# Vehicle Detection

This was last project of Udacity SCND Term-1

**Please Note**: I have not used HOG+SVM for this project. I have used an algorithm called **YOLO**. I am fulfilling all the technical rubric points, just by method other than HOG+SVM.

## Algorithm YOLO: You Only Look Once

YOLO reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.
In this project we will implement tiny-YOLO. Full details of the network, training and implementation are available in the paper - http://arxiv.org/abs/1506.02640

YOLO divides the input image into an SxS grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes.

Confidence is defined as (Probability that the grid cell contains an object) multiplied by (Intersection over union of predicted bounding box over the ground truth).

**Confidence = Pr(Object) x IOU_truth_pred.**

f no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

Each bounding box consists of 5 predictions:

1.x

2.y

3.w

4.h

5.confidence

The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.

Each grid cell also predicts C conditional class probabilities, Pr(Class | Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B.

At test time we multiply the conditional class probabilities and the individual box confidence predictions,

**Pr(Class|Object) x Pr(Object) x IOU_truth_pred = Pr(Class) x IOU_truth_pred**

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

So at test time, the final output vector for each image is a **S x S x (B x 5 + C)** length vector

# Architecture

The model architecture consists of **9 convolutional layers**, followed by **3 fully connected layers**. Each convolutional layer is followed by a **Leaky RELU activation** function, with **alpha of 0.1**. The first **6** convolutional layers also have a **2x2 max pooling** layers.

I have used keras to build model and tensorflow as backend

Below is complete architecture of YOLO.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| convolution2d_1 (Convolution2D) | (None, 16, 448, 448) | 448 | convolution2d_input_1[0][0] |
| leakyrelu_1 (LeakyReLU) | (None, 16, 448, 448) | 0 | convolution2d_1[0][0] |
| maxpooling2d_1 (MaxPooling2D) | (None, 16, 224, 224) | 0 | leakyrelu_1[0][0] |
| convolution2d_2 (Convolution2D) | (None, 32, 224, 224) | 4640 | maxpooling2d_1[0][0] |
| leakyrelu_2 (LeakyReLU) | (None, 32, 224, 224) | 0 | convolution2d_2[0][0] |
| maxpooling2d_2 (MaxPooling2D) | (None, 32, 112, 112) | 0 | leakyrelu_2[0][0] |
| convolution2d_3 (Convolution2D) | (None, 64, 112, 112) | 18496 | maxpooling2d_2[0][0] |

| | | |
|---|---|---|
| leakyrelu_3 (LeakyReLU) convolution2d_3[0][0] | (None, 64, 112, 112) | 0 |
| maxpooling2d_3 (MaxPooling2D) leakyrelu_3[0][0] | (None, 64, 56, 56) | 0 |
| convolution2d_4 (Convolution2D) maxpooling2d_3[0][0] | (None, 128, 56, 56) | 73856 |
| leakyrelu_4 (LeakyReLU) convolution2d_4[0][0] | (None, 128, 56, 56) | 0 |
| maxpooling2d_4 (MaxPooling2D) leakyrelu_4[0][0] | (None, 128, 28, 28) | 0 |
| convolution2d_5 (Convolution2D) maxpooling2d_4[0][0] | (None, 256, 28, 28) | 295168 |
| leakyrelu_5 (LeakyReLU) convolution2d_5[0][0] | (None, 256, 28, 28) | 0 |
| maxpooling2d_5 (MaxPooling2D) leakyrelu_5[0][0] | (None, 256, 14, 14) | 0 |
| convolution2d_6 (Convolution2D) maxpooling2d_5[0][0] | (None, 512, 14, 14) | 1180160 |
| leakyrelu_6 (LeakyReLU) convolution2d_6[0][0] | (None, 512, 14, 14) | 0 |
| maxpooling2d_6 (MaxPooling2D) leakyrelu_6[0][0] | (None, 512, 7, 7) | 0 |
| convolution2d_7 (Convolution2D) maxpooling2d_6[0][0] | (None, 1024, 7, 7) | 4719616 |
| leakyrelu_7 (LeakyReLU) convolution2d_7[0][0] | (None, 1024, 7, 7) | 0 |
| convolution2d_8 (Convolution2D) leakyrelu_7[0][0] | (None, 1024, 7, 7) | 9438208 |
| leakyrelu_8 (LeakyReLU) convolution2d_8[0][0] | (None, 1024, 7, 7) | 0 |
| convolution2d_9 (Convolution2D) leakyrelu_8[0][0] | (None, 1024, 7, 7) | 9438208 |
| leakyrelu_9 (LeakyReLU) convolution2d_9[0][0] | (None, 1024, 7, 7) | 0 |
| flatten_1 (Flatten) leakyrelu_9[0][0] | (None, 50176) | 0 |

```
dense_1 (Dense)                 (None, 256)           12845312
flatten_1[0][0]
_________________________________________________________________
dense_2 (Dense)                 (None, 4096)          1052672
dense_1[0][0]
_________________________________________________________________
leakyrelu_10 (LeakyReLU)        (None, 4096)          0
dense_2[0][0]
_________________________________________________________________
dense_3 (Dense)                 (None, 1470)          6022590
leakyrelu_10[0][0]
=================================================================
Total params: 45,089,374
Trainable params: 45,089,374
Non-trainable params: 0
_________________________________________________________________
```

# Training

I have used pre-trained weights for this project.Here is the link for the same (180 MB)

## Postprocessing

The output of this network is a 1470 tensor, which contains the information for the predicted bounding boxes.

The model was trained on PASCAL VOC dataset. We use S = 7, B = 2. PASCAL VOC has 20 labelled classes so C = 20. So our final prediction, for each input image, is:

output tensor length = S x S x (B x 5 + C)

output tensor length = 7 x 7 x (2x5 + 20)

output tensor length = 1470.

The structure of the 1470 length tensor is as follows:

1.First 980 values correspons to probabilities for each of the 20 classes for each grid cell. These probabilities are conditioned on objects being present in each grid cell.

2.The next 98 values are confidence scores for 2 bounding boxes predicted by each grid cells.

3.The next 392 values are co-ordinates (x, y, w, h) for 2 bounding boxes per grid cell.

The detail of these steps are in the yolo_net_out_to_car_boxes function in the util class.

## Class score threshold

We reject output from grid cells below a certain threshold (0.2) of class scores (equation 2), computed at test time.

## Reject overlapping (duplicate) bounding boxes

If multiple bounding boxes, for each class overlap and have an IOU of more than 0.4 (intersecting area is 40% of union area of boxes), then we keep the box with the highest class score and reject the other box(es).

## Drawing the bounding boxes

The predictions (x, y) for each bounding box are relative to the bounds of the grid cell and (w, h) are relative to the whole image. To compute the final bounding box coodinates we have to multiply **w** & **h** with the width & height of the portion of the image used as input for the network.

### Results

Below are the images from the model predictions

**Evaluation Video**

I have also added the video along with submission with ‘project_video_output.mp4’

**Discussion**

The YOLO is known to be fast. In the original paper, the tiny-YOLO is reported to work at nearly 200 FPS on a powerful desktop GPU.