ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KỲ

MÔN: NHẬP MÔN THỊ GIÁC MÁY TÍNH

ĐỀ TÀI

NHẬN DIỆN XE Ô TÔ TRONG ẢNH

Giảng viên hướng dẫn: TS. Mai Tiến Dũng

Lớp: CS231.P11

Sinh viên thực hiện:

MSSV	Họ và tên
22520550	Lương Anh Huy
22520967	Hồng Khải Nguyên

Tháng 12/2024

LỜI CẢM ƠN

Trong thời đại ngày nay, vai trò của công nghệ thông tin trong đời sống con người ngày càng trở nên quan trọng và không thể thiếu. Việc áp dụng công nghệ vào các lĩnh vực khác nhau, đặc biệt là trong giao thông, đã mang lại nhiều lợi ích to lớn. Một trong những ứng dụng nổi bật là việc nhận diện xe ô tô trong ảnh, hỗ trợ các hệ thống quản lý giao thông, giám sát an ninh và tự động hóa quy trình giám sát phương tiện. Các hệ thống này không chỉ giúp tiết kiệm thời gian, giảm thiểu sai sót của con người mà còn nâng cao hiệu quả trong việc đảm bảo an toàn giao thông và quản lý phương tiện.

Với mong muốn góp phần vào việc giải quyết vấn đề trên, nhóm sinh viên chúng em đã thực hiện đồ án môn học với đề tài "Nhận diện xe ô tô trong ảnh". Đồ án được thực hiện dưới sự hướng dẫn tận tình của thầy Mai Tiến Dũng, giảng viên lớp Nhập môn Thị giác Máy tính - CS231.P11.

Thông qua đồ án này, chúng em có cơ hội áp dụng những kiến thức đã học, rèn luyện kỹ năng thực hành, đồng thời tích lũy thêm kinh nghiệm thực tế về các phương pháp xử lý và phân loại đối tượng trong hình ảnh.

Nhóm xin gửi lời cảm ơn chân thành đến thầy Mai Tiến Dũng vì đã dành thời gian và tâm huyết hướng dẫn, chỉnh sửa, góp ý, hỗ trợ chúng em trong suốt quá trình học tập và thực hiện đồ án.

Trân trọng cảm ơn!

BẢNG PHÂN CÔNG VÀ ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

Công việc	Huy	Nguyên	Tiến độ
Phân công công việc (Leader)	X		Hoàn thành
Sử dụng HOG	X	X	Hoàn thành
Sử dụng KNN	X		Hoàn thành
Sử dụng SVM		X	Hoàn thành
So sánh kết quả của các mô hình	X	X	Hoàn thành
Làm slide	X		Hoàn thành
Viết báo cáo	X	X	Hoàn thành
Chỉnh sửa sau khi được thầy nhận xét	X		Hoàn thành

MỤC LỤC

1. GIỚI THIỆU BÀI TOÁN

1.1 Lý do lựa chọn đề tài

Trong bối cảnh công nghệ ngày càng phát triển, thị giác máy tính (Computer Vision) đóng vai trò quan trọng trong nhiều lĩnh vực, từ giám sát giao thông, nhận diện khuôn mặt, đến phát triển các hệ thống xe tự lái. Một trong những bài toán cơ bản nhưng có giá trị ứng dụng cao là phát hiện và nhận diện ô tô trong hình ảnh. Việc tự động hóa quy trình này không chỉ giúp tối ưu hóa quản lý giao thông mà còn hỗ trợ phát triển các giải pháp an ninh giám sát hiện đại.

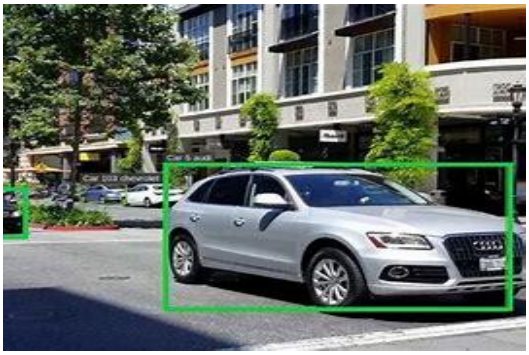
Đồ án này tập trung xây dựng hệ thống phát hiện ô tô trong ảnh bằng cách áp dụng các phương pháp trích xuất đặc trưng (HOG) kết hợp với hai thuật toán Machine Learning phổ biến là KNN (K-Nearest Neighbors) và SVM (Support Vector Machine). Thông qua việc thực hiện, nhóm mong muốn:

- Áp dụng lý thuyết đã học để giải quyết bài toán thực tế.

- Hiểu rõ và so sánh hiệu quả của các thuật toán Machine Learning trong lĩnh vực thị giác máy tính

- Nâng cao khả năng tư duy phân tích, xử lý dữ liệu và thiết kế hệ thống nhận diện hình ảnh.

Đề tài không chỉ là một bài học thực hành về Machine Learning mà còn là bước đệm quan trọng để nhóm tiếp cận với các bài toán thị giác máy tính phức tạp hơn trong tương lai.



1.2 Phát biểu bài toán

Input: Dataset gồm các ảnh đã được gắn nhãn có ô tô hoặc không có ô tô và ảnh số bất kỳ có thể có chứa hoặc không chứa ô tô trong ảnh

Output: Kết quả phân loại, cho biết hình đó có ô tô hoặc không có ô tô

2. PHƯƠNG PHÁP

2.1 Histogram of Oriented Gradient

- Histogram of Oriented Gradient (HOG) là một loại “feature discriptor”. Mục đích của “feature discriptor” là trừu tượng hóa đối tượng bằng cách trích xuất ra những đặc trưng của đối tượng đó và bỏ đi những thông tin không ích. Vì vậy, HOG được sử dụng chủ yếu để mô tả hình dạng và sự xuất hiện của một đối tượng trong ảnh.

- Bản chất của phương pháp HOG là sử dụng thông tin về sự phân bố của các cường độ gradient (intensity gradient) hoặc của hướng biên (edge directins) để mô tả các đối tượng cục bộ trong ảnh. Các toán tử HOG được cài đặt bằng cách chia nhỏ một bức ảnh thành các vùng con, được gọi là "tế bào" (cells) và với mỗi cell, ta sẽ tính toán một histogram về các hướng của gradients cho các điểm nằm trong cell. Ghép các histogram lại với nhau ta sẽ có một biểu diễn cho bức ảnh ban đầu.

- Để tăng cường hiệu năng nhận dạng, các histogram cục bộ có thể được chuẩn hóa về độ tương phản bằng cách tính một ngưỡng cường độ trong một vùng lớn hơn cell, gọi là các khối (blocks) và sử dụng giá trị ngưỡng đó để chuẩn hóa tất cả các cell trong khối. Kết quả sau bước chuẩn hóa sẽ là một vector đặc trưng có tính bất biến cao hơn đối với các thay đổi về điều kiện ánh sáng

- 5 bước chính để xây dựng một vector HOG cho hình ảnh, gồm:

+ Bước 1: Tiền xử lý

+ Bước 2: Tính gradient

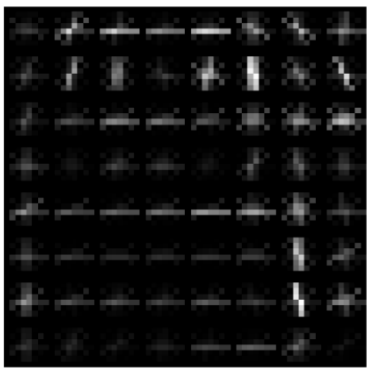
+ Bước 3: Tính vector đặc trưng cho từng ô (cells)

+ Bước 4: Chuẩn hóa các khối (blocks)

+ Bước 5: Tính toán vector HOG

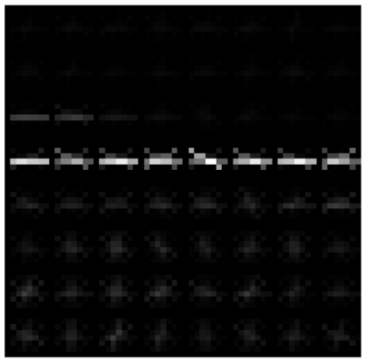
- Ví dụ minh họa khi áp dụng HOG lên hình ảnh:

Hình ảnh ô tô ban đầu



Hình ảnh sau khi áp dụng HOG

Hình ảnh không có ô tô ban đầu

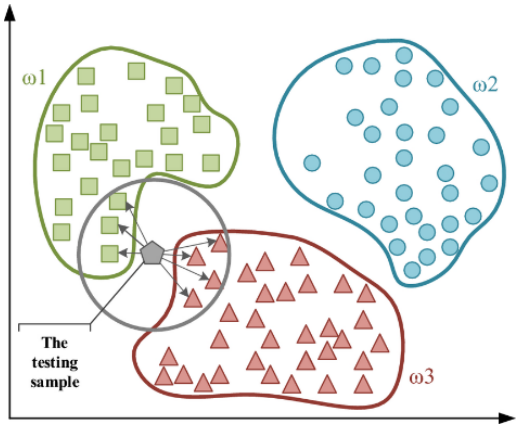


Hình ảnh sau khi áp dụng HOG

2.2 K-Nearest Neighbors

- KNN (K-Nearest Neighbors) là một trong những thuật toán học có giám sát đơn giản nhất được sử dụng nhiều trong khai phá dữ liệu và học máy. Ý tưởng của thuật toán này là nó không học một điều gì từ tập dữ liệu học (nên KNN được xếp vào loại lazy learning), mọi tính toán được thực hiện khi nó cần dự đoán nhãn của dữ liệu mới.

- Lớp (nhãn) của một đối tượng dữ liệu mới có thể dự đoán từ các lớp (nhãn) của k hàng xóm gần nó nhất.



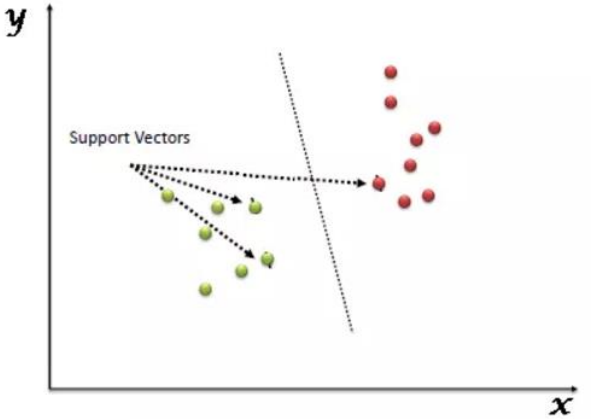
Minh họa cho thuật toán K-Nearest Neighbors

2.3 Support Vector Machine

- Support Vector Machine (SVM) là thuật toán giám sát được sử dụng nhiều trong học máy, có thể sử dụng cho cả việc phân loại hoặc hồi quy (chủ yếu là phân loại)

- Trong thuật toán này, chúng ta vẽ đồ thị dữ liệu là các điểm trong n chiều (n là số lượng đặc trưng) với giá trị của mỗi đặc trưng là một phần liên kết

- Sau đó chúng ta thực hiện tìm "hyper-plane" phân chia các lớp. "Hyper-plane" nó chỉ hiểu đơn giản là 1 đường thẳng có thể phân chia các lớp ra thành hai phần riêng biệt



Minh họa cho thuật toán Support Vector Machine

2.4 Áp dụng các phương pháp vào bài toán

2.4.1 Tiền xử lý dữ liệu và trích xuất đặc trưng

- Kiểm tra số lượng ảnh ở hai lớp là có ô tô hoặc không có ô tô để đảm bảo dữ liệu huấn luyện không bị mất cân bằng

```
n_car = len(Car_images)
n_non_car = len(Non_car_images)
img_shape = cv2.imread(Car_images[0]).shape

print('Number of car images: ', n_car)
print('Number of non-car images: ', n_non_car)
print('Images shape:',img_shape)


Number of car images:  8792
Number of non-car images:  8968
Images shape: (64, 64, 3)
```

- Thay đổi kích thước ảnh về kích thước 32x32x3 để giảm số lượng phần tử trong mảng, từ đó giảm khối lượng tính toán khi dùng ravel để chuyển hình ảnh dưới dạng mảng 3 chiều thành vector đặc trưng

```
def bin_spatial (img, size = (32,32)):
  img = cv2.resize(img, size).ravel()
  return img
```

Hàm chuyển đổi kích thước của ảnh

```
print(len(test.ravel()))
Executed at 2024.12.05 03:38:44 in 4ms

12288
```

Số lượng phần tử trong mảng khi chưa giảm kích thước

```
test = cv2.imread(Car_images[0])
resize = (32,32)
test = cv2.resize(test,resize)
print(len(test.ravel()))
Executed at 2024.12.05 00:41:48 in 2ms

3072
```

Số lượng phần tử trong mảng khi đã giảm kích thước

- Số lượng phần tử giảm xuống, khối lượng tính toán cũng sẽ giảm xuống đồng nghĩa với việc ta phải đánh đổi rằng sẽ có trường hợp đánh mất đi những thông tin quan trọng làm ảnh hưởng đến độ chính xác khi áp dụng các thuật toán máy học để huấn luyện mô hình

- Sử dụng hàm **color_histogram** để trích xuất đặc trưng màu sắc

```
def color_histogram (img, nbins=32, bins_range=(0,256)):
  channel1_hist = np.histogram(img[:,:,0], bins=nbins, range=bins_range)
  channel2_hist = np.histogram(img[:,:,1], bins=nbins, range=bins_range)
  channel3_hist = np.histogram(img[:,:,2], bins=nbins, range=bins_range)
  hist = np.concatenate((channel1_hist[0], channel2_hist[0], channel3_hist[0]))
  return hist
```

Hàm color_histogram

- Sử dụng hàm **get_hog_features** để áp dụng thuật toán HOG lên các hình. Giá trị **vis** ở đây là giá trị dùng để trực quan hóa một tấm ảnh tạm gọi là A và một tấm ảnh khi áp dụng HOG lên tấm ảnh A đó

```
def get_hog_features(img, orient, pix_per_cell, cell_per_block, vis=False, feature_vec=True):
  if vis == True:
    features, hog_image = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell),
                              cells_per_block=(cell_per_block, cell_per_block), transform_sqrt=True, visualize=vis,
                              feature_vector = feature_vec)
    return features, hog_image
  else:
    features = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell),
                   cells_per_block=(cell_per_block, cell_per_block), transform_sqrt=True,
                   visualize=vis, feature_vector=feature_vec)
    return features
```

Hàm get_hog_features

- Rút trích đặc trưng bằng hàm **extract_feature** với tham số đầu vào là mặc định

```
def extract_features(img,cspace = 'RGB', orient = 9, pix_per_cell = 8, cell_per_block = 2, hog_channel = 0,
                     spatial_size = (32,32), hist_bins = 32, hist_range = (0,256)):
  features = []
  for file in img:
    image = mpimg.imread(file)
    if cspace != 'RGB':
      if cspace == 'HSV':
        feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
      elif cspace == 'LUV':
        feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
      elif cspace == 'HLS':
        feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
      elif cspace == 'YUV':
        feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
      elif cspace == 'YCrCb':
        feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCR_CB)
    else: feature_image = np.copy(image)

    if hog_channel == 'ALL':
      hog_features = []
      for channel in range(feature_image.shape[2]):
        hog_features.append(get_hog_features(feature_image[:,:,channel], orient, pix_per_cell, cell_per_block,
                                             vis=False, feature_vec=True))
      hog_features = np.ravel(hog_features)
    else:
      hog_features = get_hog_features(feature_image[:,:,hog_channel], orient, pix_per_cell, cell_per_block,
                                      vis=False, feature_vec=True)
    spatial_features = bin_spatial(feature_image, size=spatial_size)
    hist_features = color_histogram(feature_image, nbins=hist_bins, bins_range=hist_range)
    features.append(np.concatenate((spatial_features, hist_features, hog_features)))
  return features
```

Hàm extract_features khởi tạo với các tham số mặc định

- Trích xuất vector đặc trưng cho từng ảnh:

```
spatial = 32
hist_bins = 32
colorspace = 'YCrCb'
orient = 9
pix_per_cell = 8
cell_per_block = 2
spatial_size= (32, 32)
heat_threshold= 4
hog_channel = "ALL"

car_features = extract_features(Car_images, cspace=colorspace, orient=orient,
                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                        hog_channel=hog_channel, spatial_size=(spatial, spatial),
                        hist_bins=hist_bins, hist_range=(0, 256))

non_car_features = extract_features(Non_car_images,cspace=colorspace,orient=orient,
                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                        hog_channel=hog_channel, spatial_size=(spatial, spatial),
                        hist_bins=hist_bins, hist_range=(0, 256))
```

Sử dụng tham số mặc định để trích xuất đặc trưng cho từng ảnh

2.4.2 KNN

- Đối với KNN, nhóm sử dụng phương pháp Grid Search để tìm ra tham số k tối ưu

```
knn = KNeighborsClassifier()
param_grid = {'n_neighbors' : list(range(1, 21))}
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train, y_train)
best_k = grid_search.best_params_['n_neighbors']
best_k
✓ 13m 18.1s

1
```

Tìm tham số k phù hợp cho mô hình KNN

```
best_knn = grid_search.best_estimator_
y_pred = best_knn.predict(X_test)



print(f'Model score with best k: {best_knn.score(X_test, y_test)}')
print('Confusion matrix: \n', confusion_matrix(y_test, y_pred))
print(f'\nAccuracy: {accuracy_score(y_test, y_pred)}')
print('Recall: ', recall_score(y_test, y_pred))
print('Precision: ', precision_score(y_test, y_pred))
```

Huấn luyện mô hình KNN sau khi tìm được tham số k tối ưu

- Ở bài toán này, số lượng k cho kết quả cao phần lớn là các số lẻ vì đây là bài toán phân lớp nhị phân. Vì các dữ liệu trong tập huấn luyện khi trích xuất vector đặc trưng có sự tách biệt và phân chia rõ ràng nên tham số k = 1 sẽ hiệu quả cho bài toán này. Tuy nhiên về tổng quát, số k nên là các số như 3, 5, 7 để tránh các trường hợp gây nhiễu. Bên cạnh đó, khoảng cách sử dụng khi tính toán cho mô hình sẽ là khoảng cách euclid

2.4.3 SVM

- Đối với SVM, nhóm sử dụng mô hình Linear SVM vì nó hoạt động nhanh hơn so với SVC và có thể xử lý tốt trong thời gian ngắn đối với dữ liệu lớn hoặc có số lượng đặc trưng lớn

```
svc = LinearSVC()

svc.fit(X_train, y_train)

y_pred = svc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average='weighted')
conf_matrix = confusion_matrix(y_test, y_pred)

print('Confusion matrix:')
print(conf_matrix)
print('\nAccuracy: ',accuracy)
print('Recall: ',recall)
print('Precision: ',precision)
```

Huấn luyện mô hình SVM

2.4.4 Sliding Window và Bounding Box

- Sliding window là kỹ thuật di chuyển một window với kích thước cố định qua từng vùng của ảnh. Từ những tài liệu đã được nghiên cứu bởi những người đi trước, nhóm sử dụng lại những hàm, chức năng để vẽ sliding window cho việc phát hiện đối tượng, heatmap để tăng độ tin cậy và bounding_box cuối cùng để nhận diện ô tô

```
test_image_sliding=test_images[6]
ystart_ystop_scale = [(405, 510, 1), (400, 600, 1.5), (415, 710, 2.5)]
bbox_detection_list, detections, box_vis_list = find_cars(test_image_sliding, svc, X_scaler, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins, ystart_ystop_

def draw_boxes(img, bboxes, thickness=2):
    imcopy = [np.copy(img),np.copy(img),np.copy(img)]
    color=[(255, 0, 0),(0, 255, 0),(0, 0, 255)]
    for i in range(len(bboxes)):
        for bbox in bboxes[i]:
            cv2.rectangle(img=imcopy[i], pt1=bbox[0], pt2=bbox[1],
                          color=color[i], thickness=thickness)
    return imcopy

windows_img = draw_boxes(test_image_sliding, box_vis_list)

f, ((ax1, ax2, ax3)) = plt.subplots(3, 1, figsize=(20,20))
ax1.imshow(windows_img[0])
title = "Scale: 0.9 "  + " Count: " + str(len(box_vis_list[0]))
ax1.set_title(title, fontsize=20)
ax1.axis("off")
ax2.imshow(windows_img[1])
title = "Scale: 1.4"  + " Count: " + str(len(box_vis_list[1]))
ax2.set_title(title, fontsize=20)
ax2.axis("off")
ax3.imshow(windows_img[2])
title = "Scale: 2.3"  + " Count: " + str(len(box_vis_list[2]))
ax3.set_title(title, fontsize=20)
ax3.axis("off")
```

Áp dụng kỹ thuật sliding window và vẽ bouding box dựa trên hàm có sẵn

- Quy trình áp dụng:

+ Xác định vùng quét, dựa vào 3 tham số là **ystart, ystop** và **scale**

+ Cắt và xử lý ảnh, chuyển đổi sang không gian màu YcrCb

+ Trích xuất đặc trưng và dự đoán bằng hai mô hình KNN và SVM

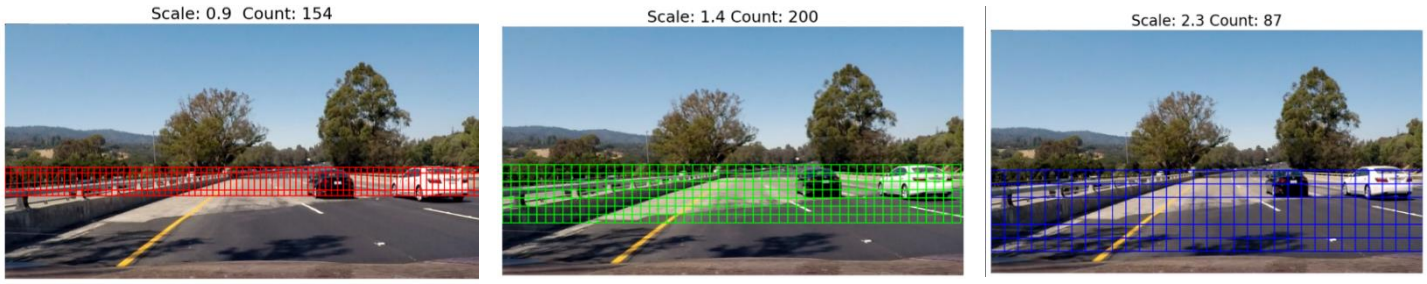
+ Vẽ bounding box đánh dấu những khu vực dự đoán là có xe

+ Xây dựng và hiển thị heatmap nhằm biểu diễn độ tin cậy của các khu vực dự đoán có xe

- Tham số ystart sẽ xác định tọa độ y bắt đầu của vùng mà ta muốn áp dụng sliding window trên ảnh, ystop sẽ xác định tọa độ y kết thúc, scale sẽ xác định tỷ lệ to hoặc nhỏ của cửa sổ dùng để quét. Ở đây nhóm sử dụng ba bộ tham số để nhận diện

```
test_image_sliding=test_images[6]
ystart_ystop_scale = [(405, 510, 1), (400, 600, 1.5), (415, 710, 2.5)]
```

Bộ ba tham số mà nhóm sử dụng cho Sliding Window



Vùng quét và kích thước của sliding window với 3 bộ số

- Kết quả khi áp dụng vào ảnh thực tế:

Áp dụng mô hình cùng kỹ thuật Sliding window và vẽ bounding box trên ảnh thực tế

3. THỰC NGHIỆM

3.1 Bộ dữ liệu

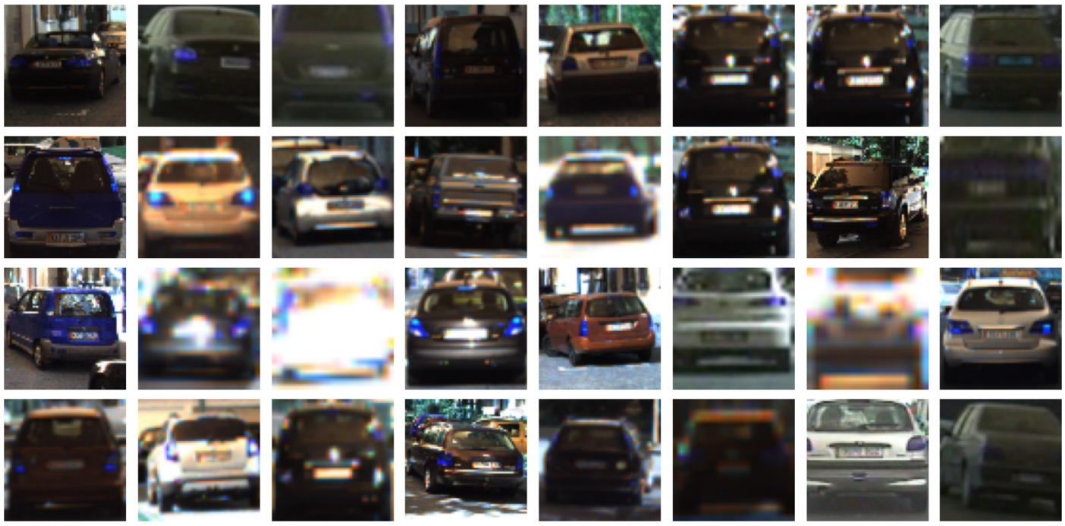
- Dữ liệu nhóm sử dụng được lấy từ github

- Đường link dẫn tới dữ liệu: https://github.com/harveenchadha/Udacity-CarND-Vehicle-Detection-and-Tracking/blob/master/vehicles.zip

- Bộ dữ liệu gồm có 8792 hình ảnh có xe ô tô và 8968 hình ảnh không có xe ô tô, ảnh có kích thước (64, 64, 3)

- Những hình ảnh được chụp từ nhiều góc khác nhau như nhìn xa, nhìn gần, nhìn từ bên trái, nhìn từ bên phải,…

- Các hình ảnh có ô tô đều đã được tiền xử lý bằng việc loại bỏ background không cần thiết



Một vài hình ảnh chứa xe ô tô trong bộ dữ liệu

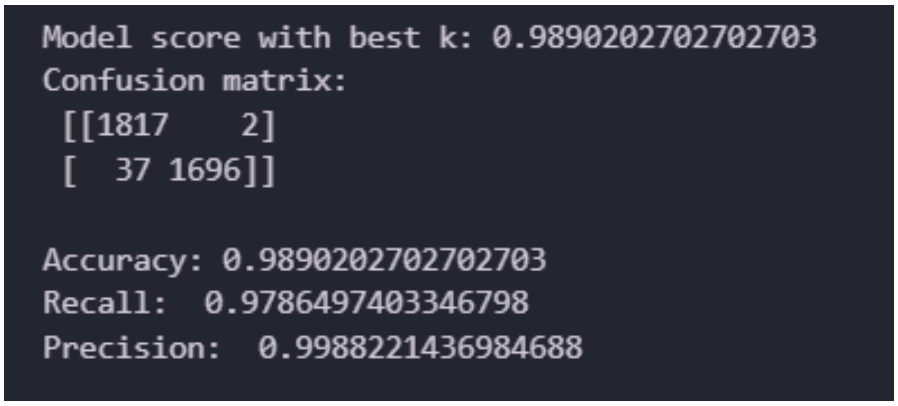
Một vài hình ảnh không chứa xe ô tô trong bộ dữ liệu

3.2 Kết quả và đánh giá trên tập kiểm tra

- Độ đo sử dụng trong bài toán: Accuracy, Recall và Precision score

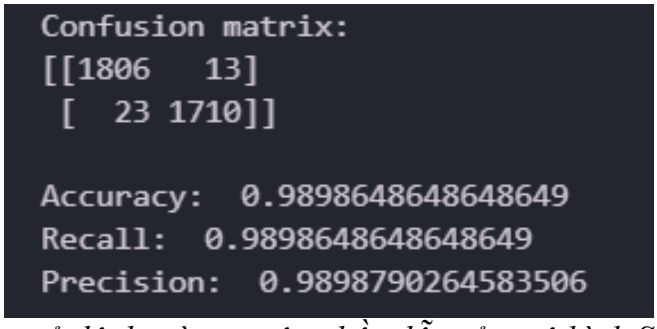
- Vì đây là bài toán phân lớp nhị phân nên đối với mỗi mô hình, nhóm chỉ sử dụng một confusion matrix để đánh giá và đưa ra kết quả cho các độ đo như accuracy, recall và precision score

- Mô hình KNN: Có 1817 trường hợp mô hình dự đoán đúng là không có xe ô tô trong ảnh và 1696 trường hợp mô hình dự đoán đúng là có xe ô tô trong ảnh. Có 2 trường hợp mô hình dự đoán là có xe ô tô nhưng thực tế là không có và 37 trường hợp mô hình dự đoán là không có xe ô tô nhưng thực tế là có ô tô



Kết quả các độ đo và ma trận nhầm lẫn của mô hình KNN

- Mô hình SVM: Có 1806 trường hợp mô hình dự đoán đúng là không có xe ô tô trong ảnh và 1710 trường hợp mô hình dự đoán đúng là có xe ô tô trong ảnh. Có 13 trường hợp mô hình dự đoán là có xe ô tô nhưng thực tế là không có và 23 trường hợp mô hình dự đoán là không có xe ô tô nhưng thực tế là có ô tô.



Kết quả độ đo và ma trận nhầm lẫn của mô hình SVM

- Tuy mang lại độ chính xác cao khi huấn luyện nhưng trong thực tế, để sử dụng hai mô hình đã nêu có thể nhận diện tốt thì cần nhiều yếu tố như kích thước vùng quét, tốc độ tính toán,…

4. KẾT LUẬN

- Đồ án đã sử dụng được phương pháp trích xuất đặc trưng HOG và các mô hình KNN, SVM để nhận diện xe ô tô từ ảnh số. Cả hai mô hình đều cho kết quả tốt, trong đó SVM cho kết quả có phần nhỉn hơn và đồng đều hơn giữa các độ đo so với KNN trên tập dữ liệu test

- Ưu điểm:

+ Dễ dàng triển khai và hiểu rõ hoạt động của hai mô hình KNN và SVM

+ Hiệu quả với dữ liệu nhỏ, cân bằng

+ Không cần tối ưu nhiều tham số khi huấn luyện mô hình ban đầu

+ Phù hợp với dữ liệu khả tuyến tính

- Hạn chế:

+ Thời gian tính toán sẽ lâu đối với ảnh có độ phân giải cao

+ Mô hình có thể sẽ gặp khó khăn trong điều kiện ánh sáng kém và ảnh nhiễu

+ Không phù hợp để áp dụng 2 mô hình vào hệ thống real-time vì tốc độ mô hình cho ra kết quả vẫn không phù hợp đối với tốc độ yêu cầu của các hệ thống hiện nay

- Hướng phát triển:

+ Tăng cường dữ liệu

+ Tối ưu hóa tham số với các kỹ thuật như Randomized Search, PCA,…

+ Thử nghiệm với các phương pháp học sâu hoặc các mô hình phân loại khác

DANH MỤC TÀI LIỆU THAM KHẢO

[1] https://github.com/vishalrk1/Vehicle-Classification-using-HOG-and-KNN?tab=readme-ov-file

[2] https://github.com/harveenchadha/Udacity-CarND-Vehicle-Detection-and-Tracking

[3] https://viblo.asia/p/tim-hieu-ve-phuong-phap-mo-ta-dac-trung-hog-histogram-of-oriented-gradients-V3m5WAwxZO7

[4] https://viblo.asia/p/tim-hieu-ve-hoghistogram-of-oriented-gradients-m68Z0wL6KkG

[5] https://viblo.asia/p/support-vector-machine-jvEla304Kkw

[6] https://viblo.asia/p/knn-k-nearest-neighbors-1-djeZ14ejKWz

[7] https://github.com/udacity/CarND-Vehicle-Detection/tree/master

[8] https://github.com/charleswongzx/Vehicle-Detection