TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO MÔN HỌC : XỬ LÝ ẢNH

ĐỀ TÀI: Tìm hiểu các đặc trưng toàn cục và ứng dụng vào bài toán phân loại ảnh

Giảng viên hướng dẫn: TS. Hoàng Văn Hiệp

Nhóm sinh viên : Nguyễn Ngọc Bình – 20160370

Nguyễn Bá Cầu - 20160399

Mục lục

Giới thiệu

Xử lý ảnh là một lĩnh vực mang tính khoa học và công nghệ. Nó là một ngành khoa học mới mẻ so với các ngành khoa học khác nhưng tốc độ phát triển rất nhanh, kích thích các trung tâm nghiên cứu, ứng dụng, đặc biệt là các phần cứng chuyên dụng cho nó.

Phân loại ảnh là một bài toán điển hình của lĩnh vực xử lý ảnh, với mục tiêu giúp cho máy tính có thể tự động nhận biết được nội dung chính trong bức ảnh, video, … giống như con người thông qua thị giác. Một trong các phương pháp để phân loại ảnh la trích chọn các đặc trưng toàn cục có trong ảnh, biểu diễn thành một vector đặc trưng và phân loại chúng bằng các thuật toán học máy cổ điển, điển hình là thuật toán Support Vector Machine (SVM). Một số các đặc trưng toàn cục điển hình về màu sắc gồm có histogram, color moment, về kết cấu ảnh có các đặc trưng như Local Binary Parttern (LBP), Histogram of Oriented Gradient (HOG)

Trong quá trình tìm hiểu và thực hiện, em có thể sẽ gặp phải những thiếu sót, em mong nhận được phản hồi và góp ý từ thầy để bài làm được trở nên hoàn thiện và đầy đủ hơn

1. Giới thiệu bài toán

1.1 Giới thiệu

Trong đề tài này, em sẽ trình bày các đặc trưng toàn cục của ảnh bao gồm:

Về màu sắc: đặc trưng về histogram, color moment

Về kết cấu ảnh: đặc trưng về LBP và HOG

Qua đó áp dụng các phương pháp này vào bài toán phân loại ảnh phương tiện giao thông đường bộ để so sánh và đánh giá các kết quả

1.2 Loading và visualize dữ liệu như sau

Code python như sau:

```
class_name = ['bicycle', 'bus', 'car', 'motorbike']
data=[]
label=[]
for i, l in enumerate(class_name):
    data_path = "vehicle_data/" + l + "/*.*"
    file = glob.glob(data_path)
    for j, f in enumerate(file):
        im = cv2.imread(f)
        data.append(im)
        label.append(i)
    print("number of " + l + " examples is : " + str(j+1))
print("number of examples is: ", len(data))
data = np.array(data)
label = np.array(label)
data, label = shuffle(data, label)
```

Output:

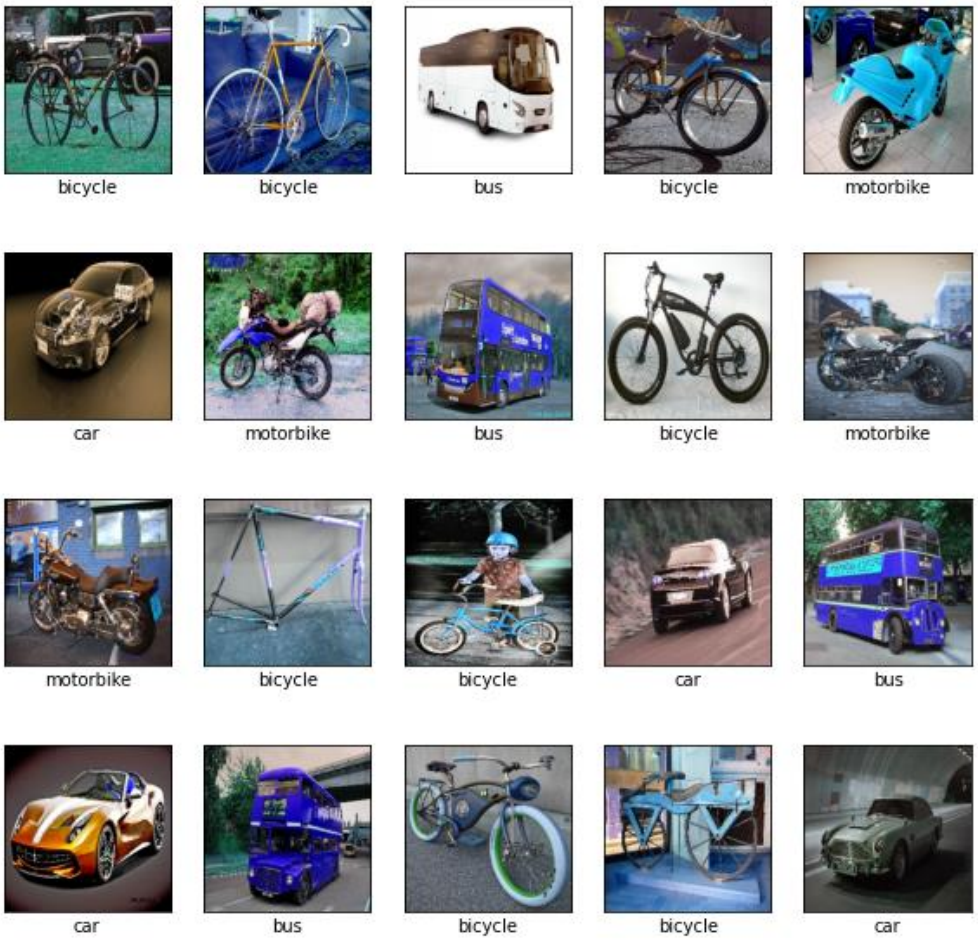
```
number of bicycle examples is : 222
number of bus examples is : 167
number of car examples is : 253
number of motorbike examples is : 255
number of examples is:  897
```

Tập dữ liệu gồm có 4 class là bicycle, bus, car, motorbike với số lượng dữ liệu của mỗi class lần lượt là 222, 167, 253, 255, và tổng số ảnh là 897, ảnh đã được chuẩn hóa về lích thước 256x256

Visualize data, show 20 ảnh đầu tiên trong bộ dữ liệu

```
plt.figure(figsize = (10, 10))
for i in range(20):
   plt.subplot(4, 5, i+1)
   plt.xticks([])
   plt.yticks([])
   plt.imshow(data[i], cmap = plt.cm.binary)
   plt.xlabel(class_name[label[i]])
```

Output:



1.3 Các hàm xử lý dữ liệu được sử dụng trong đề tài

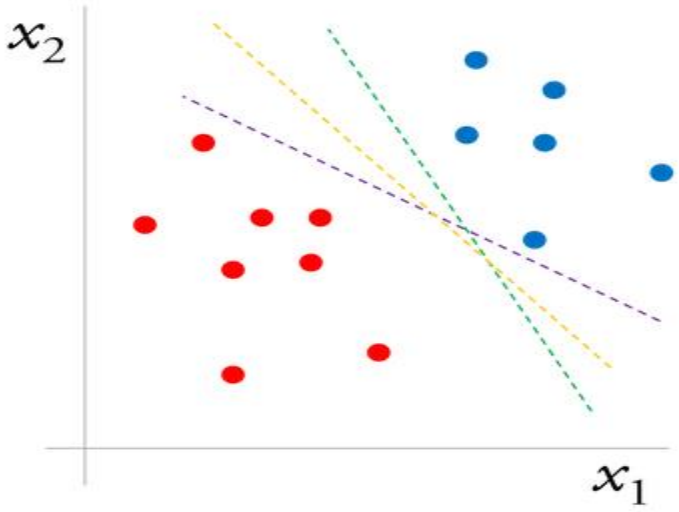
```
def cvt_hsv(X):
    m = X.shape[0]
    X_hsv = np.zeros((m, 256, 256, 3), dtype = np.uint8)
    for i in range(m):
        X_hsv[i] = cv2.cvtColor(X[i], cv2.COLOR_BGR2HSV)
    return X_hsv

def reshape_and_histogram(X):
    m, h, w = X.shape[:3]
    X_histogram = np.zeros((m, 3, 256))
    X_reshape = X.reshape((m, h*w, -1))
    X_reshape = np.transpose(X_reshape, (0, 2, 1))
    X_reshape = X_reshape.astype('uint8')
    for i in range(m):
        for j in range(3):
            (X_hist, _) = np.histogram(X_reshape[i,j], bins = np.arange(257))
            X_histogram[i,j] = X_hist
    return X_histogram

def flatten(X):
    m = X.shape[0]
    X_flatten = X.reshape((m,-1))
    return X_flatten

def normalize(X):
    norm = StandardScaler()
    X_norm = norm.fit_transform(X)
    return X_norm
```

```
def histogram(X, mode = 'bgr'):
    if mode == 'hsv':
        X = cvt_hsv(X)
    X_reshape = reshape_and_histogram(X)
    X_flatten = flatten(X_reshape)
    print('X_flatten.shape: ', X_flatten.shape)
    X_norm = normalize(X_flatten)
    return X_norm

def calculate_metrics(label, label_predict):
    #caculate accuracy and F1-score
    acc_score = accuracy_score(label, label_predict)
    fscore = f1_score(label, label_predict, average = 'weighted')
    return (acc_score, fscore)

def display_metrics(label, label_predict):
    scores = calculate_metrics(label, label_predict)
    print("Model Accuracy : {}".format(scores[0]))
    print("Model F1-Score  : {}".format(scores[1]))

def display_scores(scores):
    print("Scores          : {}".format(scores))
    print("Mean            : {}".format(scores.mean()))

def svm(X, y, C = 10, gamma = 'scale', cv = 3, scoring = 'accuracy'):
    svm = SVC(C = C, gamma = gamma)
    svm_score = cross_val_score(svm, X,y, cv= cv, scoring = scoring)
    display_scores(svm_score)
    y_pred = cross_val_predict(svm, X, y, cv = 3)
    display_metrics(y, y_pred)
    return y_pred, svm
```

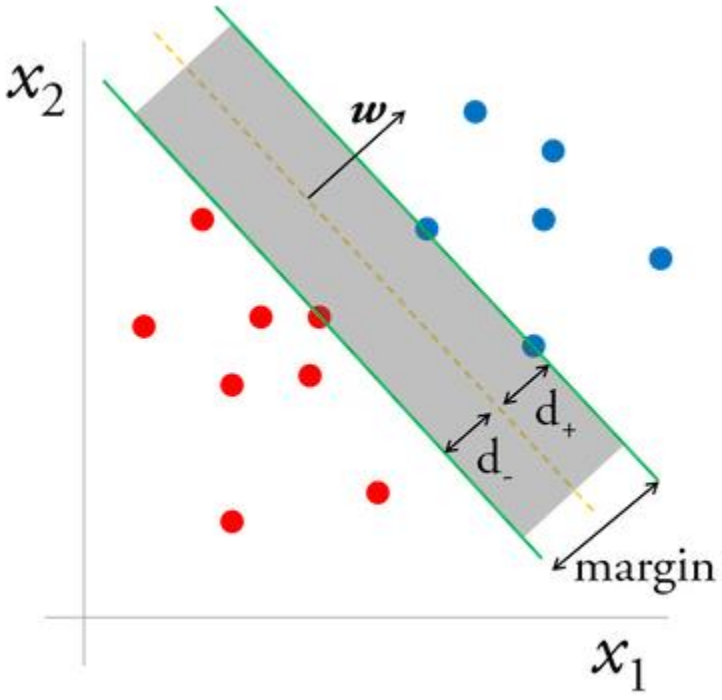
- **Hàm cvt_hsv** để chuyển định dạng màu cơ bản từ RGB sang HSV
- **Hàm reshape_and_histogram**: Chuyển đổi ma trận ảnh ban đầu và lấy histogram của 3 kênh màu
- **Hàm flatten:** Trải ma trận các điểm ảnh thành vector đặc trưng
- **Hàm normalize:** chuẩn hóa các giá trị trong ma trận X theo trung bình và phương sai
- **Hàm histogram**: Lấy histogram của ảnh truyền vào
- **Hàm caculate_metrics:** Tính toán độ chính xác của nhãn gốc và nhãn dự đoán từ mô hình, ở đây ta sử dụng 2 phép đo phổ biến là accuracy_score và f1_score
- **Hàm display_metrics:** Hiển thị các kết quả của 2 phép đo trên
- **Hàm display_score:** Hiển thị các kết quả khi thực hiện cross_validation với dữ liệu
- **Hàm svm:** Là hàm sử dụng thuật toán SVM để tiến hành phân lớp ảnh, để tránh hiện tượng overfitting (quá khớp với dữ liệu đào tạo), ta sử dụng cross_validation với k mặc định = 3 (tham khảo tại https://en.wikipedia.org/wiki/Cross-validation_(statistics))

1.4 Thuật toán SVM

Support Vector Machine (SVM) là một thuật toán học máy cổ điển thuộc nhóm Supervised learning (Học có giám sát) sử dụng để phân chia dữ liệu thành các nhóm riêng biệt. Thuật toán có độ chính xác cao với dữ liệu có số chiều vừa phải, rất phù hợp với bài toán này. Giả sử ta có một bộ data gồm các điểm màu xanh và đỏ như hình dưới



Có rất nhiều đường thẳng có thể chia 2 tập dữ liệu thành 2 lớp, tuy nhiên thuật toán SVM sẽ tìm đường phân lớp sao cho khoảng cách đến 2 điểm dữ liệu gần nhất ở mỗi class là lớn nhất như sau:

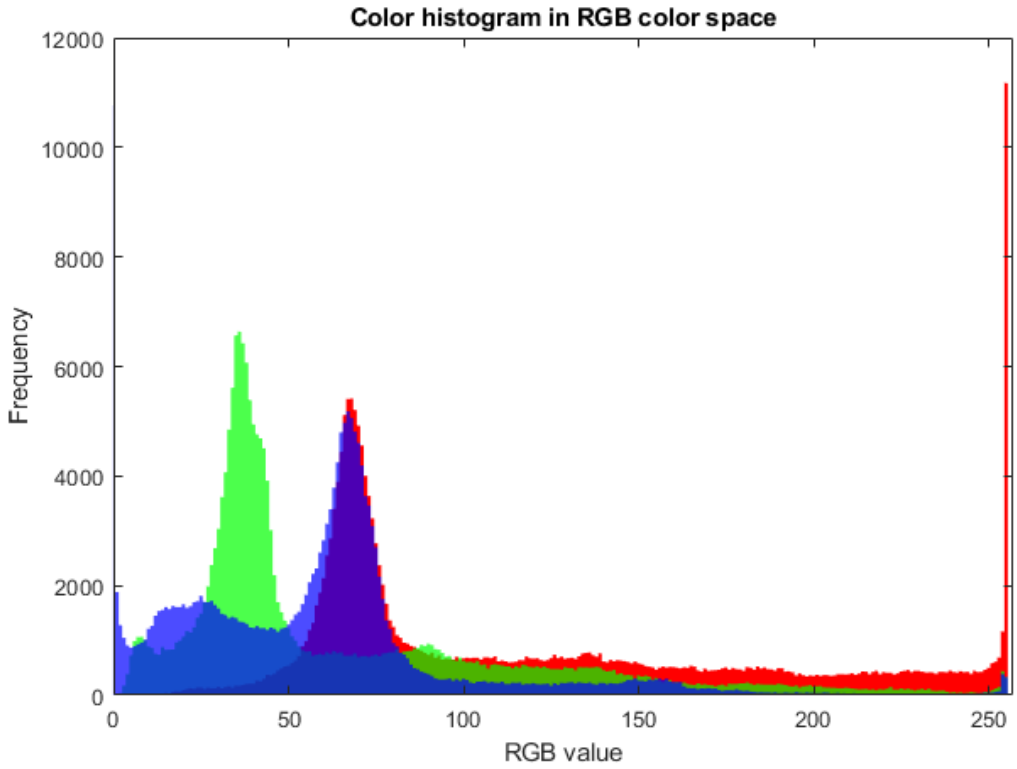


2. Các đặc trưng toàn cục

2.1 Histogram

Histogram là xác suất số lần xuất hiện của một giá trị trên tổng số lần xuất hiện của tất cả các giá trị trong một phân bố xác suất hay chính là hàm mật độ phân bố xác suất của biến ngẫu nhiên

Trong ảnh màu có 3 histogram được chia thành 256 bin ứng với 3 giá trị màu cơ bản là đỏ, xanh lam và xanh lục



2.1.1 Histogram của ảnh RGB

RGB là từ viết tắt của 3 màu cơ bản là Red(đỏ), Green (xanh lá cây), Blue (xanh dương). Mô hình màu RGB sử dụng mô hình bổ sung trong đó ánh sáng đỏ, xanh lục và xanh lam được tổ hợp với nhau theo nhiều phương thức khác nhau để tạo thành các màu khác.

Ví dụ một ảnh ở định dạng RGB, các điểm ảnh sẽ được tổ hợp từ 3 giá trị là màu đỏ, xanh lục và xanh lam, tương tự như khi pha màu với tỷ lệ thích hợp sẽ tạo ra được màu mà ta mong muốn. Do đó, ta thực hiện lấy histogram trên 3 kênh màu với giá trị từ 0 đến 255

Code python:

```
X_hist_bgr = histogram(data)
svm(X_hist_bgr, label)
```

Ta thực hiện lấy giá trị histogram trên dữ liệu gốc có định dạng RGB sau đó đưa vào mô hình SVM để huấn luyện

Đầu ra như sau:

```
Scores            : [0.45666667 0.51839465 0.47315436]
Mean              : 0.4827385593040685
Model Accuracy : 0.48272017837235226
Model F1-Score  : 0.48225087035283815
```

Độ chính xác trung bình của mô hình khi sử dụng histogram RGB là khoảng 48%

2.1.2 Histogram của ảnh HSV

Không gian màu HSV dựa trên 3 số liệu:

H (Hue): vùng màu

S (Saturation): độ bão hòa

V (Value): giá trị

Đây là hệ màu được xây dựng theo cảm nhận của con người, trực quan hơn RGB

Chuyển ảnh sang định dạng HSV sau đó lấy histogram từ 0 đến 255

```
X_hist_bgr = histogram(data, mode='hsv')
svm(X_hist_bgr, label)
```

Output:

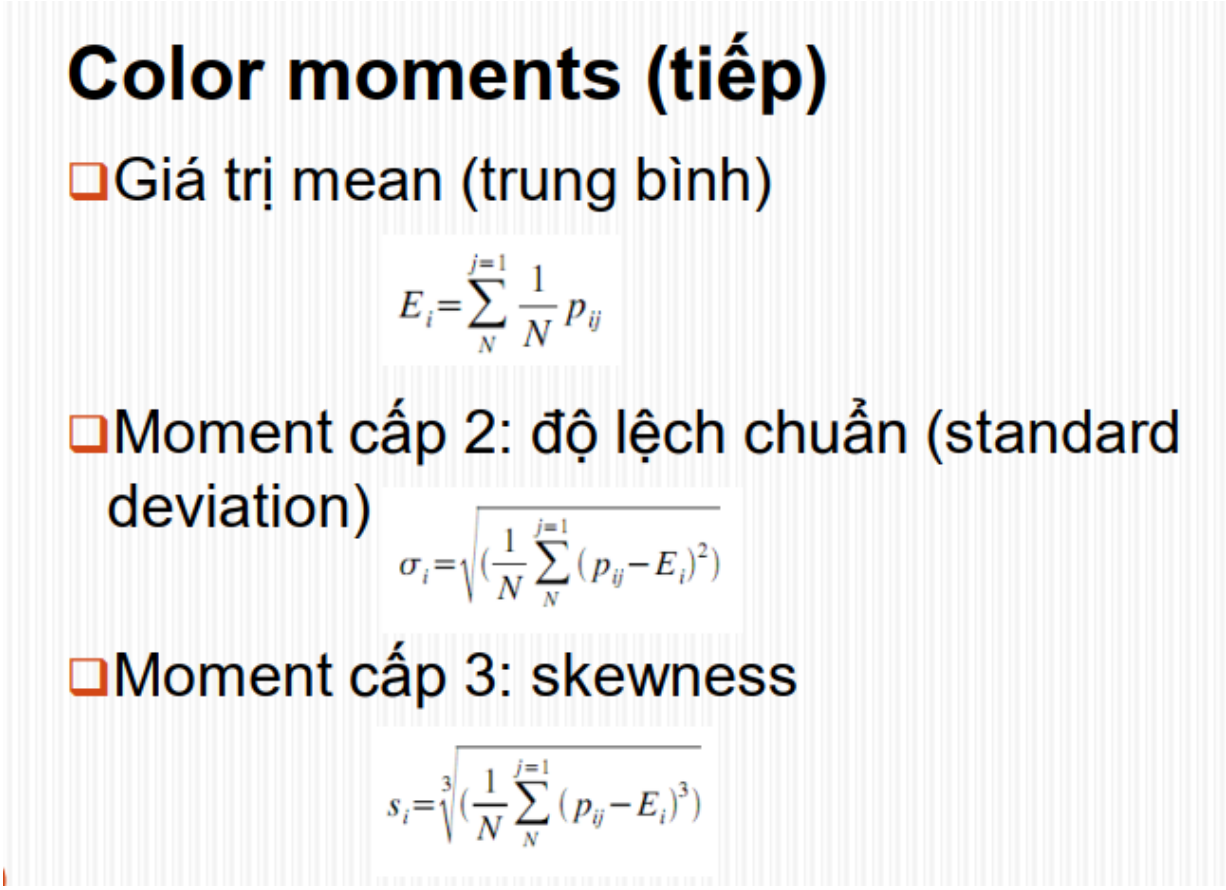
```
Scores            : [0.49       0.53846154 0.5033557 ]
Mean              : 0.5106057477198417
Model Accuracy : 0.5105908584169454
Model F1-Score  : 0.5098134992353562
```

Độ chính xác tăng thêm một chút từ 48% đến 51%

2.2 Đặc trưng về Color moment

Do bản chất của histogram là hàm mật độ phân bố xác suất nên ta có thể đại diện cho một hàm phân bố bằng các giá trị moment cụ thể là các giá trị moment cấp 1(mean), cấp 2(độ lệch chuẩn), cấp 3(skewness).

Các giá trị này được tính như sau:



Trong đó p_{ij} là giá trị của kênh màu i tại pixel có vị trí j ở trong ảnh

Xét ảnh ở trong hệ màu RGB hay HSV ta chỉ cần 9 giá trị để đại diện cho màu sắc của bức ảnh, số chiều của vector đặc trưng giảm đi rất nhiều, tang tố trong quá trình học

Tuy nhiên để kết quả chính xác hơn ta nên sử dụng trong hệ màu HSV

Code python

```
def cacul_moment_color(X_hsv):
    m = X_hsv.shape[0]
    moment_color = np.zeros((m, 9), dtype = np.float32)
    moment_matrix = np.transpose(X_hsv, (0, 3, 1,2))
    moment_matrix = moment_matrix.reshape(m, 3, 256*256)
    mean = np.mean(moment_matrix,axis = -1)
    mean = mean.reshape(m, 3, 1)
    variance = np.power(moment_matrix - mean, 2)
    do_lech_chuan = np.sqrt(np.mean(variance, axis = -1))
    skewness = moment_matrix - mean
    skewness = np.power(skewness, 3)
    skewness = np.mean(skewness, axis = -1)
    skewness = np.cbrt(skewness)
    moment_color[:, :3] = mean.reshape(m, 3)
    moment_color[:, 3:6] = do_lech_chuan.reshape(m, 3)
    moment_color[:, 6:] = skewness.reshape(m, 3)
    return moment_color

m = data.shape[0]
index = int(m/4)
train_X_moment_color1 = cacul_moment_color(data[: index])
train_X_moment_color2 = cacul_moment_color(data[index: 2*index])
train_X_moment_color3 = cacul_moment_color(data[index * 2: 3*index])
train_X_moment_color4 = cacul_moment_color(data[index * 3: ])
train_X_moment_color = np.concatenate((train_X_moment_color1,
train_X_moment_color2,
                        train_X_moment_color3, train_X_moment_color4), axis = 0)

print(train_X_moment_color.shape)
svm(train_X_moment_color, label, C=10)
```

Output:

```
(897, 9)
Scores            : [0.51333333 0.52173913 0.48322148]
Mean              : 0.5060979800927277
Model Accuracy : 0.5061315496098104
Model F1-Score  : 0.5022804997860104
```

Do các phép tính toán sử dụng trên ma trận với số chiều lớn sẽ có tốc độ chậm, nên chia dữ liệu làm 4 phần sau đó ghép lại thành một feature vector có số chiều là 897 x 9 với 897 là số ảnh training và 9 là số đặc trưng về moment của từng ảnh
Ta thấy độ chính xác vẫn tương đương như khi training bằng histogram

2.3 Đặc trưng LBP

Local binary parttern là một mô hình sử dụng để trích xuất các đặc trưng cục bộ có trong ảnh, LBP được mô tả đầu tiên vào năm 1994, tuy đã có tuổi đời rất lâu nhưng mô hình này hiện nay vẫn còn được sử dụng rất nhiều trong bài toán phân lớp ảnh.
Vector đặc trưng LBP được tạo như sau:

- Chia hình ảnh thành một lưới các hình ảnh nhỏ hơn (các ô)
- Đối với mỗi pixel trong một ô, so sánh giá trị pixel đó với 8 pixel lân cận của nó (ở phía trên bên trái, giữa bên trái, dưới cùng bên trái, trên cùng bên phải, v.v.). Theo dõi các pixel dọc theo vòng tròn, tức là theo chiều kim đồng hồ hoặc ngược chiều kim đồng hồ.
- Nếu giá trị của pixel trung tâm lớn hơn giá trị lân cận, hãy viết "0", nếu không, hãy viết "1". Điều này tạo thành một số nhị phân 8 chữ số (thường được chuyển đổi thành số thập phân để thuận tiện).
- Tính toán histogram của từng ô, thu được vector đặc trưng 256 chiều

Tuy nhiên, số chiều 256 trong một ô là khá lớn hơn nữa có nhiều parttern ít xuất hiện, Ojala tác giả gốc của bài báo đã đề xuất một khái niệm “uniform”. Một parttern được gọi là uniform nếu nó chứa nhiều nhất 2 lần đảo bit từ 1 sang 0 và từ 0 sang 1 với thứ tự các bit duyệt theo vòng tròn
Ví dụ:
- Các patterns 00000000 (0 transitions), 01110000 (2 transitions) và 11001111 (2 transitions) là uniform
- Các patterns 11001001 (4 transitions) và 01010010 (6 transitions) non uniform

Sau đó: Mỗi uniform pattern được gán một nhãn, Tất cả các non-uniform pattern được gán chung 1 nhãn. Như vậy nếu dùng (8,1) neighborhood thì sẽ có 256 pattern, trong đó có 58 uniform, nên suy ra số chiều của LBP feature là 59

Xây dựng vector đặc trưng lbp với một lưới chia hình ảnh thành 4 ô

Code python

```
def lbp(X, P = 8, R = 1):
    m = X.shape[0]
    #p = 134      ##12:134, 16:242
    p=58
    hist_X = np.zeros((m, (p+1) * 4))
    for i in range(m):
        im = X[i]
        gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
        eps = 1e-8
        lbp1 = local_binary_pattern(gray[:128, :128], P = P, R = R, method='nri_uniform')
        (hist1, _) = np.histogram(lbp1.ravel(),bins=np.arange(0, p+2),range=(0, p+1))
        lbp2 = local_binary_pattern(gray[128:256, :128], P = P, R = R, method='nri_uniform')
        (hist2, _) = np.histogram(lbp2.ravel(),bins=np.arange(0, p+2),range=(0, p+1))
        lbp3 = local_binary_pattern(gray[:128, 128:256], P = P, R = R, method='nri_uniform')
        (hist3, _) = np.histogram(lbp3.ravel(),bins=np.arange(0, p+2),range=(0, p+1))
        lbp4 = local_binary_pattern(gray[128:256, 128:256], P = P, R = R, method='nri_uniform')
        (hist4, _) = np.histogram(lbp4.ravel(),bins=np.arange(0, p+2),range=(0, p+1))
        # normalize the histogram
        hist = np.concatenate((hist1, hist2,hist3, hist4), axis = 0).astype(np.float64)
        hist /= (hist.sum() + eps)
        hist_X[i] = hist
    return hist_X

X_lbp = lbp(data)
print(X_lbp.shape)
svm(X_lbp, label, C=100)
```

Output:

```
(897, 236)
Scores            : [0.70333333 0.73244147 0.63758389]
Mean              : 0.6911195658408964
Model Accuracy : 0.6911928651059086
Model F1-Score  : 0.6914710175538469
```

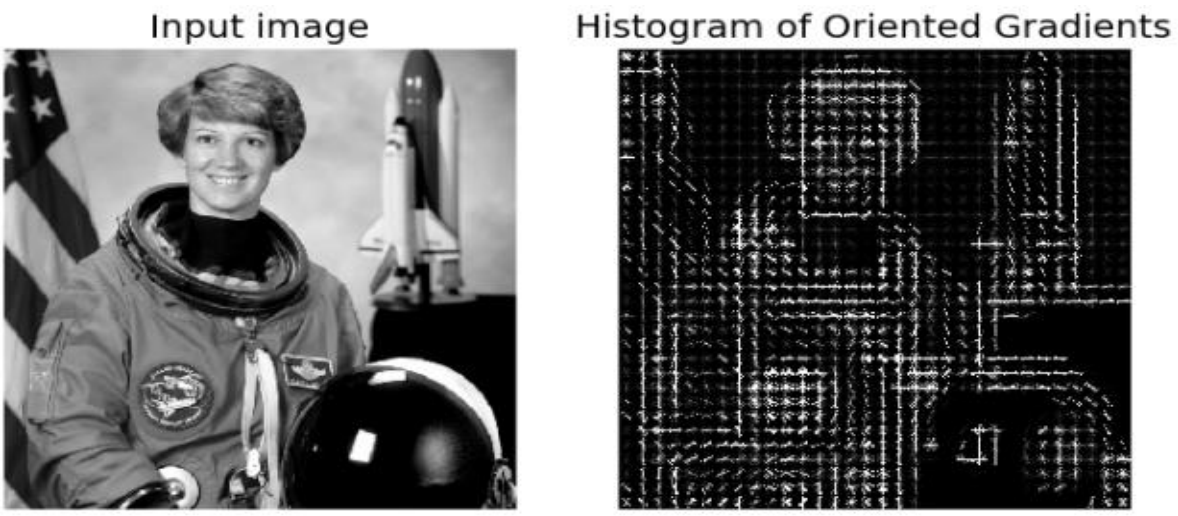
Số chiều của vector đặc trưng là 59x4 = 236, độ chính xác đã được cải thiện rất nhiều từ 50% lên 70%, qua đó có thể thấy hiệu quả của đặc trưng LBP

2.4 Đặc trưng HOG

Histogram of Oriented Gradient (Biểu đồ định hướng độ dốc) là một đặc trưng toàn cục được sử dụng rất nhiều trong xử lý ảnh và computer vision để phát hiện đối tượng có trong ảnh. Các khái niệm về HOG được nêu ra từ năm 1986 tuy nhiên cho đến năm 2005 HOG mới được sử dụng rộng rãi sau khi Navneet Dalal và Bill Triggs công bố những bổ sung về HOG. Hog tương tự như các biểu đồ edge orientation, scale-invariant feature transform descriptors(như sift, surf,..), shape contexts nhưnghog được tính toán trên một lưới dày đặc các cell và chuẩn hóa sự tương phản giữa các block để nâng cao độ chính xác. Hog được sử dụng chủ yếu để mô tả hình dạng và sự xuất hiện của một object trong ảnh. Các bài toán tính toán HOG thường gồm 5 bước:

1. Chuẩn hóa hình ảnh trước khi xử lý
2. Tính toán gradient theo cả hướng x và y .
3. Lấy phiếu bầu cùng trọng số trong các cell
4. Chuẩn hóa các block
5. Thu thập tất cả các biểu đồ cường độ gradient định hướng để tạo ra feature vector cuối cùng.

Sau khi tính toán HOG, ta thu được một vector đặc trưng được ghép bởi các histogram về độ dốc của các block trong ảnh sau khi được chia lưới thể hiện trong hình ảnh sau:



Chi tiết về thuật toán HOG có thể tham khảo tại đây hoặc tại đây

Code python

```
def hog_feature(X):
    X_hog = []
    im_hog = []
    m = X.shape[0]
    for i in range(m):
        fd, im = hog(X[i], orientations=12, pixels_per_cell=(64, 64),
                cells_per_block=(2,2), visualize=True, multichannel=True, block_norm='L2')
        X_hog.append(fd)
        im_hog.append(im)
    return np.asarray(X_hog), np.asarray(im_hog)

X_hog, im_hog = hog_feature(data)
print(X_hog.shape)
svm(X_hog, label, C=5)
```

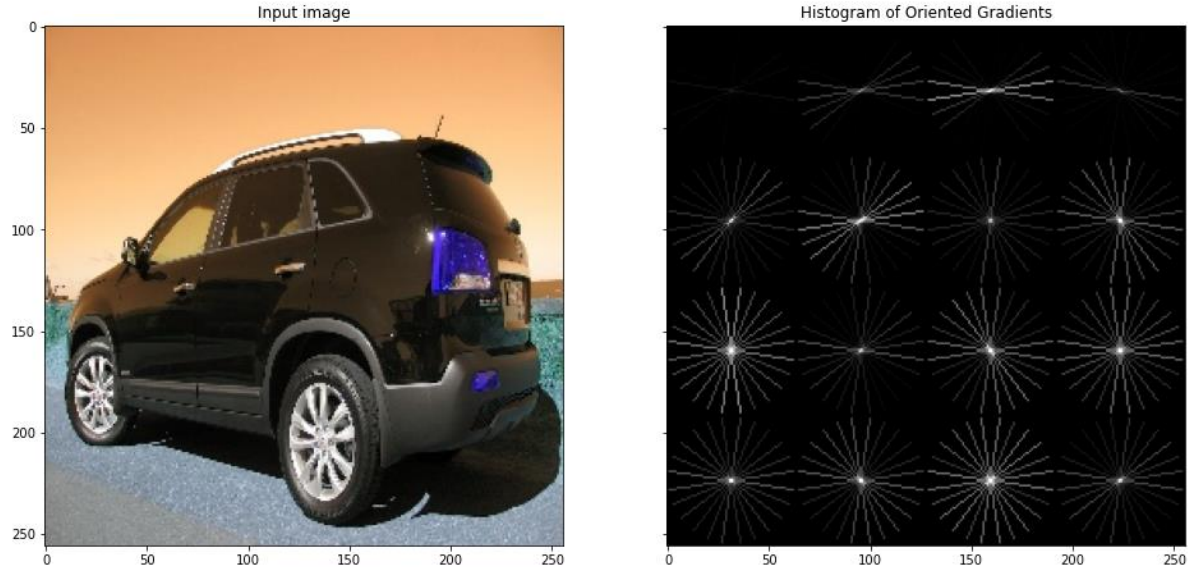
Tính toán HOG với khoảng cách giữa một bin từ 0 đến 180o là 12, block_size = (64x64), cell_per_block là (2x2) tức là mỗi block lại được chia thành ma trận 2x2

Output:

```
(897, 432)
Scores              : [0.83       0.80267559 0.84563758]
Mean                : 0.8261043897256327
Model Accuracy : 0.8260869565217391
Model F1-Score  : 0.8253006544131689
```

Đến đây, độ chính xác đã được cải thiện rất tốt từ 70% đến 82.6%, cho ta thấy hiệu quả của HOG tốt đến như thế nào với các đặc trưng trước

Hình ảnh về độ dốc được thể hiện trong hình sau:



Vậy liệu độ chính xác có tăng lên nữa không khi ta kết hợp 2 đặc trưng LBP và HOG lại với nhau. Nghiên cứu chỉ ra rằng khi kết hợp 2 đặc trưng này lại với nhau thì độ chính xác trong việc phân loại ảnh là rất lớn, ta cùng thử xem.

2.5 Kết hợp đặc trưng LBP và đặc trưng HOG

Từ các vector feature đã tìm được khi tính toán các đặc trưng về LBP và HOG, ta sẽ ghép 2 vector này lại thành một để làm đầu vào cho quá trình đào tạo SVM

```
X_lbp_norm = normalize(X_lbp)
X_hog_norm = normalize(X_hog)

X_total = np.concatenate((X_lbp_norm, X_hog_norm), axis = 1)
print(X_total.shape)
label_pred_total, clf = svm(X_total, label, C=10)
```

Output:

```
(897, 668)
Scores            : [0.85666667 0.86956522 0.83892617]
Mean              : 0.8550526861848718
Model Accuracy : 0.855072463768116
Model F1-Score  : 0.8546823023996949
```

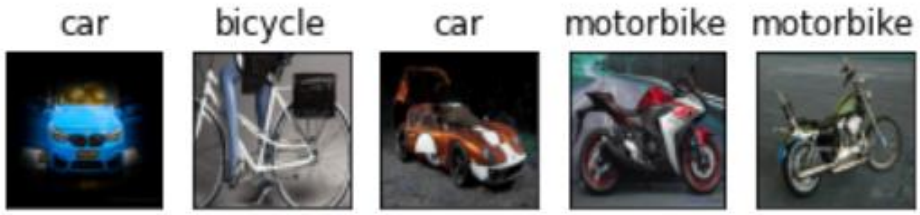
Đầu tiên ta cần chuẩn hóa 2 vector LBP và HOG về cùng một phân phối bằng hàm normalize, sau đó nối 2 vector này lại với nhau theo chiều ngang, tức là axis = 1, số chiều của vector tăng lên 668 bằng tổng của 432 feature từ HOG và 236 feature từ LBP. Sau khi kết hợp 2 đặc trưng này thì độ chính xác tăng lên khá cao là 85%, qua đó có thể thấy sự hiệu quả khi kết hợp 2 đặc trưng này lại với nhau.

Visualize kết quả của một số ảnh

```
for i in range(5 , 10):
   plt.subplot(1,5,i-4)
   plt.imshow(data[i])
   plt.title(class_name[label[i]])
   plt.xticks([])
   plt.yticks([])
   print(class_name[label[i]], end=' : ')
   print(class_name[label_pred_total[i]])
```

Kết quả như sau:

```
car : car
bicycle : bicycle
car : car
motorbike : motorbike
motorbike : motorbike
```



3. Kết luận

Trích chọn các điểm đặc trưng là công việc quan trọng và thường gặp trong các bài toán xử lý ảnh hay thị giác máy tính. Hai đặc trưng được quan tâm nhất là đặc trưng LBP và đặc trưng HOG mang lại kết quả tốt trong bài toán phân lớp ảnh và nhận diện đối tượng có trong ảnh. Đề tài sử dụng 2 thư viện cho học máy và xử lý ảnh là sklearn và skimage hỗ trợ trên ngôn ngữ Python. Em rất mong nhận được sự góp ý từ thầy để đề tài được hoàn thiện tốt hơn. Em xin chân thành cảm ơn!

Link source code: https://github.com/BinhNguyen231/lbp_hog_image_processing

4. Tài liệu tham khảo

[1] N. Dalal and B. Triggs et al, "Histograms of oriented gradients for human detection", *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*

[2] Ojala et al, “Multiresolution Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns”

[3] Slide bài giảng Xử lý ảnh – Hoàng Văn Hiệp – Đại học Bách Khoa Hà Nội