TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO
MÔN HỌC PROJECT II
Đề tài: Nhận diện biển số xe

Giảng viên hướng dẫn: **Ts. Phạm Đăng Hải**

Sinh viên thực hiện: **Nguyễn Đức Trọng**

MSSV: 20164801

Hà Nội 2019

I. Giới thiệu

1. Tổng quan

Trong những năm gần đây, trí tuệ nhân tạo đang phát triển cực kì mạnh mẽ và tác động không nhỏ tới đời sống con người. Việc học tập, tìm hiểu về trí tuệ nhân tạo là cần thiết trong thời đại công nghiệp 4.0.

Hiện nay, số lượng phương tiện tham gia giao thông hay ra vào các tòa nhà là rất lớn. Vì vậy đòi hỏi cần có một chương trình nhận diện biển số xe một cách tự động. Do đó, em đã chọn đề tài "Nhận diện biển số xe" cho môn học Project II. Chương trình không tránh khỏi những thiếu sót, mong thầy thông cảm và góp ý cho em. Em xin chân thành cảm ơn!

2. Nhiệm vụ đề tài

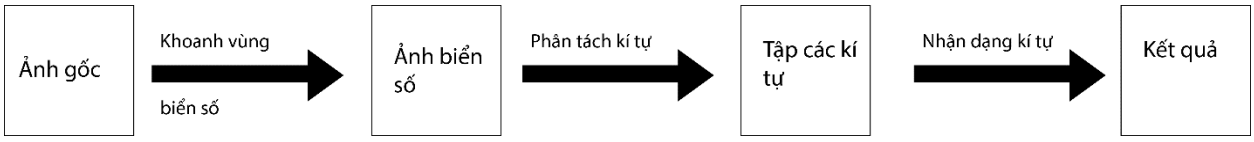
Đề tài sẽ thực hiện đọc vào hình ảnh chứa biển số xe và xuất ra biển số xe tương ứng. Chương trình hoạt động tương đối tốt trong điều kiện ánh sáng tốt, ảnh ít yếu tốt gây nhiễu, kích thước biển số đủ lớn, phù hợp với các bãi trông giữ xe. Đối với điều kiện khó khăn hơn, chương trình nhận diện ở mức trung bình, thường nhận thiếu kí tự, nhận diện lẫn lộn giữa các kí tự giống nhau.

Nội dung tìm hiểu:

- Tìm hiểu về một số thuật toán, phương pháp xử lí ảnh, áp dụng vào chương trình thông qua thư viện opencv.
- Tìm hiểu thuật toán học máy K-nearest.
- Thiết kế chương trình.

3. Khái quát hoạt động

3.1. Tổng quan



Hình 1: Quy trình hoạt động của chương trình

3.2. Khoanh vùng biển số

- Chương trình đọc vào hình ảnh ban đầu, sau đó xác định các vùng có thể là biển số dựa vào các yếu tố về kích thước, tỷ lệ chiều dài, rộng.

Hình 2: Hình ảnh gốc khi đọc vào

- Hình ảnh biển số được tách ra:

Hình 3: Hình ảnh biển số được tách ra

3.2. Phân tách các kí tự

- Từ biển số phân tách được, chương trình sẽ tìm các kí tự hợp lệ (dựa trên chiều dài, rộng, diện tích, tỷ lệ giữa dài và rộng), chia chúng thành các nhóm có kích thước, vị trí tương tự nhau. Nhóm các kí tự có nhiều phần tử nhận sẽ được chọn để nhận dạng.

3.3. Nhận dạng kí tự

Các kí tự được phân tách, thông qua k-nearest đã được train, sẽ được nhận dạng và kiểm tra lại định dạng (biển số xe máy ở Việt Nam thường có 8-10 kí tự, với 2 kí tự đầu và 5 kí tự cuối chắc chắn là số).

II. Cơ sở lý thuyết

1. Chuyển ảnh RGB sang ảnh xám (Gray):

- Ảnh xám là ảnh có giá trị mỗi điểm ảnh nằm trong khoảng [0, 255] (với độ sâu 8-bit). Ảnh RGB là ảnh gồm 3 kênh màu, mỗi điểm ảnh được biểu diễn bởi 3 giá trị R, G, B, mỗi giá trị nằm trong khoảng [0, 255].
- Để chuyển ảnh từ RGB sang ảnh xám, ta thực hiện một phép biến đổi $X = f(R, G, B)$ với X là giá trị điểm ảnh của ảnh xám; R, G, B là giá trị các kênh màu tương ứng của ảnh RGB.
- Dưới đây là một số phép biến đổi cơ bản:
    - Thuật toán 1: $X = (\max(R, G, B) + \min(R, G, B))/2$
    - Thuật toán 2: $X = (R + G + B)/3$
    - Thuật toán 3: $X = RC * R + GC * G + BC * B$ với RC = RedConstant, GC = Green-Constant, BC = BlueConstant là các hằng số được tính toán trước. Ở phương pháp này, GC thường có giá trị lớn nhất và BC có giá trị nhỏ nhất do mức độ nhạy cảm của mắt người đối với các màu là khác nhau.
- Hàm chuyển đổi ảnh RGB sang ảnh xám trong opencv:

```
cvtColor(InputArray src, OutputArray dst, COLOR_BGR2GRAY);
```

- Hàm này sử dụng thuật toán 3 để chuyển đổi, với giá trị các hằng số tương ứng là

$$RC = 0.299, GC = 0.587, BC = 0.114$$

2. Lọc nhiễu bằng Gaussian blur

- Nhiễu được định nghĩa trong xử lý ảnh là những điểm ảnh có giá trị độ xám trội hơn so với cục bộ, tức là chênh lệch lớn so với các điểm ảnh lân cận. Nhiễu ảnh được gây ra bởi các yếu tố như thời tiết, thiếu sáng, thiết bị chụp hình,…
- Gaussian blur là bộ lọc làm mờ ảnh, sử dụng lý thuyết hàm Gauss để tính toán việc chuyển đổi của các pixel

Quy trình của làm mờ ảnh bằng Gaussian blur:

- B1: Xây dựng ma trận kernel, xác định tâm của ma trận
    - Ma trận kernel có kích thước $m * n$, được xây dựng bằng công thức:

      $k(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$ với mảng một chiều

      $k(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ với mảng 2 chiều với $\frac{m}{2} \geq x \geq -\frac{m}{2}$, $\frac{n}{2} \geq y \geq -\frac{n}{2}$
    - σ là độ lệch chuẩn trong phân phối Gauss.
    - Tâm của ma trận thường là điểm nằm giữa, có toạ độ $(0,0)$.

- Trong không gian 2 chiều, 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 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.

- B2: Dịch chuyển ma trận kernel lần lượt qua tất cả điểm trên ảnh, tâm của kernel ứng với điểm ảnh đang xét. Mỗi lần dịch chuyển, thực hiện phép tích chập giữa kernel và ma trận trên ảnh tương ứng.
  - Giả sử kernel có kích thước $m * n$, ma trận kernel được thể hiện bởi hàm $k(x, y)$, ma trận của ảnh $f(x, y)$
  - Khi ấy, phép tích chập được tính như sau:

$$k(x,y) * f(x,y) = \sum_{u=-(m/2)}^{m/2} \sum_{v=-(n/2)}^{n/2} k(u,v) f(x+u, y+v)$$

  - Ví dụ:

Ma trận hình ảnh

f(1,1)	f(1,2)	f(1,3)	f(1,4)	f(1,5)
f(2,1)	f(2,2)	f(2,3)	f(2,4)	f(2,5)
f(3,1)	f(3,2)	f(3,3)	f(3,4)	f(3,5)
f(4,1)	f(4,2)	f(4,3)	f(4,4)	f(4,5)
f(5,1)	f(5,2)	f(5,3)	f(5,4)	f(5,5)

Ma trận kernel

k(-1,-1)	k(-1,0)	k(-1,1)
k(0,-1)	k(0,0)	k(0,1)
k(1,-1)	k(1,0)	k(1,1)

  - Giả sử xét điểm $(1,1)$. Khi ấy, điểm $(1,1)$ của ma trận kết quả được tính bằng $r(1,1) = f(0,0) * k(-1,-1) + f(0,1) * k(-1,0) + f(0,2) * k(-1,1) + f(1,0) * k(0,-1) + f(1,1) * k(0,0) + f(1,2) * k(0,1) + f(2,0) * k(1,-1) + f(2,1) * k(1,0) + f(2,2) * k(1,1)$

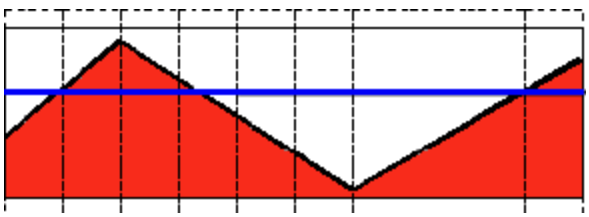
Hàm Gaussian blur trong opencv:

```
GaussianBlur(InputArray src, OutputArray dst, cv::Size kernel_size,
double sigmaX, double sigmaY,  int borderType);
```

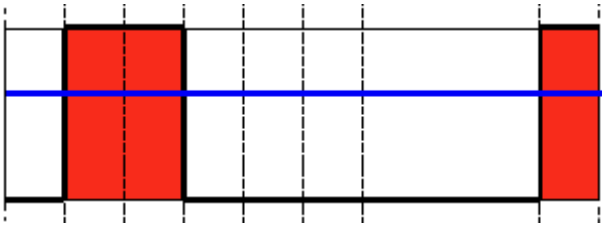
- Trong đó:
  - Src là ảnh nguồn
  - dst là ảnh đích
  - kernel_size là kích thước của kernel
  - sigmaX = σ theo chiều x
  - sigmaY = σ theo chiều y
  - borderType là phương pháp ngoại suy pixel
- Chiều rộng và chiều dài của kernel_size phải là số nguyên dương lẻ.
- Nếu sigmaX = 0, sigmaX được tính toán theo kích thước của kernel theo công thức $sigma = 0.3 * ((kernel\_size.x - 1) * 0.5 - 1) + 0.8$
- Nếu sigmaY = 0, sigmaY được gán giá trị bằng sigmaX.
- Khi σ càng lớn, ảnh hưởng của các pixel xung quanh đến pixel trung tâm càng lớn.

3. Nhị phân hoá ảnh

- Nhị phân hoá là kĩ thuật đưa ảnh từ ảnh xám đa mức về ảnh chỉ gồm 2 mức, 0 (đen) và 255 (trắng) (có thể coi là 0 và 1).
- Mục đích của nhị phân hoá là tách các đối tượng ra khỏi nền, từ đó có thể dễ dàng áp dụng các thuật toán phân tích, xử lý khác.
- Quá trình biến đổi ảnh xám thành ảnh nhị phân:
  - $dst(x,y) = \begin{cases} 255 \text{ nếu } src(x,y) > T \\ 0 \text{ nếu } src(x,y) \leq T \end{cases}$ với T là giá trị ngưỡng, thường lấy là 128.
  - Ví dụ:
    Ảnh gốc:



    Ảnh sau khi nhị phân:



- Phương pháp trên có thể đưa ra kết quả không tốt do ảnh hưởng bởi cường độ sáng không đều trong ảnh. Việc chọn ngưỡng phù hợp cho từng bức ảnh là rất khó. Vì vậy, để khắc phục điều đó, ta có thể tính toán ngưỡng cho từng vùng của bức ảnh thay vì tính toán ngưỡng chung. Đây được gọi là phương pháp phân ngưỡng động.
- Ý tưởng của nhị phân hoá ngưỡng động:
    - Chia ảnh thành nhiều khu vực nhỏ
    - Với mỗi khu vực, áp dụng thuật toán để tìm ngưỡng phù hợp
    - Áp dụng nhị phân hoá ngưỡng cho mỗi vùng với mỗi ngưỡng tìm được tương ứng

Thuật toán tính ngưỡng: thuật toán Otsu

- Bước 1: Chọn giá trị khởi tạo T, thường lấy bằng 128 ($= (maxval + minval)/2$)
- Bước 2: Phân chia các giá trị điểm ảnh thành 2 nhóm:
    - Nhóm 1: chứa tất cả điểm ảnh có giá trị $> T$
    - Nhóm 2: chứa tất cả điểm ảnh có giá trị $\leq T$
- Bước 3: Tính trung bình của các điểm ảnh trong nhóm 1, gọi là $average_1$, trung bình của các điểm ảnh trong nhóm 2, gọi là $average_2$. Lấy $T_1 = (average_1 + average_2)/2$
- Bước 4: Nếu $|T - T_1| < \delta$ cho trước, T_1 là giá trị cần tìm. Ngược lại, gán $T = T_1$ và quay trở lại bước 2.

4. Biến đổi Affine: phép quay

- Biến đổi Affine là phép biến đổi giữa các điểm trong không gian, bảo toàn điểm, đường thẳng và mặt phẳng. Hai đường thẳng song song vẫn song song qua phép biến đổi affine.
- Phép quay biến đổi điểm (x, y) thành điểm (x', y') quay một góc θ quanh điểm (X, Y).
- Quy ước: $\theta > 0$ là phép quay ngược chiều kim đồng hồ.
- Để thực hiện phép quay điểm (x, y) quanh điểm (X, Y) một góc θ, ta thực hiện các bước:
    - Tịnh tiến điểm (x, y) theo vector $\overrightarrow{(-X, -Y)}$
    - Xoay một góc θ quanh gốc toạ độ
    - Tịnh tiến theo vector $\overrightarrow{(X, Y)}$
- Gọi phép 2 tịnh tiến lần lượt là T_1, T_2, phép quay quanh gốc toạ độ là R. Ta có công thức:
$$(x', y', 1) = (x, y, 1)T_1RT_2$$
- Phép tịnh tiến:

- $(x', y', 1) = (x, y, 1)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_X & t_Y & 1 \end{pmatrix}$ với (t_X, t_Y) là vector tịnh tiến.

- Phép quay một góc θ quanh gốc toạ độ:
  - $(x', y', 1) = (x, y, 1)\begin{pmatrix} \alpha & \beta & 0 \\ -\beta & \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$ với $\alpha = \cos(\theta), \beta = \sin(\theta)$

- Tính $T_1 R T_2$:

$$T_1 R T_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X & -Y & 1 \end{pmatrix}\begin{pmatrix} \alpha & \beta & 0 \\ -\beta & \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X & Y & 1 \end{pmatrix} = \begin{pmatrix} \alpha & \beta & 0 \\ -\beta & \alpha & 0 \\ X(1-\alpha)+Y\beta & -X\beta+Y(1-\alpha) & 1 \end{pmatrix}$$

- Vậy, phép quay (x, y) một góc θ quanh điểm (X, Y) được tính bởi công thức:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & -\beta & X(1-\alpha)+Y\beta \\ \beta & \alpha & -X\beta+Y(1-\alpha) \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Phép xoay trong opencv:

```
warpAffine(InputArray src, OutputArray dst, InputArray M, Size dsize, int flag, int borderType);
```

- Trong đó:
  - M là ma trận kích thước 2×3 thể hiện phép biến đổi:

$$dst(x, y) = src(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

    hay

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

5. Tìm đường bao

5.1. Một số định nghĩa

- Định nghĩa 1: Hàng trên cùng, dưới cùng, cột bên trái nhất, bên phải nhất được gọi là khung của ảnh. Điểm (0,0) bắt đầu từ điểm trên cùng bên trái nhất.
- Định nghĩa 2: Hai điểm pixel 0 (x, y) và (p, q) được coi là liên thông nếu $|x - p| \leq 1, |y - q| \leq 1$. Hai pixel 1 (x, y) và (p, q) được coi là liên thông nếu $|x - p| + |y - q| = 1$.
- Định nghĩa 3: Nếu một thành phần liên thông của các pixel 0 chứa khung thì thành phần đó được gọi là nền (background). Ngược lại, thành phần đó là lỗ.
- Định nghĩa 4: Điểm (i, j) được gọi là điểm biên nếu (i, j) nằm trong một thành phần liên thông của các pixel 1 và tồn tại điểm lân cận (p, q) của (i, j) thuộc một thành phần liên thông của các pixel 0.

- Định nghĩa 5: Cho 2 thành phần liên thông S_1, S_2. Với mọi đường thẳng nối một điểm thuộc S_1 với một điểm trên khung, nếu tồn tại điểm thuộc S_2 nằm trên đường thẳng đó thì S_2 được gọi là bao quanh S_1. Nếu giữa S_2 và S_1 tồn tại điểm chung, S_2 được gọi là bao quanh trực tiếp với S_1.
- Định nghĩa 6: Đường bao ngoài là tập các điểm biên giữa một thành phần liên thông của các pixel 1 với một thành phần liên thông của các pixel 0 bao quanh nó trực tiếp. Đường bao lỗ là tập các điểm biên giữa một lỗ và một thành phần liên thông của pixel 1 bao quanh nó trực tiếp.
- Định nghĩa 7:
    - Đường bao cha của một đường bao ngoài giữa thành phần liên thông của pixel 1 S_1 và thành phần liên thông của pixel 0 S_2 được xác định:
        - Là đường bao lỗ giữa S_2 và thành phần liên thông của pixel 1 bao quanh S_2 trực tiếp nếu S_2 là một lỗ.
        - Là khung của ảnh nếu S_2 là nền.
    - Đường bao cha của một đường bao lỗ giữa lỗ S_1 và thành phần liên thông của pixel 1 S_2 bao quanh nó trực tiếp là đường bao ngoài của S_2 với một thành phần liên thông của pixel 0 bao quanh S_2 trực tiếp.
- Định nghĩa 8: đường bao B_n được gọi là bao quanh B_0 nếu tồn tại một chuỗi các đường bao $B_0, B_1, \dots, B_n$ sao cho B_k là cha của B_{k-1} với mọi k.

5.2. Thuật toán

- Thực hiện quét qua lần lượt các điểm ảnh để xác định đường bao.
- Quy tắc xác định đường bao: Với điểm (i, j), thoả mãn một trong hai điều kiện sau thì được xác định là điểm bắt đầu cho đường bao
    - Nếu $(i, j-1) = 0,\ (i, j) = 1$ thì (i, j) là bắt đầu của một đường bao ngoài.
    - Ngược lại, nếu $(i, j) \geq 1, (i, j+1) = 0$ thì (i, j) là bắt đầu của đường bao lỗ.
- Mỗi đường bao được tìm thấy sẽ được gán một số thứ tự, gọi là NBD.
- Quy tắc xác định cha của đường bao: Gọi B' là đường bao gần nhất tìm được, B là đường bao đang xét.
    - Nếu B và B' cùng loại, cha của B là cha của B'
    - Ngược lại, B' là cha của B.

Mô tả thuật toán:

- Đặt $F = \{f_{ij}\}$ bằng giá trị của ảnh đầu vào.
- Khởi tạo $NBD = 1$ (khung của ảnh được coi như một đường bao lỗ đặc biệt, có thứ tự là 1)
- Quét qua lần lượt các điểm ảnh, bắt đầu từ điểm $(0,0)$. Thực hiện các bước sau với mỗi pixel nếu $f_{ij} \neq 0$. Mỗi lần quét qua dòng mới, gán $LNBD = 1$ (LNBD là số thứ tự của đường bao cuối cùng được tìm thấy).
    - (1) Xác định loại cho đường viền đang xét:

        - (a) Nếu $f_{ij} = 1$ và $f_{i,j-1} = 0$ thì (i,j) là điểm bắt đầu của một đường viền ngoài. Tăng giá trị NBD, gán $(i_2, j_2) \leftarrow (i, j-1)$
        - (b) Nếu $f_{ij} \geq 1$ và $f_{i,j+1} = 0$, (i,j) là điểm bắt đầu của một đường viền lỗ. Tăng NBD, gán $(i_2, j_2) = (i, j+1)$, và $LNBD = f_{ij}$ nếu $f_{ij} > 1$.
        - (c) Còn lại, đi tới (4).
    - (2) Xác định 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 (i,j), đi theo đường viền đã được phát hiện.
        - (3.1): Bắt đầu từ (i_2, j_2), kiểm tra các pixel lân cận của (i,j) ngược chiều kim đồng hồ và tìm 1 pixel khác 0. Đặt (i_1, j_1) là pixel đầu tiên khác 0 tìm được. Nếu không tìm được pixel nào, đặt $f_{ij} = -NBD$ và đi tới (4) .
        - (3.2): gán $(i_2, j_2) \leftarrow (i_1, j_1)$ và $(i_3, j_3) \leftarrow (i, j)$.
        - (3.3): Bắt đầu từ phần từ tiếp theo của (i_2, j_2) (=(i_1, j_1) sau bước 3.2), xét ngược chiều kim đồng hồ các điểm lân cận của (i_3, j_3) (= (i,j) sau bước 3.2) và tìm phần từ khác 0 đầu tiên, đặt nó là (i_4, j_4) (Bước này là xét phần tử khác 0 tiếp theo lân cận với (i,j), giống với bước 3.1)
        - (3.4): Thay đổi giá trị của $f_{i_3 j_3}$:
            - (a): Nếu $(i_3, j_3 + 1) = 0 \rightarrow f_{i_3 j_3} = -NBD$.
            - (b): Nếu $(i_3, j_3 + 1) \neq 0$ và $f_{i_3 j_3} = 1 \rightarrow f_{i_3 j_3} = NBD$.
            - (c): Trường hợp còn lại, không thay đổi $f_{i_3 j_3}$
        - (3.5): Nếu $(i_4, j_4) = (i,j)$ và $(i_3, j_3) = (i_1, j_1)$ (quay trở lại điểm ban đầu), đi tới (4). Ngược lại, gán $(i_2, j_2) = (i_3, j_3)$, $(i_3, j_3) = (i_4, j_4)$ và quay trở lại (3.3).
    - (4): Nếu $f_{ij} \neq 1$, gán $LNBD = |f_{ij}|$ và tiếp tục quét raster từ pixel $(i, j+1)$. Thuật toán kết thúc khi đạt tới góc dưới bên phải.

6. Thuật toán K-Nearest Neighbors

- K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất trong Machine Learning. Khi training, thuật toán này không học một điều gì từ dữ liệu training, mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới.
- Với KNN, trong bài toán phân lớp, label của một điểm dữ liệu mới được suy ra trực tiếp từ K điểm dữ liệu gần nhất trong training set. Label của một test data có thể được quyết định bằng bầu chọn theo số phiếu giữa các điểm gần nhất, hoặc nó có thể được suy ra bằng cách đánh trọng số khác nhau cho mỗi trong các điểm gần nhất đó rồi suy ra label.
- Thuật toán có 2 đầu vào, một là tập các dữ liệu đã biết trước kiểu (loại) của từng dữ liệu (hay còn gọi là tập huấn luyện - training set), đầu vào thứ 2 là dữ liệu, chúng ta chưa biết kiểu (loại) dữ liệu đó. Đầu ra của thuật toán kNN là kiểu dữ liệu của đầu vào thứ 2.

- Thuật toán KNN thích hợp với việc phân loại dữ liệu, không có khả năng phân tích dữ liệu để tìm ra các thông tin có giá trị. Trong quá trình kNN hoạt động, nó phải tính toán "khoảng cách" từ dữ liệu cần xác định loại đến tất cả các dữ liệu trong tập huấn luyện (training set) nên nếu tập huấn luyện quá lớn, điều đó sẽ làm cho thời gian chạy của chương trình sẽ rất lâu.
- "Khoảng cách" từ dữ liệu cần xác định đến khoảng cách của dữ liệu trong tập huấn luyện thường được xác định bởi khoảng cách Euclid hoặc khoảng cách Manhattan
    - Khoảng cách Euclid: $d = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$ với x là dữ liệu training, y là dữ liệu cần xác định, m là chiều dài của dữ liệu.
    - Khoảng cách Manhattan: $d = \sum_{i=1}^{m}|x_i - y_i|$
- K kết quả nhỏ nhất sẽ được lấy để xét nhãn cho dữ liệu mới.

III. Thiết kế và thực hiện chương trình

1. Công nghệ sử dụng

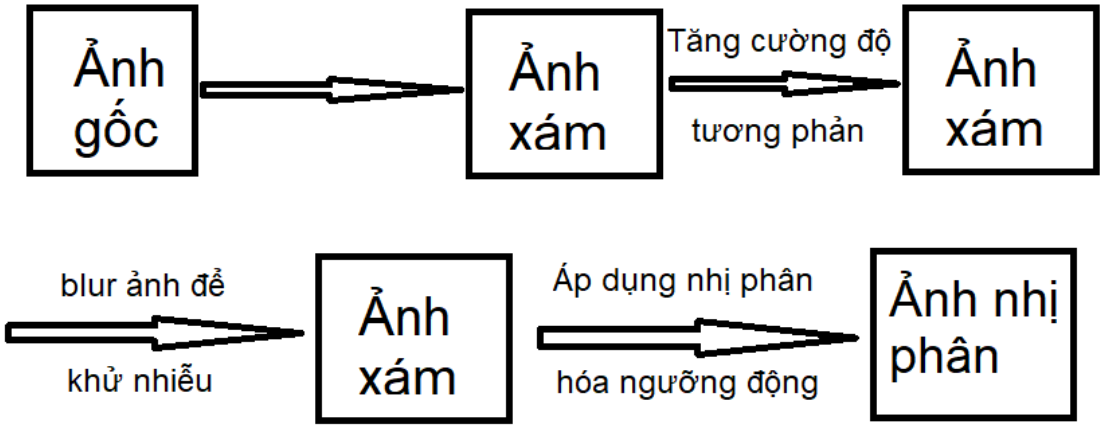
- Ngôn ngữ: C++
- Môi trường phát triển: Microsoft Visual Studio
- Thư viện: opencv 3.4.6

2. Quy trình sử dụng

3. Chi tiết hoạt động

2.1. Khoanh vùng biển số

- Hình ảnh gốc sau khi được đọc vào sẽ được tiền xử lý thông qua các bước sau:



Hình 4: Tiền xử lý ảnh

- Ảnh xám được tăng cường độ tương phản bằng cách sử dụng biến đổi top-hat và biến đổi black-hat.
- Kết quả của bước tiền xử lý là ảnh nhị phân của ảnh gốc.

Hình 5: Hình ảnh biển số gốc

Hình 6: Hình ảnh nhị phân thu được sau khi tiền xử lý

- Sau khi có ảnh nhị phân, ta tìm các contour có trong ảnh. Với mỗi contour, kiểm tra xem contour đó có khả năng là biển số hay không:
  - Chiều rộng nằm trong khoảng (MIN_PLATE_WIDTH, MAX_PLATE_WIDTH)
  - Chiều Cao nằm trong khoảng (MIN_PLATE_HEIGHT, MAX_PLATE_HEIGHT)
  - Tỷ lệ chiều cao/ chiều rộng nằm trong khoảng (MIN_PLATE_RATIO, MAX_PALTE_RATIO).
  - Chương trình lấy các giá trị như sau:

MIN_PLATE_WIDTH	70.0
MAX_PLATE_WIDTH	500.0
MIN_PLATE_HEIGHT	50.0
MAX_PLATE_HEIGHT	350.0

MIN_PLATE_RATIO	0.4
MAX_PLATE_RATIO	0.8

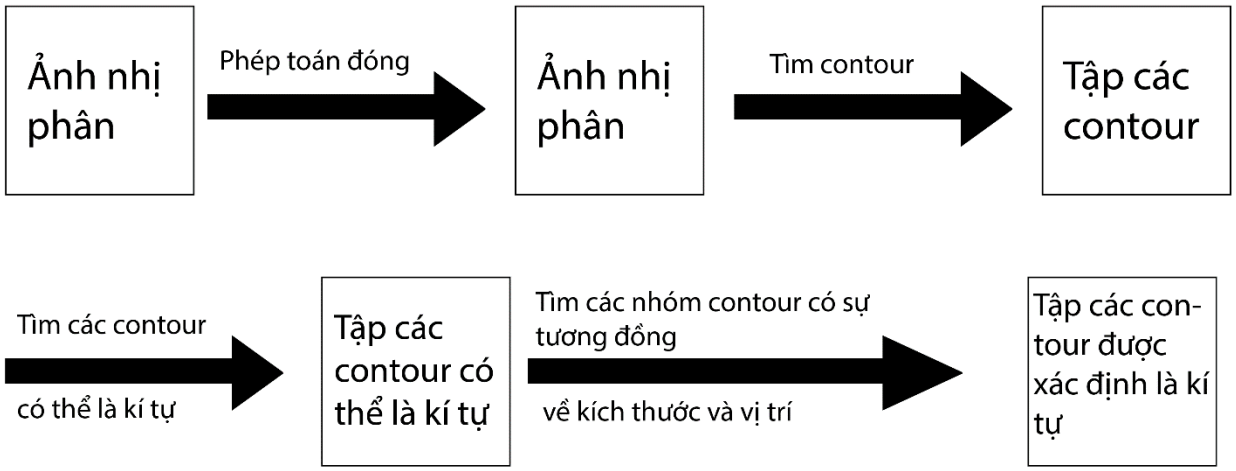
Bảng 1: Bảng giá trị kích thước biển số xe

- Các vùng biển số sau khi tìm được sẽ được xoay ngang và cắt ra từ ảnh gốc.

Hình 7: Các vùng được xác định có thể là biển số

2.2. Phân tách kí tự

- Phân tách kí tự được thực hiện theo các bước:



Hình 8: Các bước phân tách kí tự

- Phép toán đóng (Closing – thực hiện phép giãn nở (Dilation) sau đó thực hiện phép co (Erosion)) để lấp đầy các khoảng trống trên biên và các lỗ nhỏ.
- Ở bước tìm các contour có thể là kí tự: Một contour có thể là kí tự nếu:
  - Diện tích hình chữ nhật bao quanh contour > MIN_PIXEL_AREA
  - Chiều rộng > MIN_PIXEL_WIDTH
  - Chiều cao > MIN_PIXEL_HEIGHT
  - Tỷ lệ chiều rộng/chiều cao nằm trong khoảng (MIN_ASPECT_RATIO, MAX_ASPECT_RATIO)
  - Bảng các giá trị hằng số trong chương trình:

MIN_PIXEL_AREA	80
MIN_PIXEL_WIDTH	5
MIN_PIXEL_HEIGHT	15
MIN_ASPECT_RATIO	0.25
MAX_ASPECT_RATIO	1.0

Bảng 2: Bảng giá trị kích thước kí tự

- Ở bước tìm các nhóm contour có sự tương đồng: contour B được coi là tương đồng với contour A nếu:
  - Khoảng cách giữa A và B < (độ dài đường chéo của A) * `MAX_DIAG_SIZE_MULTIPLE_AWAY`
  - Tỷ lệ diện tích khác biệt < MAX_CHANGE_IN_AREA
  - Tỷ lệ chiều rộng khác biệt < MAX_CHANGE_IN_WIDTH

- Tỷ lệ chiều cao khác biệt < MAX_CHANGE_IN_HEIGHT
- Bảng giá trị hằng số:

MAX_DIAG_MULTIPLE_AWAY	3.0
MAX_CHANGE_IN_AREA	0.5
MAX_CHANGE_IN_WIDTH	0.8
MAX_CHANGE_IN_HEIGHT	0.5

Bảng 3: Bảng giá trị so sánh kí tự tương đồng

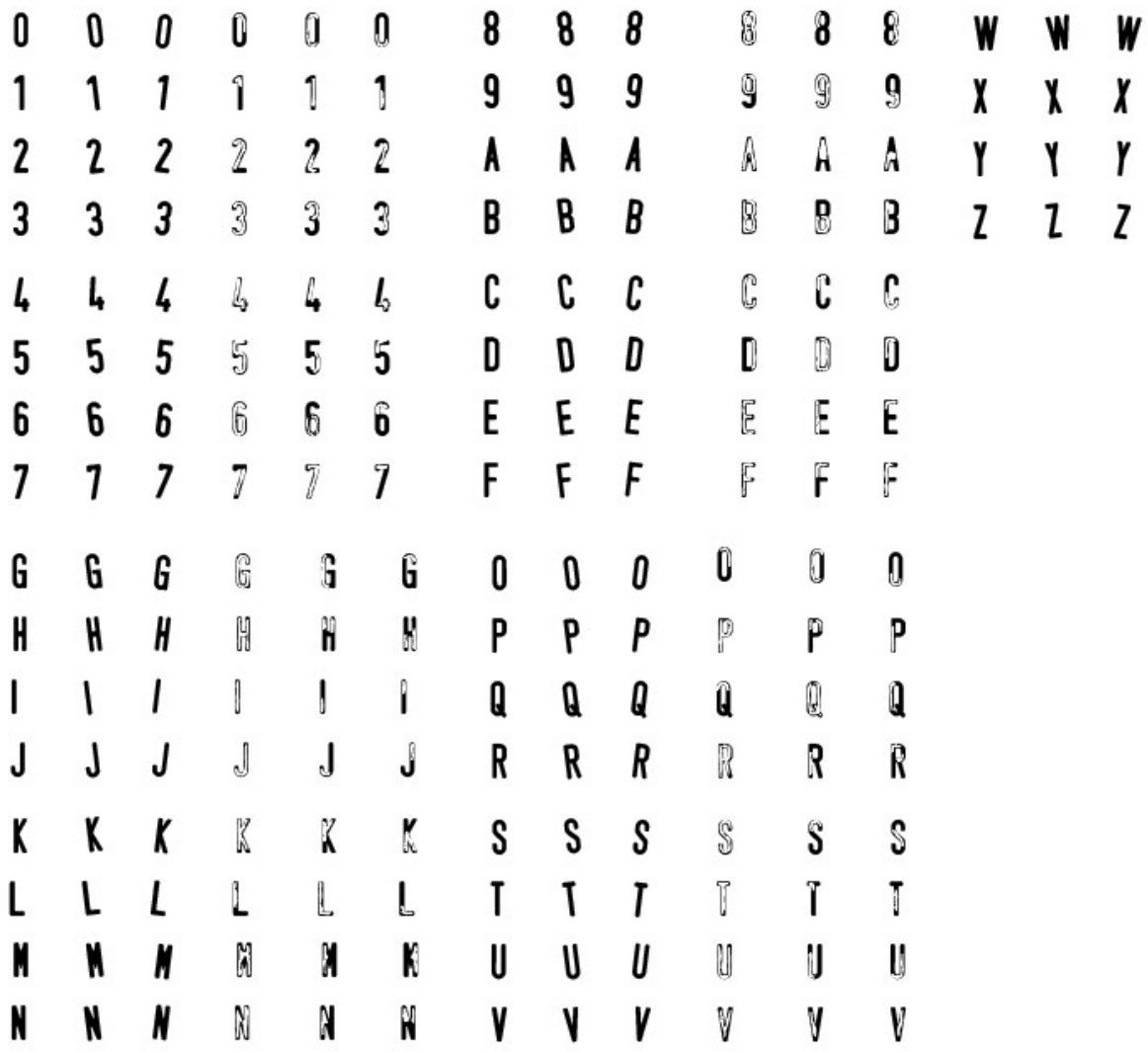
- Các kí tự được xác định ban đầu và sau khi áp dụng một số điều kiện loại trừ:

Hình 9: Phân tách kí tự

2.3. Nhận dạng kí tự

2.3.1. K-nearest neighbor

- Dữ liệu training là các file ảnh chứa kí tự được tạo ra từ font biển số xe 2 bánh của Việt Nam được biến đổi để tạo ra sự đa dạng cho dữ liệu train.

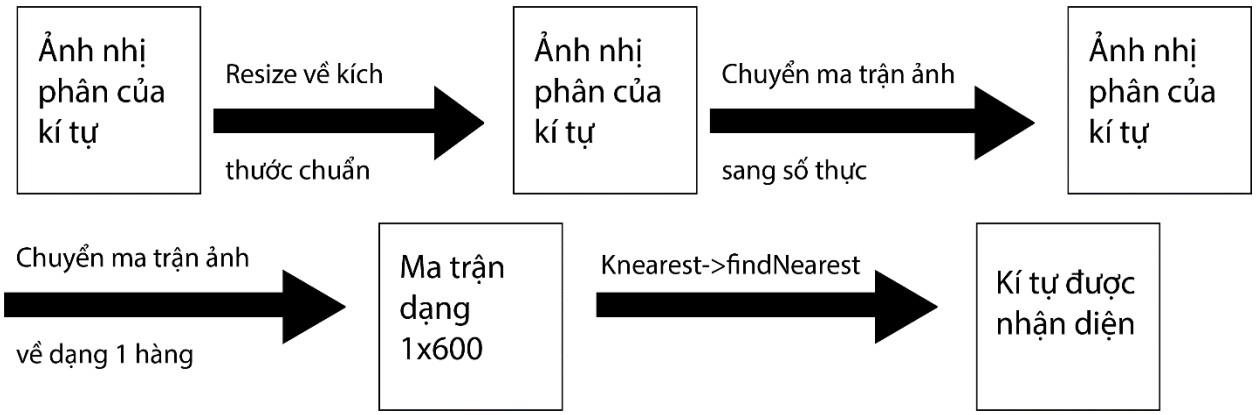


Hình 10: Dữ liệu training

- Dữ liệu training sẽ được biểu diễn bởi 2 file classifications.xml và images.xml. File image.xml chứa các biểu diễn bằng ma trận kích thước 1x600 của mỗi kí tự. File classifications.xml chứa label của các ma trận tương ứng.
- 2 file này sẽ được sử dụng để nhận dạng kí tự.

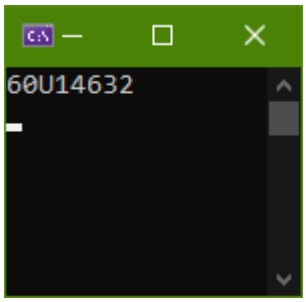
2.3.2. Quá trình nhận dạng

- Quá trình nhận dạng kí tự diễn ra như sau:



Hình 11: Quy trình nhận diện kí tự

- Kí tự được nhận dạng được in trên màn hình console:

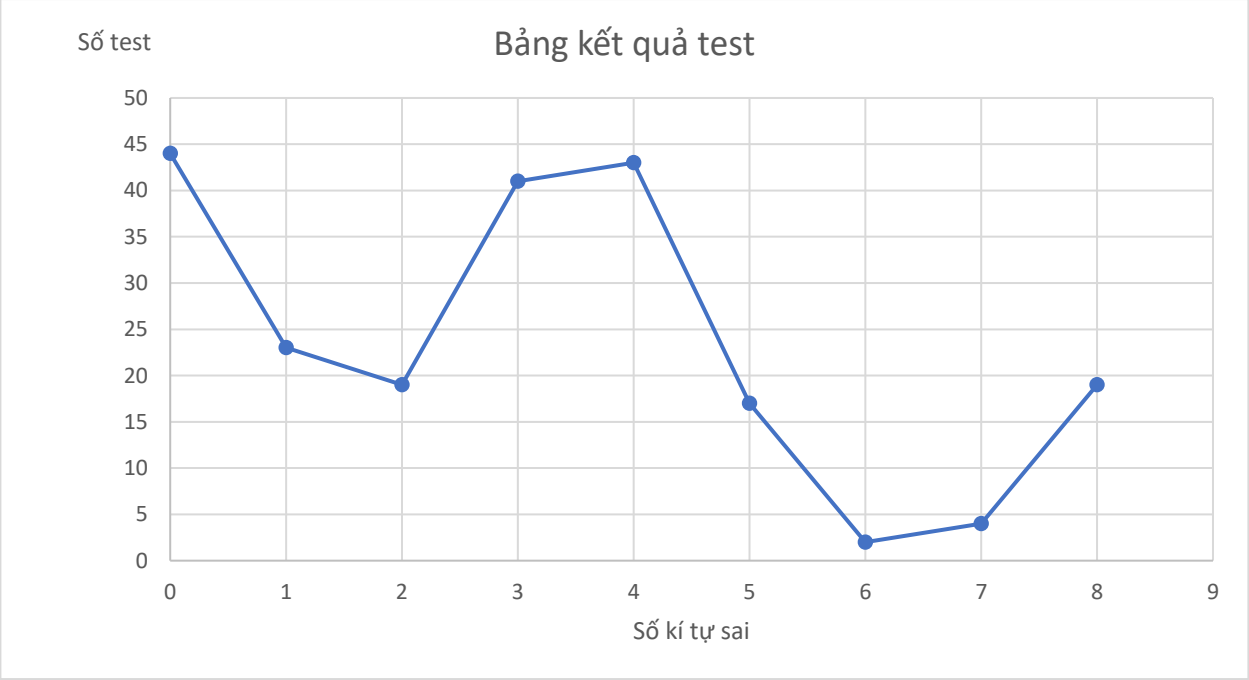


Hình 12: Kết quả nhận dạng

IV. Kết quả thực hiện

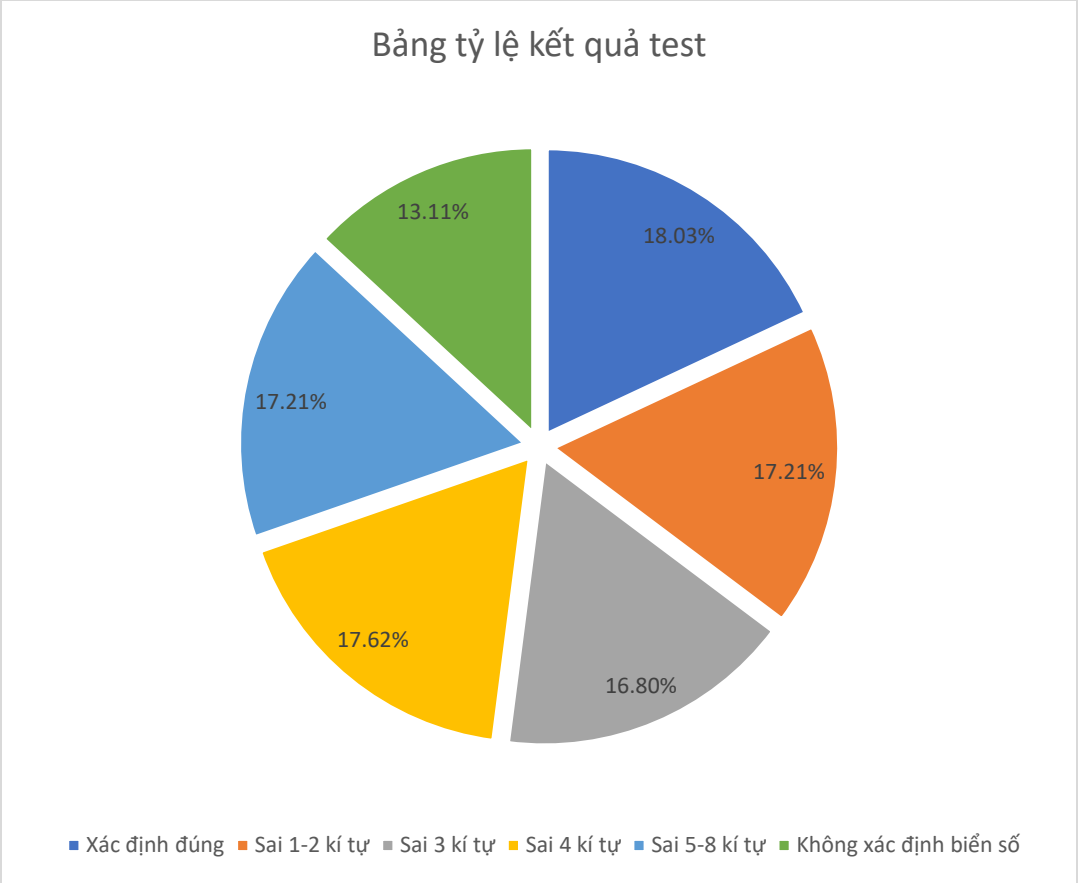
1. Kết quả chạy thử

- Chạy thử trên tập tập 244 ảnh:



Bảng 4: Bảng kết quả test

- Tỷ lệ:



Bảng 5: Tỷ lệ chính xác kết quả test

- Tỷ lệ xác định được vị trí biển số xe chiếm 86.89%
- Tỷ lệ kí tự được phân tách và nhận dạng đúng khoảng 54,45%
- Thời gian chạy cho mỗi bức ảnh: 0.3 - 0.8s

2. Nhận xét

- Chương trình hoạt động chưa hiệu quả, ảnh hưởng nhiều bởi yếu tố môi trường.
- Khi đường viền của kí tự không liền mạch, chương trình có thể không nhận ra, hoặc nhận diện 2 lần cho cùng kí tự.
- Khi 2 kí tự bị dính, chương trình nhận diện như 1 kí tự (hoặc không nhận diện kí tự đó).
- Một số kí tự thường bị nhận diện nhầm:

Kí tự đúng	Nhận diện thành
8	B
N	11 | II
10	D
18	B

5	6
G	6
A | 4	4 | A
U	0

- Ánh sáng ảnh hưởng nhiều đến việc nhận diện.

3. Hướng cải tiến

- Cải thiện thuật toán tìm vị trí biển số.
- Cải thiện thuật toán phân tách kí tự.
- Ứng dụng học máy vào việc tìm vị trí biển số và phân tách kí tự.
- Dựa trên form của biển số xe Việt Nam để xác định kí tự chính xác hơn.
- Tạo giao diện trực quan cho chương trình.

Tài liệu tham khảo

[1]: https://www.stdio.vn/articles/xu-ly-anh-voi-opencv-phong-to-thu-nho-va-xoay-anh-401#10-phep-tinh-tien-translation

[2]: https://www.stdio.vn/articles/ky-thuat-grayscale-va-nhi-phan-hoa-anh-adaptive-threshold-383

[3]: https://www.stdio.vn/articles/loc-gaussian-giao-dien-don-gian-qt-opencv-447

[4]: https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

[5]: https://machinelearningcoban.com/2017/01/08/knn/

[6]: Satoshi Suzuki and Keiichi Abe, *Topological Structural Analysis of Digitized Binary Images by Border Following* ,1985.