ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO MÔN HỌC

Đề tài:

PHÂN LOẠI BIỂN BÁO GIAO THÔNG VIỆT NAM

Môn học: Nhập môn Thị giác Máy tính

Lớp: CS231.P12

GV hướng dẫn: TS. Mai Tiến Dũng

Nhóm thực hiện:

Họ và tên	MSSV
Nguyễn Duy Hoàng	22520467
Hà Huy Hoàng	22520460

TP. Hồ Chí Minh, ngày 24 tháng 11 năm 2024

Mục lục

LỜI MỞ ĐẦU

Trong bối cảnh giao thông đường bộ tại Việt Nam ngày càng phát triển, việc quản lý và điều tiết phương tiện trở thành một thách thức lớn, đặc biệt tại các đô thị với mật độ xe cộ dày đặc. Hệ thống biển báo giao thông đóng vai trò then chốt trong việc hướng dẫn, cảnh báo và điều chỉnh hành vi của người tham gia giao thông, góp phần bảo đảm an toàn và trật tự. Tuy nhiên, thực tế cho thấy, việc nhận diện biển báo giao thông thủ công dựa trên quan sát của con người dễ gặp sai sót, đặc biệt trong các điều kiện bất lợi như ánh sáng kém, thời tiết xấu hoặc khi biển báo bị hư hỏng hay che khuất. Điều này đặt ra nhu cầu cấp thiết cho các giải pháp tự động hóa trong việc nhận diện và phân loại biển báo giao thông một cách nhanh chóng, chính xác và hiệu quả.

Hệ thống biển báo giao thông đường bộ Việt Nam được chia thành 5 nhóm chính: Cấm, Nguy hiểm, Chỉ dẫn, Hiệu lệnh và Phụ. Mỗi nhóm mang những đặc điểm hình học, màu sắc và ý nghĩa riêng biệt, tạo nên sự đa dạng nhưng cũng đồng thời đặt ra nhiều thách thức trong nhận diện và phân loại. Trước tiên, sự đa dạng về hình dáng và màu sắc của các biển báo có thể dẫn đến nhầm lẫn giữa các nhóm nếu đặc trưng không được trích xuất chính xác. Bên cạnh đó, các điều kiện môi trường như ánh sáng yếu, góc chụp không đồng nhất hoặc biển báo bị che khuất làm giảm đáng kể hiệu quả của các mô hình phân loại. Cuối cùng, việc lựa chọn và tối ưu hóa thuật toán phân loại phù hợp cũng đòi hỏi sự kết hợp chặt chẽ giữa các kỹ thuật xử lý ảnh và học máy.

Để giải quyết các thách thức này, nghiên cứu tập trung vào việc áp dụng các thuật toán học máy như K-Nearest Neighbors (KNN) và Support Vector Machine (SVM), kết hợp với các kỹ thuật trích xuất đặc trưng như HOG (Histogram of Oriented Gradients) và histogram màu sắc. Các phương pháp này không chỉ giúp mô tả chính xác đặc điểm nổi bật của biển báo mà còn tăng cường khả năng phân biệt giữa các nhóm, từ đó đạt được độ chính xác cao trong môi trường thực tế.

Mục tiêu của nghiên cứu không chỉ dừng lại ở việc xây dựng một hệ thống phân loại biển báo giao thông hiệu quả, mà còn đặt nền tảng cho việc ứng dụng trí tuệ nhân tạo trong giao thông đường bộ. Thành quả này có thể hỗ trợ các hệ thống giao thông thông minh như xe tự hành và giám sát giao thông tự động, đồng thời góp phần hiện thực hóa mục tiêu an toàn giao thông bền vững trong sự phát triển của một xã hội hiện đại.

THÔNG TIN THÀNH VIÊN

Họ tên	MSSV	Phân công	Mức độ hoàn thành
Nguyễn Duy Hoàng	22520467	Trích xuất đặc trưng (HOG), phát triển và đánh giá thuật toán (KNN)	100%
Hà Huy Hoàng	22520460	Trích xuất đặc trưng (Color Histogram), phát triển và đánh giá thuật toán (SVM)	100%

WEB DEMO QR CODE

Để truy cập web, quét mã QR dưới đây:

CHƯƠNG 1. TỔNG QUAN

1.1 Mô tả bài toán

Trong giao thông đô thị, biển báo giao thông có sự đa dạng về hình dạng, màu sắc và nội dung, kèm theo đó là số lượng biển dày đặc; điều này khiến việc nhận diện chúng trở thành thách thức lớn cho người tham gia giao thông đường bộ. Để hỗ trợ người lái xe, ta cần xây dựng một hệ thống phân loại, có khả năng nhận diện và phân loại biển báo vào thành các nhóm chính: cấm, nguy hiểm, hiệu lệnh, chỉ dẫn và biển phụ.

1.2 Lí do thực hiện bài toán

Phân loại biển báo giao thông không chỉ hỗ trợ người lái xe tránh nhầm lẫn mà còn là bước nền tảng cho các hệ thống giao thông thông minh. Nhận diện chính xác biển báo giúp phát triển các ứng dụng như bản đồ thông minh, cảnh báo tức thời... Đặc biệt, đây là yếu tố quan trọng trong hệ thống lái tự động, hỗ trợ xe tự lái tuân thủ quy định và đảm bảo an toàn giao thông.

1.3 Phát biểu bài toán

Input: Đầu vào của bài toán là một ảnh chụp biển báo giao thông chưa được gắn nhãn.

Output: Đầu ra của bài toán là một nhãn phân loại của biển báo trong ảnh, thuộc một trong các loại sau: "Cấm", "Chỉ dẫn", "Hiệu lệnh", "Nguy hiểm", "Phụ".

Với các loại biển báo được mô tả như sau:

- **Biển báo "Cấm"**: Thường có hình tròn, viền đỏ, nền trắng và ký hiệu đen, biểu thị các hành vi bị cấm như cấm vượt, cấm dừng đỗ, cấm quay đầu, v.v.
- **Biển báo "Nguy hiểm"**: Thường có hình tam giác, viền đỏ, nền vàng, dùng để cảnh báo về những nguy cơ tiềm ẩn như khúc cua gấp, đường trơn, dốc cao, v.v.
- **Biển báo "Chỉ dẫn"**: Thường có hình chữ nhật/vuông, nền xanh, ký hiệu trắng, cung cấp thông tin về hướng đi, điểm đến, hoặc các dịch vụ cần thiết như bệnh viện, bến xe buýt, bãi đỗ xe, v.v.
- **Biển báo "Hiệu lệnh"**: Thường có hình tròn, nền xanh, chỉ dẫn bắt buộc người lái xe phải tuân thủ, ví dụ như buộc phải đi thẳng, đi theo làn, giới hạn tốc độ tối thiểu, v.v.
- **Biển báo "Phụ"**: Thường là biển nhỏ có dạng hình chữ nhật hoặc vuông, được lắp kèm với các biển báo chính để bổ sung hoặc làm rõ thêm ý nghĩa, như khoảng cách, thời gian hiệu lực, các loại phương tiện áp dụng, v.v.

CHƯƠNG 2. PHƯƠNG PHÁP THỰC HIỆN BÀI TOÁN

2.1 Trích xuất đặc trưng

2.1.1 Đặc trưng Color Histogram

2.1.1.1 Định nghĩa

Color histogram là một biểu đồ thống kê thể hiện phân phối màu sắc của một hình ảnh. Nó đếm số lượng pixel trong mỗi bin (bin ở đây là range màu) và vẽ thành biểu đồ. Mỗi pixel trong hình ảnh có một giá trị màu sắc riêng, và biểu đồ màu sẽ thể hiện tần suất của các giá trị này. Color histogram là một trong những phương pháp đơn giản nhưng hiệu quả trong việc phân tích màu sắc của hình ảnh.

2.1.1.2 Nguyên lý hoạt động

Histogram (hay còn gọi là đồ thị cột) của một ảnh số với L mức xám trong miền $[0, L-1]$ [1] là một hàm rời rạc:

$$p(m) = \frac{n_m}{n}$$

Trong đó:

- m: Giá trị mức xám, $m \in [0, L-1]$.
- n_m: Số pixel có mức xám m.
- n: Tổng số pixel trong ảnh.

Khi áp dụng cho ảnh màu, mỗi kênh màu (Red, Green, Blue) sẽ có một histogram riêng, phản ánh phân phối màu trong không gian màu RGB.

Nguyên lý hoạt động của Color Histogram:

Color Histogram hoạt động bằng cách phân tích sự phân bố màu sắc trong hình ảnh thông qua việc chia không gian màu thành các khoảng (bins) và đếm tần suất các màu xuất hiện [2]. Quá trình này bao gồm các bước chính như sau:

2.1.1.2.1 Chọn không gian màu

Hình ảnh có thể được biểu diễn trong nhiều không gian màu khác nhau, chẳng hạn như:

- **RGB (Red, Green, Blue)**: Mỗi điểm ảnh được đại diện bởi ba giá trị màu cơ bản.
- **HSV (Hue, Saturation, Value)**: Không gian màu này tách biệt thông tin sắc độ, độ bão hòa và cường độ sáng, thường được sử dụng để xử lý ảnh trong điều kiện ánh sáng không đồng đều.

2.1.1.2.2 Phân chia không gian màu

Không gian màu được chia thành các khoảng màu rời rạc, gọi là bins. Số lượng bins (khoảng màu) được điều chỉnh dựa trên yêu cầu của ứng dụng:

- **Số bins ít**: Biểu diễn tổng quát, giảm kích thước đặc trưng nhưng mất chi tiết màu sắc.
- **Số bins nhiều**: Biểu diễn chi tiết hơn về màu sắc, nhưng yêu cầu tài nguyên tính toán cao hơn.

2.1.1.2.3 Đếm tần suất màu sắc

Mỗi pixel trong hình ảnh được ánh xạ vào một khoảng màu cụ thể, dựa trên giá trị màu sắc của nó. Histogram sau đó được tạo ra bằng cách đếm số lượng pixel thuộc về mỗi khoảng màu. Ví dụ:

- Nếu một bin đại diện cho các giá trị màu từ 0 đến 31, tất cả các pixel có giá trị nằm trong khoảng này sẽ được gộp chung và đếm số lượng.

Kết quả là một histogram cho thấy sự phân bố màu sắc trong ảnh.

2.1.1.3 Cài đặt trong bài toán

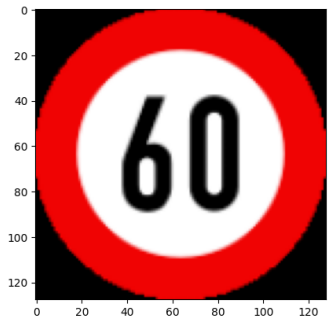
Đoạn mã Python dưới đây minh họa cách cài đặt Color Histogram sử dụng thư viện OpenCV:

```
def color_histogram(image):
    row, column, channel = image.shape[:3]
    size = row * column
    feature = []

    for k in range(channel):
        histogram = np.squeeze(cv2.calcHist(
                                [image], [k], None,
                                [BIN_SIZE], [0, 256]))
        histogram = histogram / size
        feature.extend(histogram)

    return feature
```

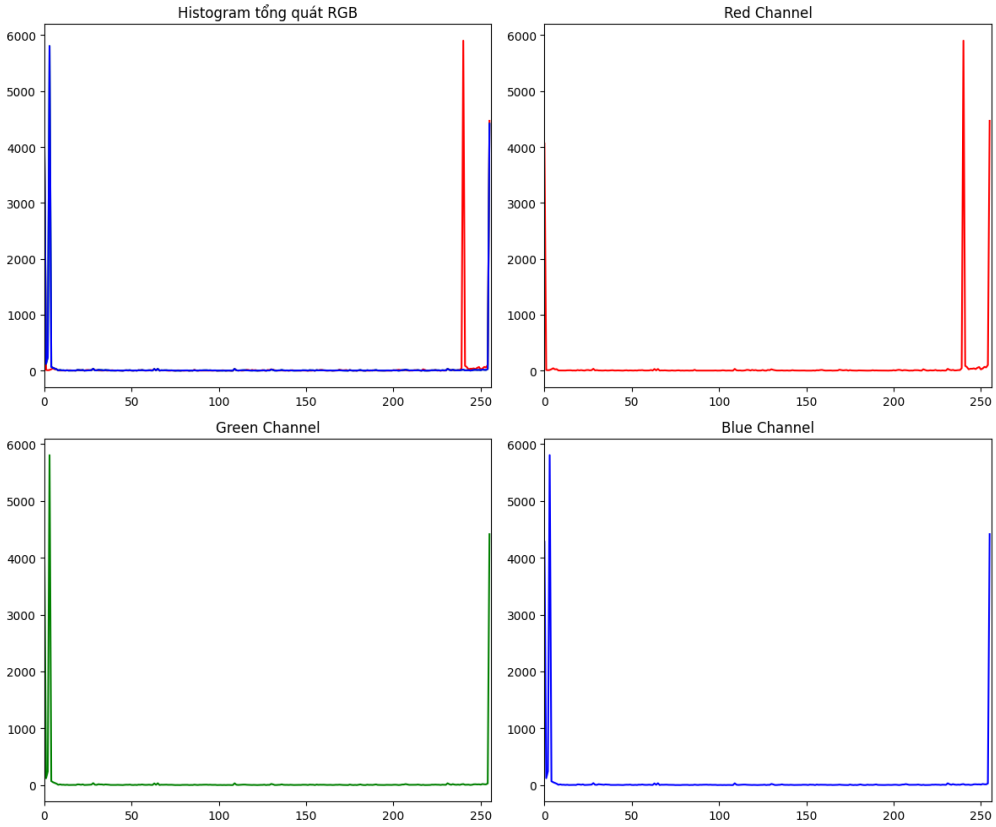
Giải thích các tham số trong `cv2.calcHist`:

- `image`: Dữ liệu ảnh (mảng NumPy).
- `[k]`: Kênh màu cần tính histogram, k thay đổi từ 0 đến 2 (Red, Green, Blue).
- `None`: Tham số mask (không sử dụng trong trường hợp này).
- `[BIN_SIZE]`: Số lượng bins trong histogram, biểu thị số lượng mức độ màu sắc trong mỗi kênh. Ví dụ, với giá trị `BIN_SIZE = 256`, có 256 bins cho mỗi kênh màu.
- `[0, 256]`: Khoảng giá trị màu, từ 0 đến 256 (không bao gồm 256).

Hình minh họa:



Hình 1: Hình ảnh biển báo.



Hình 2: Color Histogram được trích xuất từ hình ảnh biển báo (Hình 1).

2.1.1.4 Nhận xét về Color Histogram

2.1.1.4.1 Ưu điểm

- Đơn giản, dễ thực hiện và dễ dàng tích hợp vào hệ thống phân loại.
- Không phụ thuộc vào kích thước ảnh: Vì histogram chỉ phụ thuộc vào tần suất màu sắc, không bị ảnh hưởng bởi kích thước và tỷ lệ của ảnh.

2.1.1.4.2 Nhược điểm

- Không có thông tin về cấu trúc không gian: không phản ánh mối quan hệ không gian giữa các điểm ảnh, có thể ảnh hưởng đến hiệu suất trong một số bài toán phân loại phức tạp.

2.1.2 Đặc trưng HOG

2.1.2.1 Định nghĩa

HOG (Histogram of Oriented Gradients) là một phương pháp trích xuất đặc trưng từ ảnh dựa trên phân tích gradient. Các đặc trưng được xây dựng từ hướng gradient và magnitude của các pixel trong ảnh, sau đó phân nhóm chúng vào các bins (thùng chứa) để tạo thành một histogram. Phương pháp này chủ yếu được sử dụng trong các bài toán nhận dạng đối tượng và phát hiện biên của các đối tượng trong ảnh.

2.1.2.2 Nguyên lý hoạt động

HOG hoạt động bằng cách tính toán gradient của cường độ ánh sáng để xác định các cạnh và biên của đối tượng trong ảnh . Quá trình này bao gồm các bước chính như sau [3]:

2.1.2.2.1 Tính Gradient

Bước đầu tiên trong việc tính toán HOG là tính gradient của hình ảnh để xác định hướng và độ mạnh của sự thay đổi trong cường độ sáng giữa các pixel. Sử dụng bộ lọc Sobel để tính toán gradient theo hai hướng x và y, từ đó tính toán magnitude và direction của gradient từ các gradient x và y.

$$G = \sqrt{(G_x^2 + G_y^2)} \quad [3]$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \quad [3]$$

Trong đó:

- G_x và G_y là gradient theo phương x và y.
- G là magnitude của gradient.
- θ là hướng gradient.

2.1.2.2.2 Tạo ô (cell) và khối (block)

Sau khi tính gradient, ảnh được chia thành các ô nhỏ (cell) để tính toán histogram hướng trong mỗi ô. Mỗi ô có kích thước thường là 8×8 pixel. Các ô này sau đó được nhóm thành các khối (block), mỗi block gồm 2x2 ô, tức là tổng cộng 16x16 pixel.

2.1.2.2.3 Tạo histogram hướng trong mỗi ô

Mỗi ô sẽ tạo ra một histogram của các hướng gradient, trong đó hướng gradient được chia thành các bin. Thông thường, số bin là 9, với mỗi bin đại diện cho một khoảng góc từ 0 đến 180 độ, mỗi bin chiếm một khoảng góc 20 độ. Histogram này mô tả phân phối hướng gradient trong từng ô.

2.1.2.2.4 Chuẩn hóa khối

Để giảm thiểu ảnh hưởng của sự thay đổi ánh sáng và độ tương phản, histogram của các ô trong mỗi khối sẽ được chuẩn hóa. Phương pháp chuẩn hóa phổ biến là L2-norm:

$$\mathbf{v} = \frac{\mathbf{v}}{\sqrt{\|\mathbf{v}\|_2^2 + \epsilon^2}} \quad [4]$$

Trong đó:

- $\mathbf{v}$ là vector đặc trưng của khối.
- $\|\mathbf{v}\|_2$ là chuẩn L2 của vector $\mathbf{v}$.
- ϵ là một giá trị nhỏ (thường là 10^{-6}) giúp tránh chia cho 0.

2.1.2.2.5 Kết hợp đặc trưng

Cuối cùng, vector đặc trưng HOG được tạo ra bằng cách kết hợp các histogram chuẩn hóa từ tất cả các khối trong ảnh. Đây chính là đặc trưng HOG được sử dụng cho các tác vụ nhận dạng đối tượng.

2.1.2.3 Cài đặt trong bài toán

Đoạn mã Python dưới đây minh họa cách cài đặt HOG bằng thư viện 'skimage':

```python
from skimage.feature import hog

def hog_features(image):
    hog_features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                       cells_per_block=(2, 2), visualize=False,
                       block_norm='L2-Hys', transform_sqrt=True,
                       channel_axis=2)
    return hog_features
```

Giải thích các tham số trong hàm `hog`:

- `image`: Ảnh đầu vào, có thể là ảnh màu (RGB) hoặc ảnh xám (grayscale).
- `orientations`: Số lượng bins trong histogram hướng gradient. Trong ví dụ này, giá trị là 9, có nghĩa là các hướng gradient sẽ được phân chia thành 9 bins, mỗi bin đại diện cho một phạm vi góc từ 0 đến 180 độ.
- `pixels_per_cell`: Kích thước của mỗi ô (cell) trong ảnh, được xác định là 8×8 pixel. Mỗi cell sẽ tính toán gradient và độ mạnh (magnitude) của pixel trong đó.
- `cells_per_block`: Số lượng ô (cell) trong một block, được xác định là 2x2 ô, tức là mỗi block có kích thước 16×16 pixel.
- `visualize`: Điều khiển việc có hay không trực quan hóa các đặc trưng HOG dưới dạng ảnh. Nếu `True`, hàm sẽ trả về cả ảnh biểu diễn các đặc trưng HOG.
- `block_norm`: Phương pháp chuẩn hóa trong mỗi block. "L2-Hys"là một phương pháp chuẩn hóa phổ biến trong HOG, giúp giảm ảnh hưởng của ánh sáng và độ tương phản. Phương pháp chuẩn hóa L2-Hys [4] có thể được mô tả bằng các bước sau:

$$\mathbf{v}' = \frac{\mathbf{v}}{\sqrt{\|\mathbf{v}\|_2^2 + \epsilon^2}} \quad \text{(bước 1: chuẩn hóa L2)}$$

Sau đó, giá trị của vector $\mathbf{v}'$ sẽ bị cắt (clip) nếu vượt quá ngưỡng 0.2:

$$\mathbf{v}' = \min(\mathbf{v}', 0.2) \quad \text{(bước 2: cắt giá trị)}$$

Cuối cùng, chuẩn hóa lại vector sau khi cắt:

$$\mathbf{v}' = \frac{\mathbf{v}'}{\sqrt{\|\mathbf{v}'\|_2^2 + \epsilon^2}} \quad \text{(bước 3: chuẩn hóa L2 lại sau khi cắt)}$$

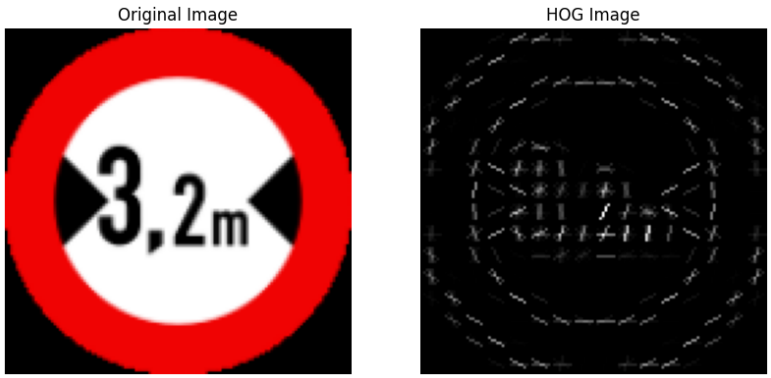
Trong đó:

- $\mathbf{v}$ là vector đặc trưng của block.
- $\|\mathbf{v}\|_2$ là chuẩn L2 của vector $\mathbf{v}$.
- ϵ là một hằng số nhỏ (thường là 10^{-6}) để tránh chia cho 0.
- $\mathbf{v}'$ là vector đã được chuẩn hóa và cắt.

Phương pháp chuẩn hóa L2-Hys giúp giảm ảnh hưởng của sự thay đổi độ sáng và độ tương phản đồng thời làm ổn định quá trình huấn luyện mô hình.

- `transform_sqrt`: Áp dụng phép biến đổi căn bậc 2 vào các giá trị độ sáng trong ảnh trước khi tính gradient. Điều này giúp giảm bớt sự ảnh hưởng của sự khác biệt lớn về độ sáng giữa các phần khác nhau của ảnh.
- `channel_axis`: Tham số này chỉ định kênh màu của ảnh. Nếu ảnh là ảnh màu, tham số này giúp hàm phân tích các kênh màu riêng biệt (Red, Green, Blue).

Hình minh họa:



Hình 3: (Trái) Hình ảnh biển báo và (Phải) Đặc trưng HOG được trích xuất từ ảnh biển báo.

2.1.2.4 Nhận xét về HOG

2.1.2.4.1 Ưu điểm

- **Tính hiệu quả cao trong nhận dạng đối tượng:** HOG rất phù hợp trong các bài toán phát hiện đối tượng, đặc biệt là trong nhận dạng khuôn mặt và các đối tượng có biên rõ ràng.
- **Khả năng chống lại sự thay đổi độ sáng và độ tương phản:** Phương pháp chuẩn hóa L2 giúp giảm thiểu sự ảnh hưởng của các thay đổi ánh sáng và độ tương phản, làm tăng khả năng nhận diện đối tượng trong môi trường khác nhau.

2.1.2.4.2 Nhược điểm

- **Tính toán phức tạp:** Quá trình tính toán gradient và chuẩn hóa có thể tốn nhiều tài nguyên tính toán, đặc biệt là với các ảnh có độ phân giải cao.
- **Không hiệu quả với các đối tượng có biên mờ hoặc có kết cấu phức tạp:** HOG chủ yếu hoạt động tốt với các đối tượng có biên rõ ràng, trong khi các đối tượng mờ hoặc không rõ ràng có thể làm giảm hiệu quả của phương pháp này.

2.1.3 Tổng hợp đặc trưng

```python
def blur_image(image):
    # gaussian_blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
    blurred_image = cv2.medianBlur(image, 5)
    return blurred_image

def extract_features(images):
    blurred_images = [blur_image(image) for image in images]
    color_features = [color_histogram(image) for image in blurred_images]
    hog_features = [hog(image) for image in blurred_images]
    combined_features = [np.concatenate((color_feature, hog_feature))
                         for color_feature, hog_feature
                         in zip(color_features, hog_features)]
    return combined_features
```

Giải thích code:

- `blur_image`: Hàm này áp dụng phương pháp làm mờ lên ảnh đầu vào. Có hai phương pháp làm mờ được sử dụng trong đó:
  - `GaussianBlur`: Phương pháp này áp dụng bộ lọc Gaussian để làm mờ ảnh. Bộ lọc Gaussian sử dụng hàm Gaussian để tính toán trọng số cho mỗi pixel trong ảnh, giúp làm mịn các chi tiết ảnh. Công thức Gaussian là:

    $$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad [5]$$

    Trong đó:
    - x, y là tọa độ pixel trong không gian ảnh.
    - σ là độ lệch chuẩn, điều chỉnh độ rộng của bộ lọc (càng lớn σ, ảnh càng mờ).
  - `medianBlur`: Đây là phương pháp làm mờ sử dụng bộ lọc trung vị, trong đó mỗi pixel trong ảnh được thay thế bằng giá trị trung vị của một vùng lân cận. Công thức cho median blur là:

    $$I'(x, y) = \text{median}\left(I(x', y') \mid (x', y') \in \text{kernel}\right) \quad [6]$$

    Trong đó:
    - $I(x, y)$ là giá trị pixel tại vị trí (x, y).

    * kernel là vùng lân cận (có thể có kích thước 3x3, 5x5, v.v.) quanh pixel (x, y).

  So sánh giữa `GaussianBlur` và `medianBlur`:

  - **Gaussian Blur**: Là phép làm mờ ảnh dựa trên trọng số, trong đó mỗi pixel trong ảnh được thay thế bằng giá trị trung bình có trọng số của các pixel xung quanh, với trọng số tuân theo phân phối Gaussian. Phương pháp này giúp làm mờ các chi tiết nhỏ và giảm nhiễu, nhưng có thể làm mờ các biên ảnh (edges) và mất chi tiết quan trọng. Nhiễu dạng Gaussian thường xuất hiện khi tín hiệu ảnh bị nhiễu do các yếu tố ngẫu nhiên như cảm biến máy ảnh hoặc môi trường ánh sáng không ổn định, và nó có phân bố chuẩn.
  - **Median Blur**: Là phép làm mờ ảnh bằng cách thay thế mỗi pixel trong ảnh bằng giá trị trung vị của vùng lân cận. Phương pháp này rất hiệu quả trong việc loại bỏ các nhiễu xuất hiện đột ngột và không có quy luật, nhưng không làm mờ các chi tiết ảnh mạnh như Gaussian Blur. Median Blur giữ được biên ảnh rõ ràng hơn và không làm mất chi tiết quá nhiều. Tuy nhiên, nó không hiệu quả trong việc giảm nhiễu dạng Gaussian vì không sử dụng trọng số và chỉ dựa vào giá trị trung vị. (Nhiễu dạng Gaussian là dạng nhiễu có phân bố chuẩn (Gaussian distribution), thường gặp trong các tín hiệu ảnh bị nhiễu do các yếu tố ngẫu nhiên như ánh sáng không ổn định hoặc cảm biến máy ảnh. Nó ảnh hưởng đến tất cả các pixel trong ảnh theo phân bố chuẩn.)

- `blurred_images`: Là một danh sách chứa các ảnh đã được làm mờ bằng phương pháp `blur_image`.

- `color_features`: Là một danh sách chứa các đặc trưng màu sắc của các ảnh đã được làm mờ. Hàm `color_histogram` tính toán histogram màu cho mỗi ảnh. Đây là một cách phổ biến để trích xuất đặc trưng màu sắc từ ảnh.

- `hog_features`: Là một danh sách chứa các đặc trưng HOG (Histogram of Oriented Gradients) của các ảnh đã được làm mờ. Hàm `hog` tính toán các đặc trưng hướng gradient của mỗi ảnh. HOG là một phương pháp đặc trưng mạnh mẽ để nhận dạng đối tượng, đặc biệt trong các tác vụ nhận dạng hình ảnh.

- `combined_features`: Là một danh sách chứa các đặc trưng kết hợp giữa `color_features` và `hog_features`. Hàm `np.concatenate` được sử dụng để ghép nối các đặc trưng màu sắc và HOG thành một vector đặc trưng duy nhất cho mỗi ảnh.

Cuối cùng, hàm `extract_features` trả về danh sách các đặc trưng kết hợp cho tất cả các ảnh đầu vào.

2.2 Thuật toán máy học

2.2.1 Thuật toán K-Nearest Neighbors (KNN)

2.2.1.1 Nguồn gốc

- **1951 - Evelyn Fix & Joseph Hodges:** Đưa ra khái niệm K-nearest neighbors (KNN) như một phương pháp phân loại không tham số, không

giả định về phân phối dữ liệu [7, 8].

- **1967 - Thomas Cover & Peter Hart:** Mở rộng và phát triển thuật toán KNN, chứng minh tính nhất quán trong lý thuyết quyết định [9].

Tính Nhất Quán (Consistency): Khi số lượng dữ liệu huấn luyện tăng lên vô hạn, thuật toán KNN sẽ hội tụ và đạt được hiệu suất tối ưu tương tự như phương pháp phân loại lý tưởng (tức là phương pháp có tỷ lệ lỗi thấp nhất có thể).

2.2.1.2 Nguyên lý hoạt động

Thuật toán K-Nearest Neighbors (KNN) bao gồm ba bước chính [10] như sau:

2.2.1.2.1 Tính khoảng cách

KNN tính toán khoảng cách giữa điểm cần dự đoán và tất cả các điểm trong tập huấn luyện. Một số loại khoảng cách phổ biến bao gồm:

- **Euclidean Distance:**

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

  Khoảng cách này đo độ dài đoạn thẳng nối hai điểm p và q.

- **Manhattan (Cityblock) Distance:**

$$d(p,q) = \sum_{i=1}^{n}|p_i - q_i|$$

  Được tính bằng tổng độ dài các đoạn song song với các trục tọa độ.

- **Cosine Similarity (Ngược lại là Cosine Distance):**

$$\text{cosine\_distance}(p,q) = 1 - \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}}$$

  Đo lường độ tương đồng giữa hai vector, thường được sử dụng trong các bài toán phân tích văn bản hoặc dữ liệu nhiều chiều.

- **Squared Euclidean Distance:**

$$d(p,q) = \sum_{i=1}^{n}(p_i - q_i)^2$$

  Đây là bình phương của khoảng cách Euclidean, thường được dùng để giảm chi phí tính căn bậc hai.

- **Chi-Square Distance:**

$$d(p,q) = \sum_{i=1}^{n}\frac{(p_i - q_i)^2}{p_i + q_i}$$

  Sử dụng phổ biến trong bài toán xử lý dữ liệu histogram.

- **Bhattacharyya Distance:**

$$d(p, q) = -\ln\left(\sum_{i=1}^{n}\sqrt{p_i q_i}\right)$$

Được dùng để đo độ tương đồng giữa hai phân phối xác suất.

- **Intersection Distance:**

$$d(p, q) = \sum_{i=1}^{n}\min(p_i, q_i)$$

Hiệu quả khi làm việc với các giá trị không âm, ví dụ như histogram.

2.2.1.2.2 Tìm K điểm gần nhất

Sau khi tính toán khoảng cách, thuật toán chọn K điểm dữ liệu có khoảng cách ngắn nhất tới điểm dữ liệu cần dự đoán. K là một tham số được chọn trước, ảnh hưởng đến hiệu suất của thuật toán. Nếu K quá nhỏ, mô hình có thể dễ bị nhiễu. Nếu K quá lớn, mô hình có thể mất đi độ chính xác do ảnh hưởng của các điểm không liên quan.

2.2.1.2.3 Dự đoán nhãn

KNN dự đoán nhãn của điểm mới dựa trên các nhãn của K điểm gần nhất. Các phương pháp dự đoán nhãn bao gồm:

- **Biểu quyết đa số (Majority Voting):** Nhãn được xác định bằng nhãn xuất hiện nhiều nhất trong K điểm gần nhất. Công thức:

$$\hat{y} = \arg\max_{y}\sum_{i=1}^{K}\mathbb{I}(y_i = y)$$

Trong đó:

  - $\mathbb{I}(y_i = y)$ là hàm chỉ thị, nhận giá trị 1 nếu $y_i = y$, và 0 trong trường hợp ngược lại.

- **Trọng số theo khoảng cách (Distance-Weighted Voting):** Mỗi láng giềng được gán một trọng số tỷ lệ nghịch với khoảng cách. Công thức:

$$\hat{y} = \arg\max_{y}\sum_{i=1}^{K}\frac{\mathbb{I}(y_i = y)}{d(p, q_i) + \epsilon}$$

Trong đó:

  - $d(p, q_i)$ là khoảng cách từ điểm cần dự đoán p tới láng giềng q_i.
  - ϵ là một giá trị rất nhỏ để tránh chia cho 0.

2.2.1.3 Cài đặt thuật toán

```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(
    n_neighbors=5,
    weights='distance',
    leaf_size=30,
    metric='euclidean'
)
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
```

Giải thích các tham số

- `n_neighbors`: Số lượng láng giềng gần nhất được sử dụng để xác định nhãn của một điểm mới.
  - Giá trị này ảnh hưởng trực tiếp đến độ chính xác và độ nhạy của mô hình.
  - Với `n_neighbors=5`, nhãn của điểm mới được xác định bởi 5 láng giềng gần nhất trong không gian đặc trưng.
- `weights`: Quy định cách tính trọng số của các láng giềng khi dự đoán nhãn.
  - `'uniform'`: Mọi láng giềng có trọng số như nhau.
  - `'distance'`: Láng giềng gần hơn có trọng số lớn hơn (dựa trên khoảng cách).
- `leaf_size`: Quy định kích thước lá (leaf) trong cây tìm kiếm K-D Tree hoặc Ball Tree.
  - **Giá trị nhỏ**: Tăng tốc độ truy vấn vì cây sẽ có nhiều tầng hơn và mỗi lá sẽ chứa ít điểm, giúp tìm kiếm nhanh chóng trong không gian phân tán. Tuy nhiên, cây sẽ chiếm nhiều bộ nhớ hơn vì có nhiều node.
  - **Giá trị lớn**: Giảm độ sâu của cây, giúp tiết kiệm bộ nhớ nhưng lại làm tăng thời gian truy vấn vì phải kiểm tra nhiều điểm trong các node lớn.
  - **Giá trị mặc định** là 30.

  Ví dụ:
  - `leaf_size = 1`: Cây sẽ có nhiều tầng với mỗi node chỉ chứa 1 điểm, giúp truy vấn nhanh nhưng yêu cầu bộ nhớ cao.
  - `leaf_size = 50`: Cây sẽ có ít tầng với mỗi node chứa 50 điểm, tiết kiệm bộ nhớ nhưng truy vấn sẽ chậm hơn vì phải kiểm tra nhiều điểm trong node.
- `metric`: Định nghĩa loại khoảng cách được sử dụng để tính toán láng giềng. Các công thức metric đã được giải thích ở phần **Nguyên lý hoạt động**.

2.2.2 Thuật toán Support Vector Machine (SVM)

2.2.2.1 Nguồn gốc

- **1974 - Vladimir Vapnik và Alexey Chervonenkis:** Phát triển lý thuyết VC (Vapnik–Chervonenkis), đặt nền tảng lý thuyết cho các thuật toán học máy dựa trên biên tối đa và khả năng tổng quát hóa, cơ sở cho việc xây dựng SVM [11].
- **1982, 1995 - Vladimir Vapnik:** Giới thiệu thuật toán SVM, tập trung vào mô hình phân loại tuyến tính dựa trên nguyên lý biên tối đa. SVM sử dụng siêu phẳng tối ưu để phân chia các lớp dữ liệu, giảm thiểu lỗi phân loại [12].
- **1992 - Bernhard Boser, Isabelle Guyon và Vladimir Vapnik:** Đề xuất kernel trick, mở rộng khả năng của SVM để giải quyết các bài toán phân loại phi tuyến [13].

2.2.2.2 Nguyên lý hoạt động

SVM tìm kiếm một siêu phẳng (hyperplane) tối ưu trong không gian đặc trưng để phân chia các lớp dữ liệu sao cho khoảng cách giữa siêu phẳng và các điểm dữ liệu gần nhất (support vectors) là lớn nhất. Mục tiêu là tối đa hóa biên (margin), tức là khoảng cách giữa siêu phẳng và các điểm dữ liệu của các lớp khác nhau.

2.2.2.2.1 Tìm siêu phẳng tối ưu

Mục tiêu của SVM là tìm một siêu phẳng tối ưu trong không gian đặc trưng sao cho khoảng cách giữa siêu phẳng và các điểm gần nhất từ mỗi lớp (support vectors) là lớn nhất. Điều này tương đương với việc tối đa hóa biên (margin), được tính bởi:

$$\text{Margin} = \frac{2}{\|\mathbf{w}\|} \quad [14]$$

Trong đó:

- $\mathbf{w}$ là vector trọng số của siêu phẳng.

2.2.2.2.2 Hàm mục tiêu trong SVM

Hàm mục tiêu của SVM là tối thiểu hóa hàm chi phí, bao gồm hai thành phần: một phần về sai số (penalty) và một phần về biên tối đa. Công thức hàm mục tiêu là:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i \quad [13]$$

Trong đó:

- $\mathbf{w}$ là vector trọng số.
- b là bias (hệ số bù).
- y_i là nhãn của điểm dữ liệu ($y_i = \pm 1$).
- $\mathbf{x}_i$ là điểm dữ liệu.

2.2.2.2.3 Kernel Trick (Nếu dữ liệu phi tuyến)

Nếu dữ liệu không thể phân chia một cách rõ ràng bằng siêu phẳng trong không gian gốc (dữ liệu phi tuyến), SVM sử dụng kernel trick để ánh xạ dữ liệu vào không gian đặc trưng cao hơn, nơi phân chia trở thành tuyến tính. Hàm kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ có thể tính toán khoảng cách trong không gian cao mà không cần tính toán trực tiếp các tọa độ trong không gian đó. Một số hàm kernel phổ biến [15]

- **Linear kernel:**
$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$
  - Kernel tuyến tính đơn giản, sử dụng tích vô hướng giữa hai điểm dữ liệu.
- **Polynomial kernel:**
$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^d$$
  - Với d là bậc của đa thức, giúp mô hình học các biên phân tách phi tuyến phức tạp.
- **Radial Basis Function (RBF) kernel:**
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$
  - Kernel Gaussian, sử dụng khoảng cách Euclid để tính toán ảnh hưởng của các điểm dữ liệu.
- **Sigmoid kernel:**
$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\alpha \mathbf{x}_i^T \mathbf{x}_j + c\right)$$
  - Kernel sigmoid mô phỏng hàm kích hoạt trong mạng nơ-ron.
  - Tham số α điều chỉnh độ dốc, và c là hằng số dịch chuyển.

2.2.2.2.4 Dự đoán nhãn

Sau khi tìm được siêu phẳng tối ưu, SVM sử dụng siêu phẳng này để phân loại điểm mới. Điểm mới sẽ được phân vào lớp bên kia của siêu phẳng, tùy thuộc vào phía của siêu phẳng mà nó nằm. Hàm phân loại là:

$$\hat{y} = \text{sign}(z), \quad \text{với } z = \mathbf{w} \cdot \mathbf{x} + b$$

Trong đó:

- z là giá trị kết hợp tuyến tính của vector trọng số, vector đặc trưng và bias.
- $\mathbf{w}$ là vector trọng số.
- $\mathbf{x}$ là vector đặc trưng của điểm cần phân loại.
- b là bias.
- $\text{sign}(z)$ trả về $+1$ nếu $z \geq 0$ và -1 nếu $z < 0$.

2.2.2.3 Cài đặt thuật toán

```
from sklearn.svm import SVC

svc_model = SVC(C=1.0, degree=3, gamma=0.1, kernel='poly')
svc_model.fit(X_train, y_train)
y_pred = svc_model.predict(X_test)
```

Giải thích các tham số

- `C`: Tham số điều chỉnh mức phạt đối với các lỗi phân loại.
  - Giá trị nhỏ (ví dụ: C=0.1): Mô hình cho phép một số lỗi khi phân loại, điều này giúp mô hình dễ dàng chấp nhận sự không hoàn hảo trong dữ liệu và có khả năng tổng quát tốt hơn khi áp dụng vào dữ liệu mới.
  - Giá trị lớn (ví dụ: C=10): Mô hình yêu cầu phân loại chính xác hơn, ít chấp nhận lỗi, nhưng dễ dẫn đến overfitting.
- `degree`: Bậc của đa thức trong kernel='poly'. Quyết định độ phức tạp của mô hình khi sử dụng kernel đa thức.
  - Với `degree=3`, hàm kernel sẽ tương ứng với một đa thức bậc ba, giúp mô hình có khả năng học các đường biên phi tuyến phức tạp hơn.
- `gamma`: Hệ số quyết định mức độ ảnh hưởng của một điểm dữ liệu đơn lẻ trong hàm kernel.
  - Giá trị nhỏ (ví dụ: gamma=0.1) làm tăng mức độ ảnh hưởng, dẫn đến một mô hình tổng quát hóa tốt hơn nhưng ít chi tiết hơn.
  - Giá trị lớn (ví dụ: gamma=10) làm giảm mức độ ảnh hưởng, mô hình tập trung hơn vào các điểm dữ liệu gần, dễ overfit.
- `kernel`: Hàm kernel được sử dụng để ánh xạ dữ liệu từ không gian gốc sang không gian đặc trưng cao hơn.
  - `'poly'`: Sử dụng kernel đa thức, phù hợp cho các bài toán với biên phi tuyến phức tạp có thể được biểu diễn bằng đa thức.
  - `'linear'`: Sử dụng kernel tuyến tính.
  - `'rbf'`: Kernel Gaussian, phù hợp cho các biên phân tách phi tuyến mượt.
  - `'sigmoid'`: Kernel sigmoid, mô phỏng hàm kích hoạt của mạng nơ-ron.

2.3 Phương pháp GridSearchCV

GridSearchCV là một kỹ thuật quan trọng trong học máy, được sử dụng để tối ưu hóa tham số siêu (*hyperparameters*) nhằm tìm ra các giá trị tốt nhất cho một mô hình. Hiệu suất của một mô hình học máy phụ thuộc rất lớn vào các tham số siêu, vì chúng kiểm soát cách thức mô hình học và đưa ra quyết định. Tuy nhiên, việc xác định giá trị tối ưu cho các tham số siêu thường không thể biết trước và đòi hỏi quá trình thử nghiệm.

2.3.1 Nguyên lý hoạt động

GridSearchCV tự động hóa quá trình thử nghiệm các giá trị tham số siêu bằng cách thực hiện kiểm tra toàn diện trên tất cả các tổ hợp giá trị được xác định trước trong một không gian tham số. Sau khi thử nghiệm, GridSearchCV sẽ chọn ra tổ hợp tham số mang lại hiệu suất tốt nhất dựa trên một tiêu chí đánh giá nhất định, chẳng hạn như *accuracy*, *precision*, *recall*, hay *f1-score*.

2.3.2 Nhận xét chung về GridsearchCV

2.3.2.1 Ưu điểm

Việc thử nghiệm thủ công tất cả các tổ hợp tham số có thể rất tốn thời gian và công sức, đặc biệt đối với các mô hình phức tạp và tập dữ liệu lớn. GridSearchCV giúp tự động hóa quy trình này, tiết kiệm thời gian hơn so với phương pháp thủ công, đồng thời đảm bảo rằng mọi tổ hợp tham số đều được xem xét một cách hệ thống.

2.3.2.2 Nhược điểm

Quá trình thử nghiệm có thể rất tốn kém về mặt thời gian khi số lượng tham số và giá trị của chúng là rất lớn. Với các mô hình phức tạp, GridSearchCV có thể phải thử nghiệm một số lượng tổ hợp tham số cực kỳ lớn, điều này làm tăng thời gian huấn luyện. Hơn nữa, GridSearchCV không tối ưu hoá được việc lựa chọn tham số trong một không gian rộng lớn và có thể bỏ sót các kết quả tốt hơn nếu không gian tham số không được định nghĩa hợp lý.

2.3.3 Ứng dụng

GridSearchCV được tích hợp trong thư viện `scikit-learn`, thuộc gói `model_selection`. Nó hỗ trợ nhiều kỹ thuật đo lường hiệu suất để đảm bảo rằng mô hình không chỉ đạt hiệu suất cao trên tập huấn luyện mà còn hoạt động tốt trên dữ liệu chưa nhìn thấy, giúp giảm thiểu hiện tượng quá khớp (*overfitting*).

Cú pháp cơ bản của GridSearchCV thường bao gồm các thành phần chính [16]:

- `estimator`: Mô hình học máy cần tối ưu hóa, ví dụ `KNeighborsClassifier`, `SVC`, `RandomForestClassifier`, v.v. Đây là tham số bắt buộc phải có.
- `param_grid`: Một từ điển hoặc danh sách các từ điển, chứa các tham số cần tối ưu và các giá trị tương ứng để thử nghiệm.
- `cv`: Số lượng lần chia nhỏ dữ liệu để thực hiện *cross-validation*. Khi `cv=k`, dữ liệu sẽ được chia thành k tập con. Trong mỗi lần thử nghiệm, một tập con được sử dụng để đánh giá và $k-1$ tập con còn lại được dùng để huấn luyện. Điều này đảm bảo mô hình được đánh giá toàn diện và hiệu quả.
- `scoring`: Tiêu chí đánh giá hiệu suất của mô hình. GridSearchCV hỗ trợ nhiều độ đo khác nhau, như `'accuracy'`, `'f1_macro'`, `'roc_auc'`, v.v., tùy thuộc vào bài toán cụ thể. Người dùng có thể chọn tiêu chí phù hợp để tối ưu hóa mô hình.
- `verbose`: Mức độ chi tiết khi hiển thị kết quả trong quá trình tìm kiếm. Tham số này có thể nhận các giá trị từ 0 đến 3, với `verbose=0` là mức yên lặng hoàn toàn và `verbose=3` hiển thị thông tin chi tiết nhất.

2.3.4 GridSearchCV cho KNN

Trong bài toán này, GridSearchCV được áp dụng để tối ưu hóa các siêu tham số cho mô hình k-Nearest Neighbors (KNN). Các siêu tham số cần tối ưu hóa bao gồm:

- `n_neighbors`: Số lượng hàng xóm gần nhất được xét để dự đoán.
- `weights`: Phương pháp tính trọng số khi lấy trung bình (`uniform` sử dụng trọng số đều, `distance` sử dụng trọng số ngược khoảng cách).
- `leaf_size`: Kích thước lá trong cây KD-Tree, ảnh hưởng đến tốc độ tìm kiếm lân cận.
- `metric`: Hàm khoảng cách được sử dụng để đo độ tương đồng giữa các điểm, bao gồm:
  - `cityblock`: Khoảng cách Manhattan.
  - `cosine`: Khoảng cách Cosine.
  - `sqeuclidean`: Khoảng cách Euclid bình phương.
  - `chi_square_distance`: Khoảng cách χ^2.
  - `bhattacharyya_distance`: Khoảng cách Bhattacharyya.
  - `intersection_distance`: Khoảng cách giao nhau.

Cài đặt GridSearchCV để tìm tham số tốt nhất cho mô hình KNN:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Định nghĩa không gian tham số
param_grid = {
    'n_neighbors': [3, 4, 5, 6, 7, 10],
    'weights': ['uniform', 'distance'],
    'leaf_size': [5, 10, 20, 30, 40, 50],
    'metric': [
        'cityblock', 'cosine',
        'sqeuclidean', 'chi_square_distance',
        'bhattacharyya_distance', 'intersection_distance'
    ]
}

# Khởi tạo mô hình k-NN
knn_model = KNeighborsClassifier()

# Grid Search
grid_search_knn = GridSearchCV(
    knn_model,
    param_grid,
    cv=3,
    scoring='f1_macro',
    verbose=3
)

# Huấn luyện mô hình
grid_search_knn.fit(X_train, y_train)
```

Giải thích các tham số chính:

- `cv=3`: Số lượng lần chia nhỏ dữ liệu để thực hiện Cross-Validation. Khi `cv=3`, dữ liệu được chia thành 3 tập con (folds). Trong mỗi lần thử nghiệm, một tập được dùng để đánh giá và hai tập còn lại được sử dụng để huấn luyện. Điều này đảm bảo tính tổng quát và ổn định trong đánh giá mô hình.

- `scoring='f1_macro'`: Đây là tiêu chí đánh giá hiệu năng của mô hình trong Grid Search. Với `f1_macro`, F_1-score được tính riêng cho từng lớp, sau đó trung bình cộng các giá trị này. Điều này đặc biệt hữu ích trong trường hợp dữ liệu không cân bằng, vì nó cân nhắc đóng góp từ tất cả các lớp thay vì chỉ tập trung vào lớp có số lượng lớn.

  Công thức tính `f1_macro`:

  $$F_{\text{1-macro}} = \frac{1}{N} \sum_{i=1}^{N} F_{1,i}$$

  Trong đó:

  - N: Số lượng lớp.
  - $F_{1,i}$: F_1-score của lớp thứ i, được tính như sau:

    $$F_{1,i} = \frac{2 \cdot P_i \cdot R_i}{P_i + R_i}$$

  - P_i (Precision của lớp i):

    $$P_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i}$$

  - R_i (Recall của lớp i):

    $$R_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i}$$

  - $\text{TP}_i, \text{FP}_i, \text{FN}_i$: Số lượng True Positive, False Positive, và False Negative của lớp i.

- `verbose`: Điều khiển mức độ chi tiết của thông báo trong quá trình chạy GridSearchCV.

  - `verbose=0` (mặc định): Không in ra bất kỳ thông tin nào trong quá trình tìm kiếm.
  - `verbose=1`: Chỉ in thông tin cơ bản, ví dụ số lượng tổ hợp siêu tham số đã chạy và đang chạy.
  - `verbose=2`: In chi tiết hơn, bao gồm thông tin về từng lần huấn luyện trên các tập con (folds) của dữ liệu.
  - `verbose=3` trở lên: In thông tin rất chi tiết, bao gồm cả kết quả đánh giá từng tổ hợp siêu tham số trên mỗi fold.

Tham số tốt nhất của KNN sau khi dùng GridsearchCV: `{'leaf_size': 5, 'metric': 'cityblock', 'n_neighbors': 3, 'weights': 'distance'}`

2.3.5 GridSearchCV cho SVM

Tương tự như KNN, GridSearchCV được áp dụng để tối ưu hóa siêu tham số cho SVM. Không gian siêu tham số bao gồm:

- `C`: Hệ số điều chỉnh trade-off giữa độ rộng biên và lỗi phân loại.
- `kernel`: Hàm kernel, quyết định cách dữ liệu được ánh xạ sang không gian đặc trưng. Bao gồm các giá trị: `linear`, `poly`, `rbf`, và `sigmoid`.
- `gamma`: Tham số điều chỉnh cho các kernel phi tuyến như `rbf`, ảnh hưởng đến độ mịn của mô hình.
- `degree`: Bậc của hàm đa thức, chỉ áp dụng cho kernel `poly`.

Cài đặt GridSearchCV để tìm tham số tốt nhất cho mô hình SVM:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Định nghĩa không gian tham số
param_grid = {
    'C': [0.1, 0.2, 0.3, 0.4],
    'kernel': ['rbf', 'linear', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto', 0.1, 0.01, 0.001],
    'degree': [2, 3, 4],
}

# Khởi tạo mô hình SVM
svm_model = SVC(random_state=42)

# Grid Search
grid_search_svm = GridSearchCV(
    estimator=svm_model,
    param_grid=param_grid,
    cv=3,
    scoring='f1_macro',
    verbose=3
)

# Huấn luyện mô hình
grid_search_svm.fit(X_train, y_train)
```

Giải thích các tham số chính: Tương tự KNN, GridSearchCV với SVM cũng sử dụng các tham số `cv=3`, `scoring='f1_macro'`, và `verbose=3`.

Tham số tốt nhất của SVM sau khi dùng GridsearchCV: `{'C': 0.1, 'degree': 3, 'gamma': 0.1, 'kernel': 'poly'}`

CHƯƠNG 3. THỰC NGHIỆM VÀ KẾT QUẢ

3.1 Bộ dữ liệu

3.1.1 Tổng quan về bộ dữ liệu

Bộ dữ liệu sử dụng trong nghiên cứu được xây dựng từ nhiều nguồn khác nhau nhằm đảm bảo tính đa dạng, chất lượng và độ phù hợp với bài toán phân loại biển báo giao thông tại Việt Nam. Cụ thể:

- **Dữ liệu kế thừa:** Bộ dữ liệu gốc được lấy từ đồ án của một nhóm nghiên cứu trước, đã qua quá trình lọc và gắn nhãn lại để đảm bảo tính chính xác [17].
- **Ảnh thực tế:** Một số lượng lớn ảnh được chụp bổ sung tại các khu vực giao thông xung quanh Làng Đại học, bao gồm các khu vực như: cầu vượt Linh Xuân, dọc Xa lộ Hà Nội, Ga Dĩ An, và Xa lộ Đại Hàn...
- **Ảnh tham khảo:** Một phần dữ liệu được thu thập từ các nguồn trực tuyến như Google, sau đó chọn lọc kỹ càng để đảm bảo chất lượng và sự tương đồng với các ảnh thực tế.

3.1.2 Cấu trúc bộ dữ liệu

Bộ dữ liệu gồm tổng cộng 1565 ảnh, được chia thành hai tập:

- **Tập huấn luyện (Training set):** Gồm 1415 ảnh, chiếm khoảng 90% dữ liệu, được sử dụng để huấn luyện mô hình.
- **Tập kiểm tra (Test set):** Gồm 150 ảnh, chiếm khoảng 10% dữ liệu, được sử dụng để đánh giá hiệu quả của mô hình.

Bộ dữ liệu được chia thành 5 nhóm (class) chính dựa trên loại biển báo giao thông:

- **Cấm**
- **Chỉ dẫn**
- **Hiệu lệnh**
- **Nguy hiểm**
- **Phụ**

Bảng dưới đây thể hiện số lượng ảnh huấn luyện và kiểm tra cho từng nhóm:

Nhóm (Class)	**Số lượng ảnh huấn luyện**	**Số lượng ảnh kiểm tra**
Cấm	330	34
Chỉ dẫn	320	33
Hiệu lệnh	272	31
Nguy hiểm	268	29
Phụ	225	23

Bảng 1: Số lượng ảnh trong bộ dữ liệu theo từng nhóm

3.2 Độ đo

Để đánh giá hiệu suất của mô hình, các độ đo phổ biến trong bài toán phân loại đã được sử dụng, bao gồm:

3.2.1 Accuracy

Khái niệm: Accuracy đo lường tỷ lệ dự đoán đúng trên tổng số dự đoán. **Công thức:**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Trong đó:

- TP (True Positive): Số mẫu dương được dự đoán đúng.
- TN (True Negative): Số mẫu âm được dự đoán đúng.
- FP (False Positive): Số mẫu âm bị dự đoán nhầm là dương.
- FN (False Negative): Số mẫu dương bị dự đoán nhầm là âm.

3.2.2 Precision

Khái niệm: Precision đo lường tỷ lệ mẫu dự đoán đúng trong số các mẫu được dự đoán là dương. **Công thức:**

$$\text{Precision} = \frac{TP}{TP + FP}$$

Trong đó:

- TP (True Positive): Số mẫu dương được dự đoán đúng.
- FP (False Positive): Số mẫu âm bị dự đoán nhầm là dương.

3.2.3 Recall

Khái niệm: Recall thể hiện tỷ lệ mẫu dương thực sự được phát hiện chính xác trong số các mẫu dương thực tế. **Công thức:**

$$\text{Recall} = \frac{TP}{TP + FN}$$

Trong đó:

- TP (True Positive): Số mẫu dương được dự đoán đúng.
- FN (False Negative): Số mẫu dương bị dự đoán nhầm là âm.

3.2.4 F1-score

Khái niệm: F1-score là trung bình điều hòa giữa Precision và Recall, cung cấp một thước đo cân bằng. **Công thức:**

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.3 Kết quả thực nghiệm

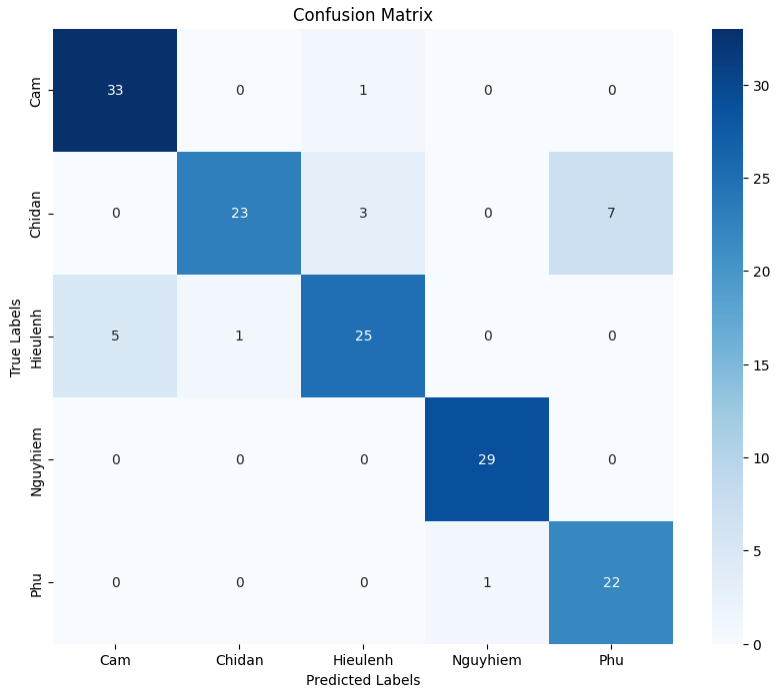
3.3.1 Tổng quan về các kết quả

3.3.1.1 KNN

Kết quả dự đoán của mô hình KNN được tóm tắt như sau:

- Nhóm *Cấm*: Dự đoán đúng 33 biển báo. Chỉ có 1 biển bị nhầm lẫn với nhóm *Hiệu lệnh*, cho thấy KNN nhận diện tốt các đặc trưng của nhóm này.
- Nhóm *Chỉ dẫn*: Dự đoán đúng 23 biển báo. Tuy nhiên, có 3 biển bị nhầm với nhóm *Hiệu lệnh* và 7 biển nhầm với nhóm *Phụ*, điều này cho thấy khó khăn trong việc phân biệt giữa các biển nền xanh và biển có chi tiết phức tạp.
- Nhóm *Hiệu lệnh*: Dự đoán đúng 25 biển báo. 5 biển bị nhầm với nhóm *Cấm* và 1 biển bị nhầm với nhóm *Chỉ dẫn*, điều này có thể bắt nguồn từ sự tương đồng về hình dạng.
- Nhóm *Nguy hiểm*: Dự đoán đúng toàn bộ 29 biển báo, cho thấy khả năng nhận diện rất tốt nhờ đặc trưng viền đỏ hình tam giác dễ nhận diện.
- Nhóm *Phụ*: Dự đoán đúng 22 biển báo. Chỉ có 1 biển bị nhầm lẫn với nhóm *Nguy hiểm*, điều đó cho thấy hiệu suất nằm ở mức ổn đối với nhóm này.

Nhận xét: Mô hình KNN đạt hiệu suất cao đối với nhóm *Cấm* và *Nguy hiểm*, nhờ các đặc trưng dễ nhận diện (đối với biển *Cấm* thì chúng thường có hình tròn với viền đỏ, còn với biển *Nguy hiểm* thì chúng thường có hình dạng là tam giác). Tuy nhiên, các nhầm lẫn chủ yếu xảy ra với nhóm *Chỉ dẫn* và *Phụ*, cũng như giữa nhóm *Hiệu lệnh* và *Cấm*.



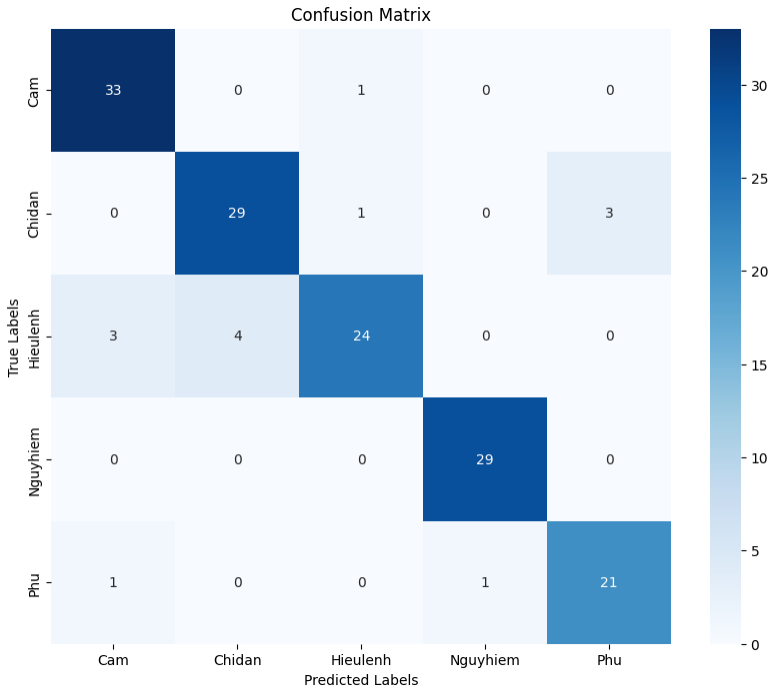
Hình 4: Confusion Matrix của mô hình KNN

3.3.1.2 SVM

Kết quả dự đoán của mô hình SVM được tóm tắt như sau:

- **Nhóm *Cấm***: Dự đoán đúng 33 biển báo. Chỉ có 1 biển bị nhầm lẫn với nhóm *Hiệu lệnh*, và không có sự nhầm lẫn với các nhóm khác. Điều này cho thấy SVM phân biệt rất tốt nhóm *Cấm*.
- **Nhóm *Chỉ dẫn***: Dự đoán đúng 29 biển báo. Tuy nhiên, có 1 biển bị nhầm với nhóm *Hiệu lệnh* và 3 biển nhầm với nhóm *Phụ*. Điều này phản ánh sự cải thiện so với KNN nhưng vẫn còn nhầm lẫn giữa các biển nền xanh với chi tiết phức tạp.
- **Nhóm *Hiệu lệnh***: Dự đoán đúng 24 biển báo. Có 3 biển nhầm với nhóm *Cấm* và 4 biển nhầm với nhóm *Chỉ dẫn*, phản ánh rằng các chi tiết phức tạp trong nhóm này vẫn là một thách thức đối với mô hình.
- **Nhóm *Nguy hiểm***: Dự đoán đúng toàn bộ 29 biển báo mà không có bất kỳ sự nhầm lẫn nào. Điều này cho thấy SVM nhận diện xuất sắc các đặc trưng nổi bật của nhóm *Nguy hiểm*.
- **Nhóm *Phụ***: Dự đoán đúng 21 biển báo. Có 1 biển bị nhầm lẫn với nhóm *Cấm* và 1 biển nhầm với nhóm *Nguy hiểm*. Điều này cho thấy hiệu suất của SVM ở nhóm này vẫn còn hạn chế, dù có cải thiện hơn KNN.

Nhận xét: Mô hình SVM cải thiện đáng kể so với KNN ở nhóm *Chỉ dẫn*, với số lượng biển báo được dự đoán đúng tăng lên đáng kể. Nhóm *Nguy hiểm* tiếp tục đạt hiệu suất tối đa, không có nhầm lẫn. Tuy nhiên, nhóm *Hiệu lệnh* vẫn gây khó khăn cho mô hình, với các biển báo bị nhầm lẫn với nhóm *Cấm* và *Chỉ dẫn*. Điều này cho thấy cần tập trung vào việc cải thiện khả năng phân biệt chi tiết phức tạp và tương đồng giữa các nhóm này.



Hình 5: Confusion Matrix của mô hình SVM

3.3.2 Nhận xét về kết quả

Classification report cung cấp thông tin chi tiết về các chỉ số như *precision*, *recall*, và *F1-score* cho từng nhóm biển báo, qua đó đánh giá hiệu suất toàn diện của mô hình KNN. Đây là một công cụ quan trọng để hiểu rõ hơn về những điểm mạnh và hạn chế trong khả năng phân loại của mô hình trên từng loại biển báo cụ thể.

3.3.2.1 KNN

Mô hình KNN đạt độ chính xác tổng thể (*accuracy*) ở mức 88%, thể hiện khả năng phân loại tốt trên tập dữ liệu kiểm tra. Tuy nhiên, các chỉ số hiệu suất của từng nhóm biển báo cho thấy sự khác biệt đáng kể:

- **Nhóm *Cấm***: Đạt hiệu suất cao nhất với *recall* là 97% và *F1-score* là 92%. Mô hình nhận diện rất tốt các biển báo thuộc nhóm này nhờ đặc trưng hình học và màu sắc nổi bật, giúp dễ dàng phân biệt với các nhóm khác.
- **Nhóm *Chỉ dẫn***: Mặc dù có *precision* cao (96%), *recall* chỉ đạt 70%, dẫn đến *F1-score* là 81%. Điều này cho thấy mô hình dễ bỏ sót các biển báo thuộc nhóm này, đặc biệt là những biển báo có nhiều chi tiết hoặc tương đồng với các nhóm khác.
- **Nhóm *Hiệu lệnh***: Hiệu suất khá tốt với *F1-score* là 83%. Tuy nhiên, mô hình vẫn gặp khó khăn trong việc phân biệt nhóm này với các nhóm khác do chi tiết phức tạp hoặc sự tương đồng giữa các biển báo, chẳng hạn như về màu sắc hoặc hình dạng.
- **Nhóm *Nguy hiểm***: Đây là nhóm được phân loại tốt nhất, đạt cả *recall* và *precision* rất cao, lần lượt là 100% và 97%. Điều này cho thấy mô hình phát hiện chính xác hầu hết các biển báo thuộc nhóm này nhờ vào đặc điểm viền đỏ tam giác rất dễ nhận biết.
- **Nhóm *Phụ***: Hiệu suất khá tốt với *recall* là 96% và *F1-score* là 85%. Tuy nhiên, *precision* chỉ đạt 76%, phản ánh sự nhầm lẫn giữa nhóm này và các nhóm khác, đặc biệt là những biển báo có kích thước nhỏ hoặc ít nổi bật trong hình ảnh.

Nhìn chung, KNN hoạt động tốt với các nhóm biển báo có đặc trưng hình học và màu sắc nổi bật như *Cấm* và *Nguy hiểm*. Tuy nhiên, các nhóm như *Chỉ dẫn*, *Hiệu lệnh*, và *Phụ* vẫn gây khó khăn do đặc điểm phức tạp hoặc sự tương đồng giữa các nhóm, dẫn đến tỷ lệ bỏ sót và nhầm lẫn đáng kể.

	precision	recall	f1-score	support
Cam	0.87	0.97	0.92	34
Chidan	0.96	0.70	0.81	33
Hieulenh	0.86	0.81	0.83	31
Nguyhiem	0.97	1.00	0.98	29
Phu	0.76	0.96	0.85	23
accuracy			0.88	150
macro avg	0.88	0.89	0.88	150
weighted avg	0.89	0.88	0.88	150

Hình 6: Classification Report của mô hình KNN

3.3.2.2 SVM

Mô hình SVM đạt độ chính xác tổng thể (*accuracy*) ở mức 91%, vượt trội hơn so với KNN. Các chỉ số *precision*, *recall*, và *F1-score* cho từng nhóm biển báo cũng thể hiện sự cải thiện đáng kể:

- **Nhóm *Cấm***: Mô hình đạt *recall* là 97% và *F1-score* là 93%, cho thấy khả năng nhận diện khá tốt đối với nhóm này.
- **Nhóm *Chỉ dẫn***: Cải thiện rõ rệt với cả *precision* và *recall* đạt 88%, dẫn đến *F1-score* là 88%. Điều này thể hiện rằng SVM giảm đáng kể tỷ lệ bỏ sót so với KNN.
- **Nhóm *Hiệu lệnh***: Mặc dù *precision* cao (92%), *recall* chỉ đạt 77%, dẫn đến *F1-score* là 84%. Nhóm này vẫn là thách thức với SVM do đặc trưng phức tạp của các biển báo.
- **Nhóm *Nguy hiểm***: Hiệu suất rất cao, với *recall* đạt 100% và *F1-score* là 98%, tương tự KNN. Mô hình phân loại chính xác hầu hết các biển báo thuộc nhóm này.
- **Nhóm *Phụ***: Với *recall* là 91% và *F1-score* là 89%, nhóm này đạt hiệu suất cao hơn KNN, cho thấy khả năng phân biệt của SVM tốt hơn đối với nhóm có đặc điểm ít rõ ràng hơn.

Nhìn chung, SVM vượt trội hơn KNN trong việc nhận diện các nhóm biển báo phức tạp như *Chỉ dẫn* và *Phụ*, nhờ vào khả năng tối ưu hóa siêu phẳng phân tách. Tuy nhiên, nhóm *Hiệu lệnh* vẫn gây khó khăn cho mô hình do sự đa dạng trong đặc trưng biển báo.

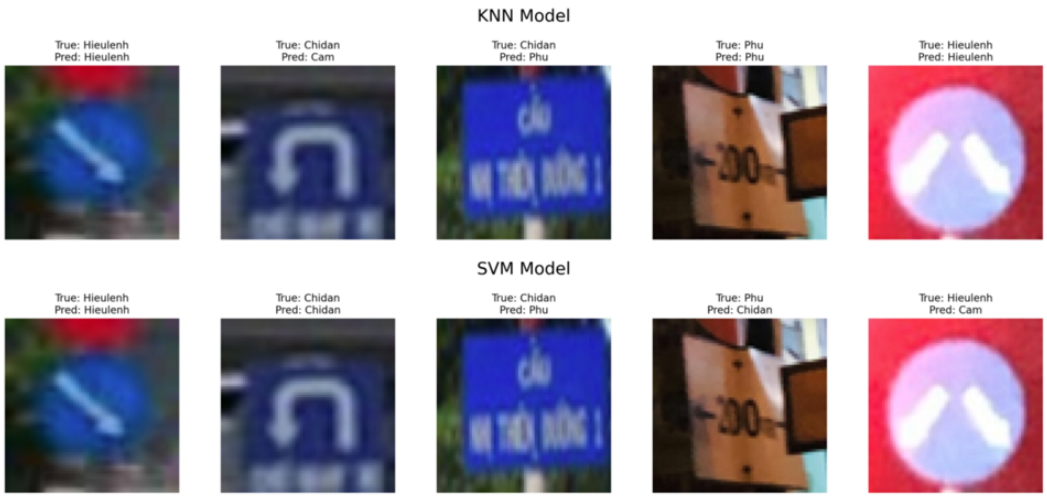
	precision	recall	f1-score	support
Cam	0.89	0.97	0.93	34
Chidan	0.88	0.88	0.88	33
Hieulenh	0.92	0.77	0.84	31
Nguyhiem	0.97	1.00	0.98	29
Phu	0.88	0.91	0.89	23
accuracy			0.91	150
macro avg	0.91	0.91	0.91	150
weighted avg	0.91	0.91	0.91	150

Hình 7: Classification Report của mô hình SVM

3.4 Minh họa các ảnh phân loại

3.4.1 Ảnh từ tập test

Dưới đây là các ảnh từ tập kiểm tra, được phân loại bởi các mô hình KNN và SVM.



Hình 8: Ảnh từ tập test

3.4.2 Ảnh không tồn tại trong bộ dữ liệu

Các ảnh dưới đây không tồn tại trong bộ dữ liệu huấn luyện và kiểm tra, và được phân loại bởi các mô hình KNN và SVM.

Hình 9: Ảnh không tồn tại trong bộ dữ liệu

KẾT LUẬN

Kết luận chung

- Với tập dataset đã được chuẩn bị, trong ba mô hình mà nhóm chọn, mô hình **SVM** đạt độ chính xác cao nhất với **91%**. Mô hình này cho thấy khả năng phân loại tốt đối với hầu hết các nhóm biển báo.

- Nhóm *Cấm* được phân loại tốt nhất trên cả ba mô hình, với độ chính xác cao. Nguyên nhân có thể do số lượng ảnh của nhóm *Cấm* cao hơn so với các nhóm khác, và đặc trưng của biển báo *Cấm* (thường có viền và nền màu đỏ) giúp mô hình dễ dàng nhận diện nhờ sự khác biệt màu sắc mạnh mẽ, ít bị nhầm lẫn với các biển báo khác.

- Nhóm *Hiệu lệnh* và *Chỉ dẫn* thường bị nhầm lẫn do màu sắc và hình dạng tương đồng. Điều này thể hiện sự cần thiết phải cải thiện khả năng phân biệt chi tiết giữa các biển báo này.

- Điều kiện môi trường có thể gây khó khăn cho việc nhận diện biển báo, chẳng hạn như khi biển báo bị che khuất bởi ánh sáng yếu, thời tiết xấu như mưa hoặc sương mù, hoặc khi hình ảnh có chất lượng kém. Những yếu tố này ảnh hưởng đến khả năng quan sát và phân loại chính xác của mô hình.

Hướng phát triển

- **Mở rộng và cải thiện dataset:**

- Thu thập thêm dữ liệu, đặc biệt là những biển báo trong các điều kiện môi trường khác nhau (ánh sáng yếu, mưa, sương mù, các góc chụp khác nhau) để tăng tính đa dạng và khả năng tổng quát hóa của mô hình.

- Sử dụng dữ liệu từ nhiều nguồn khác nhau, bao gồm cả hình ảnh từ các camera khác nhau, để cải thiện độ chính xác và tính tổng quát của mô hình.

- **Cải thiện tiền xử lý dữ liệu:**

- Áp dụng các kỹ thuật tiền xử lý như cân bằng sáng và tăng cường ảnh để cải thiện chất lượng hình ảnh đầu vào.

- Phát hiện và loại bỏ nhiễu trong hình ảnh nhằm nâng cao độ chính xác của mô hình.

- **Nâng cấp mô hình học máy:**

- Triển khai các mô hình học sâu để cải thiện khả năng nhận diện và phân loại biển báo giao thông.

- **Cải thiện đặc trưng trích xuất:**

- Kết hợp nhiều phương pháp trích xuất đặc trưng để tăng cường độ chính xác và khả năng phân loại của mô hình.

TÀI LIỆU THAM KHẢO

Tài liệu

[1] Rafael C. Gonzalez **and** Richard E. Woods. *Digital Image Processing, Second Edition*. 2019. URL: https://www.researchgate.net/publication/333856607_Digital_Image_Processing_Second_Edition.

[2] *Demystifying Color Histograms*. URL: https://zilliz.com/learn/demystifying-color-histograms.

[3] *Histogram of Oriented Gradients (HOG)*. URL: https://learnopencv.com/histogram-of-oriented-gradients/.

[4] *Histogram of Oriented Gradients (HOG) for Multiclass Image Classification and Image Recommendation*. URL: https://medium.com/swlh/histogram-of-oriented-gradients-hog-for-multiclass-image-classification-and-image-recommendation-cf0ea2caaae8.

[5] M. S. Nixon **and** A. S. Aguado. *Feature Extraction and Image Processing*. Page 86. 2018. URL: https://scian.cl/scientific-image-analysis/wp-content/uploads/2018/09/Feature.extraction.and_.image_.processing_Nixon-Aguado-1.pdf.

[6] M. S. Nixon **and** A. S. Aguado. *Feature Extraction and Image Processing*. Page 89. 2018. URL: https://scian.cl/scientific-image-analysis/wp-content/uploads/2018/09/Feature.extraction.and_.image_.processing_Nixon-Aguado-1.pdf.

[7] E. Fix **and** J.L. Hodges. "The nearest neighbor method for classification". **in***JSTOR*: (1951). URL: https://www.jstor.org/stable/1403796?seq=1.

[8] Evelyn Fix **and** Joseph Hodges. *K-Nearest Neighbor History*. 1951. URL: https://holypython.com/knn/k-nearest-neighbor-knn-history/.

[9] Thomas Cover **and** Peter Hart. "Nearest neighbor pattern classification". **in***IEEE Transactions on Information Theory*: (1967). URL: https://ieeexplore.ieee.org/document/1053964/authors#authors.

[10] *K-Nearest Neighbor Principle*. URL: https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4.

[11] Vladimir Vapnik **and** Alexey Chervonenkis. *Pattern Recognition Theory*. 1974. URL: https://cml.rhul.ac.uk/resources/PatternRecognition/index.htm.

[12] Vladimir Vapnik **and** Corinna Cortes. "Support-Vector Networks". **in***Machine Learning*: (1995). URL: http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf.

[13] Bernhard Boser, Isabelle Guyon **and** Vladimir Vapnik. "A Training Algorithm for Optimal Margin Classifiers". **in***Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*: (1992). URL: https://dl.acm.org/doi/pdf/10.1145/130385.130401.

[14] Math StackExchange. *Why is the SVM margin equal to* $\frac{2}{\|\mathbf{w}\|}$*?* URL: https://math.stackexchange.com/questions/1305925/why-is-the-svm-margin-equal-to-frac2-mathbfw.

[15] GeeksforGeeks. *Support Vector Machine Algorithm.* URL: https://www.geeksforgeeks.org/support-vector-machine-algorithm/.

[16] Great Learning. *GridSearchCV: A Comprehensive Guide with Examples.* URL: https://www.mygreatlearning.com/blog/gridsearchcv/.

[17] Nguyễn Hoàng Hiệp, Văn Tiến Hoàng **and** Phạm Đức Huy Hoàng. *Vietnam Traffic Signs Dataset.* URL: https://universe.roboflow.com/nkhhmh/vietnam-traffic-signs-hkrxj/dataset/2.