Machine Learning Lab #2 SVM

Steps:

- 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

Questions

- *How many features we could get per color channel per orientation?*
- *Select the colorspace that will have the highest test set accuracy.*
- *Find the optimal number of orientations bins that will give the highest test set accuracy.*
- *Please determine the optimal number of spatial bins and number of histogram bins*.
- *How did you decide what scales to search and how much to overlap windows?*
- *How do you use heatmap to reduce false positives and dupes*

Process

Calculate the features per color channel per orientation

- imageSize (64,64)
- pizels_per_cell (8,8)
- cells_per_block (2,2)
- blockSize (16,16)
- **blockstep 16**
- features : $((64-16)/16+1)^2 * (2*2) = 64$

We could get **64** features per color channel per orientation.

Split the dataset into training set and testing set

Randomly splitting data into train and test sets will result in highly correlated images beween two datasets.To solve the problem, I choose about **20 percentage** testing images sets from different files.

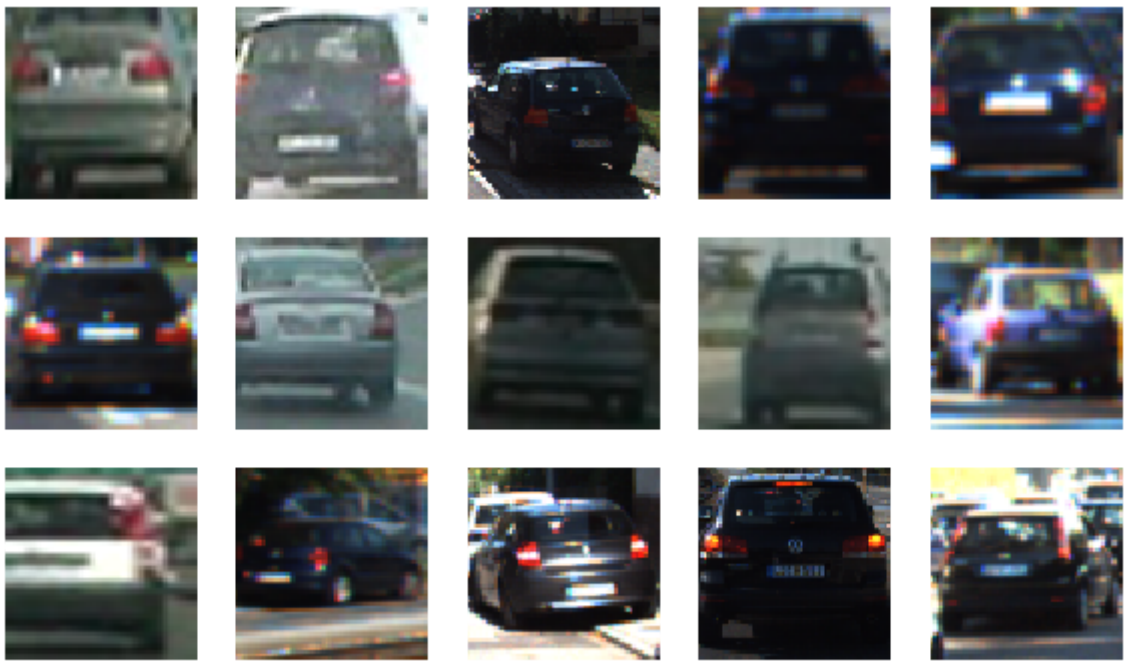
Vehicles

Using the indices of pictures from **different GTI four files**

```
split_manual("vehicles/GTI_Far/",(0, 142))
split_manual("vehicles/GTI_Left",(148, 344))
split_manual("vehicles/GTI_MiddleClose",(387, 494))
split_manual("vehicles/GTI_Right",(686, 940))
```

Using the *test_size* to randomly split KITTI.

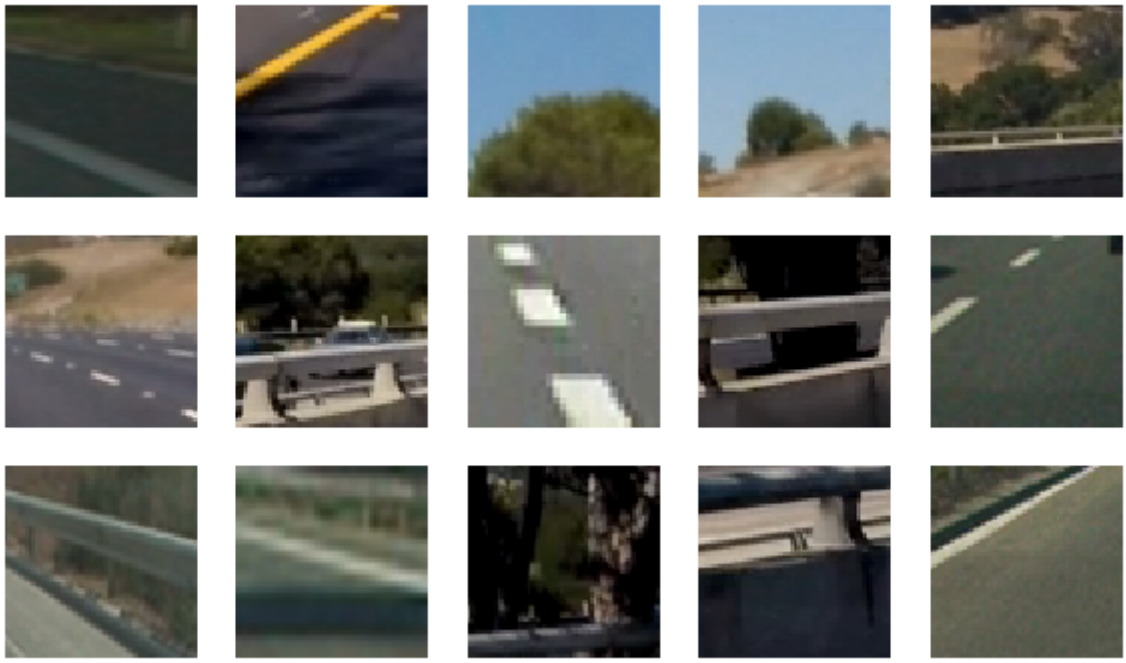
```
split_random("vehicles/KITTI_extracted/", test_size=test_size)
```



Non-vehicles

Using the *train_test_split* to randomly split non-vehicles

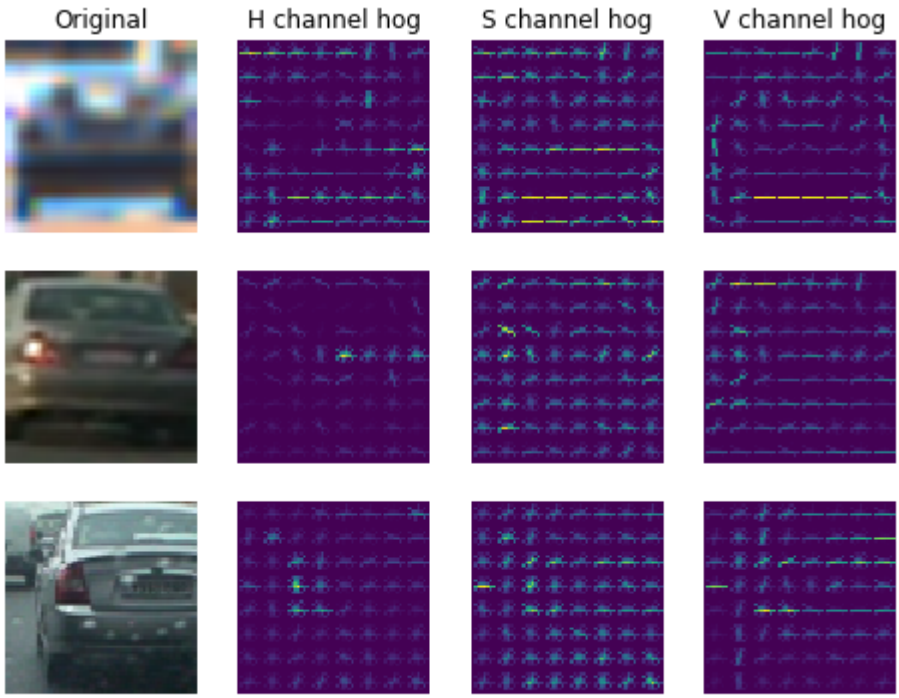
```
notcar_train_indices, notcar_test_indices =
train_test_split(range(len(notcar_imgs)), test_size=test_size,
random_state=0)
```



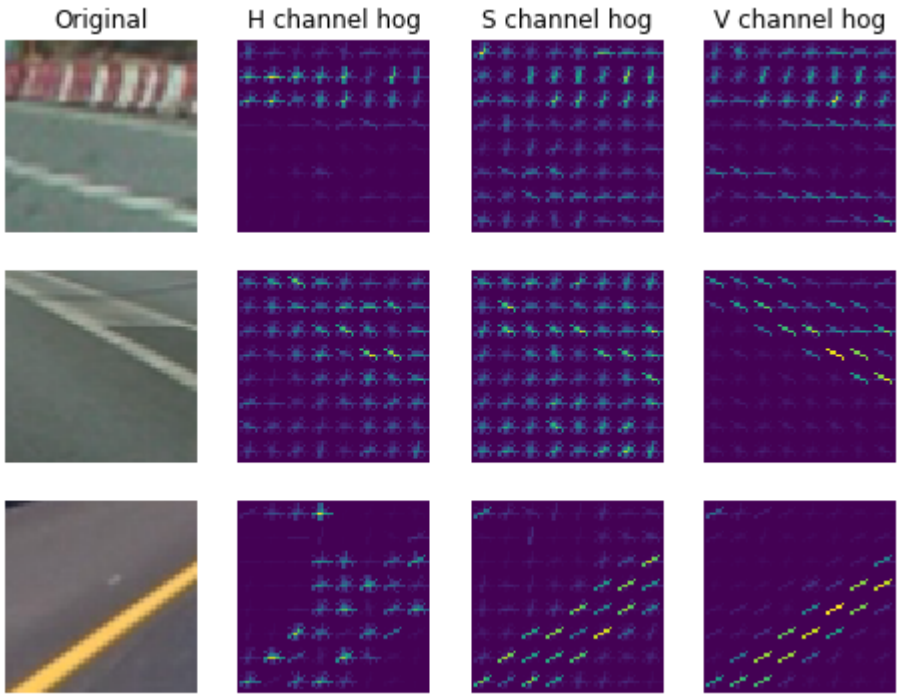
Plot HOG features using HSV colorspace

- `skimage.hog()` to extract HOG features.
- imageSize (64,64)
- pizels_per_cell (8,8)
- cells_per_block (2,2)
- orient = 9

Vehicles



Non-vehicles



Run the linear SVM classifier (colorspace and orientation bins)

Run linear SVM classifier for all colorspace RGB, HSV, HLS, YUV, YCrCb, LUV

- YCrCb colorspace had the highest test set accuracy
- All colorspaces had similar scores except RGB which had lower score than the rest.

colorspace	Accuracy
RGB	0.96325
LUV	0.98272
YUV	0.98354
HSV	0.98217
HSL	0.98217
YcrCb	0.98546

The colorspace which has the highest accuracy is **YcrCb**

Run linear SVM classifier for kinds of orientation bins[8,9,10,11]

orientations bins	Accuracy

orientations bins	Accuracy
8	0.98382
9	0.98546
10	0.98409
11	0.98629

The orientations bin which has the highest accuracy is **11**

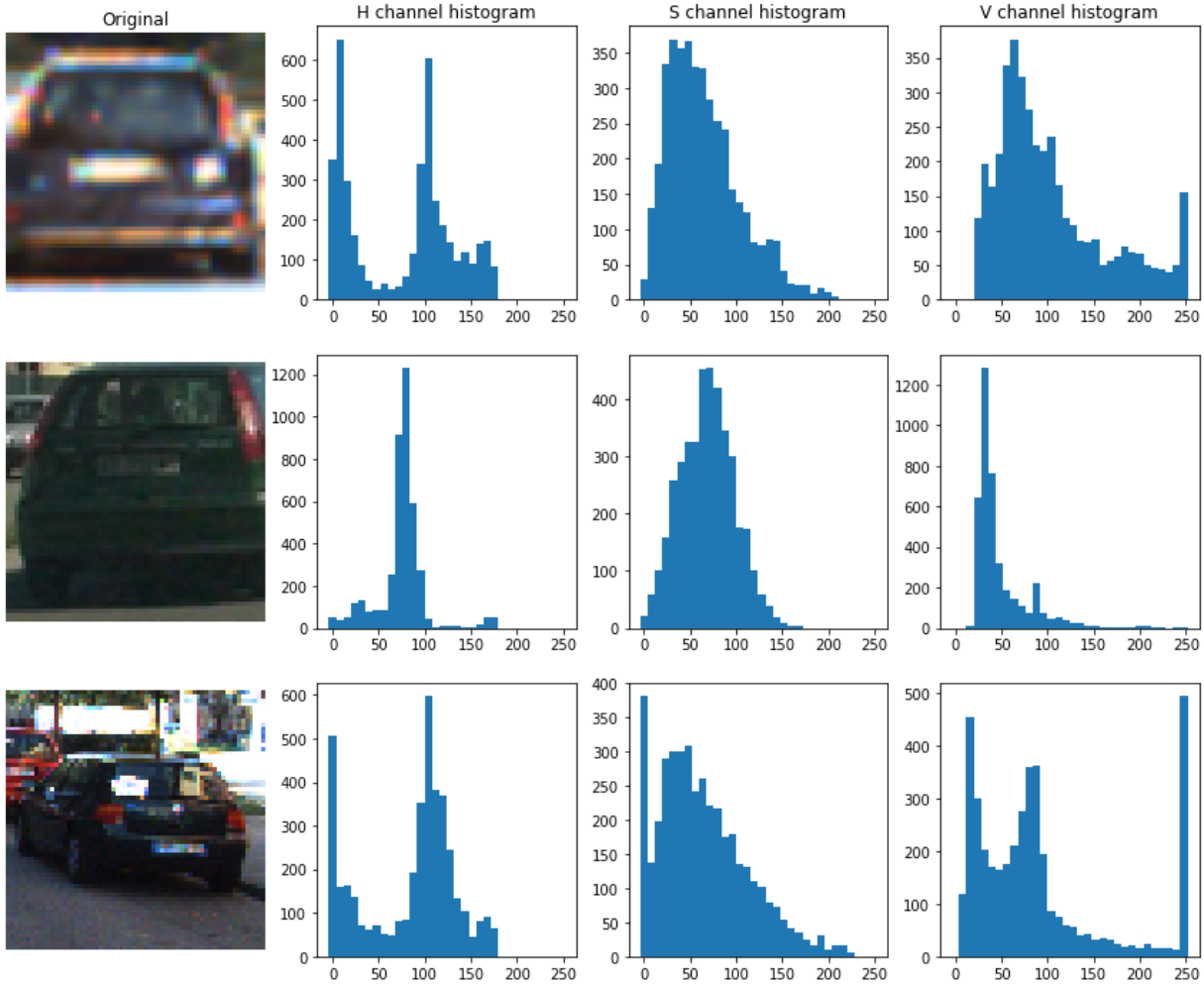
Color histogram features and spatial features

Two types of color features are considered :

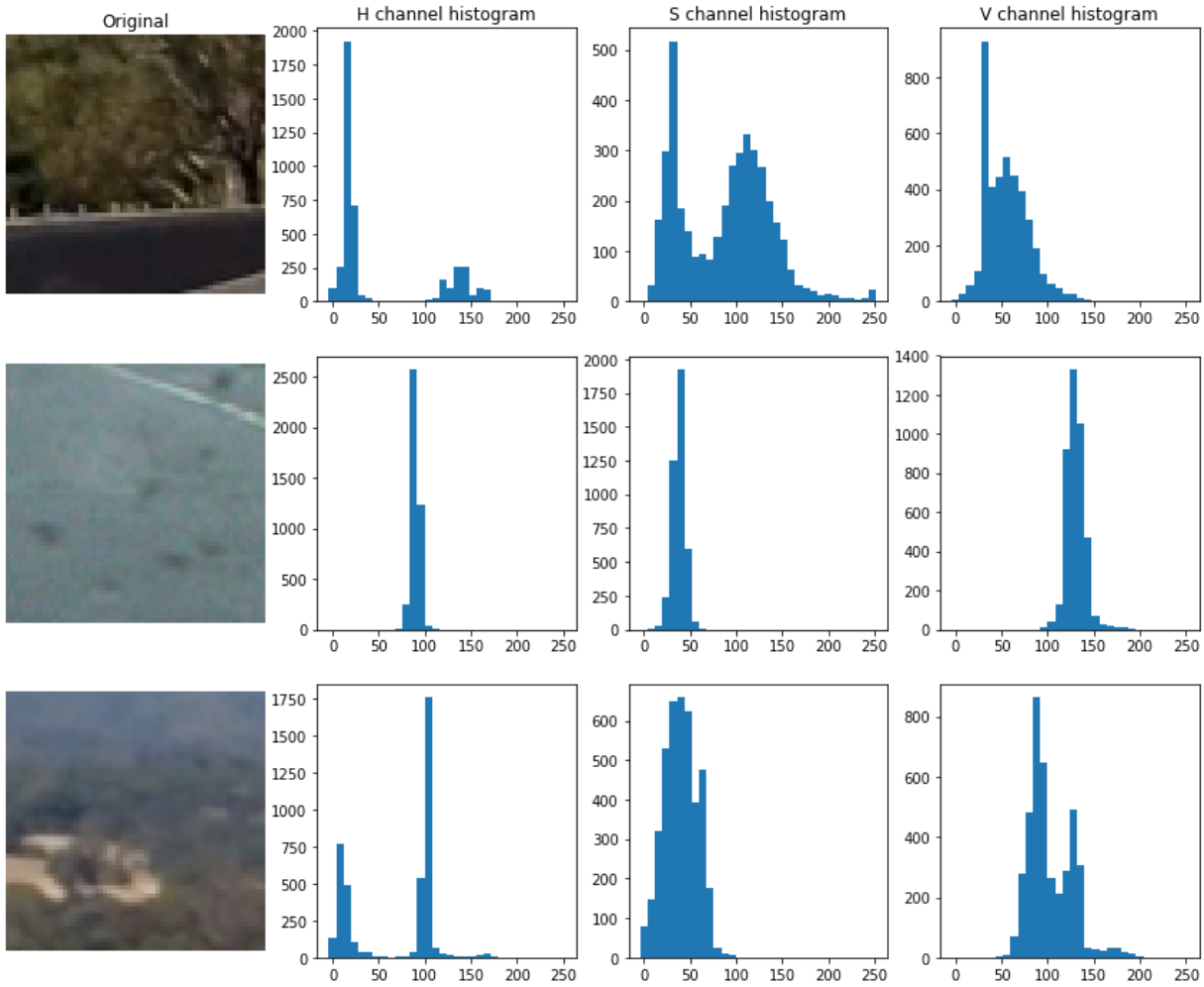
- Spatial binning of color channels
- Histogram of color channels

Here are some examples of color histograms using HSV colorspace for vehicles and non-vehicles datasets suing 32 bins :

Vehicles



Non-vehicles



- imageSize (32,32)
- pizels_per_cell (8,8)
- cells_per_block (2,2)

(spatial, hist)	Accuracy
(8, 16)	0.9605
(8, 32)	0.9679
(8, 64)	0.9679
(16, 16)	0.9515
(16, 32)	0.9487
(16, 64)	0.9545
(32, 16)	0.9438
(32, 32)	0.9506
(32, 64)	0.9619

The optimal number of spatial bins and number of histogram bins is **(8, 32)**

Linear SVM classifier

HOG, color histogram and spatial features were combined into a single 1-D feature vector using parameters determined before.

- colorspace : YCrCb
- orientations : 11
- pixels per cell : (8,8)
- cells per block : (2,2)
- color histogram bins :
- spatial bins = (32,32)

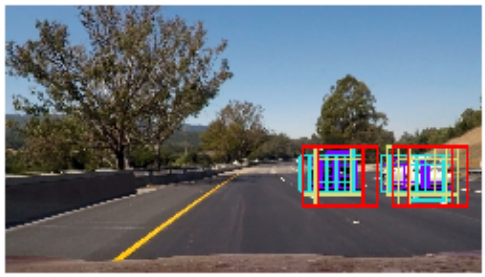
Features were normalized using *sklearn.preprocessing.StandardScaler()*. I explored colorspaces for the combined feature vector. YCrCb still gives the best overall test scores. Further, cross-validation was used to determine the penalty parameter ‘C’ in the linear SVM classifier. Test set accuracy of 99.2% was obtained.

Sliding Window

sliding-window technique is used to find cars in video frames. Four scales were chosen, each with their own search area :

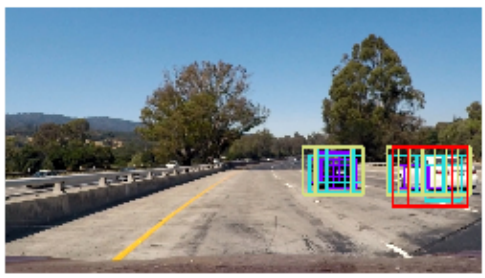
Scale	y-range
1.0	(380,508)
1.5	(380,572)
2.0	(380,636)
2.5	(380,700)

where default `scale=1.0` corresponds to window size of 64x64 pixels. Note that `y=380` is approximately the horizon position.



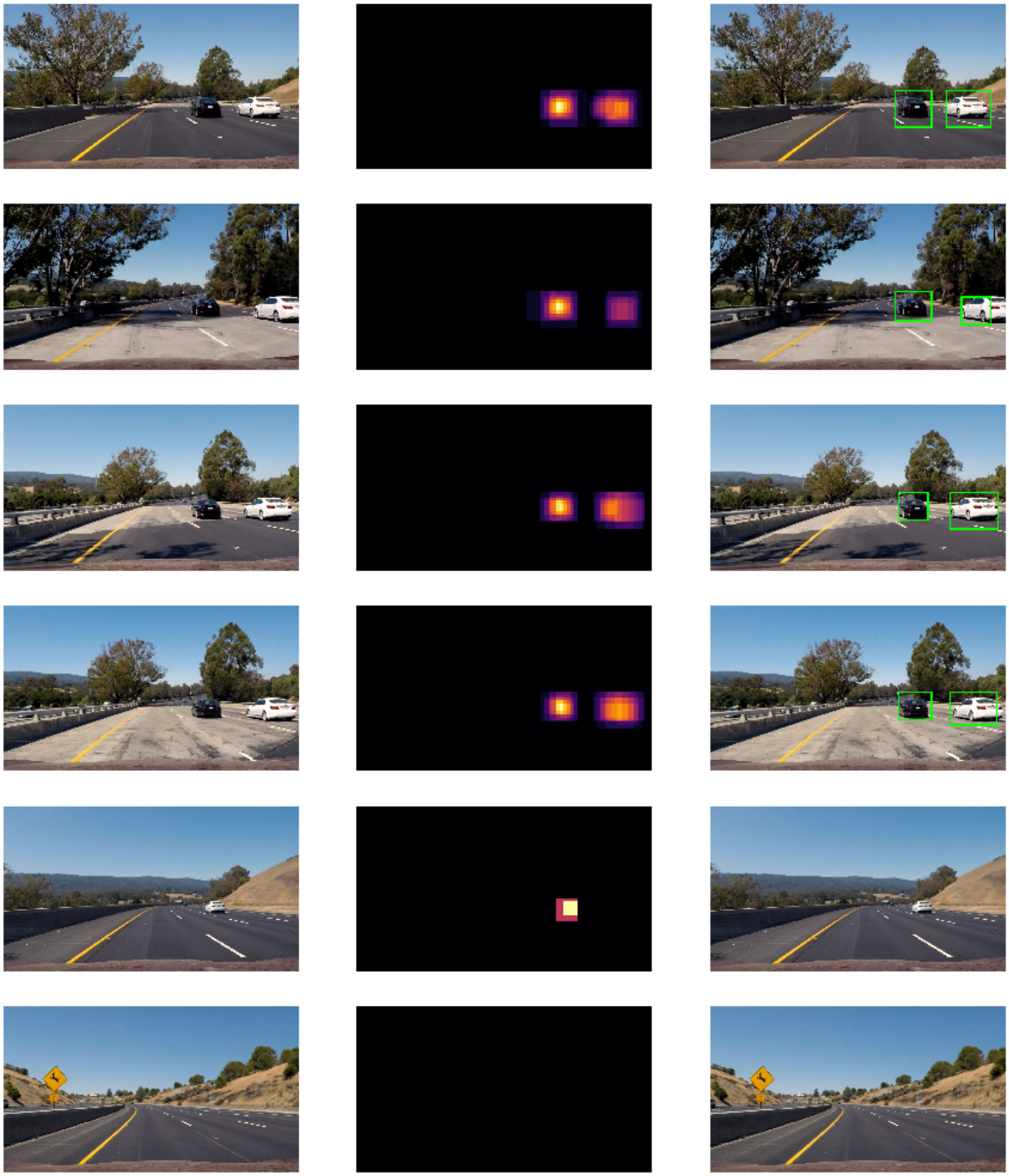






Heatmap

After positive detections in each image, the last step is to combine the detections to actually identify vehicle locations. For this step, I created a heatmap and then thresholded that map to identify vehicle positions.
`scipy.ndimage.measurements.label()` function was used to identify individual blobs in the heatmap.
constructed bounding boxes to cover the area of each blob detected. (`Project.ipynb` for implementation).



Vedio

To make the video implementation more robust against false positives, a slight modification of above procedure is implemeted :

- The sliding window technique is applied to each frame of the video and the resulting detections upto 25 frames (or 1 second) are stored in memory.
- Heatmap is constructed using the last 25 frames and a threshold=8 is applied.
- Using scipy.ndimage.measurements.label(), blobs in the heatmap from last step are identified as vehicles and bounding boxes drawn.
- Updates to the heatmap are performed every 2 frames. If a vehicle was detected in previous update, a small bias in the heatmap is applied at the positions where vehicles were last detected.
  To accomplish the above task, I have written a class vehicle detection.

Reference

- https://github.com/jessicayung/self-driving-car-nd/tree/master/p5-vehicle-detection (https://github.com/jessicayung/self-driving-car-nd/tree/master/p5-vehicle-detection)
- https://blog.csdn.net/qrlhl/article/details/60883604 (https://blog.csdn.net/qrlhl/article/details/60883604)
- https://medium.com/@ksakmann/vehicle-detection-and-tracking-using-hog-features-svm-vs-yolo-73e1ccb35866?spm=a2c4e.11153940.blogcont71662.17.84412198rgfnoa (https://medium.com/@ksakmann/vehicle-detection-and-tracking-using-hog-features-svm-vs-yolo-73e1ccb35866?spm=a2c4e.11153940.blogcont71662.17.84412198rgfnoa)
- https://github.com/tatsuyah/vehicle-detection (https://github.com/tatsuyah/vehicle-detection)

Machine Learning Pre Lab #3 TensorFlow

Run linear_regression.py

```
➜  TF_beginner python3 linear_regression.py
```

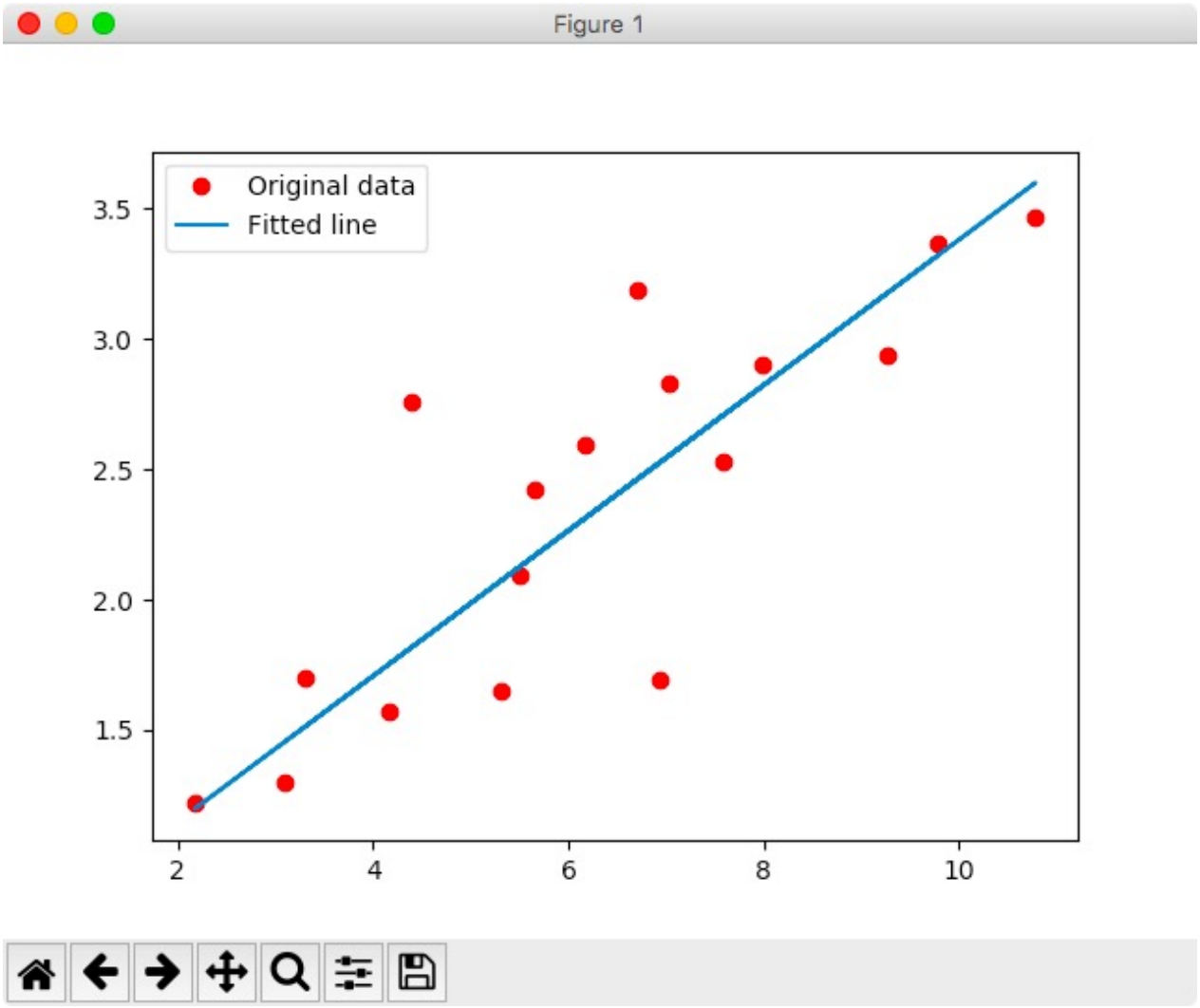


Figure 1
Original data
Fitted line
3.5
3.0
2.5
2.0
1.5
2
4
6
8
10

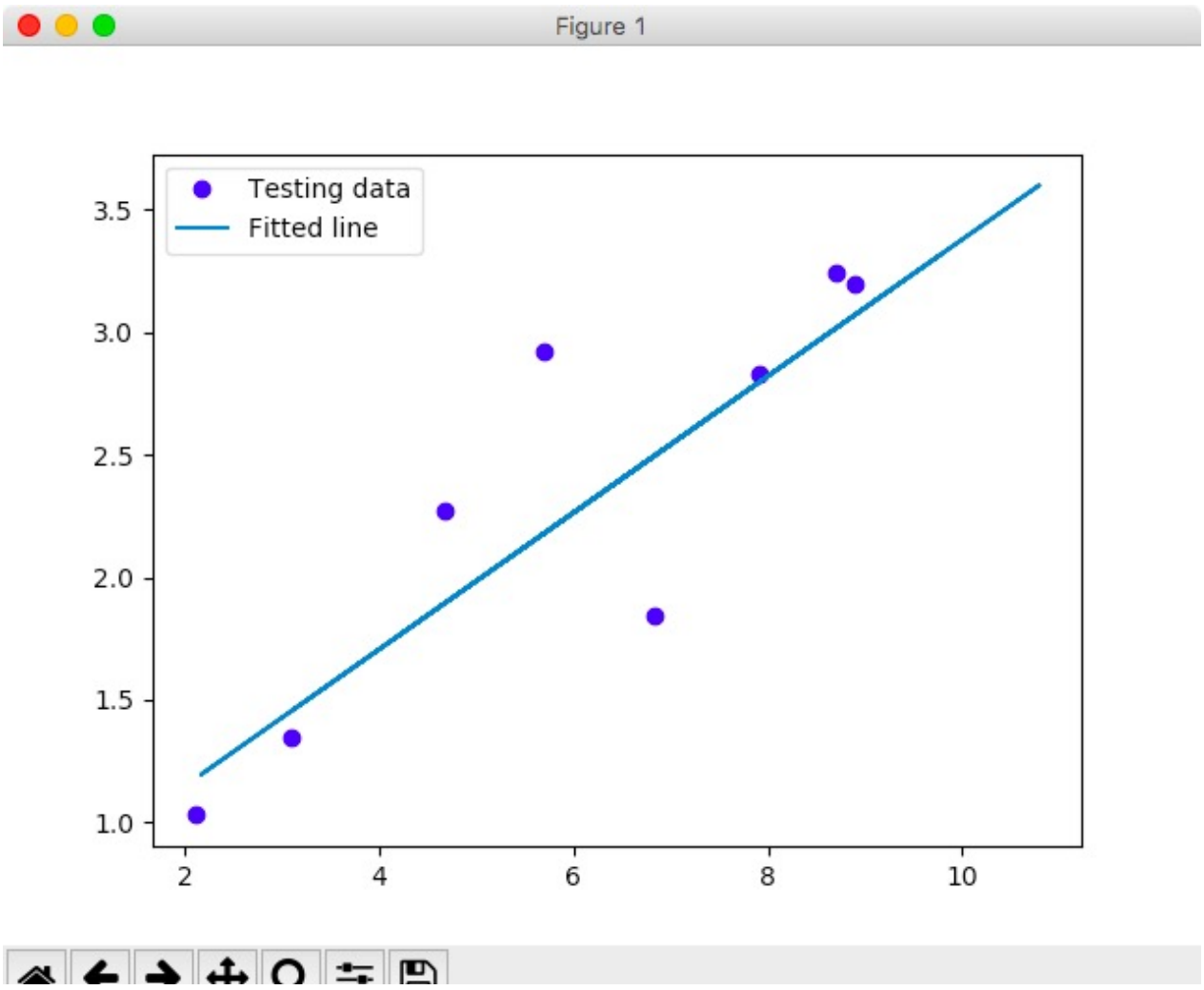


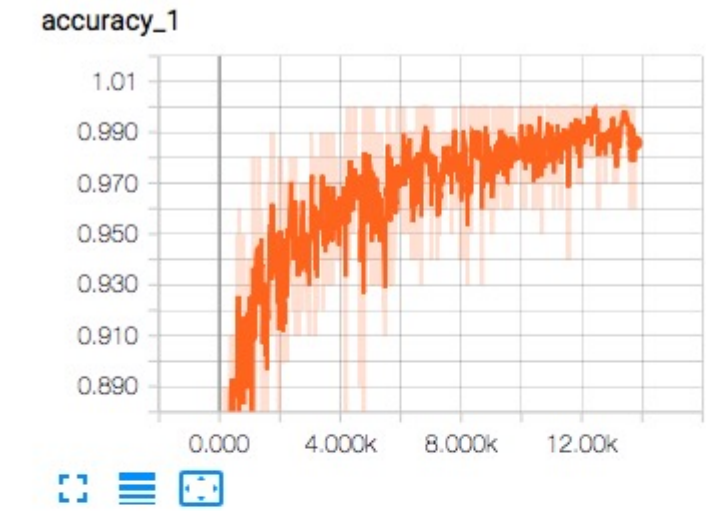
Figure 1
Testing data
Fitted line
3.5
3.0
2.5
2.0
1.5
1.0
2
4
6
8
10

Machine Learning Pre Lab #4 CNN

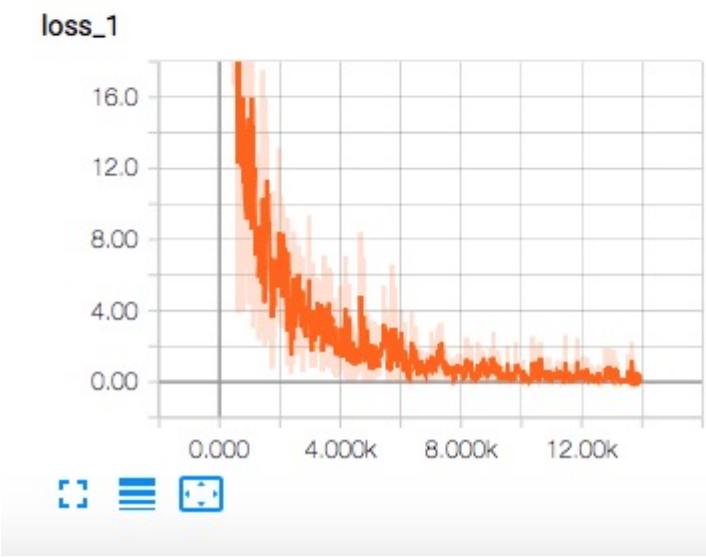
Visualize

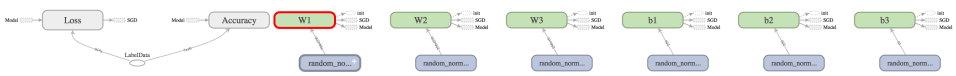
```
➜  Pre_lab_cnn tensorboard --logdir tmp/tensorflow_logs/example/
```

accuracy_1

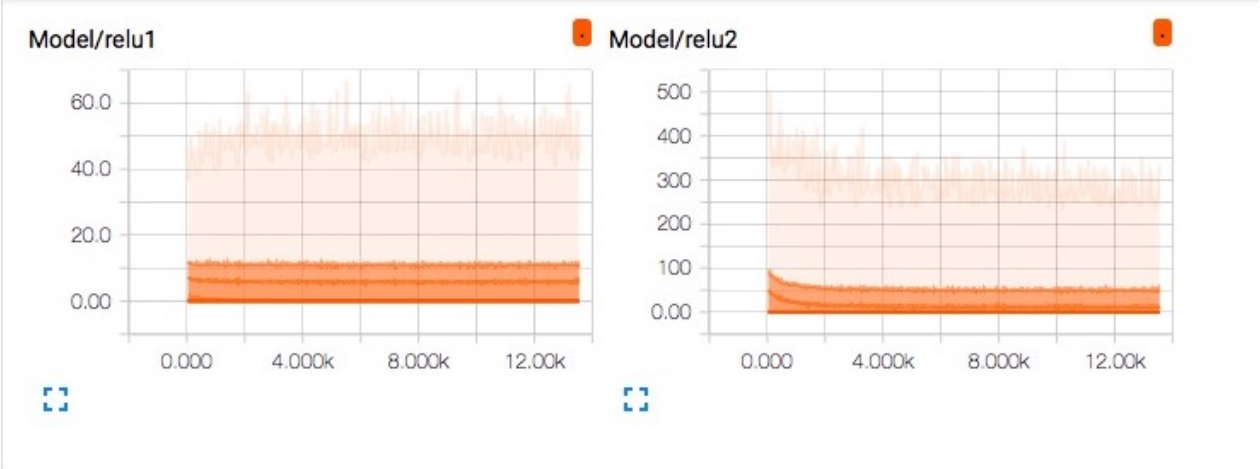


loss_1





Model



W1_0

Explain the meaning of loss function, SGD and activation function.

- **Loss functions** are helpful to train a neural network. Given an input and a target, they calculate the loss, i.e difference between output and target variable. Loss functions fall under three major category : Regressive loss functions, Classification loss functions, Embedding loss functions.

- **SGD**(Stochastic gradient descent) known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization.
- **Activation function** a node of activation function defines the output of that node given an input or set of inputs. Nonlinear activation functions allow such networks to compute nontrivial problems using only a small number.
  Activation functions include Sigmoid、TanHyperbolic(tanh)、ReLu、 softplus and softmax.