**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**Viện Công nghệ Thông tin và Truyền thông**

****

**Báo cáo Project II**

**Đề tài:**

***Nhận diện biển số xe bằng OpenCV***

Sinh viên thực hiện:

Lê Quang Huy 20161813

Giáo viên hướng dẫn: **Đỗ Duy Hiệp**

**HÀ NỘI – 2019**

Nội dung

[1. Mô tả đề tài: 3](#_Toc10276167)

[2. Tiền xử lý: 3](#_Toc10276168)

[2.1. Chuyển ảnh sang dạng ảnh xám (Grayscale): 3](#_Toc10276169)

[2.2. Lọc số ảnh (Image Filters): 3](#_Toc10276170)

[2.3. Phân ngưỡng ảnh (Threshold): 4](#_Toc10276171)

[3. Xác định vùng biển số: 6](#_Toc10276172)

[3.1. Tìm đường viền (Contours): 6](#_Toc10276173)

[3.2. Xác định vùng có khả năng là biển số: 8](#_Toc10276174)

[4. Nhận dạng kí tự: 9](#_Toc10276175)

[4.1. Thư viện Tesseract: 10](#_Toc10276176)

[4.2. Cắt kí tự trên vùng biển số: 10](#_Toc10276177)

[4.3. Giải thuật dự đoán k-NN: 10](#_Toc10276178)

[4.4. Giải thuật phân lớp SVM: 11](#_Toc10276179)

# Mô tả đề tài:

# Tiền xử lý:

## Chuyển ảnh sang dạng ảnh xám (Grayscale):

* Ảnh xám là ảnh có giá trị màu của mỗi điểm ảnh nằm trong khoảng [0, 255]. Ảnh màu RGB là ảnh mà mỗi điểm ảnh có giá trị màu được tổng hợp bởi 3 giá trị màu Red, Green, Blue, mỗi giá trị nằm trong khoảng [0, 255].
* Với mỗi điểm ảnh trong ảnh màu, giá trị R, G, B được nhân với các hằng số nhất định để tạo ra giá trị mới. Ví dụ giá trị màu của điểm ảnh Y có thể được tính bởi

Y =

* Sử dụng hàm chuyển ảnh sang dạng xám trong OpenCV:

cv::cvtColor(InputArray src, OutputArray dst, COLOR\_BGR2GRAY);

* Trong hàm này giá trị màu của điểm ảnh Y được tính bởi:

*Y* = 0.299*R*+0.587*G*+0.114*B*

## Lọc số ảnh (Image Filters):

* Một hệ thống dùng để làm biến dạng sự phân bố tần số của các thành phần tín hiệu theo các chỉ tiêu gọi đã cho được gọi là bộ lọc số. Sử dụng các bộ lọc làm mở (Blur), làm trơn (Smooth) để giảm nhiễu ảnh và tăng độ chính xác.
* Bộ lọc số ảnh gồm:
  + Một ma trận lọc (Mask)
  + Một phép toán được định nghĩa trước thực hiện phép toán trên các điểm ảnh được bao phủ mới ma trận lọc.
* Quy trình:
  + Xác định điểm trung tâm của ma trận lọc
  + Tại điểm (x, y) đang xét (trùng với tâm ma trận lọc), thực hiện phép toán lọc trên các điểm lân cận (bị mặt nạ bao phủ);
  + Ghi nhận kết quả phép lọc là giá trị mức xám của điểm ảnh (x, y) trong ảnh đầu ra.
  + Lần lượt trượt ma trận lọc tới những điểm chưa xét. Lặp lại bước 2 Quá trình lọc kết thúc khi điểm trung tâm của mặt nạ lần lượt thăm hết từng điểm ảnh của ảnh đầu vào, ta thu được kết quả ảnh đã được lọc ở đầu ra.
* Ví dụ: với bộ lọc tuyến tính và ma trận lọc có dạng 3x3:

w(x, y) = w(−1, −1)f(x − 1, y − 1) + w(−1,0)f(x − 1, y) + ⋯ + w(0,0)f(x, y) + ⋯ + w(1,1)f(x + 1, y + 1).

|  |  |  |
| --- | --- | --- |
| w(-1, -1) | w(0, -1) | w(1, -1) |
| w(-1, 0) | w(0, 0) | w(1, 0) |
| w(-1, 1) | w(0, 1) | w(1, 1) |

|  |  |  |
| --- | --- | --- |
| f(x-1, y-1) | f(x, y-1) | f(x+1, y-1) |
| f(x-1, y) | f(x, y) | f(x+1, y) |
| f(x-1, y+1) | f(x, y+1) | f(x+1, y+1) |

Ma trận điểm ảnh Ma trận lọc

* Áp dụng bộ lọc tuyến tính trong lọc nhiễu, làm trơn: Đầu ra của một bộ lọc không gian tuyến tính là mức trung bình của các điểm ảnh lân cận bị bao phủ bởi mặt nạ lọc (Mean Filter - bộ lọc trung bình). Bằng cách thay thế các giá trị của mỗi điểm ảnh trong hình ảnh bằng mức trung bình của cường độ các điểm lân cận được giảm các chuyển tiếp "mạnh" trong cường độ. Vì nhiễu ngẫu nhiên thường bao gồm các chuyển đổi mạnh trong cường độ nên ứng dụng của lọc tuyến tính là giảm nhiễu.
* Làm mờ gauss (Gaussian Blur): Làm mờ ảnh bằng hàm gauss:

G(x, y) =

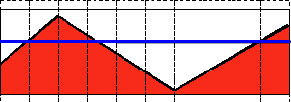
Trong không gian 2 chiều hàm này công thức này sản sinh ra những đường viền là những đường tròn đồng tâm, tuân theo logic phân tán Gauss từ điểm trung tâm. Giá trị này dùng để xây dựng ma trận lọc. Phép toán sẽ được sử dụng ở đây là phép tích chập. Giá trị mới của điểm ảnh sẽ là trung bình của các điểm ảnh xung quanh nó. Trong ma trận lọc này, giá trị của điểm trung tâm là lớn nhất và giá trị sẽ nhỏ hơn với các điểm càng xa trung tâm. Tính chất này giúp giữ lại đường viền và biên cũng như làm mờ một cách đồng bộ hơn so với các bộ lọc tuyến tính khác.

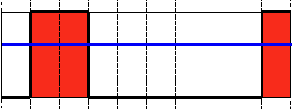
* Hàm Gaussian Blur trong OpenCV:

cv::GaussianBlur(InputArray src, OutputArray dst, cv::Size kernel\_size, double *sigmaX*, double *sigmaY,* int borderType);

## Phân ngưỡng ảnh (Threshold):

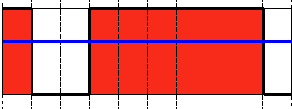
* Phân ngưỡng ảnh là phương pháp phân đoạn ảnh đơn giản nhất. Từ dạng ảnh xám, ảnh sẽ được chuyển sang dạng nhị phân. Mục đích phương pháp này là tách các vùng của hình ảnh tương ứng với các đối tượng muốn phân tích. Sự phân tích này dựa trên sự thay đổi cường độ giữa các điểm ảnh thuộc vùng muốn phân tích và các điểm ảnh nền liền kề. Ảnh đã phân ngưỡng chỉ có hai giá trị là 0 (đen) hoặc 255 (trắng). Có thể coi đen là 0 và trắng là 1, ảnh đầu ra là ảnh nhị phân.

Ví dụ với một ngưỡng T, giá trị các điểm ảnh được biểu diễn trong hình sau, kết quả sau khi phân ngưỡng sẽ là:



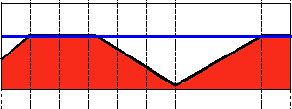
Thresh binary

dst(x, y) =



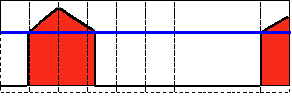
Thresh binary inverted

dst(x, y) =

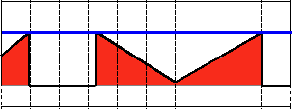


Truncate

dst(x, y) =

Threshold to Zero

dst(x, y) =



Threshold to Zero

dst(x, y) =

* Sử dụng phân ngưỡng trong OpenCV:

cv::threshold(InputArray src, OutputArray dst, double thresh, double maxVal, int type)

* Phương pháp trên dễ đưa ra kết quả không tốt nếu ảnh bị ảnh hưởng bởi cường độ sáng không đều, hơn nữa việc chọn ngưỡng phù hợp là rất khó.
* Một cách chọn ngưỡng phổ biến: thuật toán otsu:
  + **Bước 1:** Xác định T1. Giá trị cho T1 ban đầu nên chọn là (0+255) / 2 = 128.
  + **Bước 2:** Phân loại thành 2 nhóm điểm ảnh:
    - Type1: tất cả các điểm ảnh cường độ <= T
    - Type2: tất cả các điểm ảnh có cường độ > T
  + **Bước 3:** Tính giá trị cường độ sáng trung bình cho Type1 (iAverage1) và Type2 (iAverage2).
  + **Bước 4:** Tính giá trị T2 theo công thức (iAverage1 + iAverage2) /2.
  + **Bước 5:** So sánh T1 và T2. Nếu giá trị chênh lệch của T1 và T2 <= Delta (một giá trị cho trước) thì T2 chính là **T** cần tìm. Nếu giá trị chênh lệch của T1 và T2 > Delta Deltal thì quay lại **Bước 1.**
* Otsu là phương pháp phân ngưỡng phổ biến để phân chia ảnh thành 2 lớp background và foreground, phương sai trong lớp là tối thiểu và phương sai giữa các lớp là tối đa.
* Để giải quyết vấn đề cường độ sáng thay đổi trong ảnh, người ta đưa ra phương pháp phân ngưỡng khác là phân ngưỡng động. Phương pháp này chia ảnh thành nhiều khu vực, với mỗi khu vực chọn ra một giá trị ngưỡng T phù hợp với khu vực đấy. Phương pháp phân ngưỡng cho mỗi khu vực tương tự trên.

cv::adaptiveThreshold(InputArray src, OutputArray dst, double maxVal, int adaptiveMethod, int thresholdType, int blockSize, double C);

* Kết quả của phép phân ngưỡng đơn giản với ngưỡng 127:

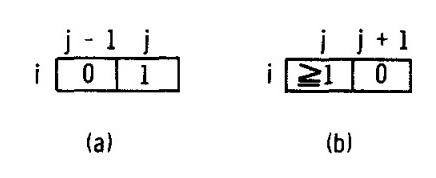
 

# Xác định vùng biển số:

* Sau bước tiền xử lý ta thu được ảnh đã phân ngưỡng.

## Tìm đường viền (Contours):

* Quét ảnh nhị phân đầu vào bằng phép quét raster (quét ngang từng dòng) và ngắt quét raster khi pixel được tìm thấy thỏa mãn điều kiện cho điểm bắt đầu của viền ngoài (2a) hoặc viền lỗ (2b). Nếu thỏa mãn cả 2 điều khiện trên, được coi là điểm bắt đầu của viền ngoài và chỉ định một id (số thứ tự) cho đường mới tìm đc. Số thứ tự của đường viền được kí hiệu là NBD.

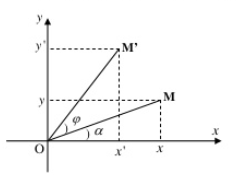


* + Hình (a) là điều kiện xác định đường viền ngoài, hình (b) xác định đường viền lỗ
* Xác định đường viền cha của đường viền mới: trong quá trình quét raster, lưu lại số thứ tự LNBD của đường viền gần nhất (gọi là B’). Đường viền này sẽ là cha hoặc anh em của đường viền mới. Quy tắc xác định:
  + Nếu B và B’ cùng loại thì B và B’ là anh em (có cùng cha)
  + Ngược lại, B’ là cha của B
* Theo đường viền được tìm thấy từ điểm bắt đầu, đánh dấu các pixel của viền:
  + Nếu đường viền đang xét nằm giữa 0-component chứa pixel (p,q+1) và 1-component chứa pixel (p,q), thay đổi giá trị (p,q) thành -NBD
  + Ngược lại, đặt (p,q) thành NBD trừ khi pixel (p,q) nằm trên đường viền đã được xét.
  + 2 điều kiện trên ngăn cản pixel (p,q) trở thành đường viền theo điểm bắt đầu của đường viền lỗ đã xét. Giá trị NBD dương và âm tương ứng với các nhãn “l” và “r” của đường viền theo thuật toán trong [1].
* Sau khi xác định một đường viền, raster tiếp tục chạy từ điểm đang tạm ngắt.
* Tính chất:
  + Điểm trên trái của một 1-component thỏa mãn hình (a) phía trên. Đường viền B đã xét từ pixel là đường viền ngoài (tức tồn tại 0-component chứa bao quanh . Sau khi đường viền này đã xét, đường viền tương tự sẽ k bao giờ xét nữa.
  + là điểm trên cùng bên trái của một lỗ thỏa mãn hình (b), đường viền B đã xét từ pixel là đường viền lỗ (1-component chứa bao quanh lỗ ).
  + Các đường viền có tương ứng 1-1 với các 1-component hoặc lỗ.
* Hoạt động:
* Trong thuật toán này, khi nói lân cận, nghĩa là 4 (hoặc 8) lân cận liên kết xung quanh trong trường hợp 4 (hoặc 8) liên kết xung quanh.
* Gọi ảnh đầu vào là . Khởi tạo NBD ban đầu = 1 (khung của F tạo thành một đường viền lỗ đặc biệt, lấy thứ tự là 1).
* Quét qua ảnh bằng raster và thực hiện các bước sau với mỗi pixel . Mỗi lần bắt đầu quét qua một dòng mới của ảnh, gán LNBD = 1:
  + (1): Chọn loại cho đường viền đang xét
    - (a) Nếu và thì là điểm bắt đầu của một đường viền ngoài. Tăng giá trị NBD, gán
    - (b) Nếu và , là điểm bắt đầu của một đường viền lỗ. Tăng NBD, gán , và LNBD nếu .
    - (c) Còn lại, đi tới (4).
  + (2): Chọn cha của đường viền đang xét: dựa vào loại của đường viền đang xét và đường viền ứng với số thứ tự LNBD (đường viền cuối cùng gặp trên cùng dòng), quyết định đường viền cha của đường viền đang xét theo quy tắc đã nếu phía trên.
  + (3): Từ điểm bắt đầu , đi theo đường viền đã được phát hiện.
    - (3.1): Bắt đầu từ , kiểm tra các pixel lân cận của ngược chiều kim đồng hồ và tìm 1 pixel khác 0. Đặt là pixel đầu tiên khác 0 tìm được. Nếu không tìm được pixel nào, đặt và đi tới (4) .
    - (3.2): gán và .
    - (3.3): Bắt đầu từ phần từ tiếp theo của ( = sau bước 3.2), xét ngược chiều kim đồng hồ các điểm lân cận của ( = sau bước 3.2) và tìm phần từ khác 0 đầu tiên, đặt nó là (Bước này là xét phần tử khác 0 tiếp theo lân cận với , giống với bước 3.1)
    - (3.4): Thay đổi giá trị của :
      * (a): Nếu → .
      * (b): Nếu và → .
      * (c): Trường hợp còn lại, không thay đổi
    - (3.5): Nếu và (quay trở lại điểm ban đầu), đi tới (4). Ngược lại, gán , và quay trở lại (3.3).
  + (4): Nếu , gán và tiếp tục quét raster từ pixel . Thuật toán kết thúc khi đạt tới góc dưới bên phải.

## Xác định vùng có khả năng là biển số:

* Duyệt toàn bộ đường viền đã tìm được, sử dụng hàm minAreaRect để bao lấy đường viền, kết quả trả về là hình chữ nhật.
* Xác định vùng có khả năng của biển số qua các đặc trưng:
  + Chiều cao, chiều rộng
  + Tỉ lệ điểm ảnh trắng/đen.
  + Diện tích.
* Với tập các hình chữ nhật tìm được, tìm điểm chính giữa và góc của hình chữ nhật. Sử dụng phép xoay để lấy hình chữ nhật tại vị trí đúng. Để thực hiện phép xoay ta sử dụng phép biến đổi Affine.

cv::minAreaRect(InputArray points);

* Phép xoay trong hệ tọa độ cực:
* 
* Hay

Ta thu được ma trận biến đổi trong hệ Oxy:

* Phép biến đổi Affine trong hệ tọa độ đồng nhất

Q = P.M

Trong đó Q là ma trận kết quả, P là ma trận ban đầu và M là ma trận biến đổi. Ta có ma trận của phép quay và tịnh tiến trên trong hệ tọa độ đồng nhất:

* Ma trận biến đổi với phép quay quanh tâm bất kì là kết quả của phép quay và tịnh tiến (tịnh tiến tâm quay về gốc tọa độ, thực hiện phép quay và tịnh tiến lại vị trí ban đầu): M = MT(xc, yc) . MQ . MT(-xc, -yc)
* Để lấy ma trận xoay của phép biến đổi affine trong OpenCV:
* Để thực hiện phép biến đổi Affine trong OpenCV:

cv::getRotationMatrix2D(Point2f center, float angle, float scale);

* Kết quả phép biến đổi này xoay ảnh đúng 1 góc bằng với góc của hình chữ nhật. Lúc này vùng này đã xoay về đúng góc, cắt vùng chữ nhật đấy ra.

cv::warpAffine(InputArray src, OutputArray dst, Mat rotationMatrix, cv::Size size);

# Nhận dạng kí tự:

* Với tập những vùng có khả năng là biển số thu được, đưa vào bộ nhận dạng kí tự.

## Thư viện Tesseract:

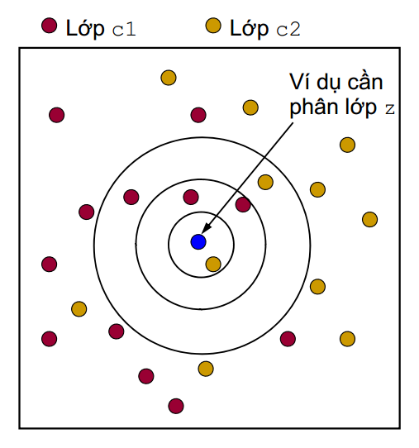
* Tesseract là thư viện OCR nổi tiếng do độ chính xác cao hơn hẳn các thư viên khác. Tesseract có thể chạy độc lập hoặc tích hợp với OpenCV đều được. Nếu chạy độc lập thì Tesseract sử dụng thư viện leptonica để đọc hình ảnh.

## Cắt kí tự trên vùng biển số:

* Việc cắt kí tự trên vùng biển số được thực hiện tương tự với việc cắt biển số ở trên.

## Giải thuật dự đoán k-NN:

* Giải thuật phân lớp k-NN:
  + Đầu vào của bài toán  
    • Mỗi ví dụ x được biểu diễn là một vector n chiều trong không gian các vector X ∈ R­­­n  
    • x = (x1, x2,…, xn) trong đó xi (∈R) là một số thực
  + Bài toán phân lớp (classification)



Để phân lớp cho một ví dụ mới z, xác đinh tập NB(z) là k ví dụ có khoảng cách gần nhất với z tính theo 1 hàm khoảng cách d. Phân z vào lớp nào chiếm số đông trong NB(z).

* Quá trình học:
  + Xử lý dữ liệu học: đọc ảnh các kí tự trong tập training, chuyển các kí tự đấy qua dạng ảnh xám, và chuyển ảnh sang dạng vector

Mat.convertTo(OutputArray dst, CV\_32FC1);

Mat.reshape(1, 1);

* + Mỗi ảnh đọc vào, đẩy 1 vector vào tập train và 1 vector là nhãn của nó vào tập classfication

Mat traindata.push\_back(Mat train);

Mat classification.push\_back(char class);

* + Gọi hàm k-NN của OpenCV:

Ptr<KNearest> knn = cv::ml::KNearest::create();

knn->train(traindata, ROW\_SAMPLE, classification);

* Nhận dạng một ví dụ mới:
  + Chuyển các kí tự đấy qua dạng ảnh xám, và chuyển ảnh sang dạng vector.

Mat.convertTo(OutputArray dst, CV\_32FC1);

Mat.reshape(1, 1);

* + Dự đoán nhãn của ví dụ mới:
* Chia tập dữ liệu thành 2 phần 1 phần để train và một phần để test, kết quả đoán nhận đạt 85.7143%.

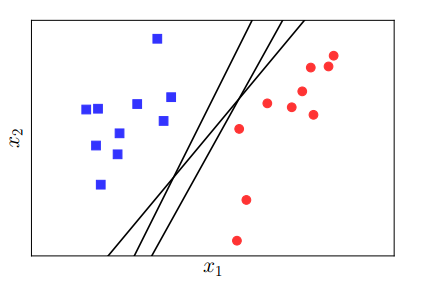
Mat detect(30, 30, CV\_32F);

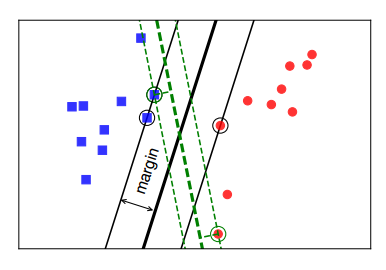
knn->findNearest(src, 1, detect);

float fltChar = (float)detect.at<float>(0, 0);

result = (char) (int) fltChar;

## Giải thuật phân lớp SVM:

* + Đầu vào của bài toán  
    • Mỗi ví dụ x được biểu diễn là một vector n chiều trong không gian các vector X ∈ R­­­n  
    • x = (x1, x2,…, xn) trong đó xi (∈R) là một số thực
  + Bài toán phân lớp (classification)
    - Quá trình học: Giả sử có tập điểm (vector), cần tìm một siêu phẳng phân chia 2 lớp này thành 2 phần riêng biệt. Trong hình vẽ bên có thể thấy ta tìm được vô số siêu phẳng để phân lớp 2 dữ liệu xanh và đỏ.



* + - Margin của lớp: khoảng cách nhỏ nhất từ 1 điểm thuôc lớp đến mặt phẳng phân chia. Ta cần phải cực đại margin của các lớp vì margin rộng hơn đem đến hiệu quả phân lớp tốt hơn.
    - Giải đoạn phân lớp: Khi một có ví dụ mới, xác định xem ví dụ này thuộc vào phân lớp nào.
* Quá trình học:
  + Xử lý dữ liệu học: đọc ảnh các kí tự trong tập training, chuyển các kí tự đấy qua dạng ảnh xám, và chuyển ảnh sang dạng vector

Mat.convertTo(OutputArray dst, CV\_32FC1);

Mat.reshape(1, 1);

* + Mỗi ảnh đọc vào, đẩy 1 vector vào tập train và 1 vector là nhãn của nó vào tập classfication

Mat traindata.push\_back(Mat train);

Mat classification.push\_back(char class);

* + Gọi hàm k-NN của OpenCV:

Ptr<SVM> svm = cv::ml::SVM::create();

svm->train(traindata, ROW\_SAMPLE, classification);

* Nhận dạng một ví dụ mới:
  + Chuyển các kí tự đấy qua dạng ảnh xám, và chuyển ảnh sang dạng vector.

Mat.convertTo(OutputArray dst, CV\_32FC1);

Mat.reshape(1, 1);

* + Dự đoán nhãn của ví dụ mới:

float r = svm->predict(temp.reshape(1, 1));

result = (char) (int) r;

* Chia tập dữ liệu thành 2 phần 1 phần để train và một phần để test, kết quả đoán nhận đạt 87.8419%.