Vehicle Detection

using Histogram of Oriented Gradients and Support Vector Machines

2019

Felix Jorczik, Philipp Rothmann

Institute for Electrical Engineering in Medicine
University of Lübeck

Structure

Introduction

Preprocessing

Histogram of Oriented Gradients (HOG)

Support Vector Machine (SVM)

Post Processing

Results

The Task

- Detect vehicle in video footage
- Camera in a cars wind-shield
- Highway environment

Example Video

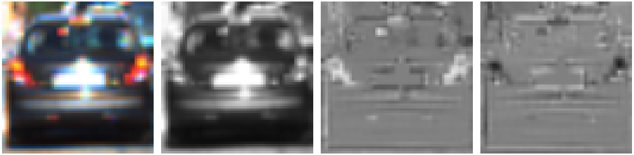
example video

The Pipeline

1. Frame extraction
2. Preprocessing
   $\rightarrow$ Colorspace Conversion
   $\rightarrow$ Sliding Windows
3. Feature computation
   $\rightarrow$ Histogram of Oriented Gradient (HOG)
4. Classification
   $\rightarrow$ Support Vector Machine (SVM)
5. False positive detection
   $\rightarrow$ Heat Map

Colorspace Conversion

- Best results with YCbCr
- luminance information extracted in Y
- color change
- Feature computation for each channel seperately



Sliding Window

Generating subimages in different sizes

Sliding Window

Histogram of Oriented Gradients (HOG)

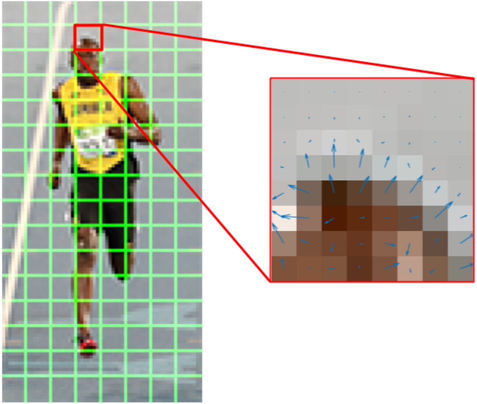
- Popular since Dalal and Triggs used them for pedestrian detection (2005)
- Good representation of object shapes
- Distribution of gradients
- Quite small feature descriptor

HOG Pipeline

1. Gradient computation
2. Orientation binning
3. Block normalisation

Gradient Computation

- Edge detection through convolution with simple kernel $[-1, 0, 1]$
- Calculation of the magnitude and direction
- Seperation in cells (pixel per cells)



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

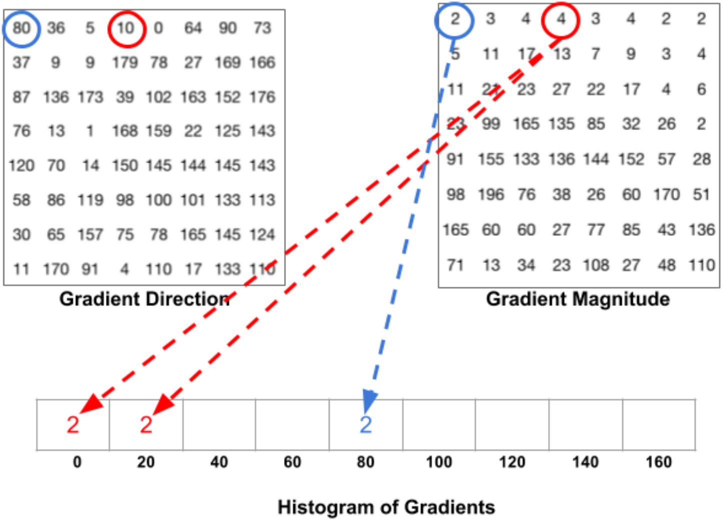
Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

https://www.learnopencv.com/histogram-of-oriented-gradients/

Orientation Binning

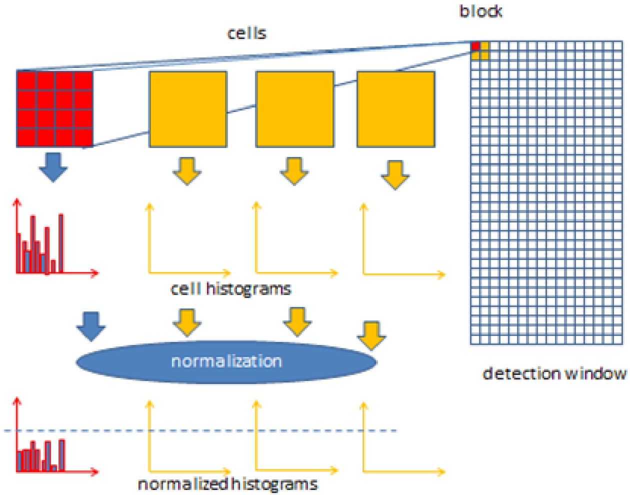


https://www.learnopencv.com/histogram-of-oriented-gradients/

Block Normalisation

- Reduce light sensitivity, contrast and other variations
- Blocks grouped from cells
- Normalisation function is applied on all blocks

Block Normalisation

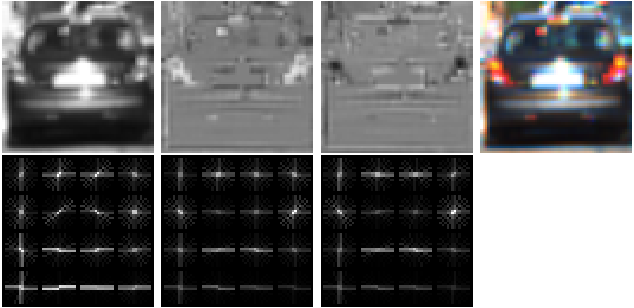


https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor

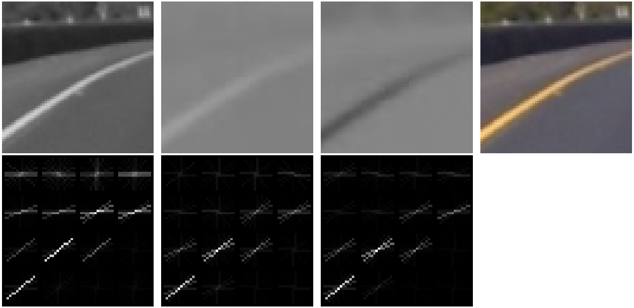
Feature Vector

- Image Size:
  - $64px * 64px * 3 * 1Byte \sim= 12MB$ (8 bit integer)
- HOG-Vector:
  - color x blocks x blocks x cells x cells x orientation bins
  - $3x3x3x2x2x12 = 1296$ dimensional vector
  - $1269 * 4Byte \sim= 5MB$ (32 bit float)

Example of a car image



Example of a none car image



Support Vector Machine(SVM)

- Original algorithm invented by Vapnik and Chervonenkis in 1963
- Supervised learning method for classification and regression
- Mainly used in image and handwriting recognition

Linear Perceptron

- The SVM is based on the concept of linear perceptrons
- Linear separability of the data is required

Linear Perceptron (learning phase)

- Works as a boolean function
- Learning phase: the input vector is multiplied by a weight vector and a bias is added

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w.x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w.x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

Linear Perceptron (learning phase)

- If the classification is wrong (input vector label contradicts output) the input vector is added to/subtracted from the weight vector
- Repitition until the algorithm reaches convergence

Algorithm: Perceptron Learning Algorithm

```
P ← inputs with label 1;
N ← inputs with label 0;
Initialize w randomly;
while !convergence do
    Pick random x ∈ P ∪ N ;
    if x ∈ P and w.x < 0 then
        w = w + x ;
    end
    if x ∈ N and w.x ≥ 0 then
        w = w − x ;
    end
end
//the algorithm converges when all the
 inputs are classified correctly
```

Linear Perceptron (learning phase)

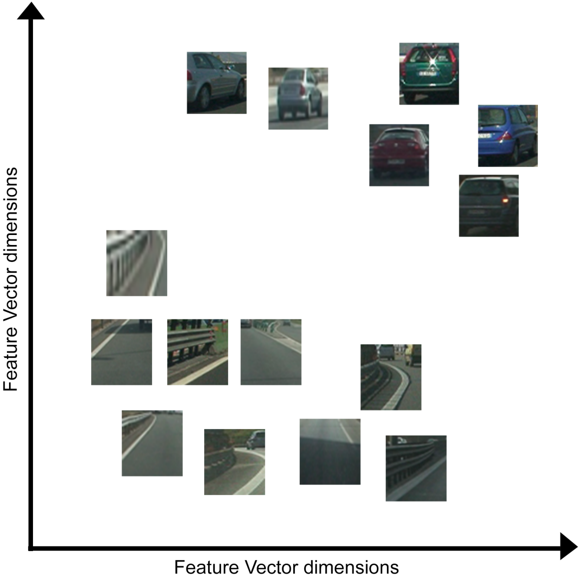
- The result is a learned weight and bias (a hyperplane)
- A new input vector from a datapoint is evaluated by the function and gives a binary output

Algorithm: Perceptron Learning Algorithm

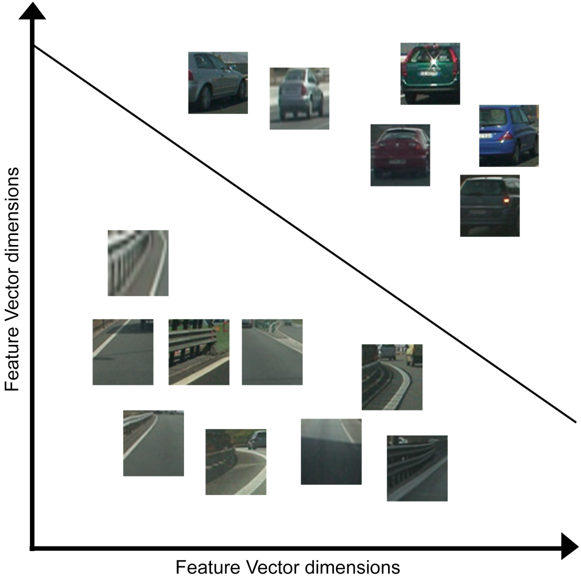
$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

https://cdn-images-1.medium.com/max/1600/1*PbJBdf-WxR0Dd0xHvEoh4A.png

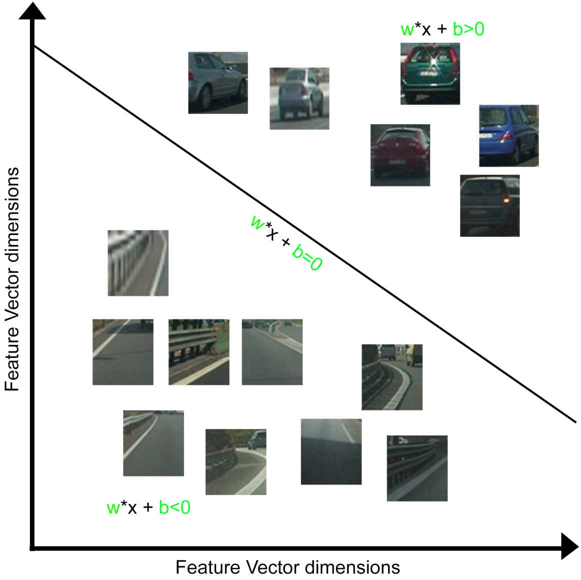
Linear Perceptron



Linear Perceptron

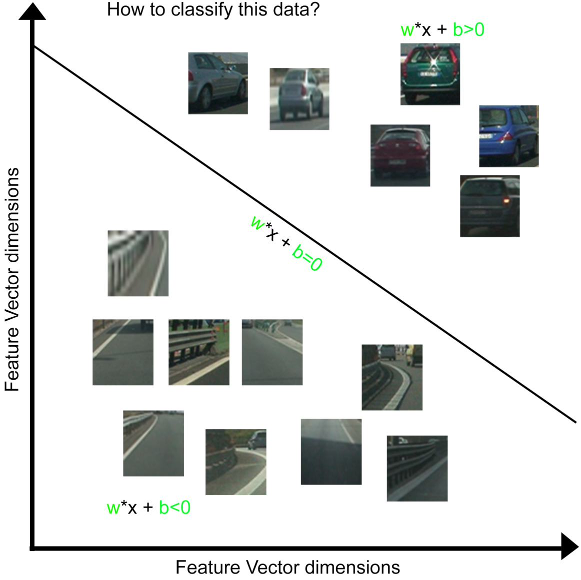


Linear Perceptron

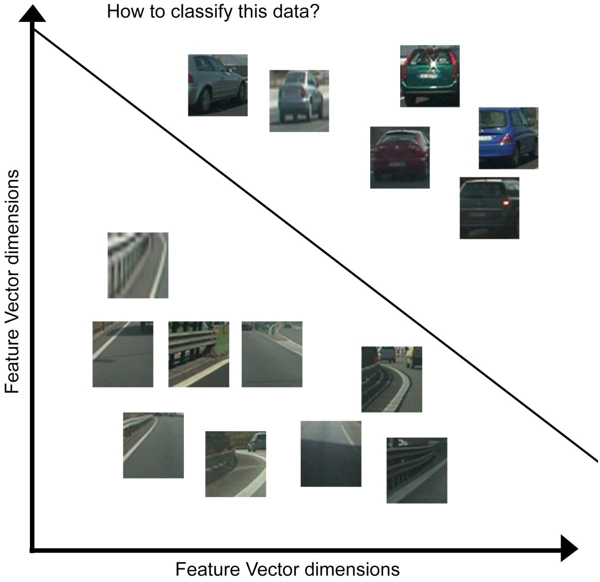


Linear Perceptron

How to classify this data?

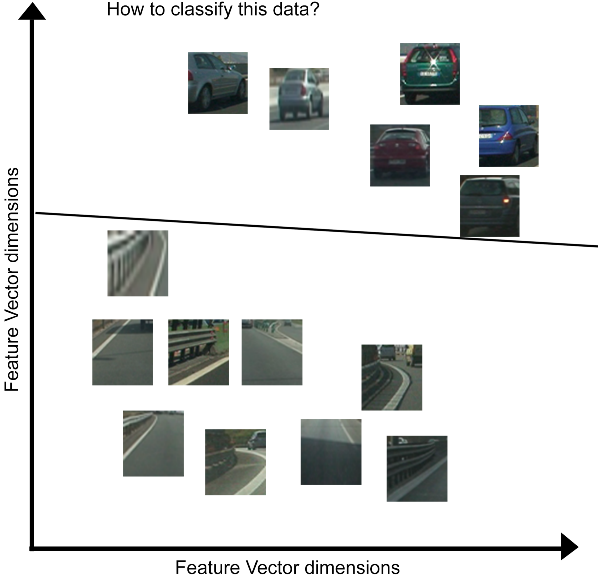


Linear Perceptron



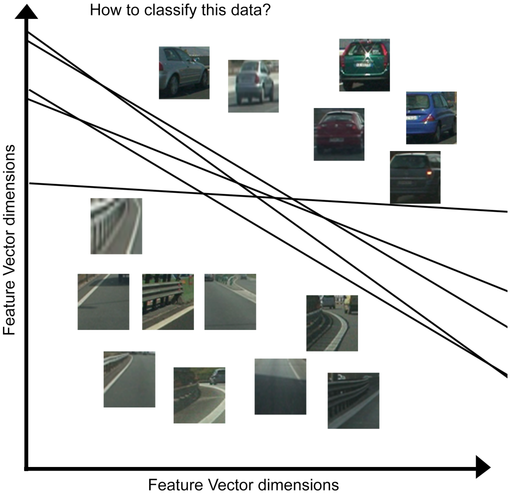
Linear Perceptron

How to classify this data?

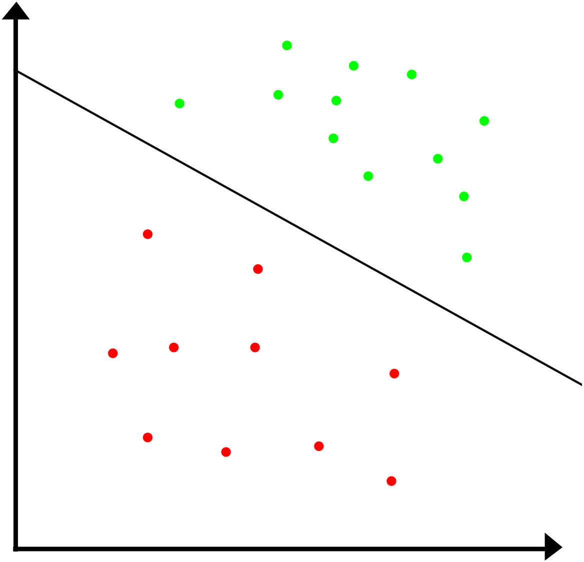


Linear Perceptron

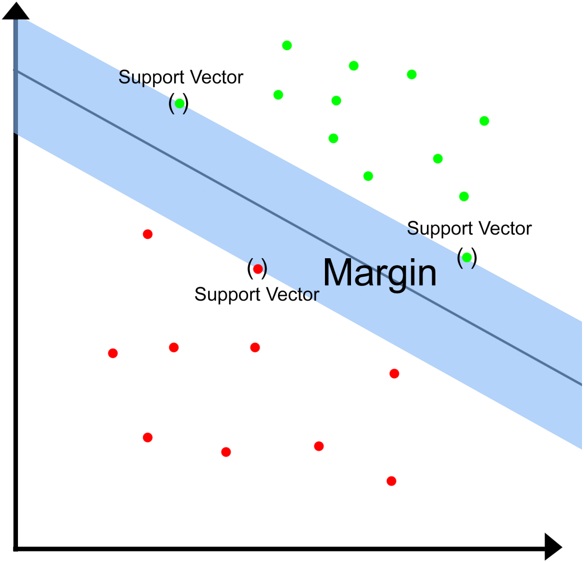
- Depending on the order of the input vectors the final seperating hyperplane is different



Linear Perceptron



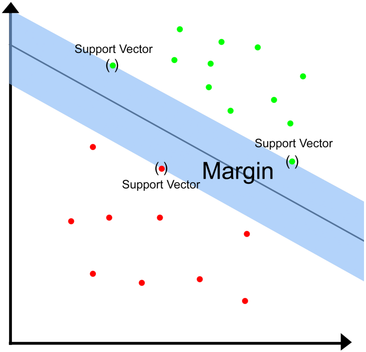
SVM



SVM

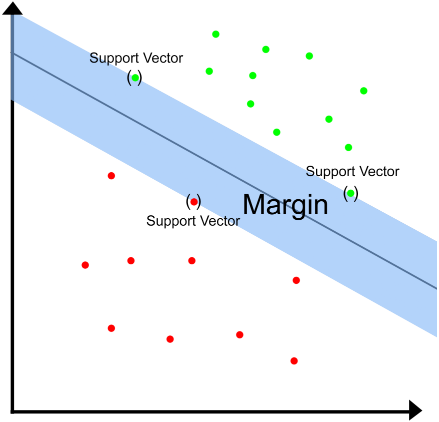
- Closest datapoints (Support Vectors) define the margin's size
- Maximizing the margin returns an optimal hyperplane

$$M = \frac{2}{||w||}$$



SVM

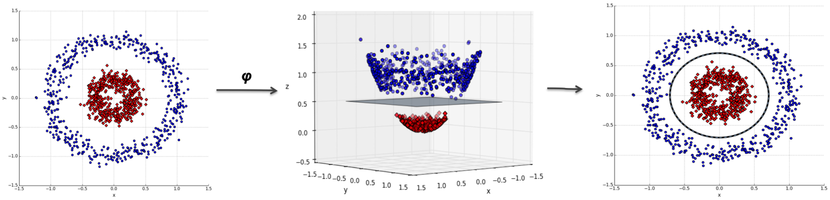
- Maximizing $M = \frac{2}{||w||}$ is equal to minimizing $||w||$
- This is solved by a Lagrange Multiplier



SVM (Nonlinear case)

- The kernel trick is solving the problem of nonlinear data (where separating planes do not exist)
- Nonlinear data is transformed to emulate linearity
- There are polynomial, radial and many more kernel methods

SVM (Nonlinear case)



http://beta.cambridgespark.com/courses/jpm/figures/mod5_kernel_trick.png

Heat Map

- Origin from thermal cameras
- Most common use in website analysis or human recognition (China)
- Uncomplicated and intuitive visualisation

Heat Map

- Made of monochrome grayscale images, binary masks or even colormaps
- Visualize informations density

Heat Map

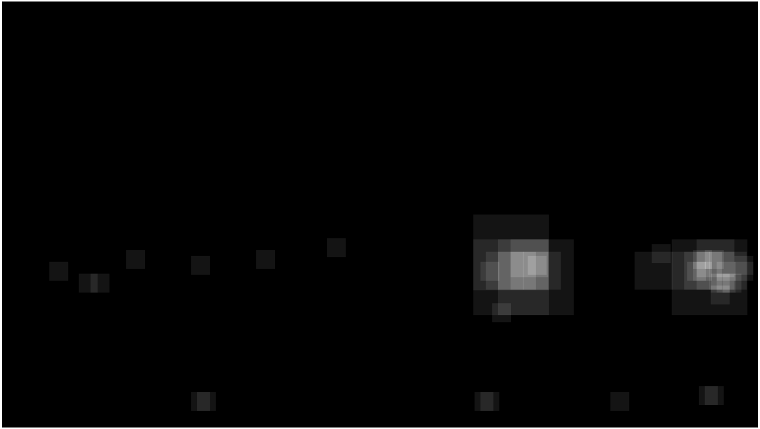
- Heat maps compensate the SVMs inaccuracy (false positives)

Heat Map

- Heat maps compensate the SVMs inaccuracy (false positives)

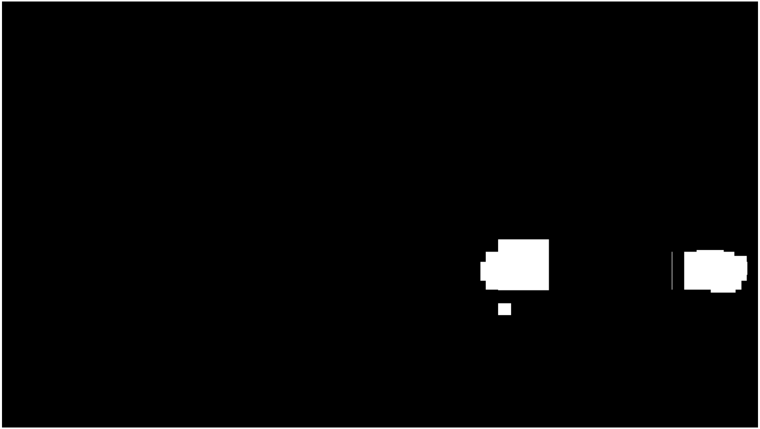
Heat Map

- The more rectangles overlap in an area, the brighter it gets in the heat map



Heat Map

- A threshold (of 3 or more overlappings) eliminates false positives



Example Video

example video

Results

- Continously improving the HOG paramters
- Calculating the HOG for each individual color dimension and appending them to one feature vector improved it
- YCrCb showed an SVM's accuracy of 99% (grayscale reached 96%)

Results

- More pixels per cell led to a more general represantation of vehicles
- Less pixels would have been too detailed
- Gamma correction reduced noise

Results

- Training the SVM with only one dataset made it fail on different test images
- Adding another dataset with about 6000 images resulted in more diversity

Results

- The issue of false positives was improved by adjusting the sliding windows
- The Heat map with a threshold was very rewarding
- Extending the Heat map on consecutive frames added more stability

Results (limitations)

- Situations differentiating from highways might not work well
- Very close and far vehicles will not be detected
- Live detection is not feasible with our current implementation

Conclusion

- Combining HOG's and SVM's we implemented a basic vehicle detector
- Still far away from being practical for autonomous systems