# Vehicle Detection Project

This is a project for Udacity self-driving car Nanodegree program. The aim of this project is to detect the vehicles in a dash camera video. The implementation of the project is in the file vehicle_detection.ipynb.

## Introduction to object detection

Detecting vehicles in a video stream is an object detection problem. An object detection problem can be approached as either a classification problem or a regression problem. As a classification problem, the image are divided into small patches, each of which will be run through a classifier to determine whether there are objects in the patch. Then the bounding boxes will be assigned to locate around patches that are classified with high probability of present of an object. In the regression approach, the whole image will be run through a convolutional neural network to directly generate one or more bounding boxes for objects in the images.

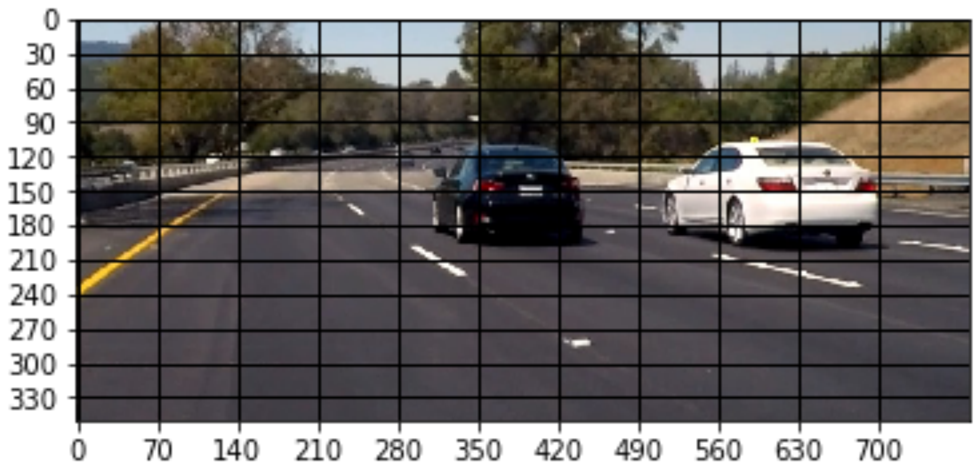
| classification | regression |
|---|---|
| Classification on portions of the image to determine objects, generate bounding boxes for regions that have positive classification results. | Regression on the whole image to generate bounding boxes |
| 1. sliding window + HOG 2. sliding window + CNN 3. region proposals + CNN | generate bounding box coordinates directly from CNN |
| RCNN, Fast-RCNN, Faster-RCNN | SSD, YOLO |

In this project, we will use tiny-YOLO v2, since it's easy to implement and are reasonably fast.

## The Tiny-YOLO v2

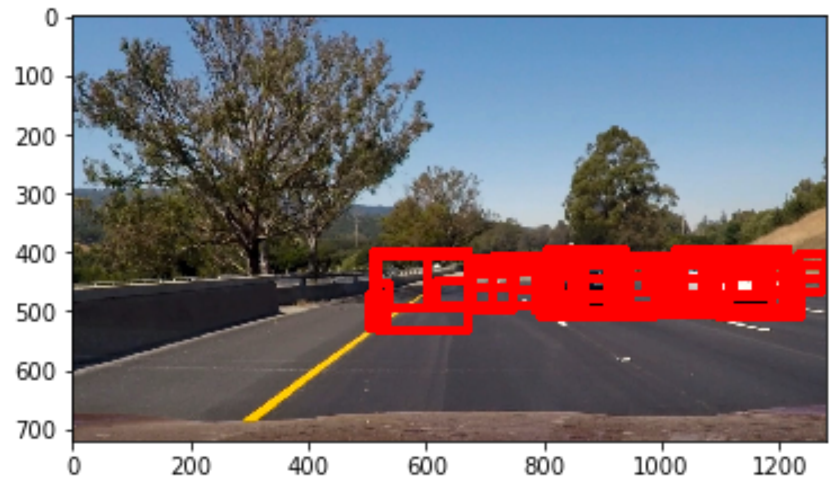
YOLO takes a completely different approach. It's not a traditional classifier that is repurposed to be an object detector. YOLO actually looks at the image just once (hence its name: You Only Look Once) but in a clever way.

YOLO divides up the image into a grid of 13 by 13 cells.



Each of these cells is responsible for predicting 5 bounding boxes. A bounding box describes the rectangle that encloses an object.

YOLO also outputs a *confidence score* that tells us how certain it is that the predicted bounding box actually encloses some object. This score doesn't say anything about what kind of object is in the box, just if the shape of the box is any good.



For each bounding box, the cell also predicts a *class*. This works just like a classifier: it gives a probability distribution over all the possible classes. The version of YOLO we're using is trained on the PASCAL VOC dataset (20 classes).

The confidence score for the bounding box and the class prediction are combined into one final score that tells us the probability that this bounding box contains a specific type of object.

Since there are 13×13 = 169 grid cells and each cell predicts 5 bounding boxes, we end up with 845 bounding boxes in total. It turns out that most of these boxes will have very low confidence scores, so we only keep the boxes whose final score is 30% or more (you can change this threshold depending on how accurate you want the detector to be).

From the 845 total bounding boxes we only kept these three because they gave the best results. But note that even though there were 845 separate predictions, they were all made at the same time — the neural network just ran once. And that's why YOLO is so powerful and fast.

## Architecture of the convolutional neural network:

The architecture of YOLO is simple, it's just a convolutional neural network. This neural network only uses standard layer types: convolution with a 3×3 kernel followed by batch-normalization layer and max-pooling with a 2×2 kernel.
In total, there are 9 convolutional layers, 8 batch normalization layers and 6 pooling layers.
The very last convolutional layer has a 1×1 kernel and exists to reduce the data to the shape 13×13×125. This 13×13 should look familiar: that is the size of the grid that the image gets divided into.

So we end up with 125 channels for every grid cell. These 125 numbers contain the data for the bounding boxes and the class predictions. Why 125? Well, each grid cell predicts 5 bounding boxes and a bounding box is described by 25 data elements:

- x, y, width, height for the bounding box's rectangle

- the confidence score
- the probability distribution over the 20 classes

Filter out the bounding boxes which has object threshold greater than 55%.

To remove the overlapping bounding boxes, we use a technique called NonMax Suppression.

The algorithm used by the `nonMaxSuppression()` function is quite simple:

1. Start with the bounding box that has the highest score.
2. Remove any remaining bounding boxes that overlap it more than the given threshold amount (i.e. more than 45%).
3. Go to step 1 until there are no more bounding boxes left

This removes any bounding boxes that overlap too much with other boxes that have a higher score. It only keeps the best ones.

## Model Summary:

```
Layer     (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 416, 416, 16)      432
_________________________________________________________________
norm_1 (BatchNormalization)  (None, 416, 416, 16)      64
_________________________________________________________________
leaky_re_lu_139 (LeakyReLU)  (None, 416, 416, 16)      0
_________________________________________________________________
max_pooling2d_107 (MaxPoolin (None, 208, 208, 16)      0
_________________________________________________________________
conv_2 (Conv2D)              (None, 208, 208, 32)      4608
_________________________________________________________________
norm_2 (BatchNormalization)  (None, 208, 208, 32)      128
_________________________________________________________________
leaky_re_lu_140 (LeakyReLU)  (None, 208, 208, 32)      0
_________________________________________________________________
max_pooling2d_108 (MaxPoolin (None, 104, 104, 32)      0
_________________________________________________________________
conv_3 (Conv2D)              (None, 104, 104, 64)      18432
_________________________________________________________________
norm_3 (BatchNormalization)  (None, 104, 104, 64)      256
_________________________________________________________________
leaky_re_lu_141 (LeakyReLU)  (None, 104, 104, 64)      0
_________________________________________________________________
max_pooling2d_109 (MaxPoolin (None, 52, 52, 64)        0
_________________________________________________________________
conv_4 (Conv2D)              (None, 52, 52, 128)       73728
_________________________________________________________________
norm_4 (BatchNormalization)  (None, 52, 52, 128)       512
_________________________________________________________________
leaky_re_lu_142 (LeakyReLU)  (None, 52, 52, 128)       0
_________________________________________________________________
max_pooling2d_110 (MaxPoolin (None, 26, 26, 128)       0
_________________________________________________________________
conv_5 (Conv2D)              (None, 26, 26, 256)       294912
_________________________________________________________________
norm_5 (BatchNormalization)  (None, 26, 26, 256)       1024
```

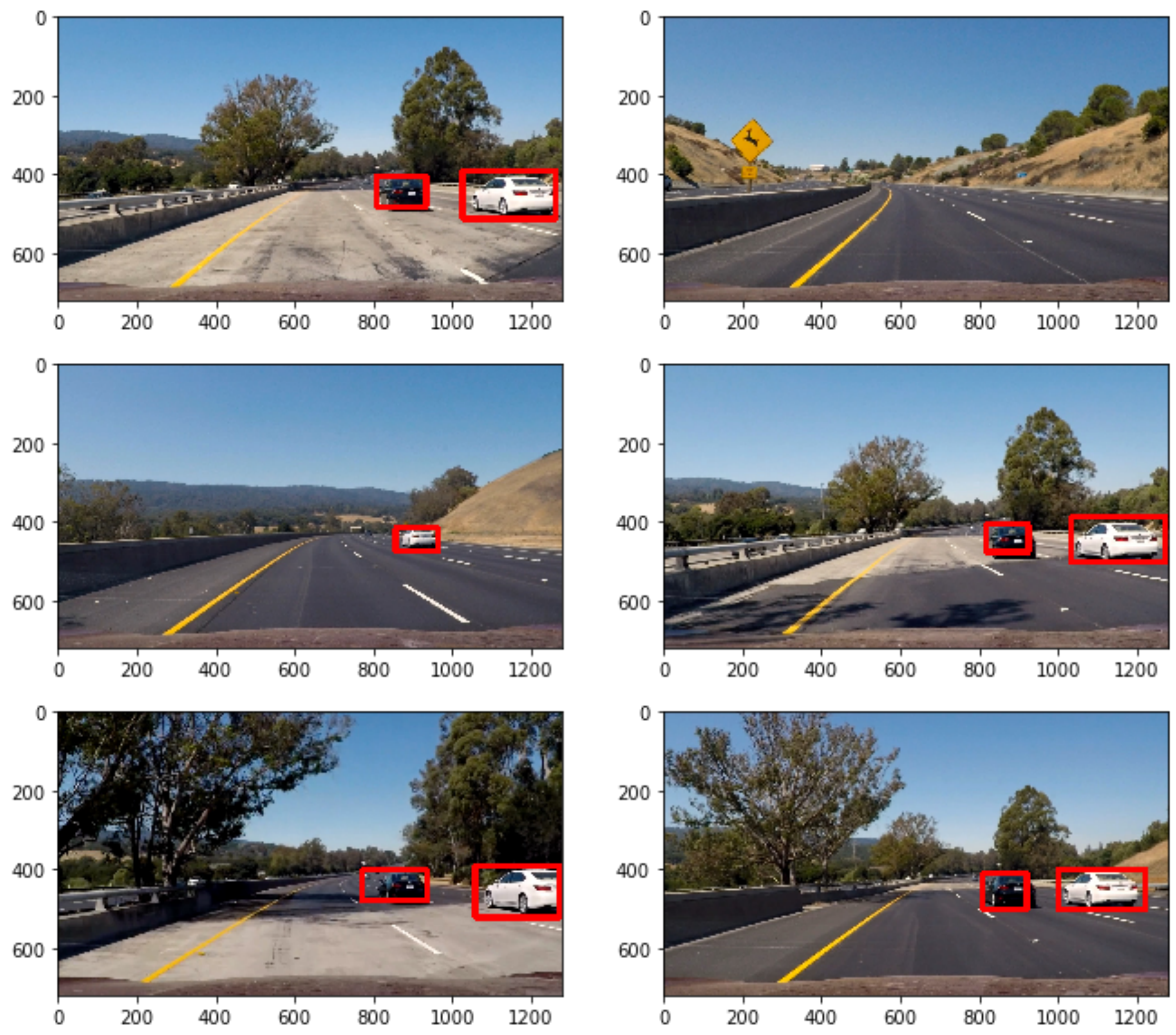
```
_________________________________________________________________
leaky_re_lu_143 (LeakyReLU)  (None, 26, 26, 256)       0         
_________________________________________________________________
max_pooling2d_111 (MaxPoolin (None, 13, 13, 256)       0         
_________________________________________________________________
conv_6 (Conv2D)              (None, 13, 13, 512)       1179648   
_________________________________________________________________
norm_6 (BatchNormalization)  (None, 13, 13, 512)       2048      
_________________________________________________________________
leaky_re_lu_144 (LeakyReLU)  (None, 13, 13, 512)       0         
_________________________________________________________________
max_pooling2d_112 (MaxPoolin (None, 13, 13, 512)       0         
_________________________________________________________________
conv_7 (Conv2D)              (None, 13, 13, 1024)      4718592   
_________________________________________________________________
norm_7 (BatchNormalization)  (None, 13, 13, 1024)      4096      
_________________________________________________________________
leaky_re_lu_145 (LeakyReLU)  (None, 13, 13, 1024)      0         
_________________________________________________________________
conv_8 (Conv2D)              (None, 13, 13, 1024)      9437184   
_________________________________________________________________
norm_8 (BatchNormalization)  (None, 13, 13, 1024)      4096      
_________________________________________________________________
leaky_re_lu_146 (LeakyReLU)  (None, 13, 13, 1024)      0         
_________________________________________________________________
conv_9 (Conv2D)              (None, 13, 13, 125)       128125    
=================================================================
Total params: 15,867,885
Trainable params: 15,861,773
Non-trainable params: 6,112
_________________________________________________________________
```

## Use pretrained weights:

Training the YOLO network is time consuming. We will download the pretrained weights (trained on VOC Pascal dataset) from https://pjreddie.com/darknet/yolo/

### Results

The following shows the results for several test images with a threshold of 0.17. We can see that the cars are detected:



## References:

https://github.com/allanzelener/YAD2K
http://machinethink.net/blog/object-detection-with-yolo/
https://pjreddie.com/darknet/yolo/
https://pjreddie.com/media/files/papers/YOLO9000.pdf