P. Vehicle Dectection and Tracking

`CarND`

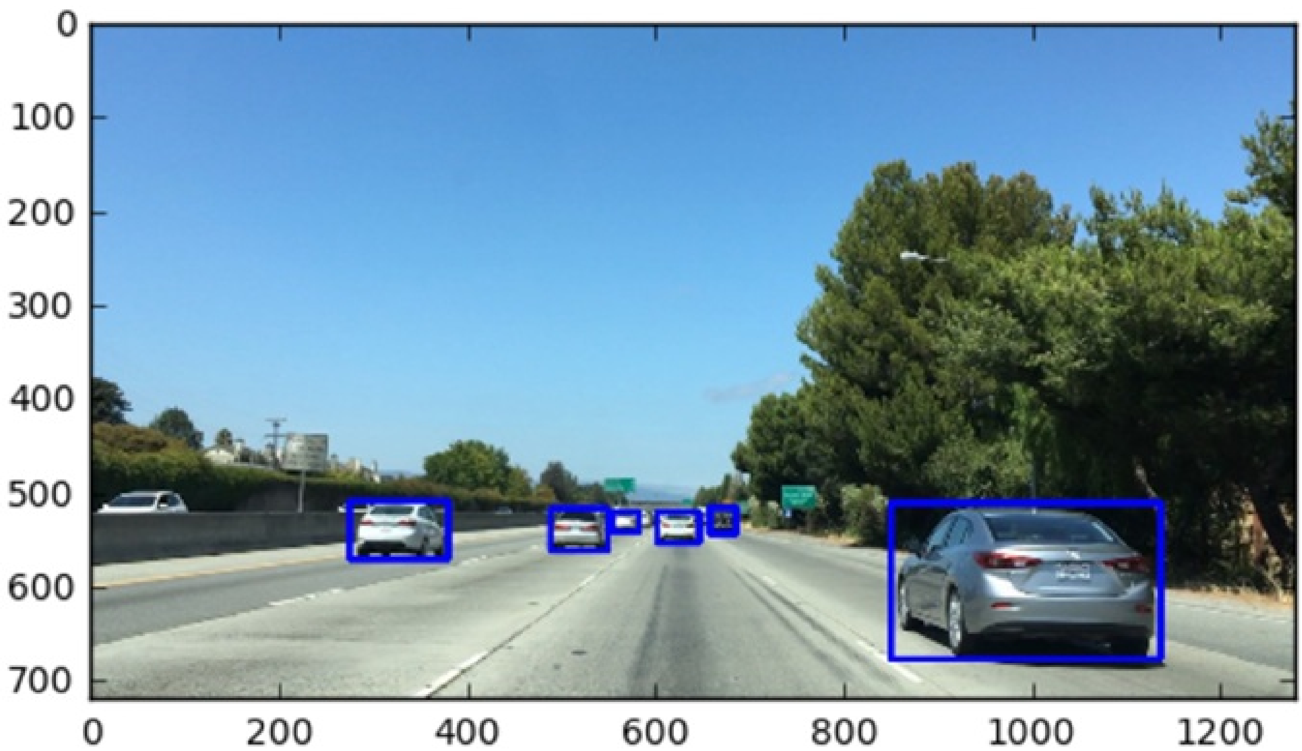
> 作业地址：https://github.com/morgengc/CarND-Vehicle-Detection

这一章课程的核心内容是，对图像进行特征提取，得到特征向量，将特征向量输入分类器模型，对模型进行调参，最后得到一个良好的分类器模型。

记住，不同于神经网络方法，本章分类模型的输入并不是图像本身，而是图像的特征向量。因此，全篇费了老大的劲在讲述如何进行图像的特征提取，包括基于颜色的方法（颜色直方图和空域分级）和基于形状的方法（HOG）。

5. Manual Vehicle Detection

OpenCV 提供了一个 `cv2.rectangle()` 函数，在图像的给定区域内，绘制一个矩形，如下图：

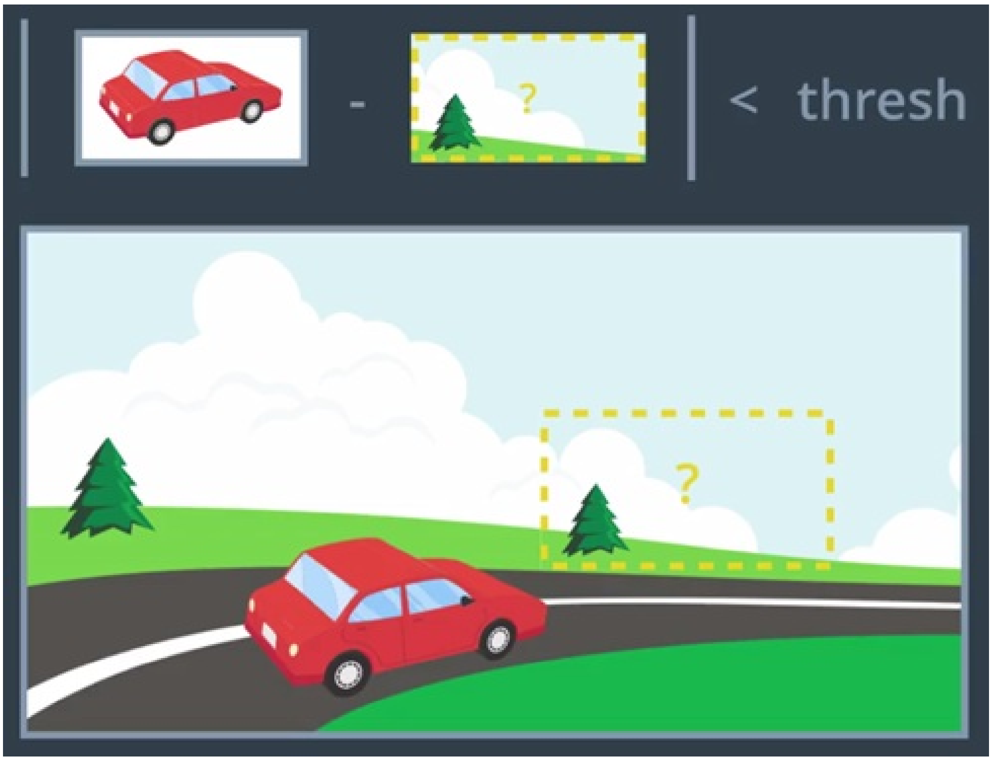


代码实现如下：

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

image = mpimg.imread('bbox-example-image.jpg')

# Define a function that takes an image, a list of bounding boxes,
# and optional color tuple and line thickness as inputs
# then draws boxes in that color on the output

def draw_boxes(img, bboxes, color=(0, 0, 255), thick=6):
    # make a copy of the image
    draw_img = np.copy(img)
    for bbox in bboxes:
        cv2.rectangle(draw_img, bbox[0], bbox[1], color, thick)
    # draw each bounding box on your image copy using cv2.rectangle()
    # return the image copy with boxes drawn
    return draw_img # Change this line to return image copy with boxes

# Add bounding boxes in this format, these are just example coordinates.
bboxes = [((275, 572), (380, 510)), ((488, 563), (549, 518)), ((554, 543), (582, 522)),
          ((601, 555), (646, 522)), ((657, 545), (685, 517)), ((849, 678), (1135, 512))]

result = draw_boxes(image, bboxes)
plt.imshow(result)
```

9. Template Matching

模板匹配的原理是，事先手动给定要识别的物体，然后在目标图像中选取同样大小的一块区域，进行一个比较，当差值小于给定阈值时，就认为匹配到了想要的物体。



OpenCV 提供了 `cv2.matchTemplate(image, templ, method) -> result` 函数专门用于模板匹配，通过在输入图像上滑动，对实际的模板图像块和输入图像进行匹配。如果输入图像 image 的大小为 $W * H$，模板 templ 的大小为 $w * h$，那么最后的结果 result 将会是一个数组，它的每一个点代表输入图像上的左上角，表示每次滑动的起始点，那么 result 的大小将是 $(W - w + 1) * (H - h + 1)$。

这么多个匹配结果中，我们需要从中挑选出匹配最好的那一张，只需要从 result 中找到一个最小值点即可，这个点代表了结果图像的左上角。通过 `cv.MinMaxLoc(arr, mask=None) -> (minVal, maxVal, minLoc, maxLoc)` 函数来实现，最小值由 `minVal` 表示，对应的点由 `minLoc` 表示。

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

image = mpimg.imread('bbox-example-image.jpg')
#image = mpimg.imread('temp-matching-example-2.jpg')
templist = ['cutout1.jpg', 'cutout2.jpg', 'cutout3.jpg',
            'cutout4.jpg', 'cutout5.jpg', 'cutout6.jpg']

# Here is your draw_boxes function from the previous exer
```

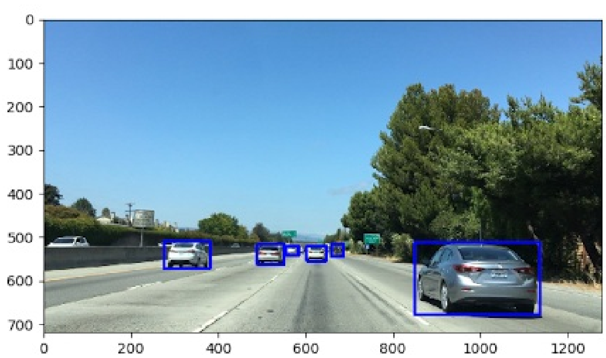
```
cise
def draw_boxes(img, bboxes, color=(0, 0, 255), thick=6):
    # Make a copy of the image
    imcopy = np.copy(img)
    # Iterate through the bounding boxes
    for bbox in bboxes:
        # Draw a rectangle given bbox coordinates
        cv2.rectangle(imcopy, bbox[0], bbox[1], color, thick)
    # Return the image copy with boxes drawn
    return imcopy


# Define a function to search for template matches
# and return a list of bounding boxes
def find_matches(img, template_list):
    # Define an empty list to take bbox coords
    bbox_list = []
    # Define matching method
    # Other options include:
    #     'cv2.TM_CCORR_NORMED'
    #     'cv2.TM_CCOEFF'
    #     'cv2.TM_CCORR',
    #     'cv2.TM_SQDIFF'
    #     'cv2.TM_SQDIFF_NORMED'
    method = cv2.TM_CCOEFF_NORMED
    # Iterate through template list
    for temp in template_list:
        # Read in templates one by one
        tmp = mpimg.imread(temp)
        # Use cv2.matchTemplate() to search the image
        result = cv2.matchTemplate(img, tmp, method)
        # Use cv2.minMaxLoc() to extract the location of the best match
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
        # Determine a bounding box for the match
        w, h = (tmp.shape[1], tmp.shape[0])
        if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
            top_left = min_loc
        else:
            top_left = max_loc
```

```
        bottom_right = (top_left[0] + w, top_left[1] + h)
        # Append bbox position to list
        bbox_list.append((top_left, bottom_right))
        # Return the list of bounding boxes

    return bbox_list

bboxes = find_matches(image, templist)
result = draw_boxes(image, bboxes)
plt.imshow(result)
```

滑动计算时，我们考察的是模板图像和目标区域的协方差 `TM_CCOEFF_NORMED`，除此以外，还有很多其他方法，比如：

方法	说明
TM_SQDIFF	平方差匹配法：该方法采用平方差来进行匹配；最好的匹配值为0；匹配越差，匹配值越大。
TM_CCORR	相关匹配法：该方法采用乘法操作；数值越大表明匹配程度越好。
TM_CCOEFF	相关系数匹配法：1表示完美的匹配；-1表示最差的匹配。
TM_SQDIFF_NORMED	归一化平方差匹配法
TM_CCORR_NORMED	归一化相关匹配法
TM_CCOEFF_NORMED	归一化相关系数匹配法

匹配结果还不错哈，如下左图。然而，来自于同一个视频的后面几秒钟的图像，运行此代码后得到的结果如下右图所示，可以看出效果非常差。



所以，这一节实际上是告诉我们，有这么一个模板匹配的方法，但在无人车的场景下，几乎没法使用这个技术。究其原因，那是因为这种场景下，物体的大小、方向是一直不断变化的，你没法用一个固定的模板来匹配，而模板匹配技术则要求模板和目标之间的差异不能太大。

实用模板匹配技术比较好的场景是，我们的模板是从输入图像中提取出来的一个块图像，从这个块图像返回去在输入图像中检索（比如我要在某个页面检索一个表情图像出现的位置），这种情况下，精度非常之高，但确实没太大用处。

11. Color Histogram Features

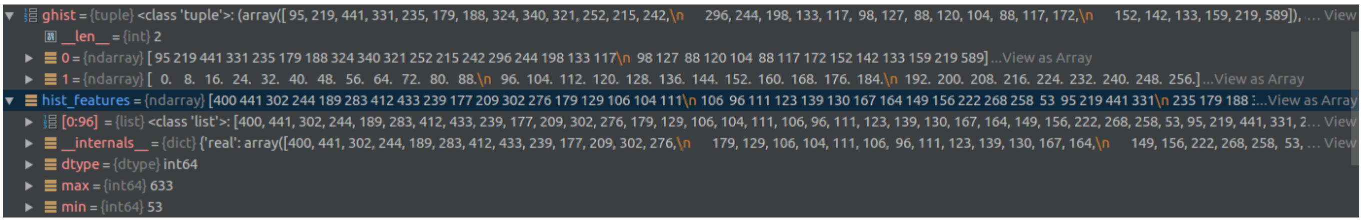
模板匹配是一种对原始像素强度进行统计的技术，对颜色出现的顺序、位置都有极高的要求。而颜色直方图则刻意减少了这些要求，侧重于考虑模板内的各种颜色成分的分布情况，是一种更普适的方法。不过这种方法也不是特别可靠，正像下图所示的，问题区域的颜色分布也会导致错误识别。

12. Histograms of Color

首先通过 `np.histogram()` 函数，分别求出 RGB 三个通道的直方图分布。该函数返回值 `rhist` 是一个 tuple 二元组，其中 `rhist[1]` 表示直方图每个柱的右边缘，因为本例设置了 `bins=32`，所以就有 33 个值：`[0, 8, 16, 24, ..., 256]`；而 `rhist[0]` 则代

表了每个柱范围内，有多少个点，比如 `[400, 441, 302, ..., 258, 53]`，一共 32 个数，所有值加起来就是红色像素的个数，应该等于原始图片的 `长度*高度`。

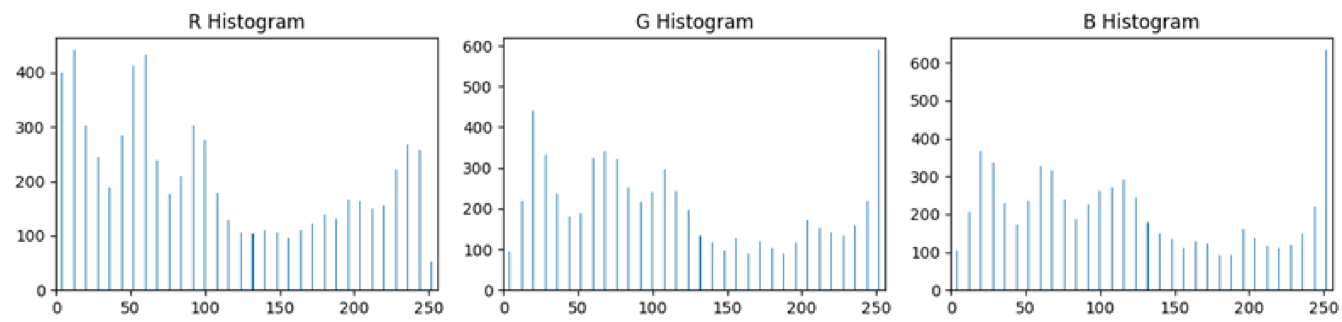
可以看出，无论是 RGB 哪个通道，`hist[1]` 这个值都是一样的，所以下面代码中求取立柱中线时，只使用 R 通道即可，GB 通道是一样的值。而 `hist[0]` 则各异，可以通过 `np.concatenate()` 将他们组合起来，组合前后值的变化可以参见下图：




```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

image = mpimg.imread('cutout1.jpg')

# Define a function to compute color histogram features
def color_hist(img, nbins=32, bins_range=(0, 256)):
    # Compute the histogram of the RGB channels separately
    rhist = np.histogram(img[:,:,0], bins=nbins, range=bins_range)
    ghist = np.histogram(img[:,:,1], bins=nbins, range=bins_range)
    bhist = np.histogram(img[:,:,2], bins=nbins, range=bins_range)
    # Generating bin centers
    bin_edges = rhist[1]
    bin_centers = (bin_edges[1:]  + bin_edges[0:len(bin_edges)-1])/2
    # Concatenate the histograms into a single feature vector
    hist_features = np.concatenate((rhist[0], ghist[0], bhist[0]))
    # Return the individual histograms, bin_centers and feature vector
    return rhist, ghist, bhist, bin_centers, hist_feature
```

```
s

rh, gh, bh, bincen, feature_vec = color_hist(image, nbins=32, bins_range=(0, 256))

# Plot a figure with all three bar charts
if rh is not None:
    fig = plt.figure(figsize=(12,3))
    plt.subplot(131)
    plt.bar(bincen, rh[0])
    plt.xlim(0, 256)
    plt.title('R Histogram')
    plt.subplot(132)
    plt.bar(bincen, gh[0])
    plt.xlim(0, 256)
    plt.title('G Histogram')
    plt.subplot(133)
    plt.bar(bincen, bh[0])
    plt.xlim(0, 256)
    plt.title('B Histogram')
    fig.tight_layout()
else:
    print('Your function is returning None for at least one variable...')
```

显示结果如下图：



16. Spatial Binning of Color

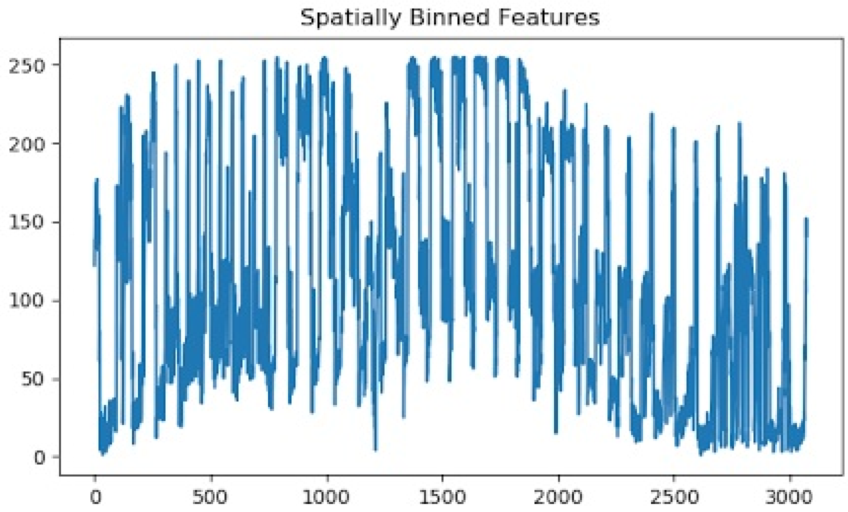
如果手动将包含汽车的模板图像缩小到 32*32 大小，肉眼仍然能够辨别出这是汽车，所以将图像缩小后，仍然保留了汽车的特征，而这个时候得到的 32*32*3=3072 个值就可以称为模板图像的特征向量。

本节的目的就是对颜色进行空域分级（Spatial Binning），得到图像的特征向量：

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Read in an image
# You can also read cutout2, 3, 4 etc. to see other examples
image = mpimg.imread('cutout1.jpg')

# Define a function to compute color histogram features
# Pass the color_space flag as 3-letter all caps string
# like 'HSV' or 'LUV' etc.
# KEEP IN MIND IF YOU DECIDE TO USE THIS FUNCTION LATER
# IN YOUR PROJECT THAT IF YOU READ THE IMAGE WITH
# cv2.imread() INSTEAD YOU START WITH BGR COLOR!
def bin_spatial(img, color_space='RGB', size=(32, 32)):
    # Convert image to new color space (if specified)
    if color_space != 'RGB':
        if color_space == 'HSV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
        elif color_space == 'LUV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2LUV)
        elif color_space == 'HLS':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
        elif color_space == 'YUV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
        elif color_space == 'YCrCb':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    else:
        feature_image = np.copy(img)
    # Use cv2.resize().ravel() to create the feature vector
    features = cv2.resize(feature_image, size).ravel()
    # Return the feature vector
```

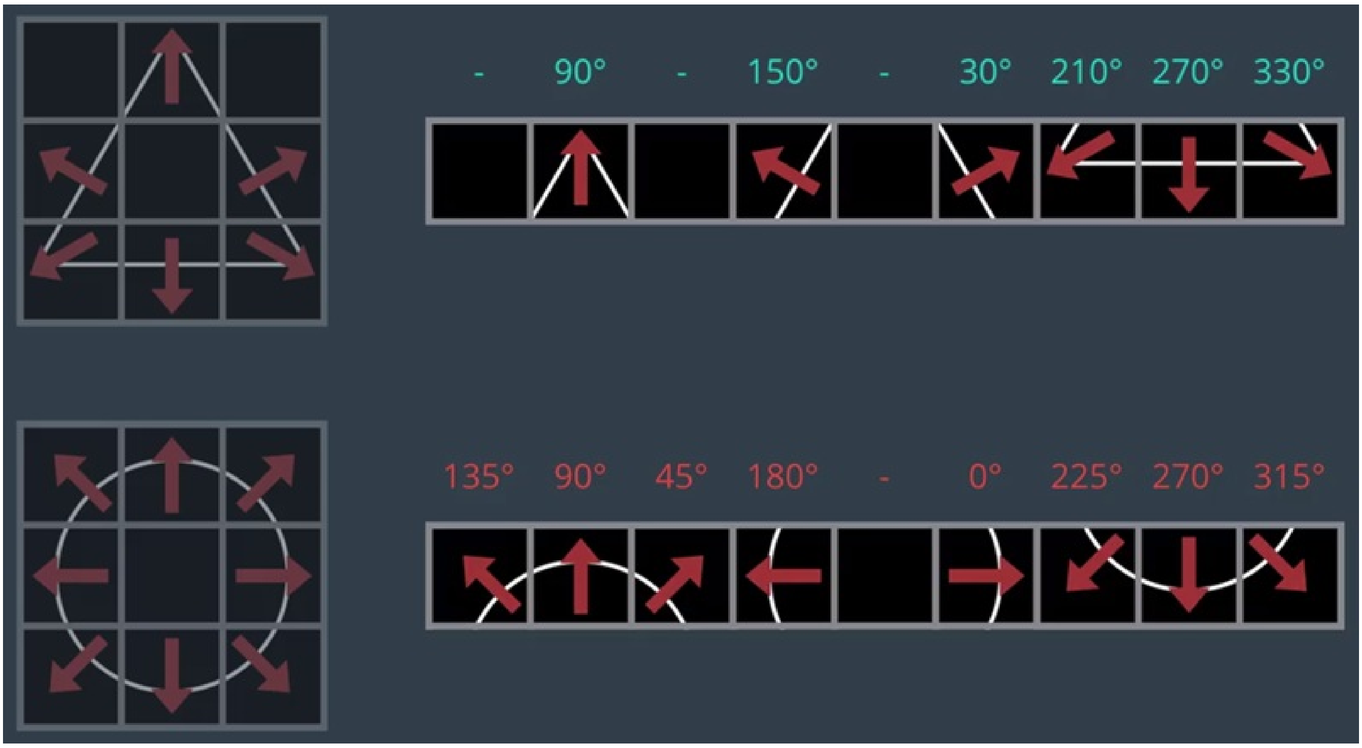
```
    return features

feature_vec = bin_spatial(image, color_space='RGB', size=(32, 32))

# Plot features
plt.plot(feature_vec)
plt.title('Spatially Binned Features')
```

得到的图形如下：



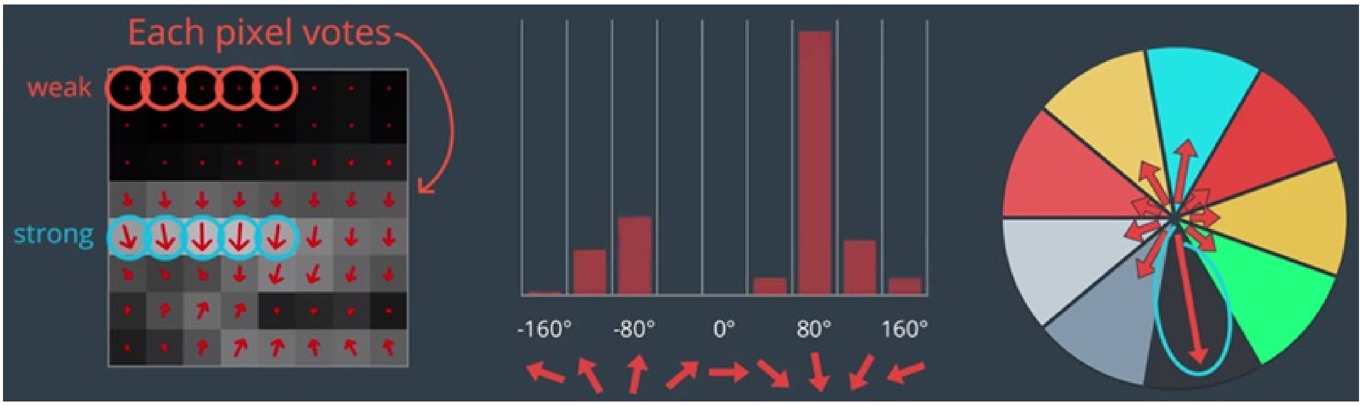
17. Gradient Features

除了颜色信息以外，梯度仍然能够反映物体的一些信息。基本想法是，将图形划分成网格，再将网格数据组合起来进行识别。

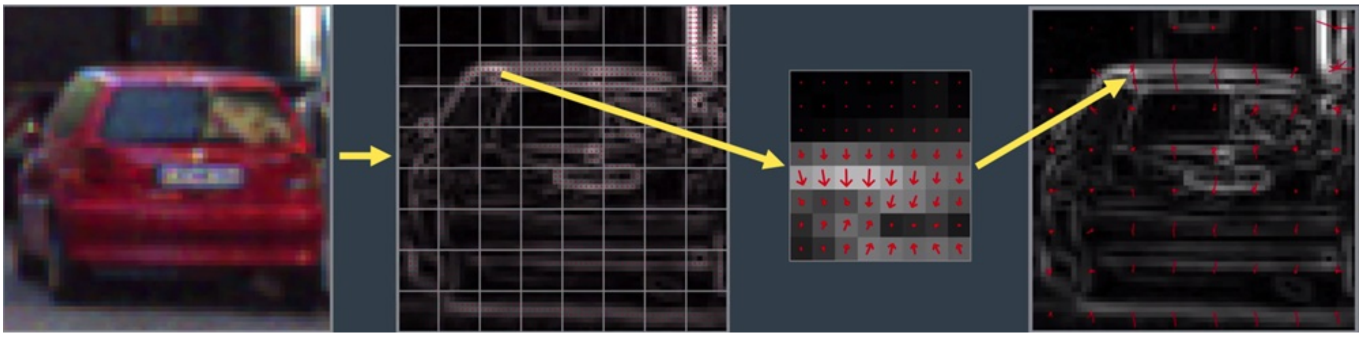


18. HOG Features

本节介绍的是方向梯度直方图（Histogram of Oriented Gradient, HOG）。把一个 64*64 的模板图像划分成 8*8 的网格，分别计算每个网格中的 64 个点的梯度大小和方向，如下图左图；中图为方向梯度直方图，它统计的是位于某个角度区间的点数量，但这个图有一个致命缺陷，如果某个梯度的大小非常大，它在直方图的权重也只有 1，也就是压根没考虑梯度的大小；而右图则综合考虑了梯度方向和梯度大小，它将每个区域的梯度大小相加，最长的一根则代表了这个 8*8 网格的特征。这就是 HOG 特征。



回顾一下这个变换的过程，首先是将原始图像转变成梯度图像，然后划分网格，对每个网格分别统计梯度的方向和大小，得到 HOG 特征，然后以 64 个 HOG 特征来刻画这幅梯度图像。过程如下：



19. Data Exploration

本节实现了一个 `data_look` 函数，它将测试集的数据信息打印出来。

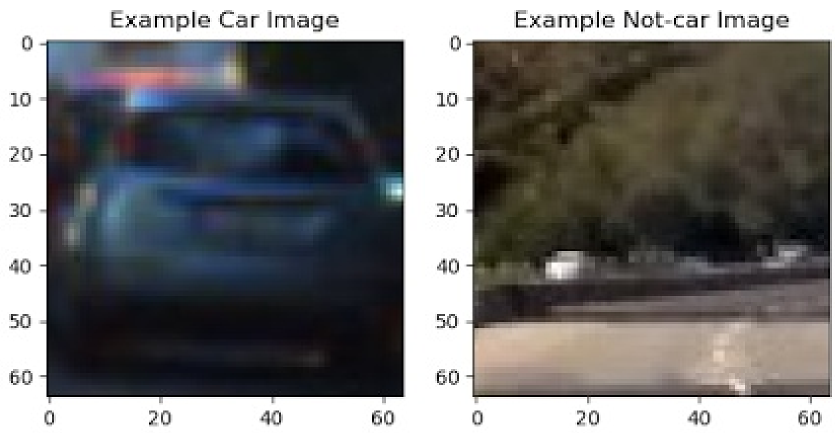
```python
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
#from skimage.feature import hog
#from skimage import color, exposure
# images are divided up into vehicles and non-vehicles

images = glob.glob('*.jpeg')
cars = []
notcars = []

for image in images:
    if 'image' in image or 'extra' in image:
        notcars.append(image)
    else:
        cars.append(image)

# Define a function to return some characteristics of the dataset
def data_look(car_list, notcar_list):
    data_dict = {}
    # Define a key in data_dict "n_cars" and store the number of car images
    data_dict["n_cars"] = len(car_list)
    # Define a key "n_notcars" and store the number of notcar images
```

```
    data_dict["n_notcars"] = len(notcar_list)
    # Read in a test image, either car or notcar
    example_img = mpimg.imread(car_list[0])
    # Define a key "image_shape" and store the test image shape 3-tuple
    data_dict["image_shape"] = example_img.shape
    # Define a key "data_type" and store the data type of the test image.
    data_dict["data_type"] = example_img.dtype
    # Return data_dict
    return data_dict

data_info = data_look(cars, notcars)

print('Your function returned a count of',
      data_info["n_cars"], ' cars and',
      data_info["n_notcars"], ' non-cars')
print('of size: ',data_info["image_shape"], ' and data type:',
      data_info["data_type"])
# Just for fun choose random car / not-car indices and plot example images
car_ind = np.random.randint(0, len(cars))
notcar_ind = np.random.randint(0, len(notcars))

# Read in car / not-car images
car_image = mpimg.imread(cars[car_ind])
notcar_image = mpimg.imread(notcars[notcar_ind])


# Plot the examples
fig = plt.figure()
plt.subplot(121)
plt.imshow(car_image)
plt.title('Example Car Image')
plt.subplot(122)
plt.imshow(notcar_image)
plt.title('Example Not-car Image')
```

输出为：

```
Your function returned a count of 1196  cars and 1199  no
```

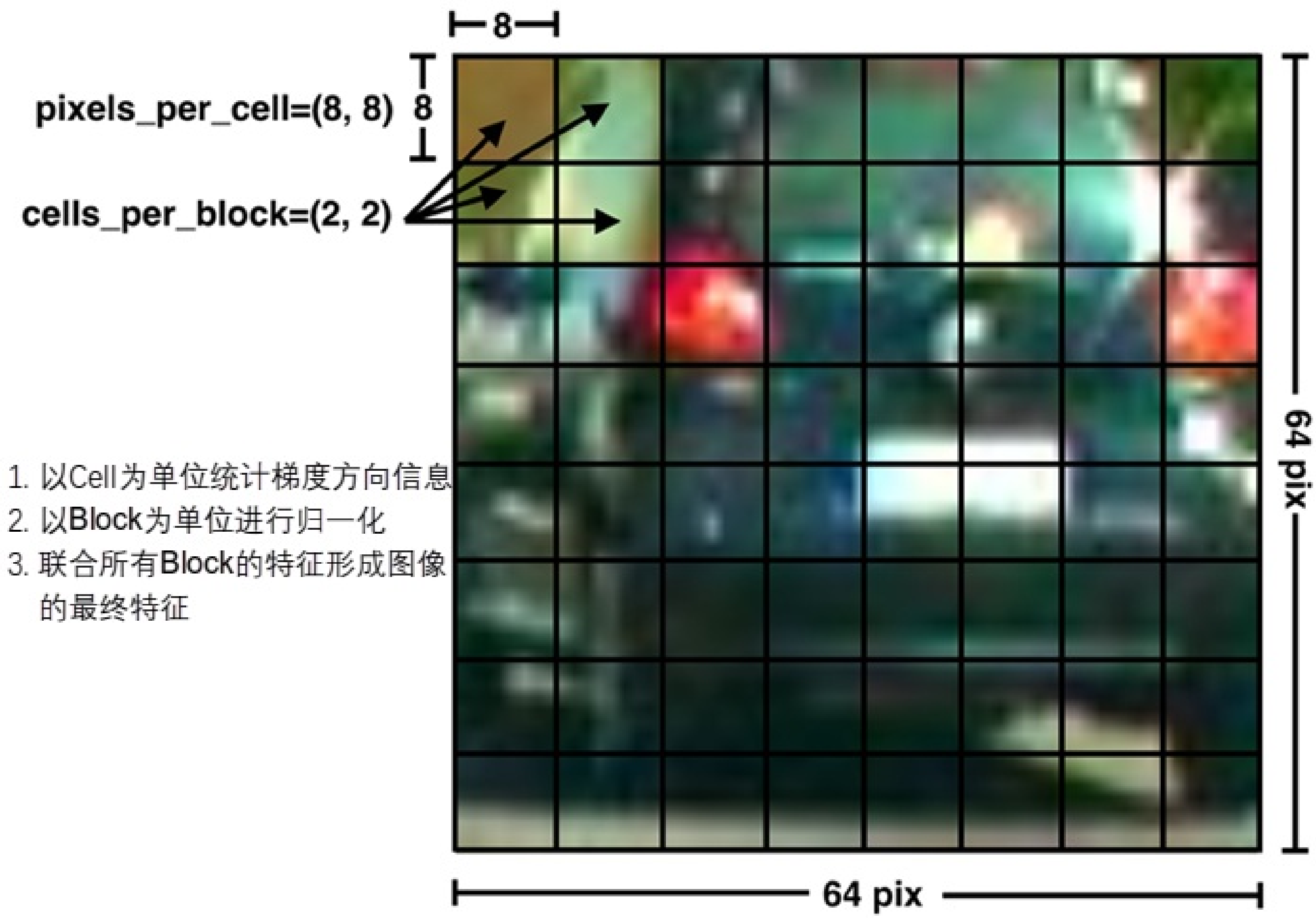
```
n-cars
of size:  (64, 64, 3)  and data type: uint8
```



20. scikit-image HOG

HOG 特征，histogram of oriented gradient，梯度方向直方图特征，作为提取基于梯度的特征，HOG 采用了统计的方式(直方图)进行提取。其基本思路是将图像局部的梯度统计特征拼接起来作为总特征。局部特征在这里指的是将图像划分为多个 Block，每个 Block 内的特征进行联合以形成最终的特征。具体来说：

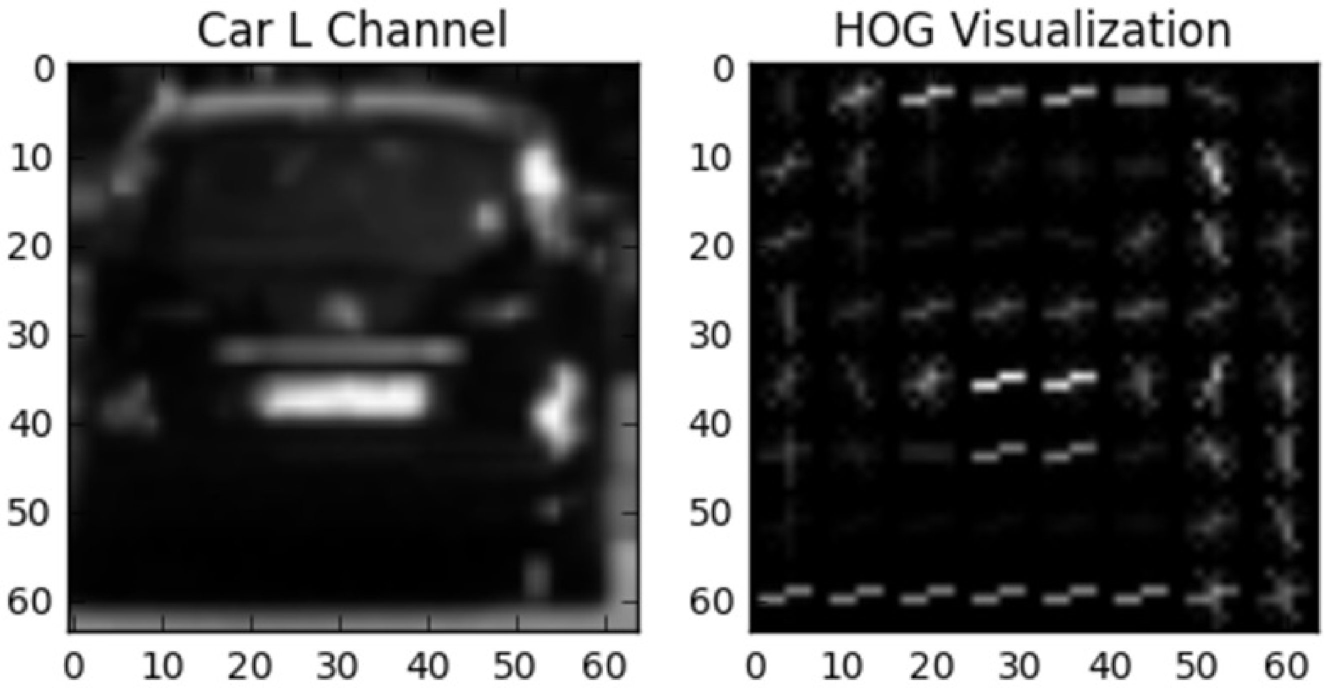
- 将图像分块，以 Block 为单位，每个 Block 以一定的步长在图像上滑动，以此来产生新的 Block
- Block 作为基本的特征提取单位，在其内部再次进行细分: 将 Block 划分为(一般是均匀划分) NxN 的小块，每个小块叫做 Cell
- Cell 是最基本的统计单元，在 Cell 内部，统计每个像素的梯度方向，并将它们映射到预设的 `orientations` 个方向的 bin 里面形成直方图
- 每个 Block 内部的所有 Cell 的梯度直方图联合起来并进行归一化处理（L1-norm，L2-Norm，L2-hys-norm，etc），据说这样可以使特征具有光照不变性。光照属于加性噪声，归一化之后会抵消掉关照变化对特征的影响
- 所有 Block 的特征联合起来，就是最终的 HOG 特征



"scikit-image" 包提供了 HOG 函数来进行 HOG 特征提取：

```
from skimage.feature import hog
pix_per_cell = 8
cell_per_block = 2
orient = 9

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),
    visualise=True,
    feature_vector=False)
```

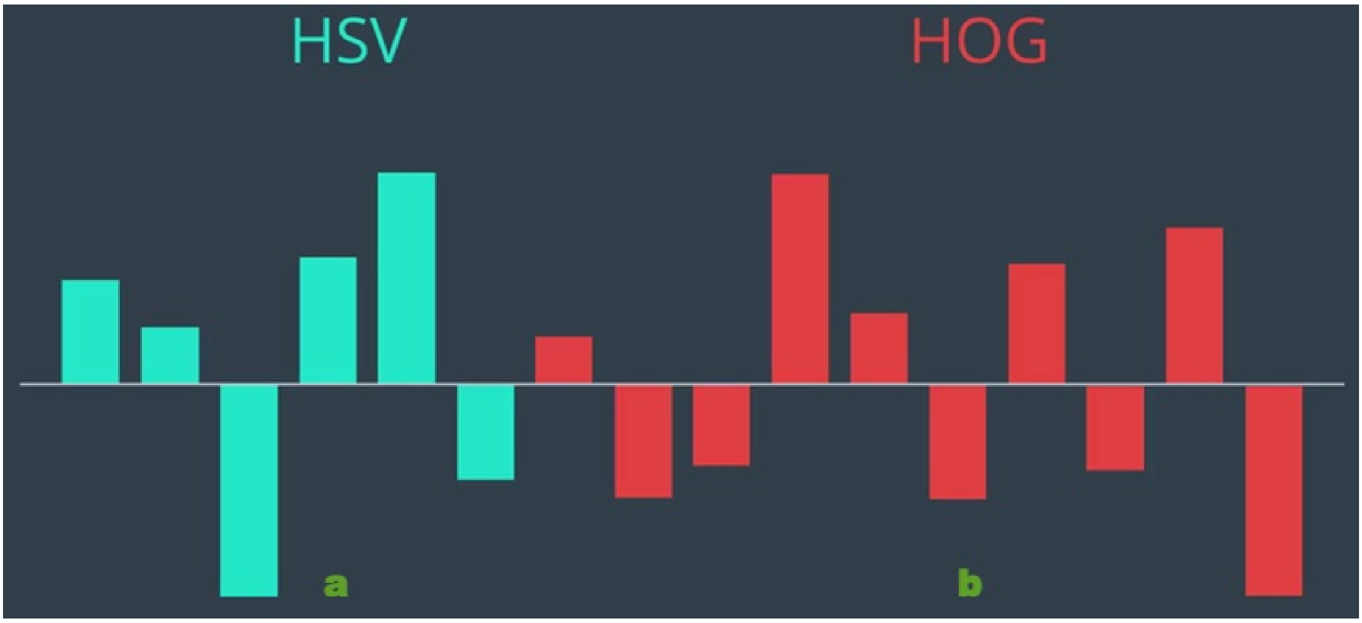
该例中，输入图像 `img` 必须是单色通道或者灰度图像，输出图像 `hog_image` 如下图右图所示：



特征向量 `features` 的大小应该为 $7*7*2*2*9=1764$，其中 $7*7$ 表示采样的 $7*7$ 个 Block，每个 Block 中有 $2*2$ 个 Cell，每个 Cell 计算 9 个梯度方向。`feature_vector=True` 表示自动将 1764 大小的 array 变成一个一维数组。

21. Combining Features

前面讲了这么多，无非就两件事情，第一件，得到颜色特征（学了两种，一种是 12 节的颜色直方图，一种是 13 节的空域分级），第二件，得到形状特征。这两组特征组合起来，就构成了图像的特征。比如说，通过 HSV 我们得到了颜色特征向量 `a`，通过 HOG 我们得到了形状特征向量 `b`，那么最简单的方法就是以 `a+b` 作为图像的最终特征。



但是需要注意，颜色特征向量和形状特征向量的大小是不一样的，它们代表不同的含义，不能简单相加，须各自归一化以后，才能相加。另外注意到，通常一种特征会比另外一种特征的数字多一些，必须要需要审视一下，其中是否包含冗余信息。比如使用决策树方法，将其中一些影响甚微的因素剔除掉。

22. Combine and Normalize Features

本例讲述了如何将两种颜色特征进行组合，一种是第 12 节提出的颜色直方图 `color_hist()`，一种是第 13 节提出的空域分级 `bin_spatial()`。

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.preprocessing import StandardScaler
import glob

# Define a function to compute binned color features
def bin_spatial(img, size=(32, 32)):
    # Use cv2.resize().ravel() to create the feature vector
    features = cv2.resize(img, size).ravel()
    # Return the feature vector
    return features

```

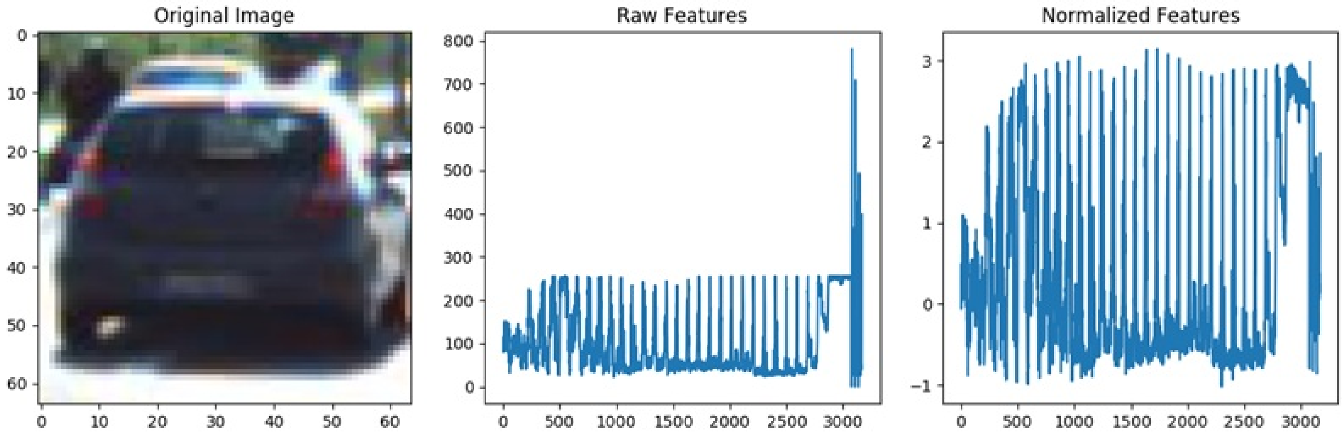
```python
# Define a function to compute color histogram features
def color_hist(img, nbins=32, bins_range=(0, 256)):
    # Compute the histogram of the color channels separately
    channel1_hist = np.histogram(img[:,:,0], bins=nbins, range=bins_range)
    channel2_hist = np.histogram(img[:,:,1], bins=nbins, range=bins_range)
    channel3_hist = np.histogram(img[:,:,2], bins=nbins, range=bins_range)
    # Concatenate the histograms into a single feature vector
    hist_features = np.concatenate((channel1_hist[0], channel2_hist[0], channel3_hist[0]))
    # Return the individual histograms, bin_centers and feature vector
    return hist_features

# Define a function to extract features from a list of images
# Have this function call bin_spatial() and color_hist()
def extract_features(imgs, cspace='RGB', spatial_size=(32, 32),
                        hist_bins=32, hist_range=(0, 256)):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        # Read in each one by one
        image = mpimg.imread(file)
        # apply color conversion if other than 'RGB'
        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)
        else: feature_image = np.copy(image)
        # Apply bin_spatial() to get spatial color features
        spatial_features = bin_spatial(feature_image, size=spatial_size)
        # Apply color_hist() also with a color space option now
        hist_features = color_hist(feature_image, nbins=hist_bins, bins_range=hist_range)
        # Append the new feature vector to the features list
        features.append(np.concatenate((spatial_features, hist_features)))
    # Return list of feature vectors
    return features

images = glob.glob('*.jpeg')
cars = []
notcars = []
for image in images:
    if 'image' in image or 'extra' in image:
        notcars.append(image)
    else:
        cars.append(image)

car_features = extract_features(
    cars,
    cspace='RGB',
    spatial_size=(32, 32),
    hist_bins=32,
    hist_range=(0, 256))
notcar_features = extract_features(
    notcars,
    cspace='RGB',
    spatial_size=(32, 32),
    hist_bins=32,
    hist_range=(0, 256))

if len(car_features) > 0:
    # Create an array stack of feature vectors
    X = np.vstack((car_features, notcar_features)).astype
```

```
(np.float64)
    # Fit a per-column scaler
    X_scaler = StandardScaler().fit(X)
    # Apply the scaler to X
    scaled_X = X_scaler.transform(X)
    car_ind = np.random.randint(0, len(cars))
    # Plot an example of raw and scaled features
    fig = plt.figure(figsize=(12,4))
    plt.subplot(131)
    plt.imshow(mpimg.imread(cars[car_ind]))
    plt.title('Original Image')
    plt.subplot(132)
    plt.plot(X[car_ind])
    plt.title('Raw Features')
    plt.subplot(133)
    plt.plot(scaled_X[car_ind])
    plt.title('Normalized Features')
    fig.tight_layout()
else:
    print('Your function only returns empty feature vectors...')
```

例子读入所有含有汽车的图片，一共 1196 张，因此特征向量大小为 $1196 * 3168$，其中颜色直方图特征向量大小为 $1196 * 32 * 3$，空域分级特征向量大小为 $1196 * 3072$，一共是 $1196 * 3168$；所有不含汽车的图片，一共 1125 张，因此特征向量大小为 $1125 * 3168$，其中颜色直方图特征向量大小为 $1125 * 32 * 3$，空域分级特征向量大小为 $1125 * 3072$，一共是 $1125 * 3168$。两者合起来，特征向量的大小一共是 $2321 * 3168$，即 `scaled_X.shape` 的值。

例子随机选取了一张含有汽车的图片，显示如下图左图；然后读取了该张图片对应的原始特征向量，它是一维的，长度为 3168，分布如中图；归一化处理后，特征向量分布如右图。



24. Labeled Data

对于机器学习算法，都知道 "garbage in, garbage out" 的原则，因此我们需要准备一个理想的数据集，否则就得不到理想的结果。准备数据集需要考虑以下几个要点：

- 输入数据都是图片，分为两类，一类是包含汽车的图片，并且标记为 "car"，另一类是不包含汽车的图片，并且标记为 "notcar"。应当尽量让这两个类型的图片样本集大小基本一致，否则分类器的预测结果很可能会偏向样本集多的那种类型
- 将样本划分为训练集和测试集。所有的图片必须事先处理成一样的大小，无论是训练集，还是测试集，亦或是一张待预测的新图片
- 保证训练集和测试集里面的图片是随机打乱的
- 归一化处理，使输入条件零均值、等方差

DATA PREPARATION STEP	PURPOSE
Prepare a balanced dataset, i.e., have as many positive as negative examples, or in the case of multi-class problems, roughly the same number of cases of each class.	To avoid having your algorithm simply classify everything as belonging to the majority class.
Random Shuffling of the data	To avoid problems due to ordering of the data
Splitting the data into a training and testing set	To avoid overfitting / improve generalization
Normalization of features, typically to zero mean and unit variance	To avoid individual features or sets of features dominating the response of your classifier

27. Parameter Tuning

视频中已经明确告诉我们，作业应该使用 SVM 作为模型。那么 SVM 模型需要注意哪些内容呢？

首先要注意 SVM 的超参数，包括 a kernel, a gamma value and a C value that minimize prediction error。SVM 调参时，必须谨记，对于线性 kernel，你只需要调试 C 参数；对于非线性 kernel，你可以调试 C 和 gamma 参数。

另一个要点是进行参数的交叉验证。Scikit-learn 提供了 `GridSearchCV()` 和 `RandomizedSearchCV()` 两种交叉验证的方法。下面我们演示使用 `GridSearchCV()` 进行交叉验证。

```
from sklearn import datasets, svm, grid_search

iris = datasets.load_iris()
parameters = {'kernel': ('linear', 'rbf'), 'C': [1, 10]}
svr = svm.SVC()
clf = grid_search.GridSearchCV(svr, parameters)
clf.fit(iris.data, iris.target)

print(clf.best_params_)
```

首先我们加载 Iris 数据集，后面我们可以通过 `iris.data` 得到数据，通过 `iris.target` 得到标签。然后我们用 `parameters` 变量指定了参数范围，最后会形成一个参数交叉组合：

-	C=1	C=10
Kernel='linear'	('linear', 1)	('linear', 10)
Kernel='rbf'	('rbf', 1)	('rbf', 10)

然后创建一个 SVM 模型，名为 `svr`。魔法发生在下一句：

```
clf = grid_search.GridSearchCV(svr, parameters)
```

The classifier is being created, we pass the algorithm (`svr`) and the dictionary of parameters to try (`parameters`) and it generates a grid of parameter combinations to try.

紧接着是第二个魔法：

```
clf.fit(iris.data, iris.target)
```

The `fit` function now tries all the parameter combinations, and returns a fitted classifier that's automatically tuned to the optimal parameter combination.

最后通过 `clf.best_params_` 来访问最优化的参数，结果为 `{'C': 1, 'kernel': 'linear'}`。

28. Color Classify

第 22 节讲述了如何将两种颜色特征进行组合，这一节使用到了这种技术。同时，本节选取了一个线性 SVM 模型，对输入图像进行建模：

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
import time
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
# NOTE: the next import is only valid
# for scikit-learn version <= 0.17
# if you are using scikit-learn >= 0.18 then use this:
# from sklearn.model_selection import train_test_split
from sklearn.cross_validation import train_test_split

# Define a function to compute binned color features
def bin_spatial(img, size=(32, 32)):
    # Use cv2.resize().ravel() to create the feature vector
    features = cv2.resize(img, size).ravel()
    # Return the feature vector
    return features

# Define a function to compute color histogram features
```

```
def color_hist(img, nbins=32, bins_range=(0, 256)):
    # Compute the histogram of the color channels separately
    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)
    # Concatenate the histograms into a single feature vector
    hist_features = np.concatenate((channel1_hist[0], channel2_hist[0], channel3_hist[0]))
    # Return the individual histograms, bin_centers and feature vector
    return hist_features

# Define a function to extract features from a list of images
# Have this function call bin_spatial() and color_hist()
def extract_features(imgs, cspace='RGB', spatial_size=(32, 32),
                        hist_bins=32, hist_range=(0, 256)):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        # Read in each one by one
        image = mpimg.imread(file)
        # apply color conversion if other than 'RGB'
        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.C
```

```
OLOR_RGB2YUV)
            else: feature_image = np.copy(image)
            # Apply bin_spatial() to get spatial color features
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            # Apply color_hist() also with a color space option now
            hist_features = color_hist(feature_image, nbins=hist_bins, bins_range=hist_range)
            # Append the new feature vector to the features list
            features.append(np.concatenate((spatial_features, hist_features)))
    # Return list of feature vectors
    return features


# Read in car and non-car images
images = glob.glob('*.jpeg')
cars = []
notcars = []
for image in images:
    if 'image' in image or 'extra' in image:
        notcars.append(image)
    else:
        cars.append(image)

# TODO play with these values to see how your classifier
# performs under different binning scenarios
spatial = 32
histbin = 32

car_features = extract_features(cars, cspace='RGB', spatial_size=(spatial, spatial),
                        hist_bins=histbin, hist_range=(0, 256))
notcar_features = extract_features(notcars, cspace='RGB', spatial_size=(spatial, spatial),
                        hist_bins=histbin, hist_range=(0, 256))

# Create an array stack of feature vectors
```

```
X = np.vstack((car_features, notcar_features)).astype(np.float64)
# Fit a per-column scaler
X_scaler = StandardScaler().fit(X)
# Apply the scaler to X
scaled_X = X_scaler.transform(X)

# Define the labels vector
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))


# Split up data into randomized training and test sets
rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(
    scaled_X, y, test_size=0.2, random_state=rand_state)

print('Using spatial binning of:',spatial,
    'and', histbin,'histogram bins')
print('Feature vector length:', len(X_train[0]))
# Use a linear SVC
svc = LinearSVC()
# Check the training time for the SVC
t=time.time()
svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to train SVC...')
# Check the score of the SVC
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
# Check the prediction time for a single sample
t=time.time()
n_predict = 10
print('My SVC predicts: ', svc.predict(X_test[0:n_predict]))
print('For these',n_predict, 'labels: ', y_test[0:n_predict])
t2 = time.time()
print(round(t2-t, 5), 'Seconds to predict', n_predict,'labels with SVC')
```

模型结果的精确度是比较高的，达到了 98.92%。

29. HOG Classify

前面一节考察的是图片的颜色特征，这一节考察形状特征。同样还是使用线性 SVM 模型，模型精度达到 95.5%，通过调参也许还可以增加精确度。

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
import time
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from skimage.feature import hog
# NOTE: the next import is only valid for scikit-learn version <= 0.17
# for scikit-learn >= 0.18 use:
# from sklearn.model_selection import train_test_split
from sklearn.cross_validation import train_test_split

# Define a function to return HOG features and visualization
def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                        vis=False, feature_vec=True):
    # Call with two outputs if vis==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,
                                  visualise=vis, feature_vector=feature_vec)
        return features, hog_image
    # Otherwise call with one output
    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,
                       visualise=vis, feature_vector=fea
```

```
ture_vec)
        return features

# Define a function to extract features from a list of images
# Have this function call bin_spatial() and color_hist()
def extract_features(imgs, cspace='RGB', orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel=0):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        # Read in each one by one
        image = mpimg.imread(file)
        # apply color conversion if other than 'RGB'
        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_RGB2YCrCb)
        else: feature_image = np.copy(image)

        # Call get_hog_features() with vis=False, feature_vec=True
        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,
```

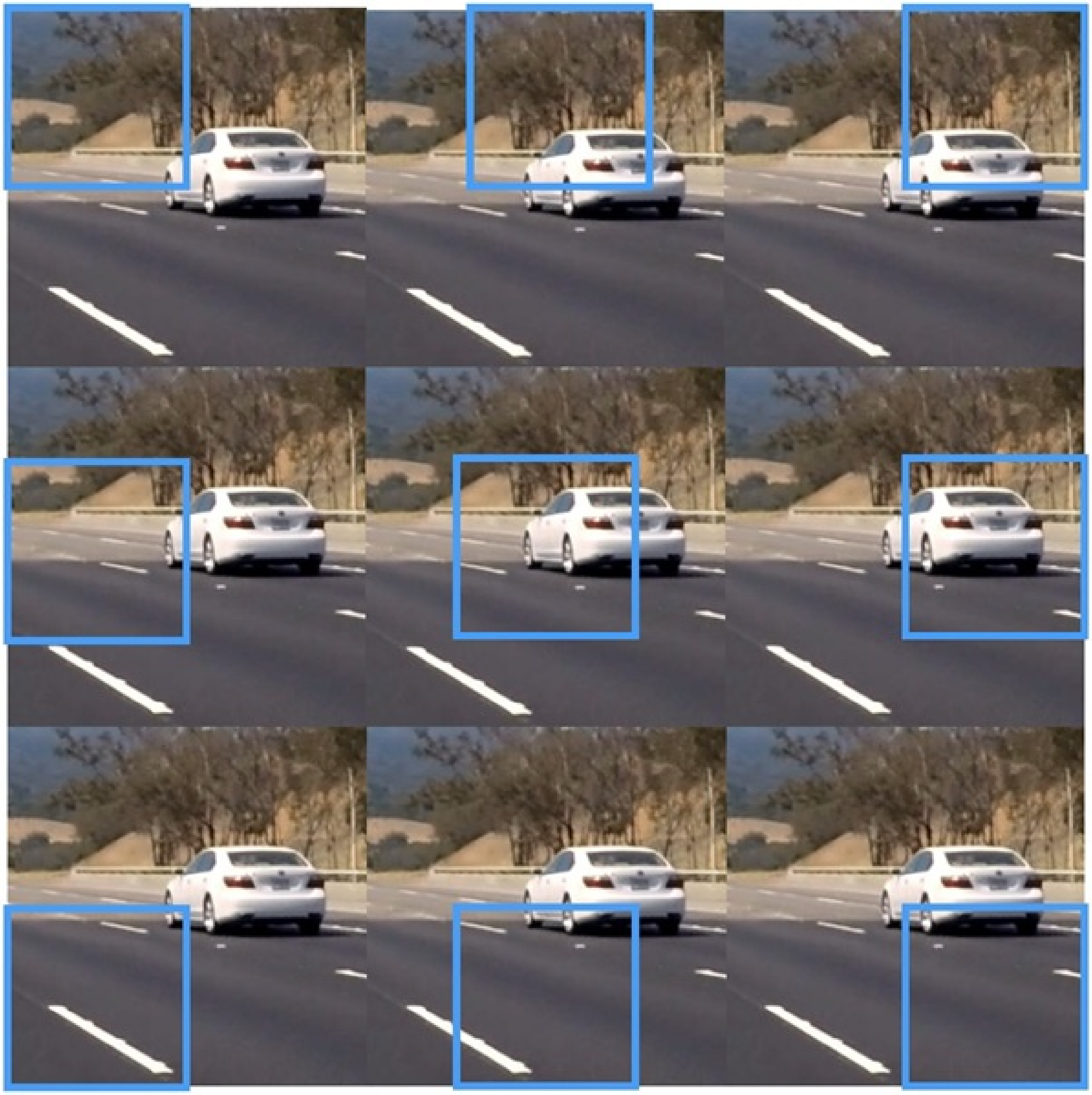
```
                                    vis=False, feature_vec=True))
            hog_features = np.ravel(hog_features)
        else:
            hog_features = get_hog_features(feature_image[:,:,hog_channel], orient,
                        pix_per_cell, cell_per_block, vis=False, feature_vec=True)
        # Append the new feature vector to the features list
        features.append(hog_features)
    # Return list of feature vectors
    return features


# Divide up into cars and notcars
images = glob.glob('*.jpeg')
cars = []
notcars = []
for image in images:
    if 'image' in image or 'extra' in image:
        notcars.append(image)
    else:
        cars.append(image)

# Reduce the sample size because HOG features are slow to compute
# The quiz evaluator times out after 13s of CPU time
sample_size = 500
cars = cars[0:sample_size]
notcars = notcars[0:sample_size]

### TODO: Tweak these parameters and see how the results change.
colorspace = 'RGB' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = 0 # Can be 0, 1, 2, or "ALL"

t=time.time()
car_features = extract_features(cars, cspace=colorspace,
```

```
orient=orient,
                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                        hog_channel=hog_channel)
notcar_features = extract_features(notcars, cspace=colorspace, orient=orient,
                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                        hog_channel=hog_channel)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to extract HOG features...')
# Create an array stack of feature vectors
X = np.vstack((car_features, notcar_features)).astype(np.float64)
# Fit a per-column scaler
X_scaler = StandardScaler().fit(X)
# Apply the scaler to X
scaled_X = X_scaler.transform(X)

# Define the labels vector
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))


# Split up data into randomized training and test sets
rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(
    scaled_X, y, test_size=0.2, random_state=rand_state)

print('Using:',orient,'orientations',pix_per_cell,
    'pixels per cell and', cell_per_block,'cells per block')
print('Feature vector length:', len(X_train[0]))
# Use a linear SVC
svc = LinearSVC()
# Check the training time for the SVC
t=time.time()
svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to train SVC...')
# Check the score of the SVC
print('Test Accuracy of SVC = ', round(svc.score(X_test,
```

```
y_test), 4))
# Check the prediction time for a single sample
t=time.time()
n_predict = 10
print('My SVC predicts: ', svc.predict(X_test[0:n_predict]))
print('For these',n_predict, 'labels: ', y_test[0:n_predict])
t2 = time.time()
print(round(t2-t, 5), 'Seconds to predict', n_predict,'labels with SVC')
```

32. Sliding Window Implementation

给定一张输入图片，我们如何去寻找图中的汽车呢？方法很简单，那就是设定一个固定的窗口，比如 64*64 大小，在原图中进行滑动，每次滑动 50% 的距离，这样就可以得到 9 个区域，判断这 9 个区域中是否有汽车即可。



随堂练习中，给出了 `slide_window` 的实现方法。注意参数 `x_start_stop` 是一个可读写的变量，默认值为 `[None, None]`，是一个 List，而不是 `(None, None)` 的 Tuple，因为程序中可能会修改该参数的值，Tuple 是不可以更改的。而参数 `xy_window` 则是一个只读变量，默认值为 `(64, 64)`，也可以传入其他值比如 `(128, 128)`，因此将其设置为 Tuple 类型使得数据更为安全。

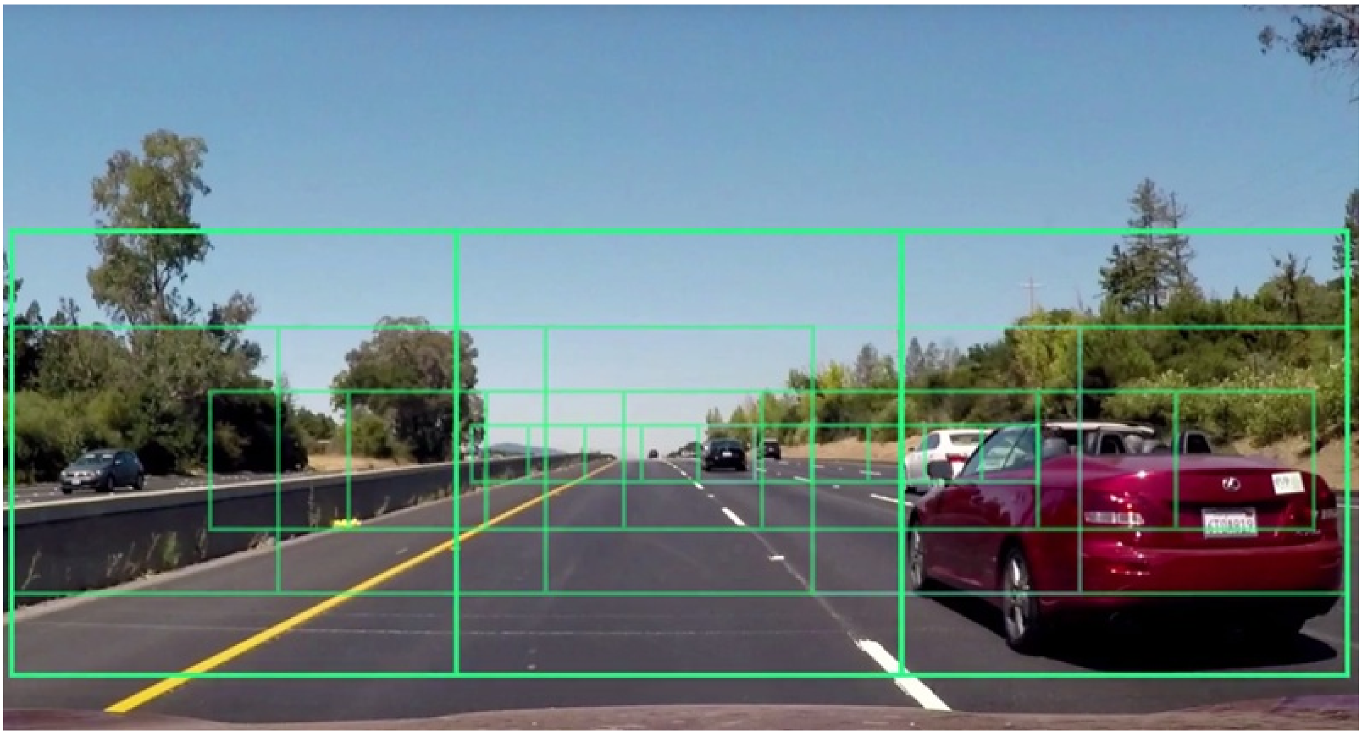
```
# Define a function that takes an image,
# start and stop positions in both x and y,
# window size (x and y dimensions),
# and overlap fraction (for both x and y)
def slide_window(img, x_start_stop=[None, None], y_start_stop=[None, None], xy_window=(64, 64), xy_overlap=(0.5, 0.5)):
```

```
    # If x and/or y start/stop positions not defined, set to image size
    if x_start_stop[0] == None:
        x_start_stop[0] = 0
    if x_start_stop[1] == None:
        x_start_stop[1] = img.shape[1]
    if y_start_stop[0] == None:
        y_start_stop[0] = 0
    if y_start_stop[1] == None:
        y_start_stop[1] = img.shape[0]
    # Compute the span of the region to be searched
    xspan = x_start_stop[1] - x_start_stop[0]
    yspan = y_start_stop[1] - y_start_stop[0]
    # Compute the number of pixels per step in x/y
    nx_pix_per_step = np.int(xy_window[0]*(1 - xy_overlap[0]))
    ny_pix_per_step = np.int(xy_window[1]*(1 - xy_overlap[1]))
    # Compute the number of windows in x/y
    nx_buffer = np.int(xy_window[0]*(xy_overlap[0]))
    ny_buffer = np.int(xy_window[1]*(xy_overlap[1]))
    nx_windows = np.int((xspan-nx_buffer)/nx_pix_per_step)
    ny_windows = np.int((yspan-ny_buffer)/ny_pix_per_step)
    # Initialize a list to append window positions to
    window_list = []
    # Loop through finding x and y window positions
    # Note: you could vectorize this step, but in practice
    # you'll be considering windows one by one with your
    # classifier, so looping makes sense
    for ys in range(ny_windows):
        for xs in range(nx_windows):
            # Calculate window position
            startx = xs*nx_pix_per_step + x_start_stop[0]
            endx = startx + xy_window[0]
            starty = ys*ny_pix_per_step + y_start_stop[0]
            endy = starty + xy_window[1]
            # Append window position to list
            window_list.append(((startx, starty), (endx, endy)))
    # Return the list of windows
```

```
        return window_list
```

33. Multi-scale Windows

因为车的大小是不固定的，所以就要用到多种尺寸的固定窗口。因为我们是检测同侧来车，所以我们仅关注图像的下半区域。



34. Search and Classify

这一节，我们就综合应用前面的知识，抽取输入样本的三种特征向量构成图像的特征向量，并且用 SVM 分类器建模。对一幅新的图片，应用模型进行预测。

```
import matplotlib.image as mpimg
import numpy as np
import cv2
from skimage.feature import hog
# Define a function to return HOG features and visualization
def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                        vis=False, feature_vec=True):
```

```
    # Call with two outputs if vis==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,
                                  visualise=vis, feature_vector=feature_vec)
        return features, hog_image
    # Otherwise call with one output
    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,
                       visualise=vis, feature_vector=feature_vec)
        return features

# Define a function to compute binned color features
def bin_spatial(img, size=(32, 32)):
    # Use cv2.resize().ravel() to create the feature vector
    features = cv2.resize(img, size).ravel()
    # Return the feature vector
    return features

# Define a function to compute color histogram features
# NEED TO CHANGE bins_range if reading .png files with mpimg!
def color_hist(img, nbins=32, bins_range=(0, 256)):
    # Compute the histogram of the color channels separately
    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,
```

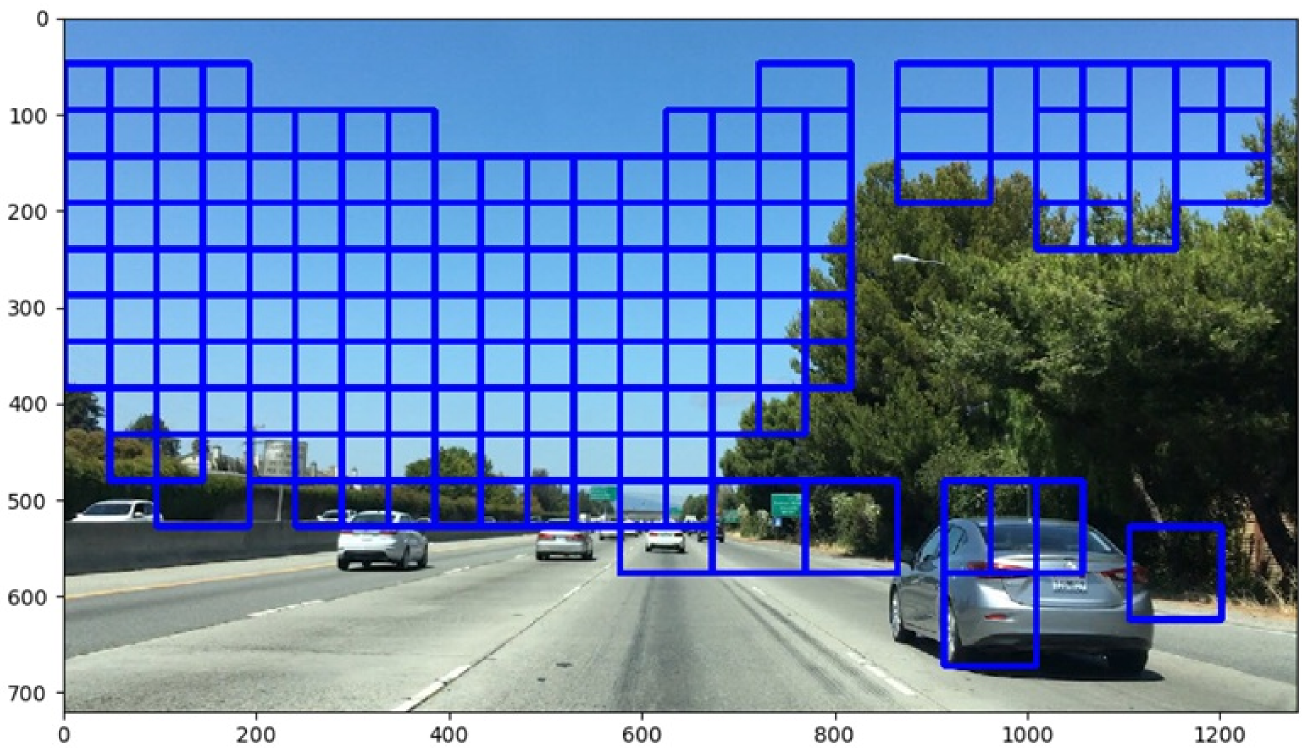
```
range=bins_range)
    # Concatenate the histograms into a single feature vector
    hist_features = np.concatenate((channel1_hist[0], channel2_hist[0], channel3_hist[0]))
    # Return the individual histograms, bin_centers and feature vector
    return hist_features

# Define a function to extract features from a list of images
# Have this function call bin_spatial() and color_hist()
def extract_features(imgs, color_space='RGB', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel=0,
                        spatial_feat=True, hist_feat=True, hog_feat=True):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        file_features = []
        # Read in each one by one
        image = mpimg.imread(file)
        # apply color conversion if other than 'RGB'
        if color_space != 'RGB':
            if color_space == 'HSV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            elif color_space == 'LUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
            elif color_space == 'HLS':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
            elif color_space == 'YUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
            elif color_space == 'YCrCb':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
```

```
        else: feature_image = np.copy(image)

        if spatial_feat == True:
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            file_features.append(spatial_features)
        if hist_feat == True:
            # Apply color_hist()
            hist_features = color_hist(feature_image, nbins=hist_bins)
            file_features.append(hist_features)
        if hog_feat == True:
        # Call get_hog_features() with vis=False, feature_vec=True
            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)
            # Append the new feature vector to the features list
            file_features.append(hog_features)
        features.append(np.concatenate(file_features))
    # Return list of feature vectors
    return features

# Define a function that takes an image,
# start and stop positions in both x and y,
# window size (x and y dimensions),
# and overlap fraction (for both x and y)
def slide_window(img, x_start_stop=[None, None], y_start_stop=[None, None],
```

```
                    xy_window=(64, 64), xy_overlap=(0.5, 0.5)):
    # If x and/or y start/stop positions not defined, set to image size
    if x_start_stop[0] == None:
        x_start_stop[0] = 0
    if x_start_stop[1] == None:
        x_start_stop[1] = img.shape[1]
    if y_start_stop[0] == None:
        y_start_stop[0] = 0
    if y_start_stop[1] == None:
        y_start_stop[1] = img.shape[0]
    # Compute the span of the region to be searched
    xspan = x_start_stop[1] - x_start_stop[0]
    yspan = y_start_stop[1] - y_start_stop[0]
    # Compute the number of pixels per step in x/y
    nx_pix_per_step = np.int(xy_window[0]*(1 - xy_overlap[0]))
    ny_pix_per_step = np.int(xy_window[1]*(1 - xy_overlap[1]))
    # Compute the number of windows in x/y
    nx_buffer = np.int(xy_window[0]*(xy_overlap[0]))
    ny_buffer = np.int(xy_window[1]*(xy_overlap[1]))
    nx_windows = np.int((xspan-nx_buffer)/nx_pix_per_step)
    ny_windows = np.int((yspan-ny_buffer)/ny_pix_per_step)
    # Initialize a list to append window positions to
    window_list = []
    # Loop through finding x and y window positions
    # Note: you could vectorize this step, but in practice
    # you'll be considering windows one by one with your
    # classifier, so looping makes sense
    for ys in range(ny_windows):
        for xs in range(nx_windows):
            # Calculate window position
            startx = xs*nx_pix_per_step + x_start_stop[0]
            endx = startx + xy_window[0]
            starty = ys*ny_pix_per_step + y_start_stop[0]
            endy = starty + xy_window[1]

            # Append window position to list
```

```python
                window_list.append(((startx, starty), (endx, endy)))
    # Return the list of windows
    return window_list

# Define a function to draw bounding boxes
def draw_boxes(img, bboxes, color=(0, 0, 255), thick=6):
    # Make a copy of the image
    imcopy = np.copy(img)
    # Iterate through the bounding boxes
    for bbox in bboxes:
        # Draw a rectangle given bbox coordinates
        cv2.rectangle(imcopy, bbox[0], bbox[1], color, thick)
    # Return the image copy with boxes drawn
    return imcopy
```

得到的结果如下图所示，可以看到，尽管模型的精度比较高（这一次跑了 98.9%），但仍然有非常多的区域被错误识别。



35. Hog Sub-sampling Window Search

前面一节的 `slide_window()` 函数识别出了大量的错误区域。这一节换一个函数，叫做 `find_cars()`，在一个指定的带型区域 [ystart, yend] 之间进行窗口滑动，并且将所有识别为包含汽车的窗口显示出来。

本节之前，每当我们移动一次窗口时，我们就提取了一次窗口下面的图像的 HOG 特征，窗口移动后再次重复提取。这个过程中存在在大量的重复工作。因此本节这个函数也避免了这种做法，它直接对整个目标区域进行 HOG 特征提取，根据窗口位置提取一部分特征出来计算即可。要实现这一点非常简单，只需要在 `skimage.feature.hog()` 函数中指定 `feature_vector=True` 即可。

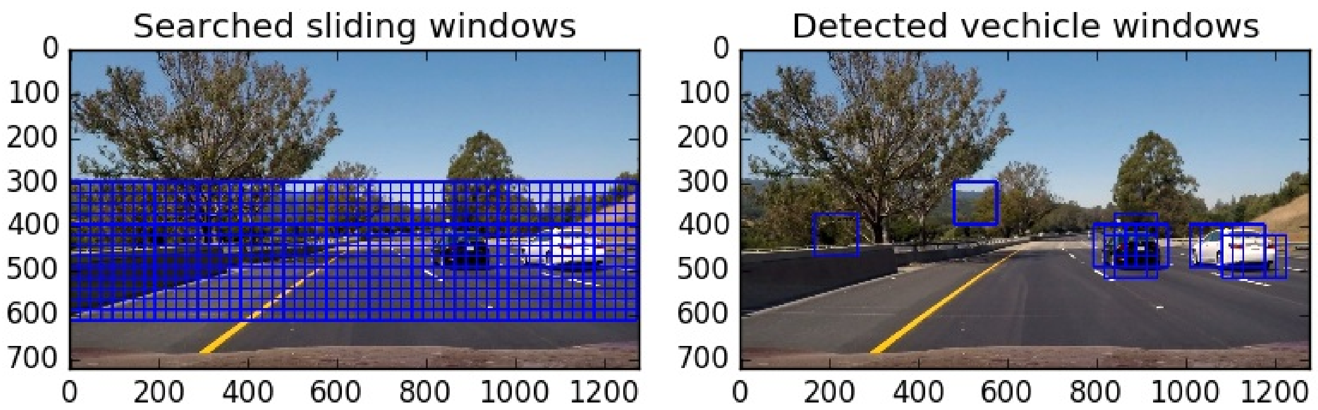
```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import pickle
import cv2
from lesson_functions import *

dist_pickle = pickle.load( open("svc_pickle.p", "rb" ) )
svc = dist_pickle["svc"]
X_scaler = dist_pickle["scaler"]
orient = dist_pickle["orient"]
pix_per_cell = dist_pickle["pix_per_cell"]
cell_per_block = dist_pickle["cell_per_block"]
spatial_size = dist_pickle["spatial_size"]
hist_bins = dist_pickle["hist_bins"]

img = mpimg.imread('test_image.jpg')
#img = mpimg.imread('bbox-example-image.jpg')

# Define a single function that can extract features using hog sub-sampling and make predictions
def find_cars(img, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins):

    draw_img = np.copy(img)
    img = img.astype(np.float32)/255

    img_tosearch = img[ystart:ystop,:,:]
    ctrans_tosearch = convert_color(img_tosearch, conv='R
```

```
GB2YCrCb')
    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1]/scale), np.int(imshape[0]/scale)))

    ch1 = ctrans_tosearch[:,:,0]
    ch2 = ctrans_tosearch[:,:,1]
    ch3 = ctrans_tosearch[:,:,2]

    # Define blocks and steps as above
    nxblocks = (ch1.shape[1] // pix_per_cell) - cell_per_block + 1
    nyblocks = (ch1.shape[0] // pix_per_cell) - cell_per_block + 1
    nfeat_per_block = orient*cell_per_block**2

    # 64 was the orginal sampling rate, with 8 cells and 8 pix per cell
    window = 64
    nblocks_per_window = (window // pix_per_cell) - cell_per_block + 1
    cells_per_step = 2  # Instead of overlap, define how many cells to step
    nxsteps = (nxblocks - nblocks_per_window) // cells_per_step
    nysteps = (nyblocks - nblocks_per_window) // cells_per_step

    # Compute individual channel HOG features for the entire image
    hog1 = get_hog_features(ch1, orient, pix_per_cell, cell_per_block, feature_vec=False)
    hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block, feature_vec=False)
    hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block, feature_vec=False)

    for xb in range(nxsteps):
        for yb in range(nysteps):
            ypos = yb*cells_per_step
            xpos = xb*cells_per_step
            # Extract HOG for this patch
```

```
            hog_feat1 = hog1[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_window].ravel()
            hog_feat2 = hog2[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_window].ravel()
            hog_feat3 = hog3[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_window].ravel()
            hog_features = np.hstack((hog_feat1, hog_feat2, hog_feat3))

            xleft = xpos*pix_per_cell
            ytop = ypos*pix_per_cell

            # Extract the image patch
            subimg = cv2.resize(ctrans_tosearch[ytop:ytop+window, xleft:xleft+window], (64,64))

            # Get color features
            spatial_features = bin_spatial(subimg, size=spatial_size)
            hist_features = color_hist(subimg, nbins=hist_bins)

            # Scale features and make a prediction
            test_features = X_scaler.transform(np.hstack((spatial_features, hist_features, hog_features)).reshape(1, -1))
            #test_features = X_scaler.transform(np.hstack((shape_feat, hist_feat)).reshape(1, -1))
            test_prediction = svc.predict(test_features)

            if test_prediction == 1:
                xbox_left = np.int(xleft*scale)
                ytop_draw = np.int(ytop*scale)
                win_draw = np.int(window*scale)
                cv2.rectangle(draw_img,(xbox_left, ytop_draw+ystart),(xbox_left+win_draw,ytop_draw+win_draw+ystart),(0,0,255),6)

    return draw_img

ystart = 400
ystop = 656
scale = 1.5
```

```
out_img = find_cars(img, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins)

plt.imshow(out_img)
```

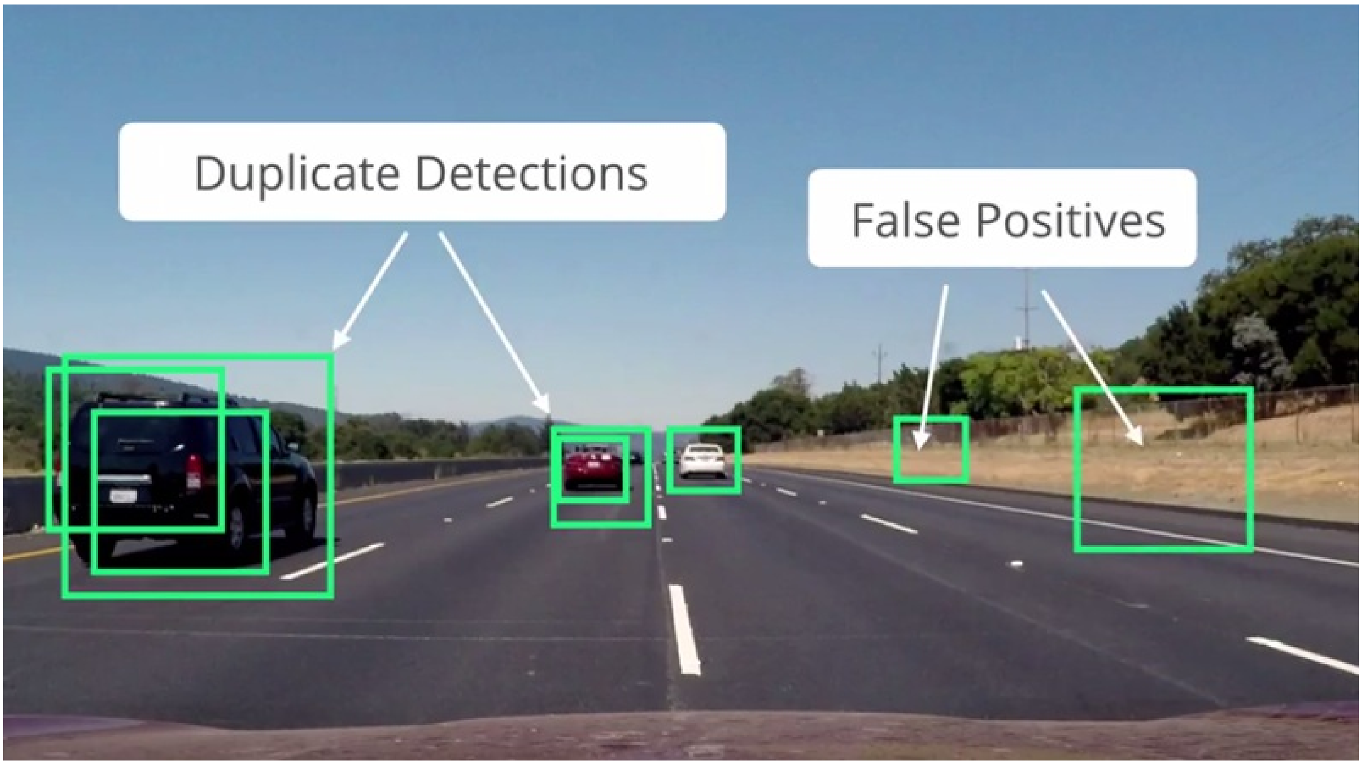
得到的结果如下图右图所示（代码和图像结果并没有对应，这个图像是从其他人的代码得到的，但思路一致）：



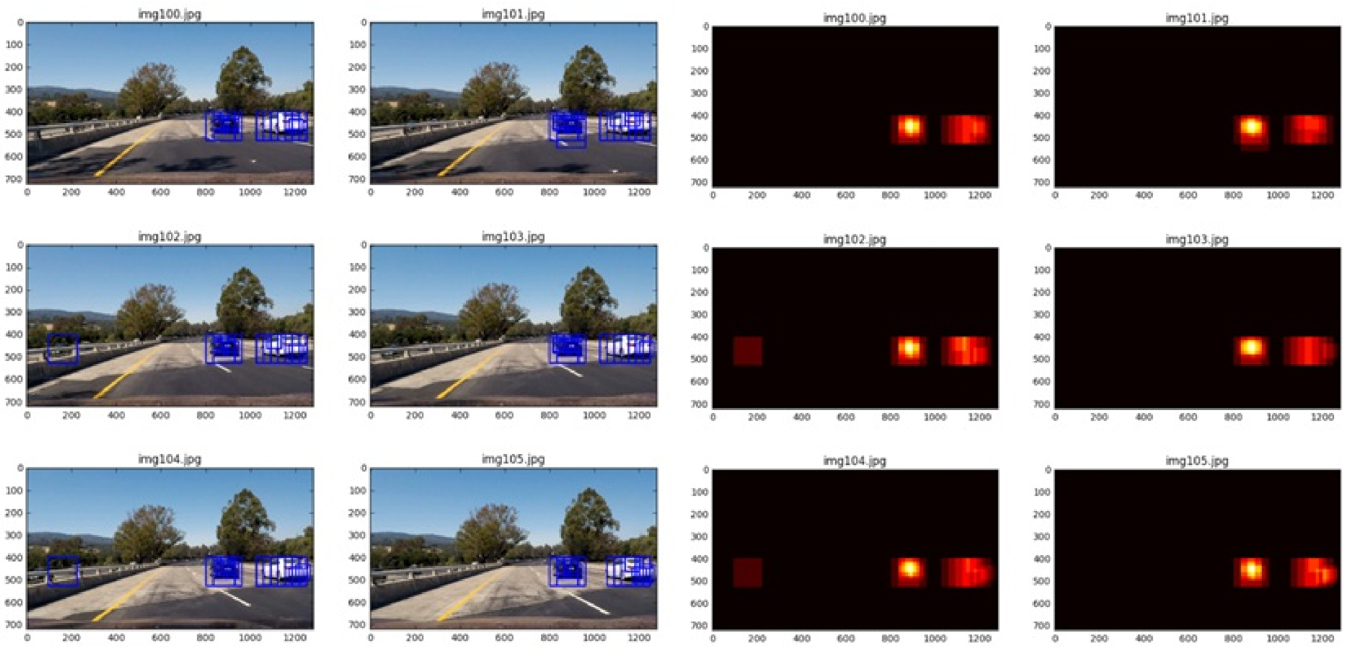
可以看出，仍然存在一些错误识别区域，需要进行误报（false positive）处理。

37. Multiple Detections & False Positives

本节需要处理两件事情，如下图所示，一是将重复识别的区域合并成一个整体，另一个是将误报区域删除。



这两件事情都可以通过引入热点图来解决。比如左边的六种情况，有一些识别出来没有误报，有一些有误报，每张图各自对应的热点图如右图所示。我们的任务就是将小于某个阈值的热点区域删除，并且将较热区域用一个矩形表示出来。

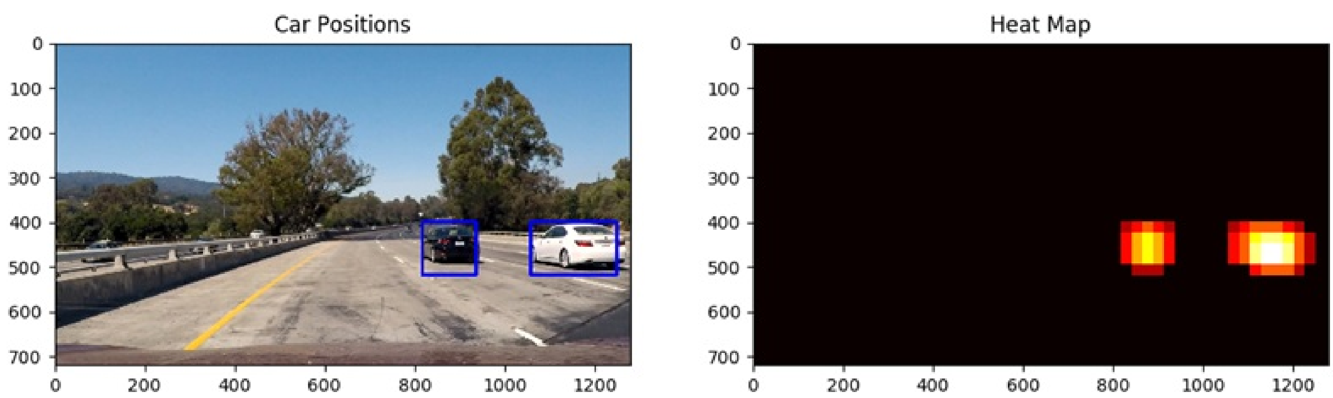



```python
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
```

```python
import pickle
import cv2
from scipy.ndimage.measurements import label

# Read in a pickle file with bboxes saved
# Each item in the "all_bboxes" list will contain a
# list of boxes for one of the images shown above
box_list = pickle.load( open( "bbox_pickle.p", "rb" ))

# Read in image similar to one shown above
image = mpimg.imread('test_image.jpg')
heat = np.zeros_like(image[:,:,0]).astype(np.float)

def add_heat(heatmap, bbox_list):
    # Iterate through list of bboxes
    for box in bbox_list:
        # Add += 1 for all pixels inside each bbox
        # Assuming each "box" takes the form ((x1, y1), (x2, y2))
        heatmap[box[0][23]:box[1][24], box[0][0]:box[1][0]] += 1

    # Return updated heatmap
    return heatmap# Iterate through list of bboxes

def apply_threshold(heatmap, threshold):
    # Zero out pixels below the threshold
    heatmap[heatmap <= threshold] = 0
    # Return thresholded map
    return heatmap

def draw_labeled_bboxes(img, labels):
    # Iterate through all detected cars
    for car_number in range(1, labels[1]+1):
        # Find pixels with each car_number label value
        nonzero = (labels[0] == car_number).nonzero()
        # Identify x and y values of those pixels
        nonzeroy = np.array(nonzero[0])
        nonzerox = np.array(nonzero[1])
        # Define a bounding box based on min/max x and y
        bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzeroy)))
        # Draw the box on the image
```

```python
        cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 6)
    # Return the image
    return img

# Add heat to each box in box list
heat = add_heat(heat,box_list)

# Apply threshold to help remove false positives
heat = apply_threshold(heat,1)

# Visualize the heatmap when displaying
heatmap = np.clip(heat, 0, 255)

# Find final boxes from heatmap using label function
labels = label(heatmap)
draw_img = draw_labeled_bboxes(np.copy(image), labels)

fig = plt.figure()
plt.subplot(121)
plt.imshow(draw_img)
plt.title('Car Positions')
plt.subplot(122)
plt.imshow(heatmap, cmap='hot')
plt.title('Heat Map')
fig.tight_layout()
```

结果为：



40. Tips and Tricks for the Project

本节讲到了一些注意事项：

- Extract HOG features just once for the entire region of interest in each full image / video frame。这可以通过设置 `skimage.feature.hog()` 函数的参数为 `feature_vec=False`。
- Make sure your images are scaled correctly. 随堂练习时，数据集是 JPG 格式的，而作业的数据集使用的是 PNG 格式。`matplotlib image` 和 `cv2.imread()` 的行为非常不一致，需要处处小心。
- Be sure to normalize your training data. 使用 `sklearn.preprocessing.StandardScaler()`。
- Random shuffling of data. 不知该如何处理。