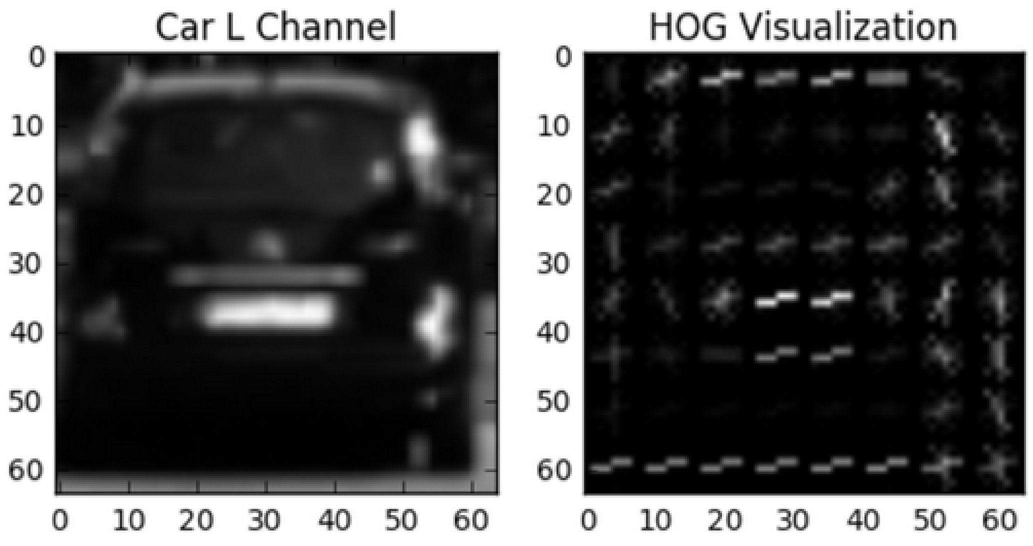
P5 Vehicle Detection

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) 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.

Histogram of Oriented Gradients (HOG)

The Histogram of Oriented Gradients technique counts the occurrences of gradient orientation in localized portions of an image. Following left image shows a car and right image shows the HOG Visualization of that car.



Using HOG features, I will be trying to differentiate cars from none car features in images and correctly identify the cars in the image (video).

HOG Parameters

First I tried using the parameters used in the lessons. There it was found that it works only for some images.

Then I started optimizing the parameters. After trying different parameter combinations, I found following HOG parameters giving good results.

```
### TODO: Tweak these parameters and see how the results change.
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_size = (32, 32) # Spatial binning dimensions
hist_bins = 16    # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
y_start_stop = [400, None] # Min and max in y to search in slide_window()
```

Feature Extraction

The “extract_features” method extracts spatial features, color histogram features and HOG features from a given image

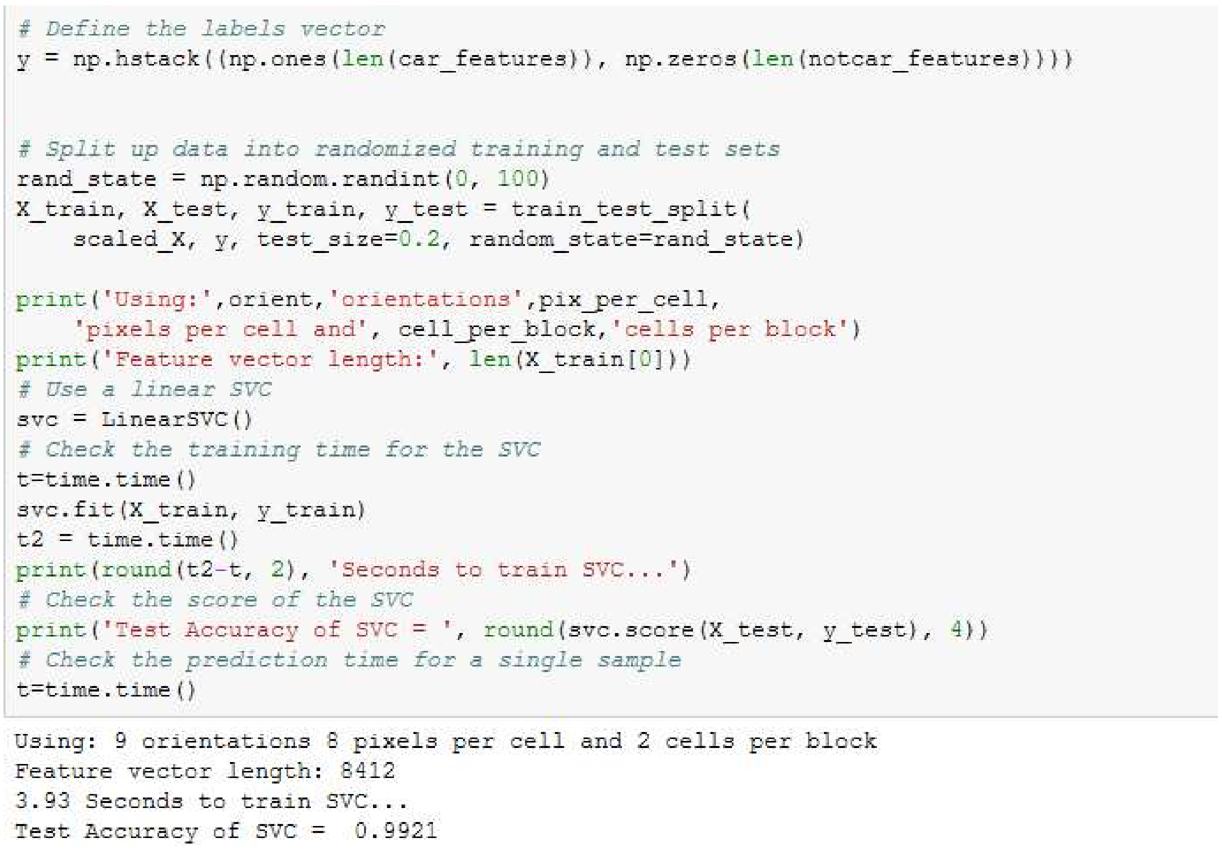
```
def extract_features(imgs, color_space='RGB', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel=0,
                        spatial_feat=True, hist_feat=True, hog_feat=True):
```

Training the Model

The features from both vehicles and none-vehicles are extracted and a linear SVM is trained

First I used 1000 images from each set. Later tried 4000, 6000 and 9000. It was found that 6000 set gave the best accuracy.

```
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)
```



```python
# 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()
```

```
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 8412
3.93 Seconds to train SVC...
Test Accuracy of SVC =  0.9921
```

Sliding Window

Sliding window is used to scan the image to find out vehicles. First I started with 64x64 window with 0.5 overlap. That window did not perform well. After experimenting it was found that:

Increasing the window size will improve the processing time taken, but reduces the detection

Increasing the overlap will improve the detection but takes long time to process.

After doing many experiments a widow of 100x100 pixels with 0.9 overlap is selected to scan the images.

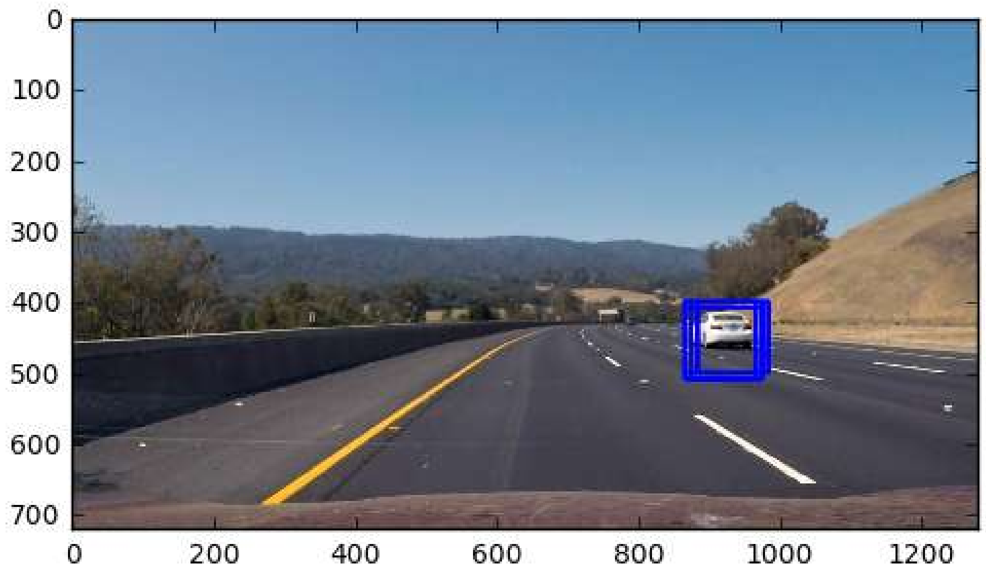
When the training model had less images to train, the size of the window had to be small (64x64) to get a marginal result. For 6000 images it was found that we can use 100x100 pixel window with 0.9 overlap.

Scaling

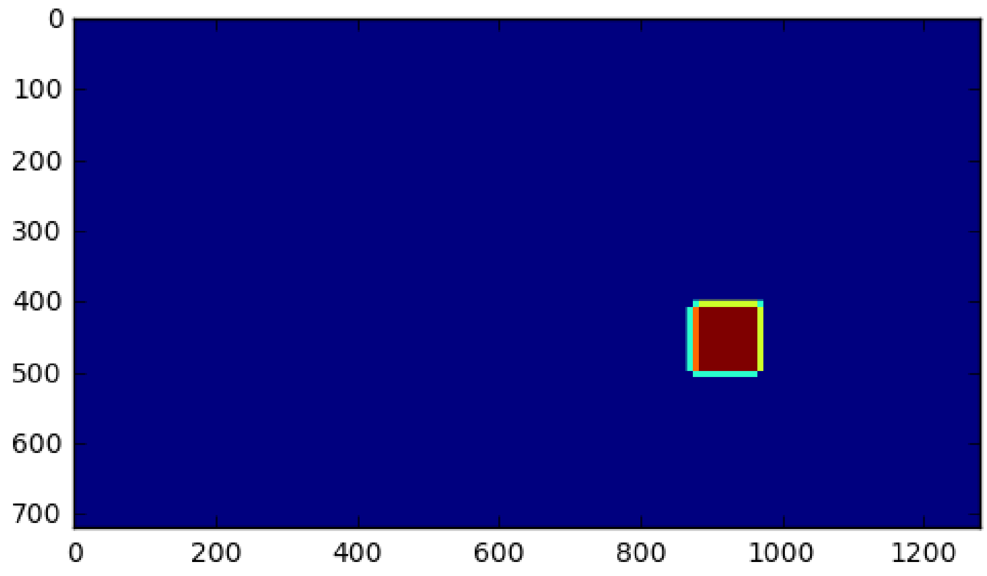
I tried to use different scaled levels for the sliding window. Finally I used 100x100 window with 0.9 overlap and 150x150 window with 0.8 overlap.

False Detection and Multiple Detection

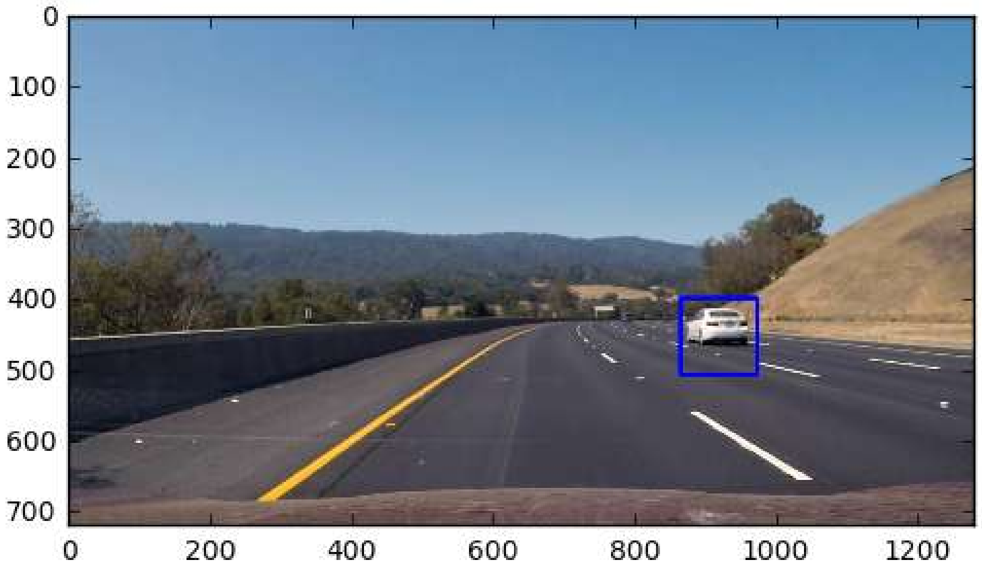
False detection Multiple Detection are a common issue. This was reduced by implementing Heat Map and applying a threshold of 1. Then I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. Later in the pipeline, the vehicles were only detected only when they were detected in two consecutive frames.



Multiple Detection



Heat Map



Thresholded and labeled Output

Pipeline

First pipeline was implemented by

1. Read given image
2. Apply a sliding window
3. Extract YCrCb 3 channel HOG features, Spatially binned color features and color histogram features.
4. Predict
5. Apply heat map and thresholding and false detection rejection
6. Plot final output

Different parameter combinations were tried out to get the best performance. For that, I started measuring the time taken to process a image.

Learning Data Set – 2000
Window – 64x64, 0.9 overlap
Y_start, Y_stop = 400,720
Time taken to process image – **75.8 seconds.**

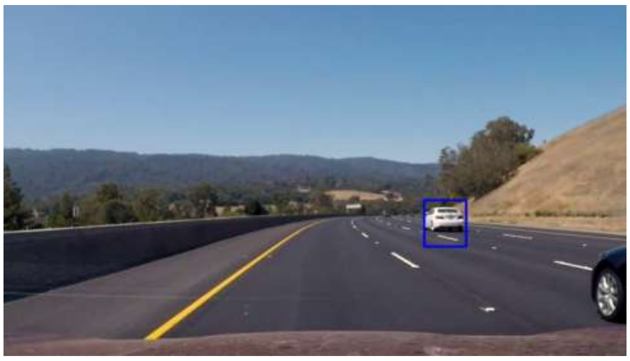
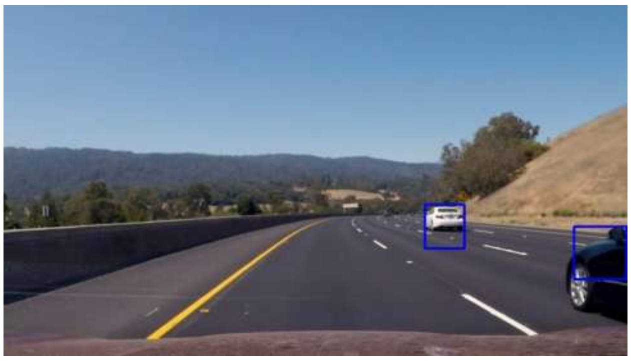
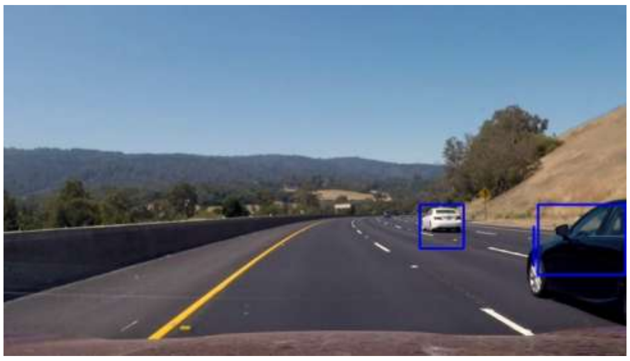
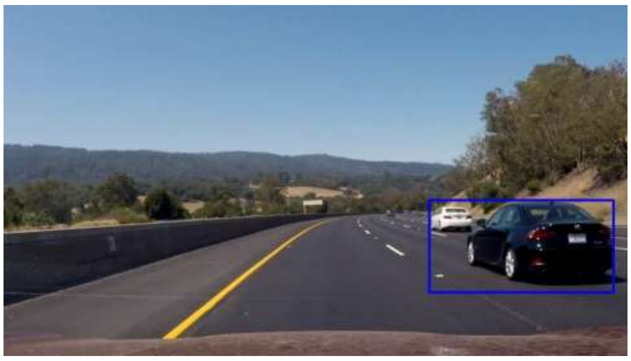
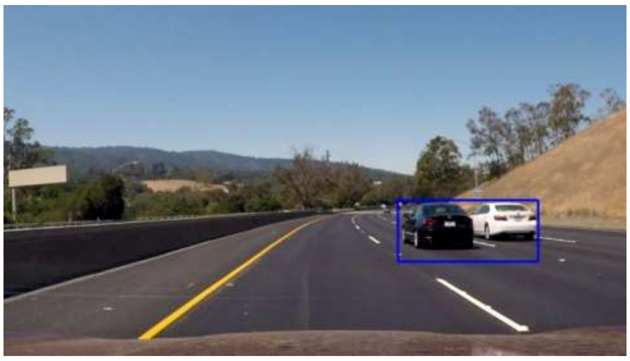
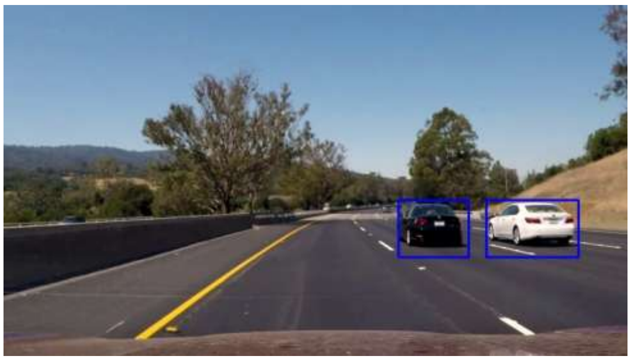
Learning Data Set – 4000
Window – 72x72, 0.9 overlap
Y_start, Y_stop = 400,720
Time taken to process image – **50.38 seconds.**
Learning Data Set – 6000
Window – 100x100, 0.9 overlap
Y_start, Y_stop = 400,720
Time taken to process image – **23.64 seconds.**

Learning Data Set – 6000
Window – 100x100, 0.9 overlap
Y_start, Y_stop = 400,600
Time taken to process image – **11.82 seconds.**

Learning Data Set – 6000
Window1 – 100x100, 0.9 overlap
    X_start, X_stop = 400,1200
    Y_start, Y_stop = 400,500
Window2 – 150x150, 0.8 overlap
    X_start, X_stop = 0,1200
    Y_start, Y_stop = 450,600
Time taken to process image – **1.18 seconds.**

Reducing the scanning X and Y range significantly improved the processing time.

Here are some of the output images

Video processing using the pipeline

The project video was processed using the final pipeline. There also, I used heat map, thresholding and labeling to reduce false detection and multiple detections.

Each frame took about 1.1 seconds to process. To process the whole video, it took about 20 minutes.

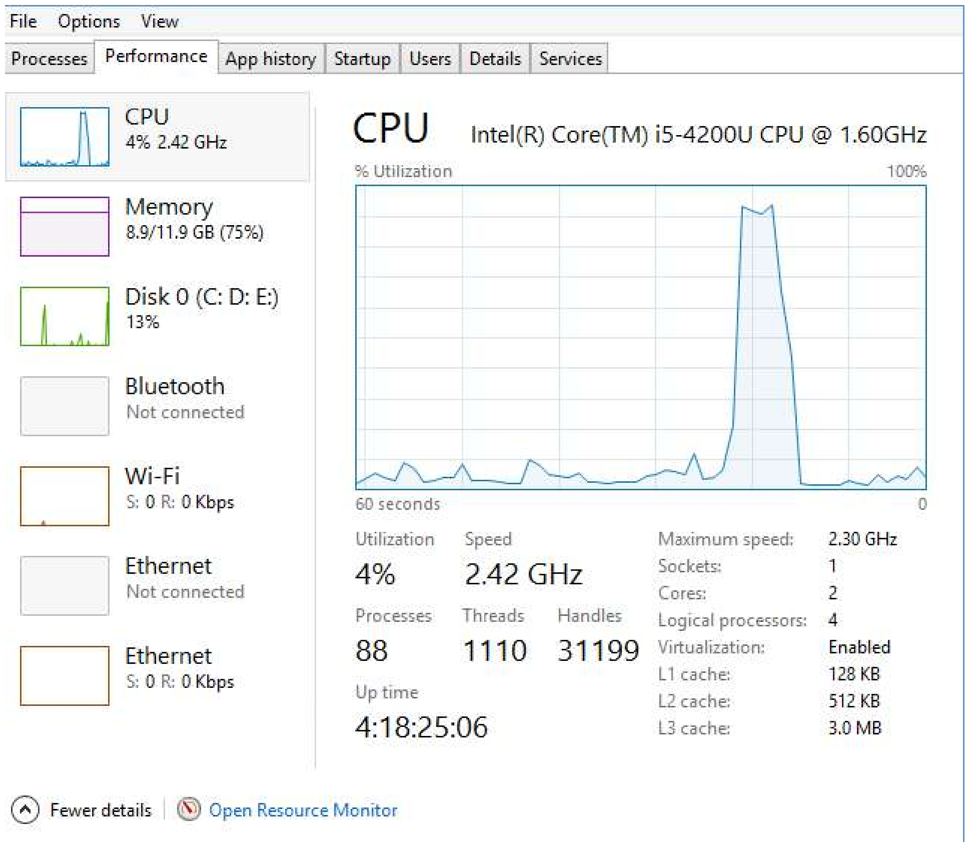
Discussion

Depends on the training set, the detection accuracy get change. Using small set made it reduce its accuracy. Also, using a large set over fit the model and reduced the accuracy.

We have not check the accuracy of the pipeline in different lighting conditions. There is a possibility that results may get change on the lighting conditions.

There were many false detections. This could be reduced by using a better trained model.

The best time to process a frame was about 1.1 seconds. In a real-time application this should be done within 40 milliseconds. That means the processing power should be increased by 30 times. This could be achieved by using GPU based system.



CPU 4% 2.42 GHz

Memory 8.9/11.9 GB (75%)

Disk 0 (C: D: E:) 13%

Bluetooth Not connected

Wi-Fi S: 0 R: 0 Kbps

Ethernet Not connected

Ethernet S: 0 R: 0 Kbps

CPU Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz

Utilization 4%
Speed 2.42 GHz
Processes 88
Threads 1110
Handles 31199
Up time 4:18:25:06

Maximum speed:	2.30 GHz
Sockets:	1
Cores:	2
Logical processors:	4
Virtualization:	Enabled
L1 cache:	128 KB
L2 cache:	512 KB
L3 cache:	3.0 MB

CPU performance during image process