Digital Image Processing Report

Group 7

<Car Logo Recognition>

Group members:

- Nguyễn Tiến Dũng - BI9-071
- Nguyễn Minh Hiếu - BI9-104
- Đinh Gia Lượng - BI9-159
- Đàm Minh Phúc - BI9-188
- Hoàng Nhật Tân - BI-205

I. Introduction

- The presence of the cars is becoming extremely commonplace as the price of cars is getting lower everyday. As more and more people are having needs in travelling with cars, there are now so many types of cars which we can see anywhere in the world. The easiest way to recognize the types is based on its logo, which is the unique presence of a car. However, it's nearly impossible for anyone to remember every single logo of every car. Therefore, our application, Car Logo Recognition, will help you to define which type of car .

II. User Requirements

1. Purpose and scope

- The application will receive a car logo, after the training time, it will return the name of the car you need.
- With this application, you will be able to recognize your type of car easily just based on your car logo.

2. System Overview

System Overview	Detail
System name	Car Logo Recognition
System type	Image Processing
Environment	OpenCV, Python

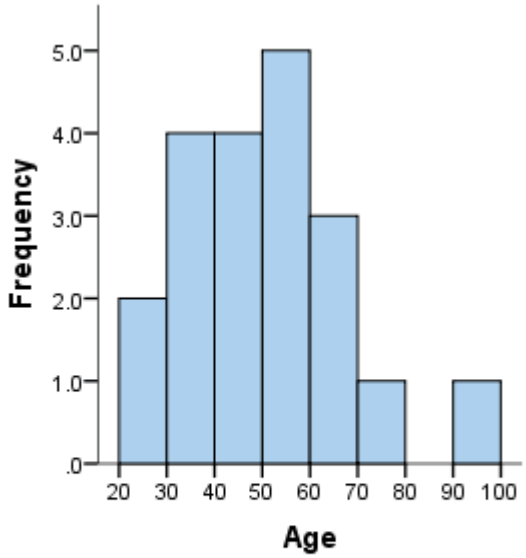
III. Methods

1. Image Processing

- Image processing is a technique used to manipulate an image through several sets of algorithms resulting in an output image or any feature abstracted from the image. The manipulations in an image include filtering out the noises, image conversions into different color spaces, blurring the image, edge detection techniques, line detection, circle detection, etc

2. Histogram

- A histogram is an approximate representation of the distribution of numerical. It is a plot that lets you discover, and show, the underlying frequency distribution (shape) of a set of continuous data. This allows the inspection of the data for its underlying distribution (e.g., normal distribution), outliers, skewness, etc. An example of a histogram, and the raw data it was constructed from, is shown below:



36	25	38	46	55	68	72	55	36	38
67	45	22	48	91	46	52	61	58	55

- Difference between a bar chart and a histogram:
  The major difference is that a histogram is only used to plot the frequency of score occurrences in a continuous data set that has been divided into classes, called bins. Bar charts, on the other hand, can be used for a great deal of other types of variables including ordinal and nominal data sets.

3. Histogram of Oriented Gradients (HOG)

- The histogram of oriented gradients is a feature descriptor used in computer vision and image processing for the purpose of object detection.
- HOG descriptors are mainly used to describe the structural shape and appearance of an object in an image, making them excellent descriptors for object classification. However, since HOG captures local intensity gradients and edge directions, it also makes for a good texture descriptor.

- The HOG descriptor returns a real-valued feature vector. The dimensionality of this feature vector is dependent on the parameters chosen for the `orientations`, `pixels_per_cell`, and `cells_per_block` parameters mentioned above.
- Calculating the HOG:

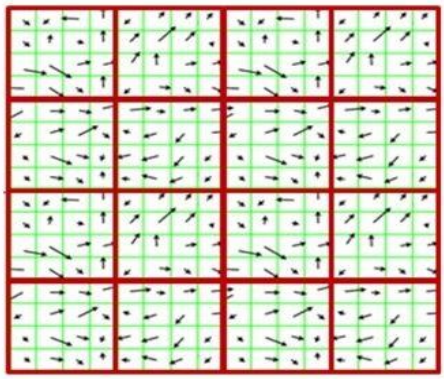
+ Normalizing the image prior before we train.
    - Gamma/Power Law Normalization: In this case, we take the log(p) in the input image.
    - Square-Root Normalization: Here, we take the $\sqrt{(p)}$ of each pixel p in the input image. By definition, square-root normalization compresses the input pixel intensities far less than gamma normalization.
    - Variance Normalization: Here, we compute both the mean μ and standard deviation σ of the input image. All pixels are mean centered by subtracting the mean from the pixel intensity, and then normalized through dividing by the standard deviation: .$p' = (p - \mu)/\sigma$

+Gradient Calculation

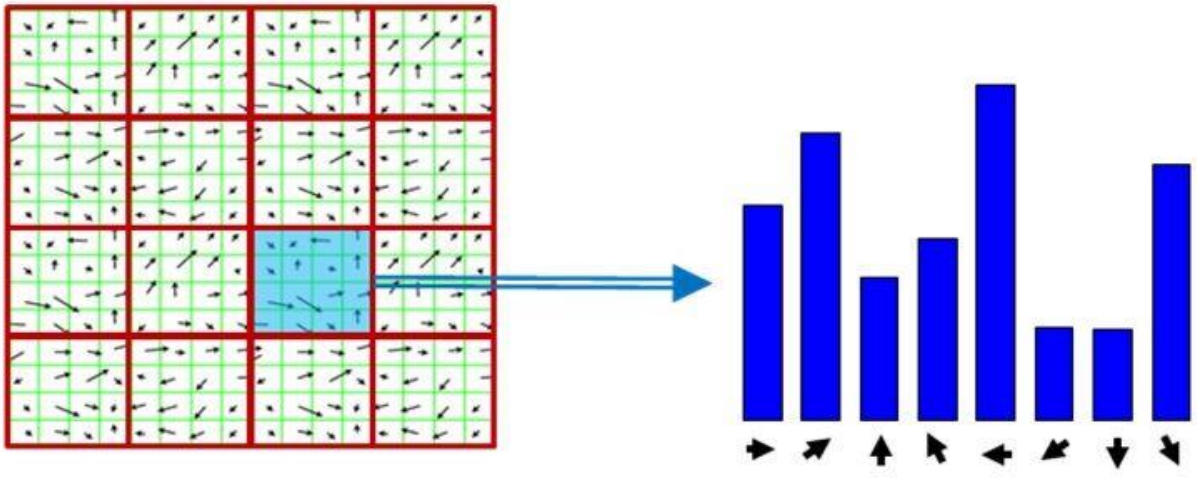
- To get the Gradient image, we will use the convolution: $G_x = I \star D_x$and $G_y = I \star D_y$ with I is the input image, $D_x$is the filter of the x-direction and D_y is the filter of the y-direction.
- After observing the gradient image, we can calculate the gradient magnitude representation of the image: $|G| = \sqrt{G_x^2 + G_y^2}$.
- Finally, the orientation of the gradient for each pixel is calculated:

$$\vartheta = arctan2(Gy, Gx)$$

Based on $|G|$ and ϑ, we can calculate the histogram of oriented gradients, where the bin of the histogram is based on ϑ and the weight added to given bin is based on the $|G|$.



+ Weighted votes in each cells
    - A “cell” is a rectangular region defined by the number of pixels that belong in each cell. For example, if we had a 128 x 128 image and defined our `pixels_per_cell` as 4 x 4, we would thus have 32 x 32 = 1024 cells. For each of the cells in the image, we need to construct a histogram of oriented gradients using our gradient magnitude $|G|$ and orientation ϑ mentioned above. Finally, each pixel will contribute a weighted vote in the bin - the weight of the vote is the oriented gradient $|G|$ at the given pixel. Now, we will collect and graft each of these histograms to form the last feature vector.



+ Contrast Normalization Over Blocks
    - Again, the number of blocks are rectangular. However, our units are no longer pixels, they are the cells. We usually use

2x2 or 3x3 cell_per_block to obtain the accuracy in most of the cases. For example, we have 3x3 cells and pixel_per_block= 2x2 then we will have 4 blocks. Finally, we will collect the histograms, graft them and treat them as our final feature vector.

IV. Code Implementation

1. Train car logos (train-car-logos.py)

- Set path for input, dataset and model

```
inputPath = '.../test_images/'
datasetPath = '.../car_logos/'
modelPath = '.../model/'
modelName = 'carLogoModel.pkl'
```

- Count the number of logos on the progress bar:

```
barCount = 0
for logos in os.listdir(datasetPath):
    if os.path.isdir(datasetPath+logos):
        barCount +=1
```

- Load the image: convert to gray, find edges and find contours

```
imagePath = datasetPath+logos+ '/' + imageName
print(imagePath);
image=cv2.imread(imagePath)
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
canny = cv2.Canny(gray,100,255)

contours, hierarchy=cv2.findContours(canny.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
```

- Find HOG of the logo:

```
x,y,w,h=cv2.boundingRect(maxCnt)
logo = gray[y:y + h, x:x + w]
logo = cv2.resize(logo, imageSize)
H = feature.hog(logo, orientations=9, pixels_per_cell=(10, 10),
    cells_per_block=(2, 2), transform_sqrt=True)
```

- Update HOG image and logo:

```
data.append(H)
labels.append(label)
```

- Train model:

```
print ("[INFO] Training kNN classifier")
model = KNeighborsClassifier(n_neighbors=1)
model.fit(data, labels)

print("[INFO] Saving ML Model")
joblib.dump(model,modelPath + modelName)
print('[INFO] ML Model Trained, Time Taken: ' + str(round((time.time() - initialTime),3)) + ' sec')
print('[INFO] ML Model Name: ' + str(modelName))
modelSize = round(os.stat(modelPath+modelName).st_size / (1024 * 1024),3)
print('[INFO] ML Model size: ' + str(modelSize) + ' MB')
return model
```

2. Recognize car logos (recognize-car-logos.py)

- Set path for test folder and model:

```
inputPath='.../test_images/'
modelPath = '.../model/'
testImageName='test1.jpg'
modelName = 'carLogoModel.pkl'
```

- Load trained model:

```
print ('[INFO] Loading ML Model')
initialTime = time.time()
model=joblib.load(modelPath+modelName)
print('[INFO] ML Model Loaded ,Time Taken: ' + str(round((time.time() - initialTime),3)) + ' sec')
print('[INFO] ML Model Name: ' + str(modelName))
modelSize = round(os.stat(modelPath+modelName).st_size / (1024 * 1024),3)
print('[INFO] ML Model size: ' + str(modelSize) + ' MB')
return model
```

- Load test logo: convert to gray, resize to resolution 200x100 and find HOG of test logo.

```
image=cv2.imread(inputPath+testImageName)
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
gray=cv2.resize(gray, imageSize)

(H, hogImage)=feature.hog(gray,orientations=9,pixels_per_cell=(10,10),cells_per_block=(2,2),
                transform_sqrt=True,visualize=True)
pred = model.predict(H.reshape(1,-1))[0]
```

- Show HOG of test logo:

```
hogImage = exposure.rescale_intensity(hogImage, out_range=(0, 255))
hogImage = hogImage.astype("uint8")
cv2.imshow("HOG Image", hogImage)
```

- Show test logo with predicted label:

```
cv2.putText(image, pred.title(), (10, 35), cv2.FONT_HERSHEY_SIMPLEX, 1.0,
            (0, 0, 255), 2)
cv2.imshow("Test Image", image)
```