ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

MÔN HỌC: MÁY HỌC

ĐỀ TÀI:

PHÂN LOẠI TÍN HIỆU ĐÈN GIAO THÔNG

Lớp: CS114.K21

Giảng viên hướng dẫn:

1. Lê Đình Duy
2. Phạm Nguyễn Trường An

Sinh viên thực hiện:

1. Nguyễn Hữu Khang	18520892
2. Hồ Đăng Tuệ	18521611
3. Phan Hoàng Nguyên	18521163

Thành phố Hồ Chí Minh, ngày 03 tháng 08 năm 2020

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN
MÔN HỌC: MÁY HỌC

ĐỀ TÀI:
PHÂN LOẠI TÍN HIỆU ĐÈN GIAO THÔNG

Lớp: CS114.K21

Giảng viên hướng dẫn:

1. Lê Đình Duy
2. Phạm Nguyễn Trường An

Sinh viên thực hiện:

1. Nguyễn Hữu Khang	18520892
2. Hồ Đăng Tuệ	18521611
3. Phan Hoàng Nguyên	18521163

Thành phố Hồ Chí Minh, ngày 03 tháng 08 năm 2020

MỤC LỤC

I. GIỚI THIỆU ĐỀ TÀI PHÂN LOẠI TÍNH HIỆU ĐÈN GIAO THÔNG:

1. Tổng quan về đề tài:

- Trong thời kì công nghiệp 4.0, Trí tuệ nhân tạo đang dần phát triển và len lỏi vào từng ngóc ngách của cuộc sống. Nó phục vụ cho rất nhiều mục đích của con người như việc tự động hóa, an ninh, bảo mật,... và giao thông thông minh chính là một trong những mảng mà Trí tuệ nhân tạo được áp dụng rất nhiều.
- Giao thông thông minh là một vấn đề lớn, bao gồm rất nhiều bài toán con. Trong đó bài toán nhận diện tính hiệu đèn giao thông là bài toán rất quan trọng. Nó phục vụ cho rất nhiều mục đích như xử phạt nguội các hành vi vi phạm giao thông, xe tự hành.… Bài toán nhận diện tính hiệu đèn giao thông bao gồm 2 bài toán con cần giải quyết:
    + Phát hiện đèn giao thông (Detect)
    + Phân loại tính hiệu đèn giao thông (Classify).
- Đầu ra của bài toán phát hiện đèn giao thông là hình ảnh đèn giao thông (không bao gồm các vật thể nhiễu từ bên ngoài) đây cũng chính là đầu vào của bài toán phân loại tính hiệu đèn giao thông.

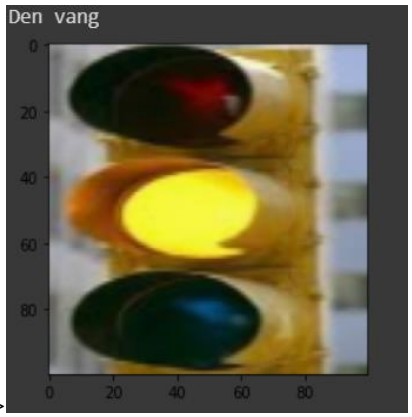
2. Mục tiêu đề tài:

- Tìm hiểu về phương pháp rút trích đặc trưng ảnh.
- Khảo sát một số thuật toán máy học.
- Thử giải quyết bài toán phân lớp đèn giao thông bằng cách lập trình truyền thống.
- Xây dựng một bộ dữ liệu dùng đề phân lớp đèn giao thông.
- Xây dựng một hàm nhận đầu vào là hình đường dẫn đến hình ảnh và cho đầu ra là kết quả phân lớp.

3. Mô tả bài toán:

- Bài toán phân loại tính hiệu đèn giao thông nhận đầu vào là một hình ảnh về đèn giao thông (chỉ chứa đèn giao thông) và cho đầu ra là kết quả dự đoán màu của đèn giao thông này.

-----Thông qua chương trình ----->



Input *Output*

II. THU THẬP DỮ LIỆU:

- Dữ liệu luôn là vấn đề lớn nhất của tất cả mô hình máy học, tất cả các mô hình đều bị ảnh hưởng rất nhiều bởi dữ liệu, dữ liệu không tốt sẽ dẫn đến kết quả đầu ra của các mô hình không tốt, vì vậy bước thu thập cũng như tiền xử lí dữ liệu gần như là bước quan trọng nhất trong toàn bộ quá trình giải quyết bài toán của ta. Trường hợp dữ liệu quá ít, hoặc bị nhiễu quá nhiều sẽ dẫn đến mô hình dễ bị học "dở" (Under-fitting) hoặc học "vẹt" (Over-fitting) nói cách khác mô hình sẽ bị thiếu tính tổng quát hóa (Generalization).
- Để có thể thu thập chính xác dữ liệu, ta cần phải hiểu rõ được bài toán mà ta đang giải quyết là gì. Ta cần phải phân loại tính hiệu của đèn giao thông, nên nhóm sẽ thu thập dữ liệu đó là ảnh (chỉ chứa đèn giao thông) để phục vụ cho mục đích phân loại.
- Cách thức thu thập: gồm 2 nguồn.

1. Internet:
    + Đầu tiên đó là một số clip camera hành trình từ youtube.
    + Thứ hai đó là các hình ảnh đèn giao thông từ google image.
    + Thứ ba đó là bộ data có sẵn (Có đính kèm ở phần tài liệu tham khảo).
2. Thực tế:
    + Đèn giao thông được nhóm tự chụp trong Sài Gòn.

Ảnh từ các nguồn trên đều là ảnh toàn cảnh, chứa rất nhiều đối tượng khác ngoài đèn giao thông.

Ví dụ:

Ảnh từ google image

Ta chỉ quan tâm đến đèn giao thông, nên sẽ có 2 cách để lấy được dữ liệu đèn trong ảnh toàn cảnh này:

+ Cách 1: Giải quyết bài toán Detection (Bài toán sẽ trả về vị trí của đèn trong ảnh, ta chỉ cần dùng code để crop vị trí này ra). Cách này khá khó, và dữ liệu ít nên có thể dẫn đến việc detect bị sai, không khả thi trong trường hợp này.

+ Cách 2: Crop tất cả đèn bằng tay, cách này tuy tốn sức nhưng dữ liệu sẽ chính xác.

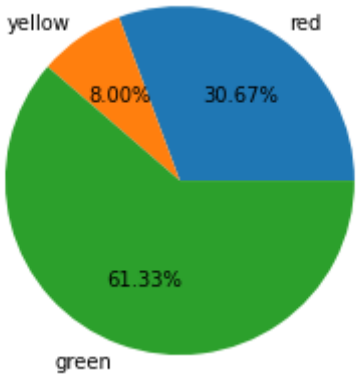
- Bộ data của nhóm là ảnh các hộp đèn giao thông, bài toán ở đây chỉ phân loại được 3 màu đèn là đỏ, vàng, xanh. Chưa thể phân loại được cả 3 đèn cùng sáng hoặc cùng tối.
- Các tấm ảnh được crop tay từ bộ dữ liệu giao thông, các video hành trình từ youtube, ảnh, video của nhóm tự chụp ở các quận của HCM …. khá mờ vì được chụp/quay khi di chuyển và qua camera có chất lượng chưa đủ tốt. Bộ data gồm có 1112 tấm trong đó có *1009 ảnh được thu thập bằng cách 1* và *103 ảnh được thu thập bằng cách 2.*

+ Số lượng ảnh đèn đỏ: 341 ảnh

+ Số lượng ảnh đèn vàng: 89 ảnh

+ Số lượng ảnh đèn xanh: 682 ảnh

Đa số các ảnh là đèn đỏ và đèn xanh, đèn vàng chiếm số lượng khá ít. Đèn vàng ít vì trong thực tế nó chỉ xuất hiện khoảng 2-3s.



Ảnh phân bố dữ liệu


.---Sau khi cắt ảnh--->


- Dữ liệu bị ảnh hưởng rất nhiều bởi ánh sáng. Ánh sáng lóa, quá mạnh làm cho màu đèn bị ảnh hưởng và thay đổi, môi trường buổi tối cũng là một thử thách:

Ảnh đèn đỏ nhưng có thể bị lẫn với đèn vàng.

Ảnh đèn xanh trong điều kiện buổi tối, bị chá thành màu xanh dương.

- Dữ liệu được chia thành 2 tập:

    + Tập train gồm 1009 ảnh được thu thập theo cách 1 (90% tổng dữ liệu).

    + Tập test gồm 103 ảnh được thu thập theo cách 2 (10% tổng dữ liệu).

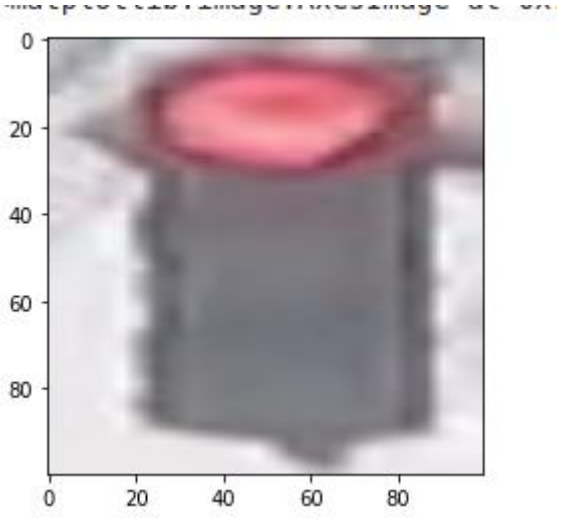
III. TIỀN XỬ LÝ DỮ LIỆU VÀ ĐỀ XUẤT PHƯƠNG PHÁP RÚT TRÍCH ĐẶC TRƯNG:

1. Tiền xử lý dữ liệu:

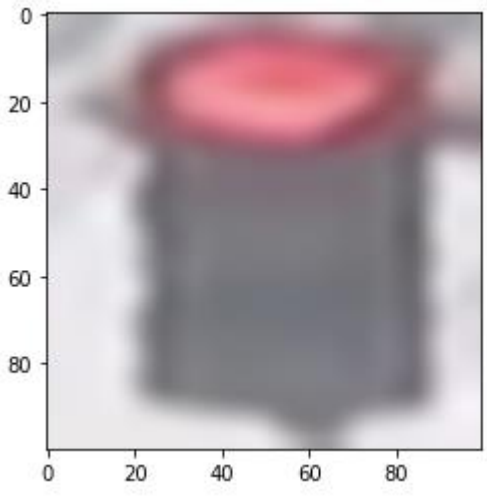
- Ảnh được sử dụng làm dữ liệu có kích thước đa dạng và khác nhau, nên nhóm sẽ đưa chúng về một kích cỡ duy nhất là 100x100 pixel, hàm resize ảnh được cung cấp bởi cv2.
- Nhóm sử dụng cách làm mờ ảnh (Blur) để khử nhiễu, cũng như làm các chi tiết được mượt mà hơn trước khi vào bước rút trích đặc trưng. Phương pháp làm mờ được sử dụng là Bộ lọc mờ trung bình (Box filter) được cung cấp sẵn bởi thư viện cv2.

```
# Làm mờ ảnh
rgb_image = cv2.blur(rgb_image, ksize = (7,7))
```

Bộ lọc mờ trung bình trong cv2



Ảnh trước khi qua bộ lọc mờ



Ảnh trước sau qua bộ lọc mờ

- Nhóm có sử dụng thêm phương pháp cân bằng sáng phối hợp với bộ lọc mờ trong bước tiền xử lí dữ liệu nhưng kết quả không tốt bằng việc không sử dụng.

2. Đề xuất phương pháp rút trích đặc trưng:

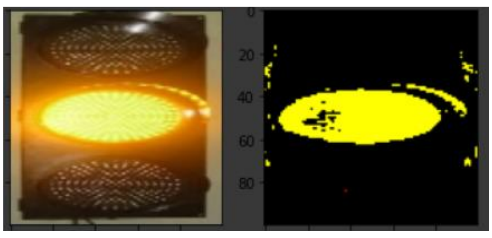
- Vì đèn giao thông trong tập dữ liệu của nhóm chỉ có một trạng thái (trong 3 trạng thái đỏ, xanh, vàng) cho nên nhóm muốn rút trích đặc trưng liên quan đến màu sắc của đèn.
- Thông qua khảo sát nhóm tìm được một số phương pháp như Color Histogram, Color Image Segmentation... Nhóm quyết định chọn phương pháp Color Image Segmentation để dùng trong bước rút trích đặc trưng.
- Ý tưởng của phương pháp này đó là ta sẽ tạo ra các ngưỡng màu (Color threshold) nếu các pixel của ảnh nằm trong các ngưỡng màu này thì ta sẽ giữ lại pixel đó.
- Sẽ có 2 ngưỡng là ngưỡng chặn trên (màu sáng) và ngưỡng chặn dưới (màu tối).

Ví dụ:

Màu đỏ tối

Màu đỏ sáng

- Hệ màu được sử dụng để rút trích đặc trưng là HSV. Vì hệ màu HSV bao gồm 3 kênh:
    + H: Hue - là giá trị biểu diễn màu sắc với 0 là đỏ, 120 là xanh lá, và 255 là xanh dương.
    + S: Saturation là giá trị chỉ độ bảo hòa của ảnh.
    + V: Value chỉ độ sáng của ảnh.
- Hệ màu này được sử dụng rất nhiều trong việc phân loại ảnh, vì nó có thể biểu diễn được độ sáng của ảnh, khác với hệ màu RGB chỉ biểu diễn được màu thông qua 3 kênh màu phối lại với nhau.



Kết quả sau khi rút trích đặc trưng

```python
#tham khảo ý tưởng: https://realpython.com/python-opencv-color-s
def feature_rgb(rgb_image):
    # Làm mờ ảnh
    rgb_image = cv2.blur(rgb_image, ksize = (7,7))
    #cân bằng sáng
    # hsv = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2HSV)
    # hsv[:,:,0] = cv2.equalizeHist(hsv[:,:,0])
    hsv = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2HSV)


    image = np.copy(hsv)
    # Ta sẽ set các ngưỡng màu cho green, red, và yellow
    #green
    lower_green = np.array([50, 25, 25])
    upper_green = np.array([86, 255, 255])
    #yellow
    lower_yellow = np.array([20,100,120])
    upper_yellow = np.array([60,255,255])
    #red
    lower_red = np.array([0, 120, 70])
    upper_red = np.array([10, 255, 255])

    # tạo ra các mask màu cho ảnh, ta sẽ lấy tất cả các pixel nằ
    mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
    mask_red = cv2.inRange(hsv, lower_red, upper_red)
    mask_green = cv2.inRange(hsv, lower_green, upper_green)

    # đưa màu của ta về thành 1 giá trị duy nhất, vì các pixel s
    # toàn bộ các pixel trên về thành một màu duy nhất (tương đư
    masked_image = np.zeros((image.shape[1],image.shape[0],3), n
    # masked_image chứa 3 channel tương đương 1 ảnh RGB, qua mỗi
    # tương ứng với màu của  channel đó
    masked_image[mask_red != 0] = [255, 0, 0]
    masked_image[mask_green != 0] = [0, 255, 0]
    masked_image[mask_yellow != 0] = [255, 255, 0]
    result = masked_image.flatten()

    return result
```

Hàm thực hiện rút trích đặc trưng cho một ảnh

IV. THỰC NGHIỆM:

1. Cách dùng máy học:

- Bộ dữ liệu được chia 90% train, và 10% test.

- Nhóm thực nghiệm trên 3 thuật toán:

    + Naive Bayes

    + K-Nearest Neighbors

+ SVM (Support vector machine)

- Hyperparamters của 3 thuật toán đều để mặc định và không thay đổi gì.

- Độ đo được sử dụng để đánh giá mô hình đó là f1 - score, và accuracy. F1 - score và accuracy càng cao thì mô hình càng tốt.

- Sau quá trình training thì nhận được kết quả predict trên tập train và tập test như sau:

+ Naive Bayes:

```
cfm Naive bayes on test set
[[23  3 20]
 [ 4  6  0]
 [ 6  0 47]]
cfm Naive bayes on train set
[[152  11 132]
 [  0  56  23]
 [  0   1 628]]
report Naive bayes on test set
              precision    recall  f1-score   support

           0       0.70      0.50      0.58        46
           1       0.67      0.60      0.63        10
           2       0.70      0.89      0.78        53

    accuracy                           0.70       109
   macro avg       0.69      0.66      0.67       109
weighted avg       0.70      0.70      0.68       109

report Naive bayes on train set
              precision    recall  f1-score   support

           0       1.00      0.52      0.68       295
           1       0.82      0.71      0.76        79
           2       0.80      1.00      0.89       629

    accuracy                           0.83      1003
   macro avg       0.88      0.74      0.78      1003
weighted avg       0.86      0.83      0.82      1003
```

Kết quả thực nghiệm với thuật toán Naive Bayes.

+ KNN:

```
cfm KNN on test set
[[22  4 20]
 [ 2  8  0]
 [ 4  0 49]]
cfm KNN on train set
[[150   1 144]
 [  2  52  25]
 [  0   0 629]]
report KNN on test set
              precision    recall  f1-score   support

           0       0.79      0.48      0.59        46
           1       0.67      0.80      0.73        10
           2       0.71      0.92      0.80        53

    accuracy                           0.72       109
   macro avg       0.72      0.73      0.71       109
weighted avg       0.74      0.72      0.71       109

report KNN on train set
              precision    recall  f1-score   support

           0       0.99      0.51      0.67       295
           1       0.98      0.66      0.79        79
           2       0.79      1.00      0.88       629

    accuracy                           0.83      1003
   macro avg       0.92      0.72      0.78      1003
weighted avg       0.86      0.83      0.81      1003
```

Kết quả thực nghiệm với thuật toán KNN.

+ SVM:

```
cfm svm on test set
[[20  0 26]
 [ 9  1  0]
 [46  0  7]]
cfm svm on train set
[[132   0 163]
 [  0  45  34]
 [  0   0 629]]
report svm on test set
              precision    recall  f1-score   support

           0       0.27      0.43      0.33        46
           1       1.00      0.10      0.18        10
           2       0.21      0.13      0.16        53

    accuracy                           0.26       109
   macro avg       0.49      0.22      0.23       109
weighted avg       0.31      0.26      0.24       109

report svm on train set
              precision    recall  f1-score   support

           0       1.00      0.45      0.62       295
           1       1.00      0.57      0.73        79
           2       0.76      1.00      0.86       629

    accuracy                           0.80      1003
   macro avg       0.92      0.67      0.74      1003
weighted avg       0.85      0.80      0.78      1003
```

Kết quả thực nghiệm với thuật toán SVM.

Nhận xét:

Kết quả thực nghiệm cho thấy accuracy và f1 - score của 2 thuật toán KNN và SVM là chênh lệch khoảng 10% nhưng không nhiều như model SVM (khoảng 60% giữa kết quả trên tập test và tập train). Có thể model SVM đã bị overfitting. Việc chênh lệch dữ liệu quá nhiều cũng có thể dẫn đến sự dư đoán thiếu chính xác của model, chỉ cần dự đoán đa số ảnh là màu xanh cũng có thể đạt được kết quả khá cao (đối với tập train).

Tuning Hyperparameters

- Nhóm thực hiện việc tuning hyperparameters cho thuật toán KNN (vì đây là thuật toán có Accuracy và f1 - score cao nhất), các parameters là:

+ n_neighbors: số lượng các điểm "hàng xóm" mà thuật toán sẽ xét khoảng cách, để dự đoán điểm dữ liệu mới.

+ weigths: đó là sự ảnh hưởng của các điểm dữ liệu "hàng xóm" đến điểm dữ liệu dự đoán nếu là "uniform" thì tất cả các điểm dự đoán xung quanh điểm dữ liệu mới đều bằng nhau, nếu là "distance" thì khoảng cách giữa các điểm dữ liệu "hàng xóm" đến điểm dữ liệu dự đoán sẽ tác động lên dự đoán của model, điểm nào càng gần điểm dữ liệu được dự đoán sẽ càng ảnh hưởng nhiều, và ngược lại nếu xa.

+ p: là cách tính khoảng cách, nếu là 1 thì cách tính khoảng cách sẽ là manhattan - distance, còn nếu là 2 sẽ là euculide - distance.

- Sử dụng GridSearch được cung cấp bởi Scikit Learn để thực hiện việc này.

```
param_grid = {'n_neighbors': [5, 7, 12, 20], 'weights': ['uniform', 'distance'], 'p' : [1, 2]}
grid = GridSearchCV(KNeighborsClassifier(),param_grid,refit=True,verbose=2, return_train_score=True)
grid.fit(features_training, labels_training)
```

- Bộ parameter tốt nhất:

```
grid.best_estimator_
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=1,
                     weights='uniform')
```

Kết quả:

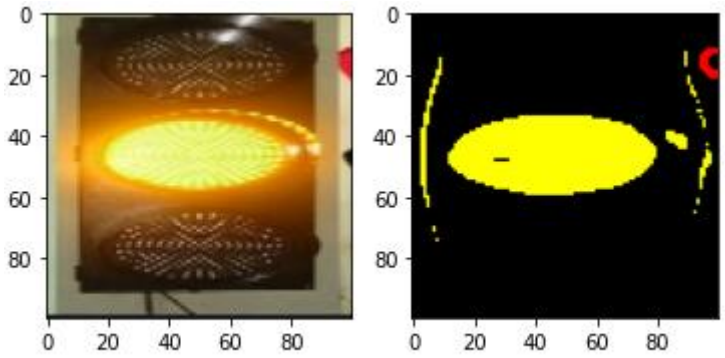
```
cfm test set
[[22  4 20]
 [ 2  8  0]
 [ 4  0 49]]
cfm train set
[[150   1 144]
 [  2  52  25]
 [  0   0 629]]
report test set
              precision    recall  f1-score   support

           0       0.79      0.48      0.59        46
           1       0.67      0.80      0.73        10
           2       0.71      0.92      0.80        53

    accuracy                           0.72       109
   macro avg       0.72      0.73      0.71       109
weighted avg       0.74      0.72      0.71       109

report train set
              precision    recall  f1-score   support

           0       0.99      0.51      0.67       295
           1       0.98      0.66      0.79        79
           2       0.79      1.00      0.88       629

    accuracy                           0.83      1003
   macro avg       0.92      0.72      0.78      1003
weighted avg       0.86      0.83      0.81      1003
```

Nhận xét:

Kết quả sau khi tuning không thay đổi nhiều so với trước khi tuning.

2. Cách dùng lập trình thuần túy:

- ***Ý tưởng:*** Sau khi rút trích đặc trưng ta sẽ nhận được một ảnh RGB chỉ chứa 3 giá trị màu sắc là đỏ (255,0,0), vàng (255, 255, 0) và xanh lá (0, 255, 0). Vì đèn chỉ ở một trạng thái (xanh, đỏ, vàng) nên ta chỉ cần đếm số pixel tương ứng với từng giá trị màu, giá trị màu nào có nhiều pixel nhất thì đèn sẽ có màu đó.



Một ảnh sau khi rút trích đặc trưng

Ví dụ đối với ảnh này, ta dễ dàng thấy được số lượng pixel màu vàng là nhiều nhất, nên ta có thể kết luận ảnh này là đèn vàng.

```
def predict_without_ML (img_bgr):
  img = feature_rgb(img_bgr)
  count_R = 0
  count_G = 0
  count_Y = 0
  R = img[:,:,0]
  G = img[:,:,1]

  for i in range (100):
    for j in range (100):
      if (R[i][j] == G[i][j] and R[i][j]== 255):
        count_Y += 1
      if (R[i][j] != G[i][j] and R[i][j] == 255):
        count_R += 1
      if (R[i][j] != G[i][j] and G[i][j] == 255):
        count_G += 1

  if ((count_R - count_Y) > 0 and (count_R - count_G) > 0):
    return 0
  if ((count_Y - count_G) > 0 and (count_Y - count_R) > 0):
    return 1
  if ((count_G - count_R) > 0 and (count_G - count_Y) > 0):
    return 2
  return 0
```

Hàm dự đoán ảnh bằng lập trình thuần túy.

- Tạo ra 2 biến đếm ở 2 channel Red và Green, nếu là màu vàng thì ở pixel thứ (i,j) của channel Red sẽ bằng với pixel (i,j) ở channel G và bằng 255, 2 màu Red và Green sẽ dễ hơn, nếu ở vị trí (i.i) mà channel Red có giá trị = 255 và không có channel nào có pixel ở vị trí (i,j) = 255 nữa thì đó sẽ là pixel màu đỏ, tương tự cho màu xanh. Sau khi đếm xong 3 màu, ta chỉ cần so sánh số lượng.

- **Kết quả**:

```
cfm test set
[[45  0  1]
 [ 2  8  0]
 [ 2  0 51]]
cfm train set
[[136   1 158]
 [ 50  25   4]
 [345  44 240]]
report test set
              precision    recall  f1-score   support

           0       0.92      0.98      0.95        46
           1       1.00      0.80      0.89        10
           2       0.98      0.96      0.97        53

    accuracy                           0.95       109
   macro avg       0.97      0.91      0.94       109
weighted avg       0.96      0.95      0.95       109

report train set
              precision    recall  f1-score   support

           0       0.26      0.46      0.33       295
           1       0.36      0.32      0.34        79
           2       0.60      0.38      0.47       629

    accuracy                           0.40      1003
   macro avg       0.40      0.39      0.38      1003
weighted avg       0.48      0.40      0.42      1003
```

- **Nhận xét**: ta thấy, ở tập train có kết quả thấp hơn tập test, có thể do tập train bị nhiễu quá nhiều, cũng như số lượng ảnh ở tập train khá nhiều.

3. Sử dụng máy học, nhưng bỏ qua bước rút trích đặc trưng:

- Duỗi thẳng tất cả các ảnh và cho máy học, không sử dụng rút trích đặc trưng

- Kết quả:

+ Naive Bayes:

```
cfm Naive bayes on test set
[[39  5  2]
 [ 3  6  1]
 [10  2 41]]
cfm Naive bayes on train set
[[231   2  62]
 [  2  68   9]
 [ 64  39 526]]
report Naive bayes on test set
              precision    recall  f1-score   support

           0       0.75      0.85      0.80        46
           1       0.46      0.60      0.52        10
           2       0.93      0.77      0.85        53

    accuracy                           0.79       109
   macro avg       0.71      0.74      0.72       109
weighted avg       0.81      0.79      0.79       109

report Naive bayes on train set
              precision    recall  f1-score   support

           0       0.78      0.78      0.78       295
           1       0.62      0.86      0.72        79
           2       0.88      0.84      0.86       629

    accuracy                           0.82      1003
   macro avg       0.76      0.83      0.79      1003
weighted avg       0.83      0.82      0.82      1003
```

+ KNN:

```
cfm KNN on test set
[[36  1  9]
 [ 3  6  1]
 [ 1  0 52]]
cfm KNN on train set
[[277   0  18]
 [  5  64  10]
 [  0   1 628]]
report KNN on test set
              precision    recall  f1-score   support

           0       0.90      0.78      0.84        46
           1       0.86      0.60      0.71        10
           2       0.84      0.98      0.90        53

    accuracy                           0.86       109
   macro avg       0.87      0.79      0.82       109
weighted avg       0.87      0.86      0.86       109

report KNN on train set
              precision    recall  f1-score   support

           0       0.98      0.94      0.96       295
           1       0.98      0.81      0.89        79
           2       0.96      1.00      0.98       629

    accuracy                           0.97      1003
   macro avg       0.97      0.92      0.94      1003
weighted avg       0.97      0.97      0.97      1003
```

+ SVM:

```
cfm svm on test set
[[45  0  1]
 [ 2  7  1]
 [ 0  0 53]]
cfm svm on train set
[[292   0   3]
 [  0  79   0]
 [  0   0 629]]
report svm on test set
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        46
           1       1.00      0.70      0.82        10
           2       0.96      1.00      0.98        53

    accuracy                           0.96       109
   macro avg       0.97      0.89      0.92       109
weighted avg       0.96      0.96      0.96       109

report svm on train set
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       295
           1       1.00      1.00      1.00        79
           2       1.00      1.00      1.00       629

    accuracy                           1.00      1003
   macro avg       1.00      1.00      1.00      1003
weighted avg       1.00      1.00      1.00      1003
```

Nhận xét: Kết quả thực nghiệm của 3 model trên tập train và test khá cao, có thể là do ảnh không bị các vật thể bên ngoài nhiễu (đã được crop).

V. DEMO:

- Clip demo được đính kèm trong báo cáo

- Hàm demo chương trình bằng máy học (model KNN): Nhận đầu vào là đường dẫn tới model, và đường dẫn đến hình ảnh (cả 2 đều được để ở Drive cá nhân)

```
def predict_demo (img_path, model_path):
  model = joblib.load(model_path)
  ori_img = cv2.imread(img_path) # Đọc ảnh cần dự đoán
  H = ori_img.shape[0] # Lấy chiều cao của ảnh
  W = ori_img.shape[1] # Lấy chiều rộng của ảnh
  img = cv2.resize(ori_img, (100,100)) #resize ảnh thành size chung của model
  _img = ori_img # copy một ảnh ori_img để hiển thị kết quả dự đoán
  _img = cv2.cvtColor(_img, cv2.COLOR_BGR2RGB)
  i = feature_rgb(img) #rút trích đặc trưng ảnh vừa đưa vào
  i = i.flatten()
  j = []
  j.append (i)
  tmp = model.predict (j)
  ori_img = cv2.cvtColor (ori_img, cv2.COLOR_BGR2RGB)
  if (tmp == 0):
    tmp = 'Den do'
    color = (255,0,0)
  if (tmp == 1):
    tmp = 'Den vang'
    color = (255,255,0)
  if (tmp == 2):
    tmp = 'Den xanh'
    color = (0,255,0)
  cv2.putText(_img, tmp, (int (H/20), int (W/2)),cv2.FONT_HERSHEY_SIMPLEX,0.4, color, 2)
  # Hiển thị kết quả
  print (tmp)
  plt.subplot (1,2,1)
  plt.imshow (ori_img)
  plt.title ('Anh goc')
  plt.subplot(1,2,2)
  plt.imshow (_img)
  plt.title ('Anh duoc du doan')
```

```
def Predict_image (img_path):
  ori_img = cv2.imread('/content/yellow.PNG') #đường dẫn chưa link đến ảnh cần dự đoán
  H = ori_img.shape[0]
  W = ori_img.shape[1]
  im = ori_img
  img = cv2.resize(ori_img, (100,100)) #resize ảnh về 1 size duy nhất
  _img = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
  tmp = predict_without_ML (img)
  ori_img = cv2.cvtColor (ori_img, cv2.COLOR_BGR2RGB)
  if (tmp == 0):
    tmp = 'Den do'
    color = (255,0,0)
  if (tmp == 1):
    tmp = 'Den vang'
    color = (255,255,0)
  if (tmp == 2):
    tmp = 'Den xanh'
    color = (0,255,0)
  cv2.putText(_img, tmp, (int (H/20), int (W/2)),cv2.FONT_HERSHEY_SIMPLEX,0.6, color, 2)
  print (tmp)
  plt.subplot (1,2,1)
  plt.imshow (ori_img)
  plt.title ('Anh goc')
  plt.subplot(1,2,2)
  plt.imshow (_img)
  plt.title ('Anh duoc du doan')
```

Hàm demo chương trình bằng cách lập trình thuần túy: nhận đầu vào là đường dẫn tới một ảnh được lưu trong drive cá nhân.

VI. KẾT LUẬN:

- Bài toán phân loại tính hiệu đèn giao thông là một bài toán tuy nhỏ nhưng cực kì quan trọng trong giao thông thông minh.
- Thông qua qua trình tìm hiểu cũng như thực hiện Đồ án, chúng em đã học thêm được nhiều kiến thức mới, tìm hiểu được thêm một số phương pháp rút trích đặc trưng màu sắc, trau dồi thêm kĩ năng tìm kiếm thông tin qua Internet, và đã viết được một chương trình nho nhỏ để dự đoán tính hiệu đèn giao thông (tuy độ chính xác không phải thật sự cao)
- Trong thời gian thực hiện báo cáo không thể tránh khỏi những thiếu sót, rất mong thầy bỏ qua cho nhóm.

VII. KHÓ KHĂN VÀ HƯỚNG PHÁT TRIỂN:

- Khó khăn:

+ Dữ liệu bởi nhiễu bởi ánh sáng khá nhiều.

+ Hạn chế về các hướng tiền xử lí dữ liệu (Dữ liệu chỉ được làm mờ để khử bớt nhiễu), do nhóm khảo sát các phương pháp này chưa tốt.

+ Vấn đề lệch dữ liệu ảnh hưởng rất lớn trong quá trình máy học (dữ liệu chủ yếu là đèn xanh).

+ Chưa có kinh nghiệm trong khâu thu thập và gáng nhãn dữ liệu dẫn đến có nhiều dữ liệu nhiễu.

- Hướng phát triển:

+ Tìm hiểu thêm phương pháp xét vị trí của các màu trong đèn (vì màu đèn giao thông thường cố định theo thứ tự đỏ - vàng - xanh từ trên xuống dưới, không áp dụng được với các đèn nằm ngang).

+ Sử dụng Deep learning để cải thiện độ chính xác (Các mạng phân loại như CNN, VGG-16).

+ Phát triển bài toán rộng ra, không chỉ phân loại màu đèn nữa, mà sẽ vừa phát hiện vị trí đèn giao thông, vừa nhận diện màu của đèn giao thông đó, sử dụng các mô hình detect hiện đại như (YOLOv3, Detectron, ...).

VIII. TÀI LIỆU THAM KHẢO VÀ NGUỒN DỮ LIỆU:

Các nguồn tham khảo dữ liệu (Không bao gồm google image):

1. **https://github.com/vatsl/TrafficLight_Detection-TensorFlowAPI#get-the-dataset**
2. **https://youtu.be/Y3HSOewZP7w**
3. **https://youtu.be/iGgogjlgE-M**
4. **https://youtu.be/EVxrZnE-uU4**
5. **https://youtu.be/abchmB6DcuE**
6. **https://youtu.be/upNHTJrEfXg**
7. **https://youtu.be/YJyexTLKo2s**
8. **https://youtu.be/QGEXZJITF58**
9. **https://youtu.be/RLjWyQmVKXg**
10. **https://youtu.be/qbR2qcbdKTA**
11. **https://youtu.be/C9J-z6JXGI0**
12. **https://youtu.be/box7pddBFdY**
13. **https://youtu.be/8USre5mc3iM**
14. **https://youtu.be/2vqWRCr5SeY**
15. **https://youtu.be/gQ6nN1ruAQU**
16. **https://youtu.be/NrgVw9UIa4I**
17. **https://youtu.be/aJ08b6R4vW0**
18. **https://youtu.be/V9MHnjaGBpA**
19. **https://youtu.be/PUPmSxGkhws**
20. **https://youtu.be/Pbnb4D_JyFI**

Tài liệu tham khảo:

1. **https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes**
2. **https://scikit-learn.org/stable/modules/svm.html#classification**
3. **https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification**
4. **https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html**

5. **https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html**
6. **https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html**
7. **https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html**
8. **https://realpython.com/python-opencv-color-spaces/**