# CS405 Machine Learning

## Lab #2 SVM

**Lab (100 points)**

In this lab, our goal is to write a software pipeline to identify vehicles in a video and apply a tight bounding box around vehicle detected. Following steps were implemented to achieve the goal:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a linear SVM classifier
- Additionally, apply a color transform and append binned color features, as well as histograms of color, to HOG feature vector
- Implement a sliding-window technique and use trained classifier to search for vehicles in images
- Run the pipeline on a video stream and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles
- Estimate a bounding box for vehicles detected

**Exercise**

There is dataset labelled as vehicles and non-vehicles and a video provided for this project. The two classes have roughly the same number of images resulting in a balanced dataset. However, there is one issue: it turns out that many of the images in the dataset were taken in quick succession resulting in very similar images. Consequently, randomly splitting data into train and test sets will result in highly correlated images between the two datasets.

- Please carefully splitting the dataset into training set and testing set

Scikit-image is an image processing toolbox. Skimage.hog() function was used to extract HOG features. There are a number of HOG parameters that can be tuned. Three of these parameters determine the size of the feature vector: number of pixels per cell pixels_per_cell, number of cells per block cells_per_block and number of orientations orientations. While a large feature vector in general carried more information, not all information may be useful for classification. From practical point of view, having too fine a grid or too many cells per block will reduce computational speed.

- Given that we have image of size 64 X 64, 8 pixels per cell, 2x2 cells per block, how many features we could get per color channel per orientation.
- There are a number of colorspaces: RGB, HSV, HLS, YUV, YCrCb and LUV, please plot HOG features using HSV colorspace with orientations=9, pizels_per_cell = (8,8), cells_per_block = (2,2)
- You need to quantify the differences for different colorspaces, please run the linear SVM classifier for each colorspace and select the colorspace that will have the highest test set accuracy.
- Similarly, please find the optimal number of orientations bins that will give the highest test set accuracy.

In addition, color histogram features and spatial features are mostly combined with HOG to improve the performance of the classifier.

```
# function to compute binned color features
def bin_spatial(img, size=(32, 32)):
    features = cv2.resize(img, size).ravel()
    return features

# function to compute color histogram features
def color_hist(img, nbins=32, bins_range=[(0, 256)]*3, visualize=False):
    channel_hist = [np.histogram(img[:,:,i], bins=nbins, range=bins_range[i]) for i in range(img.s
hape[2])]
    bin_edges = [hist[1] for hist in channel_hist]
    channel_hist = [hist[0] for hist in channel_hist]
    hist_features = np.concatenate(channel_hist)
    if visualize :
        return (channel_hist, bin_edges, hist_features)
    else :
        return hist_features
```

- Please plot color histograms using HSV colorspace for vehicles and non-vehicles datasets using 32 bins
- Please plot spatial features for vehicles and non-vehicles datasets as well
- How the test set accuracy changes while including color histogram features or spatial features, please show the results in a table.
- There are two parameters that can be tuned: number of spatial bins and number of histogram bins, please determine the optimal number of both features.
- Summarize the parameters (colorspace, orientations, pixel per cell, cells per block, color histogram, spatial bins) that give the highest test set accuracy.

Next, sliding-window technique is used to find cars in video frames using the trained classifier above.

- Please select proper sliding-window searching way (window size and searching region) to locate the car in each frame. Describe how you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?
  Tips: only a simple grid has the issue that in some windows the car was too small and in others the window only captured a small portion of the car. This results in lots of false negatives. You may implement a process for increasing the size of the windows on every iteration search of y axis. Because this can ensure that the search window is larger for the sections of the road that is close to the camera.

In order to reduce false positive and draw tighter bounding boxes, heatmap is used to filter false positives by setting a threshold.

- Please describe how do you use heatmap to reduce false positives and dupes
- Plot you test results (compute whole processing time, and draw the detecting results) in car detection in the video.

Some code examples are provided below:

```
def add_heat(heatmap, bbox_list):
    for box in bbox_list:
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1
    return heatmap

def apply_threshold(heatmap, threshold):
    heatmap[heatmap <= threshold] = 0
    return heatmap

def draw_labeled_bboxes(img, labels):
    for car_number in range(1, labels[1]+1):
        nonzero = (labels[0] == car_number).nonzero()
        nonzeroy = np.array(nonzero[0])
        nonzerox = np.array(nonzero[1])
        bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzeroy)))
        cv2.rectangle(img, bbox[0], bbox[1], (0,255,0), 6)
    return img
```

```
from scipy.ndimage.measurements import label


test_imgs = glob.glob("test_images/*.jpg")
nrows = len(test_imgs)
ncols = 3
fig, axes = plt.subplots(nrows, ncols, figsize=(5*ncols, 3*nrows))
for ax in axes.flatten():
    ax.axis('off')
for image, ax in zip(test_imgs, axes):
    img = cv2.imread(image)
    heat = np.zeros_like(img[:,:,0]).astype(np.float)
    out_img = np.copy(BGR2RGB(img))
    ax[0].imshow(BGR2RGB(img))
    for scale, i_color in zip(y_range.keys(), np.linspace(0,1,len(y_range))) :
        ystart,ystop = y_range[scale]
        bbox_list = find_cars(img, ystart, ystop, scale, clf, standard_scaler)
        out_img = draw_boxes(out_img, bbox_list,color)
        heat = add_heat(heat,bbox_list)
    ax[1].imshow(heat, cmap=plt.cm.inferno)
    heat = apply_threshold(heat,2)
    heatmap = np.clip(heat, 0, 255)
    labels = label(heatmap)
    draw_img = draw_labeled_bboxes(np.copy(BGR2RGB(img)), labels)
    ax[2].imshow(draw_img)
```

```
def exportFrames():
    """
    Code snippet to extract every image from the video and save as a JPG image.
    This was used to obtain frames from the video that caused the pipeline trouble.
    I found this code snippet on StackOverflow:
    http://stackoverflow.com/questions/33311153/python-extracting-and-saving-video-frames
    """
    vidcap = cv2.VideoCapture('project_video.mp4')
    success,image = vidcap.read()
    count = 0
    success = True
    while success:
        success,image = vidcap.read()
        cv2.imwrite("main_video_frames/frame%d.jpg" % count, image)     # save frame as JPEG file
        count += 1
    print("Complete!")
exportFrames()
```

```
def test_image_function(processing_function):
    images = []
    for i in range(6):
        count = random.randint(1,36)
        feature_image = cv2.imread("frames/frame%d.jpg" % count)
        input_img = cv2.cvtColor(feature_image, cv2.COLOR_BGR2RGB)
        result_img = processing_function(input_img)
        images.append(result_img)
    f, axarr = plt.subplots(6, 2, figsize=(30, 15))
    for i in range(len(images)):
        cur_img = images[i]
        axarr[i/2,i%2].imshow(cur_img)
    f.subplots_adjust(hspace=0)
    f.show()
```