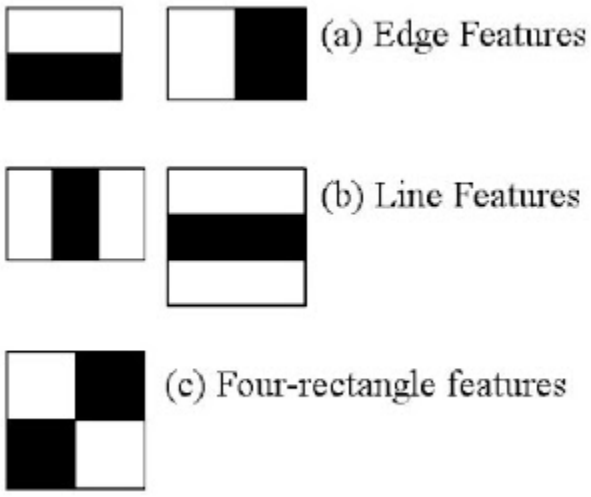


Figure 14: Các đặc trưng Haar (Trích từ OpenCV)

Bản chất Haar Cascades là một chuỗi các cửa sổ hình chữ nhật với các mức scale khác nhau lướt qua ảnh ban đầu. Trong mỗi của sổ hay kernel, ta sẽ tính giá trị cho các đặc trưng trong vùng cửa sổ đó. Các kernel này là ảnh binary với các pixel ở phần màu trắng có giá trị 1 và màu đen thì 0.

Mỗi đặc trưng là một giá trị tính từ việc lấy hiệu của tổng các pixel của hình chữ nhật màu trắng và của hình chữ nhật màu đen trong của cửa sổ đang xét. Vì thế, với mỗi kích thước cửa sổ khác nhau, mỗi vị trí khác nhau, ta phải tính rất nhiều đặc trưng, khoảng 160000 đặc trưng cần tính với kích thước kernel là 24x24.

Bước 2: Tạo ảnh tích hợp (Integral Images): Để giảm bớt chi phí tính toán lớn như vậy, ta sẽ thực hiện tính ảnh tích hợp theo công thức sau:

$$I(x,y) = i(x,y) + I(x,y-1) + I(x-1,y) - I(x-1,y-1)$$

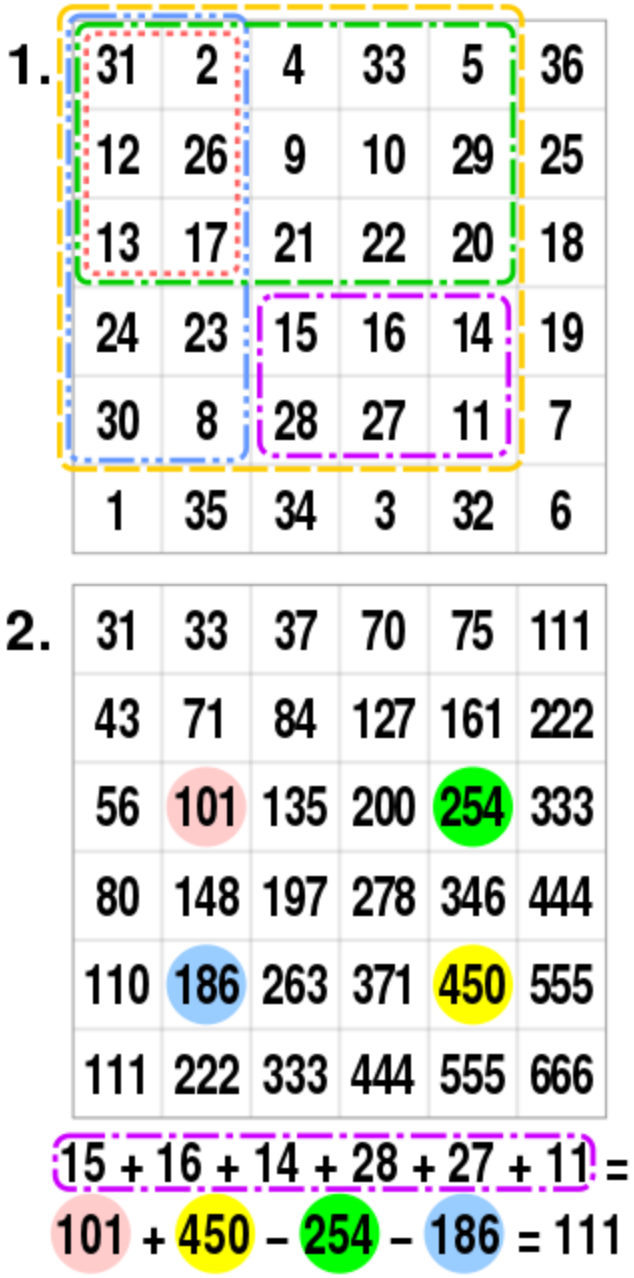
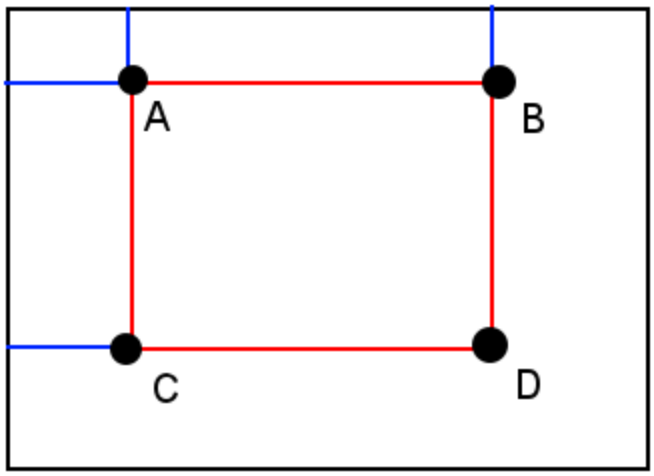


Figure 15: Ảnh 1 là ảnh gốc có kích thước (6x6). Ảnh 2 là ảnh tích hợp được tính với công thức trên (Trích từ Wiki)

Khi đó, tổng các pixel trong một vùng cửa sổ hình chữ nhật sẽ được tính như sau:

$$\sum_{\substack{x_0<x\leq x_1 \\ y_0<y\leq y_1}} i(x,y) = I(D) + I(A) - I(B) - I(C)$$



Sum = D - B - C + A

Ta có thể thấy, thay vì phải thực hiện tính lại tổng với mỗi kernel khác nhau thì giờ đây ta chỉ cần lấy tổng của 2 góc trên đường chéo chính trừ cho tổng của 2 trong đường chéo phụ trong ảnh tích hợp. Giúp tính toán nhanh hơn rất nhiều.

Bước 3: Huấn luyện Adaboost

Một sự thật là trong số các feature tính được thì đều là không phù hợp.

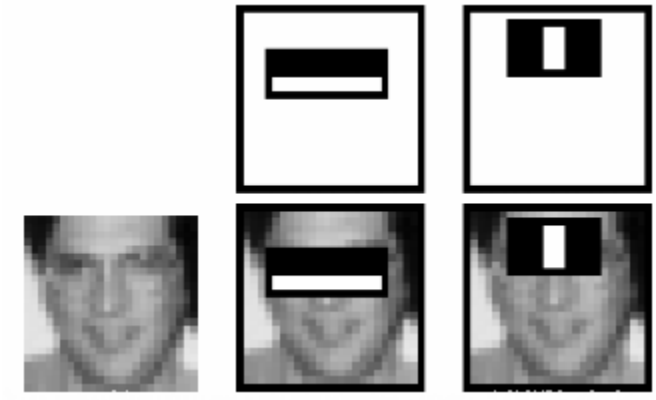


Figure 16: Ví dụ về các đặc trưng trên khuôn mặt tìm được từ Haar Cascade (Trích từ OpenCV)

Ta thấy rằng, 2 đặc trưng đầu tiên rất tốt để phát hiện mắt và phần giao giữa mắt và mũi. Nhận thấy rằng, phần mắt thường sẽ tối hơn các phần khác như: mũi và phần má. Tương tự phần giao giữa mắt và mũi (còn gọi là brige of the nose) thì nó sáng hơn hai bên mắt. Nhưng khi áp cùng kernel này lên vùng khác thì nó trở nên không thích hợp.

Để giúp chọn ra những đặc trưng tốt nhất trong hơn 160000 features thì Adaboost ra đời. Ý tưởng: tạo ra bộ phân lớp mạnh nhờ việc tổ hợp tuyến tính của các tham số của bộ phân lớp yếu.

Nó được xem là yếu bởi vì nó không thử tự mình phân lớp cho ảnh được mà phải tập hợp nhiều bộ phân lớp khác. Giống như câu, "Đoàn kết là sức mạnh".

Với mỗi cửa sổ window lướt qua ảnh, các đặc trưng Haar được tính và ta sẽ tìm ra mức threshold để tách biệt giữa có đối tượng và không có đối tượng bằng việc so sánh sự khác biệt giữa các đặc trưng đó. Dĩ nhiên sẽ tồn tại độ lỗi trượt lớp.

Nhưng ta chỉ cần đặt một ngưỡng cho độ lỗi ấy nhỏ (cỡ 0.05) thì model sẽ thông qua quá trình học để điều chỉnh tham số đến khi thỏa mức lỗi ấy hoặc đạt đủ số lượng đặc trưng mong muốn.

Sau cùng, ta thu được cỡ 6000 đặc trưng nhưng tưởng tượng với mỗi của sổ 24x24 mà ta áp 6000 đặc trưng vào để so khớp coi xem có đối tượng trong đó không thì quả là tốn thời gian.

Tác giả cũng đã có giải pháp cho việc này: đó là đa phần các vùng trong ảnh không có chứa đối tượng. Do đó, nếu khi lướt window vào một vùng ảnh kiểm tra và vùng này không có chứa đối tượng thì bỏ qua một bên, không xử lý nữa. Ta chỉ tập trung xử lý cho vùng có thể chứa đối tượng.

Bước 4: Cascading classifiers

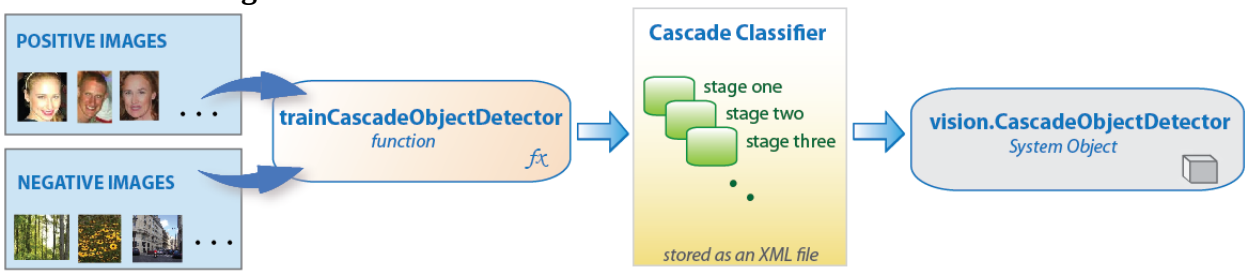


Figure 17: Qui trình hoạt động của Cascading classifiers (Trích từ: http://www.willberger.org/cascade-haar-explained/)

Khái niệm này ra đời cũng dựa trên giải pháp đó, thay vì áp 6000 đặc trưng lên một window thì ta gom nhóm các đặc trưng trong nhiều giai đoạn của bộ phân lớp và lần lượt áp các đặc trưng đó lên window đang xét. Nếu lần đầu mà đã bị loại thì ta bỏ qua window đó luôn và không xét gì tới các đặc trưng còn lại của nó nữa. Nếu thành công, nó sẽ được đưa vào giai đoạn tiếp theo và lặp lại cho đến khi nó được phân loại là có đối tượng.

Quay trở lại hệ thống ALPR nhóm phát triển. Ở đây, nhóm sử dụng model train sẵn bởi một nhóm nghiên cứu và phát triển ứng dụng ALPR ở Việt Nam: https://thigiacmaytinh.com/model-cascade-da-huan-luyen/ và nhóm cũng sử dụng bộ dataset về xe máy của trang https://thigiacmaytinh.com/tai-nguyen-xu-ly-anh/tong-hop-data-xu-ly-anh/ của nhóm này luôn.

2. Character extraction:

Bước này, nhóm sử dụng thuật toán floodfill để tô loang hết vùng kí tự trong biển số với mục đích rút trích và tách được các vùng kí tự ra khỏi ảnh ban đầu.

Cách thức:

- Trước tiên nhóm thực hiện gieo các seeds xung quanh tâm của vùng bounding box (có tính góc xoay) và thực hiện floodfill để tô.

- Sau đó, nhóm thu được ảnh mask (ảnh đã tô) và thực hiện tìm contour rồi đóng bounding box cho các vùng ảnh kí tự.

Figure 18: Ảnh mask và ảnh inverse của mask

Figure 19: Ảnh contour và đóng bounding box

- Cuối cùng, nhóm thực hiện tách các vùng ảnh dựa vào tọa độ của các bounding box.

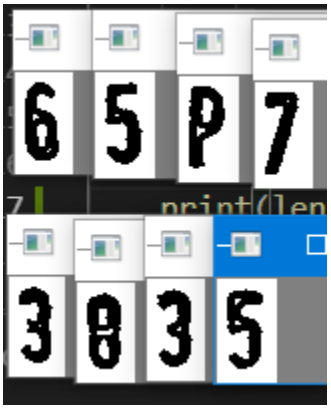
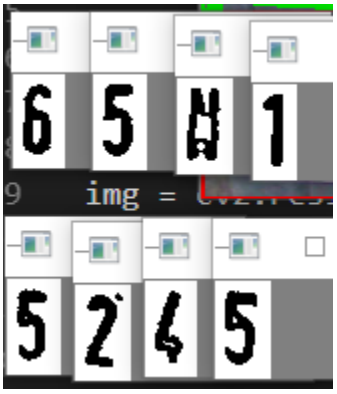
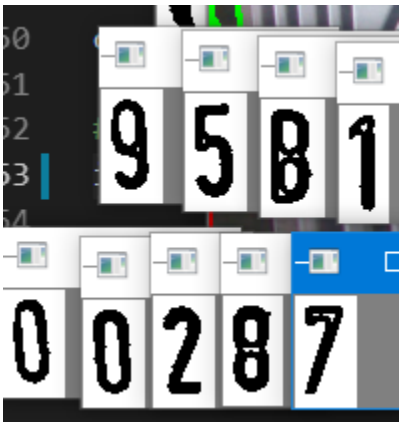
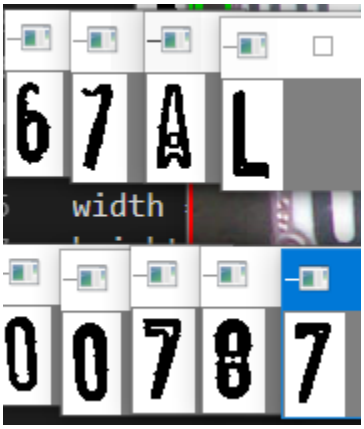


Figure 20: Ảnh nhận diện kí tự đã tách. Nhóm không hiểu vì lí do gì mà các kí tự tách ra khi bỏ vào OCR thì nó lại không nhận diện ra

Các kết quả thực nghiệm:







Nhận xét:

- Việc phát hiện biển số bằng Haar Cascade rất tốt và độ chính xác rất cao.
- Việc tách kí tự hoàn thành rất tốt, gần như trong lúc test rất hiếm trường hợp bị tách thiếu sót kí tự.
- Tuy nhiên, việc nhận diện lại không thành công. Nhóm cũng đã cố gắng chỉnh sửa rất nhiều nhưng Tesseract vẫn không cho ra kết quả nhận diện mong muốn. Nhóm cũng không rõ vì lí do gì.
- Nhìn chung, Haar Cascade hoạt động rất tốt nhưng nếu scale của ảnh quá lớn, góc nghiêng lớn hay ảnh quá tối thì vẫn không cho ra kết quả.

E Hướng dẫn sử dụng:

I. Phiên bản 1: Dành cho xe ô tô

Chương trình được chạy bằng command line.

Cú pháp:

python <Ten chuong trinh.py> -i <Input image> -o <Option: 0 hoặc 1> -m <Model: Path tới pretrained model của SVM> -c <Classify: Sử dụng SVM classifier không: 0 hoặc 1>

Ý nghĩa:

- <Ten chuong trinh.py>: file chạy chương trình
- <Input image>: đường dẫn đến ảnh input
- <Option>: Lựa chọn hiển thị tất cả các bước làm hay không. Nếu có thì nhập 1, không thì nhập 0. Mặc định chương trình sẽ để là 0 khi bạn không nhập giá trị này.
- <Model>: Truyền vào model train sẵn của SVM. Mặc định: "SVM.xml"
- <Classify>: Lựa chọn có dùng SVM để phân lớp cho biển số hay không gồm 2 giá trị: 0 hoặc 1. Chương trình mặc định là 1.

Lưu ý: Chương trình dùng argparse để giúp command line trở nên trực quan và dễ sử dụng hơn khi truyền tham sao. Bạn có thể nhập lệnh help để trở giúp:

python main.py -h

```
(opencv-env) E:\K16\Junior\TGMT\ALPR-project\Version 1>python main.py -h
usage: main.py [-h] -i INPUT [-m MODEL] [-c CLASSIFY] [-o OPTION]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Path to input image
  -m MODEL, --model MODEL
                        Path to SVM pretrained model
  -c CLASSIFY, --classify CLASSIFY
                        Use SVM to classify or not
  -o OPTION, --option OPTION
                        Show step by step
```

Ví dụ: Để chạy chương trình ALPR với tập ảnh ".\test_images\IMG_0378.jpg":

python 1612174_1612269_1612274_Lab03.py -i ".\test_images\IMG_0378.jpg" – o 0

II. Phiên bản 2: Dành cho xe máy

Chương trình được chạy bằng command line.

Cú pháp:

python <Ten chuong trinh.py> -i <Input image> -m <Model: Path tới pretrained model của Haar Cascade>

Ý nghĩa:

- <Ten chuong trinh.py>: file chạy chương trình
- <Input image>: đường dẫn đến ảnh input
- <Model>: Truyền đường dẫn của model train sẵn của Haar Cascade. Mặc định: "GreenParking_num-3000-LBP_mode-ALL_w-30_h-20.xml".

Lưu ý: Chương trình dùng argparse để giúp command line trở nên trực quan và dễ sử dụng hơn khi truyền tham sao. Bạn có thể nhập lệnh help để trở giúp:

python detect_plate.py -h

```
(opencv-env) E:\K16\Junior\TGMT\ALPR-project\Version 2>python detect_plate.py -h
usage: detect_plate.py [-h] -i INPUT [-m MODEL]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Path to input image
  -m MODEL, --model MODEL
                        Path to Haar Cascade pretrained model
```

Ví dụ:

python detect_plate.py -i E:\K16\Junior\TGMT\ALPR-project\Bike_back\2.jpg

F Tham khảo:

[1] Baggio, D. L. (2012). 5. Number Plate Recognition Using SVM and Neural Networks. In Mastering OpenCV with Practical Computer Vision Projects (6th ed., pp. 161-188). Birmingham, UK: Packt Publishing

[2] https://github.com/kagan94/Automatic-Plate-Number-Recognition-APNR.git

[3] https://github.com/MicrocontrollersAndMore/OpenCV_3_License_Plate_Recognition_Python.git

[4] https://docs.opencv.org/3.4.3/d7/d4d/tutorial_py_thresholding.html

[5] https://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html
[6] https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html
[7] https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html
[8] https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/
[9] https://cvisiondemy.com/license-plate-detection-with-opencv-and-python/
[10] Dataset cho ô tô:
http://www.medialab.ntua.gr/research/LPRdatabase.html?fbclid=IwAR0d_5jeUecAGPX2Dw23p5GFpArKSRAsLSZSD-jOU_RGhGbMOU2JydKsRq8
[11] OpenCV Haar Cascade
https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
[12] OpenCV SVM
https://docs.opencv.org/3.4/d1/d73/tutorial_introduction_to_svm.html
[13] Willberger: Deep learning Haar Casecade explained
http://www.willberger.org/cascade-haar-explained/