# AIML CAPSTONE PROJECT
# COMPUTER VISION
# OBJECT DETECTION – CAR

Interim report - April 2021

Submitted by

Ramnath Natarajan

Aravind Muthu Suthan T

Natarajan T

Kartik Muthuswamy

Batch ID: AIML Apr20B (GROUP 4)

Under the guidance of

Kaustubh V. Sakhare

**Table of Contents**

## 1. ABSTRACT

This interim report aims to document the progress of the Capstone project up to Milestone-1 and put forth future steps towards achieving Milestones 2 & 3. Emphasis is laid on the steps of Milestone-1 namely problem definition, exploratory data analytics, mapping training & testing images to its respective classes along with localization in the form of a bounding box. Further, it provides details about training & testing a basic CNN model along with it's data pre-processing. Objective is to display the image class with its bounding box, upon providing image file name or index as input. In the future, it is planned to test different models and compare them based on metrics like IOU, MAP and run time. Upon narrowing down the best suitable model, a clickable user interface will be designed to enable image selection and algorithm execution.

Keywords: *Object detection, Automotive surveillance, MobileNet, Car image detection*

## 2.INTRODUCTION

In the current world scenario, penetration of AI has become evident in every sector of technical and non-technical industries. The sub-domain of Automotive Surveillance is fast growing in terms of application & advancement since many of the traffic management control systems & security systems heavily rely upon Vehicle detection and associated statistics. With its rising popularity, the focus has been to generate huge databases of vehicle data including its images & video footages. This has brought forth challenges, at the same time, opportunities to build complex and accurate AI models to decipher such data and also identify the vehicles based on images or video.

Further, in recent years, there has been a significant increase in research interest supporting the development of autonomous vehicles, a platform capable of sensing and reacting to the immediate environment in an attempt to navigate roadways without human intervention. The task of environment sensing is known as perception, and often consists of a number of subtasks such as object classification, detection, 3D position estimation, and simultaneous localization and mapping (SLAM). In many autonomous driving systems, the object detection subtask is itself one of the most important prerequisites to autonomous navigation, as this task is what allows the car controller to account for obstacles when considering possible future trajectories.

In an effort to understand the fundamentals of this domain, this capstone project is taken up to design a Deep Learning based Car identification model & leverage its usage by designing a clickable user interface

## 3. PROBLEM DEFINITION

A dataset containing 16185 car images and pertaining annotation files are provided. The final objective is to design a Deep Learning based Car detection model along with a clickable user interface. The various stages in development of this model is phased in to 3 milestones as shown below:

- Milestone **1**:
  - Understanding the data
  - EDA
  - Mapping images to its class and displaying bounding box
  - Data pre-processing
- **Milestone 2**
  - Apply different CNN models and compare accuracy of classification
  - Apply different RCNN models for imposing bounding box
  - Conclude on best model for future prediction
- Milestone **3**
  - Design a clickable UI for selecting an image and predict its class & bounding box

## 4. LITERATURE REVIEW

Prior to finalizing the approach, literature review is carried out taking below articles as reference

4.1 Research paper on Vehicle Detection and Recognition by Sriashika Addala (May 2020)

This paper discusses the processing of automatic **vehicle detection and recognition** from **static image datasets**. The surveillance system includes detection of moving vehicles and recognizing them, counting the number of vehicles and verification of their permit with the organization. Once the vehicle has been detected, LPR (License Plate Recognition) shall be implemented. The recognized number plate shall then be processed to capture the license number. This license number will then be compared to an existing database and checked if it is valid, registered with the organization, permit's validity, if the vehicle is parked at the allotted parking location and many other parameters. The many benefits of this project would be reduced manual efforts in manual checking of each vehicle and also in maintaining manual records of the same.

**4.2** Research paper on Vehicle Detection with HOG and Linear SVM by Nikola Tomikj and Andrea Kulakov (February 2021)

In this paper, a vehicle detection system is presented by employing Histogram of Oriented Gradients (HOG) for feature extraction and linear SVM for classification. They study the influence of the color space on the performance of the detector, concluding that decorrelated and perceptual color spaces give the best results. An in-depth analysis is carried out on the effects of the HOG and SVM parameters, the threshold for the distance between features and the SVM classifying plane, and the non-maximum suppression (NMS) threshold on the performance of the detector, and they propose values that illustrate good performance for vehicle detection on images. They also discuss the issues of the approach and the reasons for its mediocre performance on videos. The described approach performs reasonably well on images and achieves mediocre performance on videos. They have shown that using HOG and linear SVM is a viable approach for vehicle detection in images, while it has some limitations for vehicle detection in videos. However, by using some simple techniques and extending the pipeline, this approach can easily overcome these limitations. They suggested that the false positives can be eliminated by checking whether the positive detections in a region are appearing in more consecutive frames.

### 4.3 Research paper on Object Detection With Deep Learning: A Review by Zhong-Qiu Zhao.Peng Zheng, Shou-Tao Xu, and Xindong Wu

Due to object detection's close relationship with video analysis and image understanding, it has attracted much research attention in recent years. In this paper, they provide a review of deep learning-based object detection frameworks. Their review begins with a brief introduction on the history of deep learning and its representative tool, namely, the convolutional neural network. Then, they focus on typical generic object detection architectures along with some modifications and useful tricks to improve detection performance further. As distinct specific detection tasks exhibit different characteristics, we also briefly survey several specific tasks, including salient object detection, face detection, and pedestrian detection.This paper provides a detailed review on deep learning-based object detection frameworks that handle different subproblems, such as occlusion, clutter, and low resolution, with different degrees of modifications on R-CNN.

**Take away from literature survey:**

- Based on the papers mentioned above and other online resources like Kaggle, GitHub, we see that the given computer vision problem statement is of type "Object Classification and Localization".
- Key challenges in developing prediction models is presence of different image sizes (aspect ratios), imbalanced data classes and scenarios with variables number of boxes as output
- Metrics for deciding a good model are IoU score, mAP and Precision & Recall

To begin with, Mobilenet is used to predict image class and bounding box. In next steps, other models will be tried and evaluated using the above metrics.

5. **DECIPHERING DATA**

Dataset is presented in the form of  three .csv files and two folders as described below

- .csv file containing image class
- Folder with training images
- Folder with testng images
- .csv file containing training annotations
- .csv file containing testing annotations

Understanding the data is explained below for each of the above items

### 5.1 Image class file and image folders

Upon reading the .csv file and checking the shape of data, we see that there are 196 rows of data, with no null values. Single column displays the different image class (car models)

```
                         0
0  AM General Hummer SUV 2000
1         Acura RL Sedan 2012
2         Acura TL Sedan 2012
3        Acura TL Type-S 2008
4        Acura TSX Sedan 2012

Number of rows,columns: (196, 1)

Checking for null values:
0    False
dtype: bool
```

Checking the number of folders under train & test category, we see the below result

```
[26] #Checking images folder
     fcount_train=(len(next(os.walk(pp_train))[1]))
     fcount_test=(len(next(os.walk(pp_test))[1]))
     print("Number of folders in training images:",fcount_train,"\n")
     print("Number of folders in testing images:",fcount_test)

     Number of folders in training images: 196

     Number of folders in testing images: 196
```

From above, we infer that total number of image classes: 196

## 5.2 Train & test annotation files

Top 5 rows in train file

```
          0    1    2     3     4    5
0  00001.jpg   39  116   569   375   14
1  00002.jpg   36  116   868   587    3
2  00003.jpg   85  109   601   381   91
3  00004.jpg  621  393  1484  1096  134
4  00005.jpg   14   36   133    99  106

Number of rows,columns: (8144, 6)

Checking for null values:
0    False
1    False
2    False
3    False
4    False
5    False
dtype: bool
```

Top 5 rows in test file

```
          0    1    2    3    4    5
0  00001.jpg   30   52  246  147  181
1  00002.jpg  100   19  576  203  103
2  00003.jpg   51  105  968  659  145
3  00004.jpg   67   84  581  407  187
4  00005.jpg  140  151  593  339  185

Number of rows,columns: (8041, 6)

Checking for null values:
0    False
1    False
2    False
3    False
4    False
5    False
dtype: bool
```

Number of train & test images is split as 8144 and 8041 respectively. Column 0 is evidently the file name of the image. Columns 1 to 4 are assumed to contain the two coordinates of the bounding box origin and diagonal point opposite to it. This can be confirmed only while checking and mapping the images.

Column 5 is assumed to be a number or ID, linking the train / test data to its image class in the other .csv file.. This is verified by counting unique values in column 5.

```
[45] # Assuming column index 5 represents image class (a number/ID)
     # Checking for non-repettiive (unique) data in column 5 of training & testing csv
     print(train.head())
     print("Number of unique values in last column of training data:",len(train[5].unique()))
     print("\n",test.head())
     print("Number of unique values in last column of testing data:",len(test[5].unique()))

          0    1    2     3     4    5
0  00001.jpg   39  116   569   375   14
1  00002.jpg   36  116   868   587    3
2  00003.jpg   85  109   601   381   91
3  00004.jpg  621  393  1484  1096  134
4  00005.jpg   14   36   133    99  106
Number of unique values in last column of training data: 196

           0    1    2    3    4    5
0  00001.jpg   30   52  246  147  181
1  00002.jpg  100   19  576  203  103
2  00003.jpg   51  105  968  659  145
3  00004.jpg   67   84  581  407  187
4  00005.jpg  140  151  593  339  185
Number of unique values in last column of testing data: 196
```

The unique values count to 196 which matches with the number of classes. Hence, Column 5 data represent index of the image class in train & test annotation files

### 5.3 Collating image class, file paths with train & test annotation files

Before proceeding to EDA, it is essential to check & create file paths using image class file and collating with test and train annotation files. This is done in two steps

#### 5.3.1 String compare image class names and folder names

Using the OS library, a list is generated containing folder names under both test and train category.

```
# Checking if folder names match with image class ID
dirlist = [ item for item in os.listdir(pp_train) if os.path.isdir(os.path.join(pp_train, item)) ]
for c in range(len(names)):
  for d in range(len(names)):
     if names.at[c,0]==dirlist[d]:
       names.at[c,1]="Matches with folder name"
       break
prob=np.where(names[1].isnull())[0][0]
print("List of folder names:\n", dirlist)
```

```
List of folder names:
 ['Acura RL Sedan 2012', 'Acura ZDX Hatchback 2012', 'Aston Martin Virage Convertible 2012', 'Acura TSX Sedan 2012',
```

String comparison of image class in .csv file is done with above list. In case of any anomaly, the same is displayed and corresponding entry is updated with closest matching folder name with help of 'difflib' library

```
    sol=get_close_matches(classes.at[prob,0],dirlist)
    print("Class updated to nearest folder match:",sol[0])
    classes.at[prob,0]=sol[0]
    classes.at[prob,1]="Updated with nearest match"
    classes.tail(25)
```

```
List of folder names:
 ['Acura RL Sedan 2012', 'Acura ZDX Hatchback 2012', 'Aston Martin Virage Convertible 2012',

 Class mismatch with folder name: Ram C/V Cargo Van Minivan 2012
Class updated to nearest folder match: Ram C-V Cargo Van Minivan 2012
```

| | 0 | 1 |
|---|---|---|
| 171 | Plymouth Neon Coupe 1999 | Matches with folder name |
| 172 | Porsche Panamera Sedan 2012 | Matches with folder name |
| 173 | Ram C-V Cargo Van Minivan 2012 | Updated with nearest match |

One anomaly was found in train annotation file for the image class "`Ram C/V Cargo Van Minivan 2012`" and the same was updated

### 5.3.2 Creating file path using image class

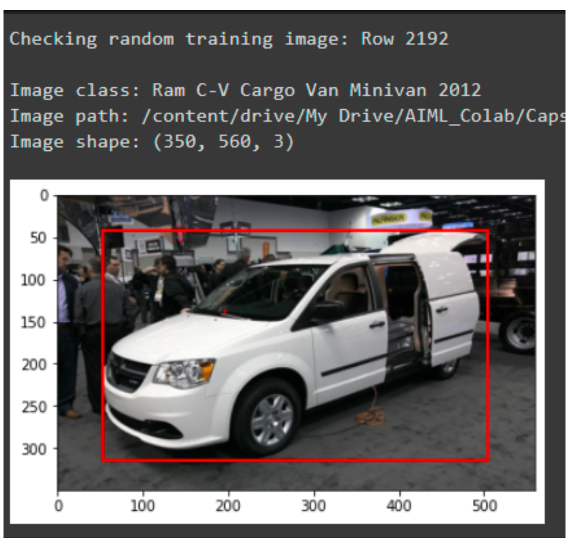
Using OS library, string concat is applied to file name, image class and project directory. The result is a single data frame (for each train and test data) as shown

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **8139** | 08140.jpg | 3 | 44 | 423 | 336 | 78 | Chrysler Town and Country Minivan 2012 | /content/drive/My Drive/AIML_Colab/Capstone/da... |
| **8140** | 08141.jpg | 138 | 150 | 706 | 523 | 196 | smart fortwo Convertible 2012 | /content/drive/My Drive/AIML_Colab/Capstone/da... |
| **8141** | 08142.jpg | 26 | 246 | 660 | 449 | 163 | Mercedes-Benz SL-Class Coupe 2009 | /content/drive/My Drive/AIML_Colab/Capstone/da... |
| **8142** | 08143.jpg | 78 | 526 | 1489 | 908 | 112 | Ford GT Coupe 2006 | /content/drive/My Drive/AIML_Colab/Capstone/da... |
| **8143** | 08144.jpg | 20 | 240 | 862 | 677 | 17 | Audi 100 Sedan 1994 | /content/drive/My Drive/AIML_Colab/Capstone/da... |

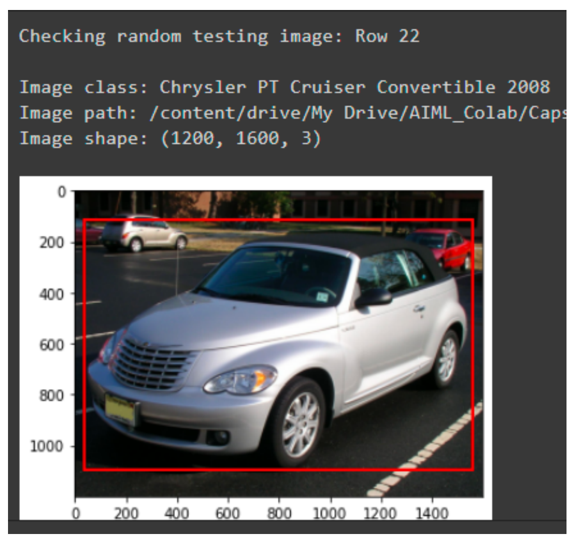
### 5.4 Checking random test and train images

Using scikit image & Matplotlib libraries, random train & test image are displayed, taking the row index (0 to 8143 for train) & ( 0 to 8040 for test) as input. Along with the image, its class and size are displayed

Random train image




Random test image




Images are successfully displayed & mapped to its class. This also proves the initial assumption that Columns 1 to 4 represent the bounding box geometry only.

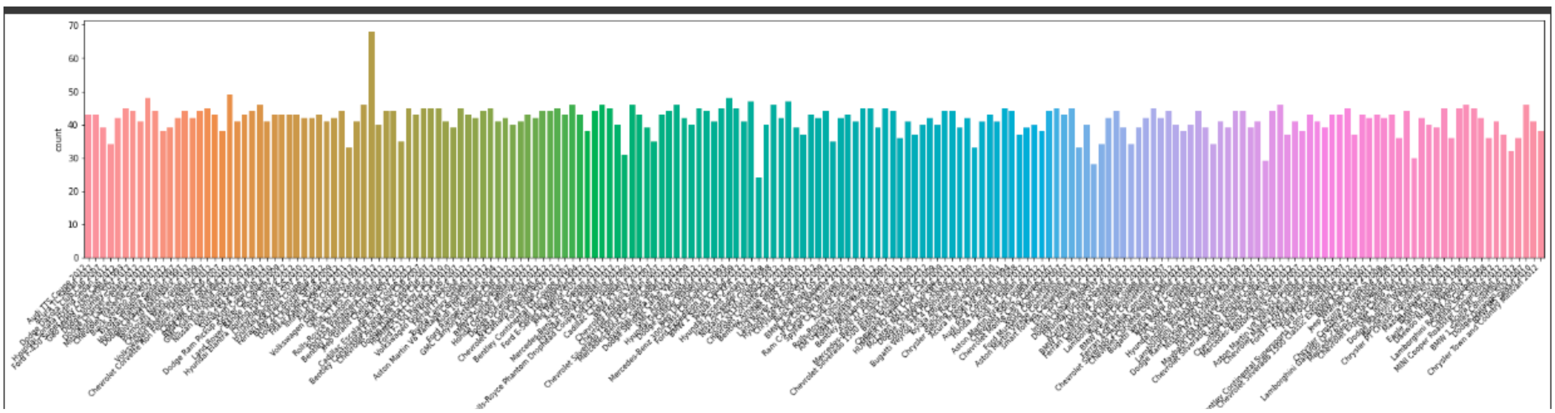
## 6. **EDA**

Image class - a categorical variable is the key focus in this EDA. The following plots are generated to see if they can reflect any meaningful inferences.

- Distribution of images amongst the classes
- Estimating number of car manufacturers
- Number of classes per car manufacturer

### 6.1 Distribution of images amongst the classes

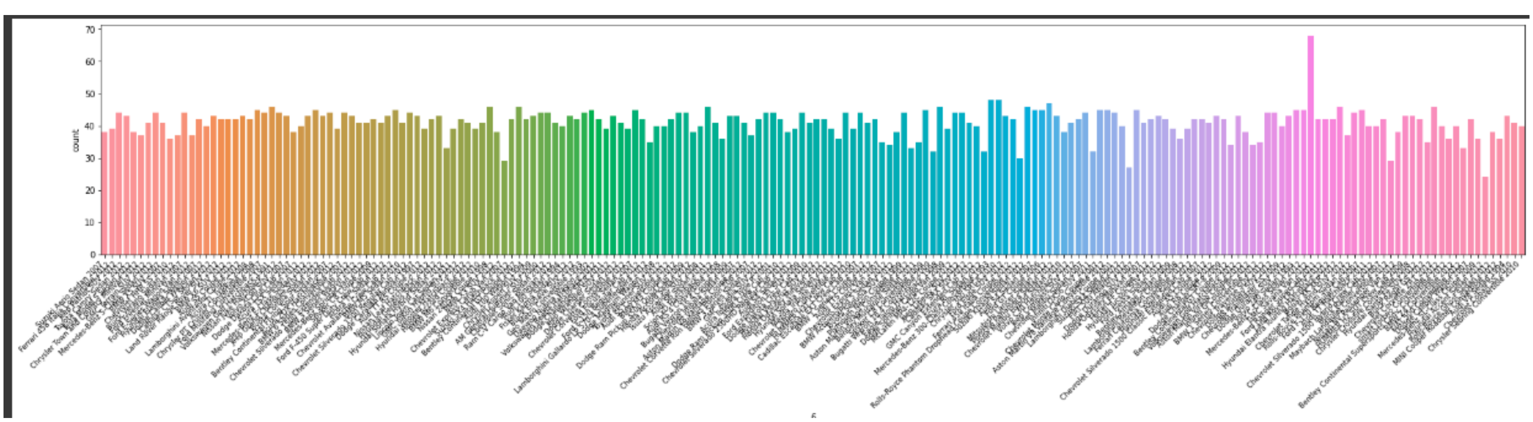
**Training data**




Class having highest number of images: `GMC Savana Van 2012    68 images`

Class having lowest number of images: `Hyundai Accent Sedan 2012    24 images`

**Testing Data**




Class having highest number of images: `GMC Savana Van 2012    68 images`

Class having lowest number of images: `Hyundai Accent Sedan 2012    24 images`

### 6.2.Estimating number of car manufacturers

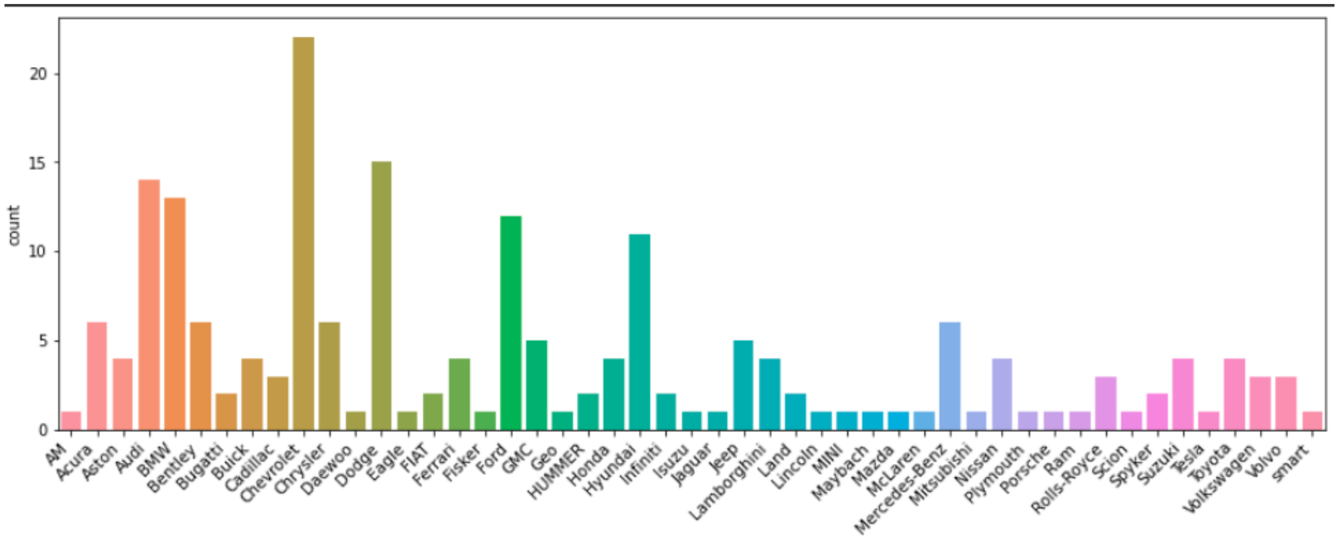
Using the split() function, car manufacturer name is extracted from the image class.

```
# Number of car manufacturers and distribution of car models / classes amongst different manufacturers
for k in range(0,len(classes)):
  temp1=classes.at[k,0].split()
  classes.at[k,2] = temp1[0]
print("Total number of car manufacturers in training dataset:",len(classes[2].unique()),"\n")

Total number of car manufacturers in training dataset: 49
```

Total number of car manufacturers considered in the dataset: 49

### 6.3 Number of classes per car manufacturer




Maximum number of images classes: `Chevrolet    22 image classes`

### 6.4 Summary of EDA

- Maximum & Minimum number of images per class is same for both training and testing dataset

  Class having highest number of images: `GMC Savana Van 2012    68 images`

  Class having lowest number of images: `Hyundai Accent Sedan 2012   24 images`

- Number of vehicle manufacturers considered in the dataset is 49, with Chevrolet having the highest number of image classes (22)

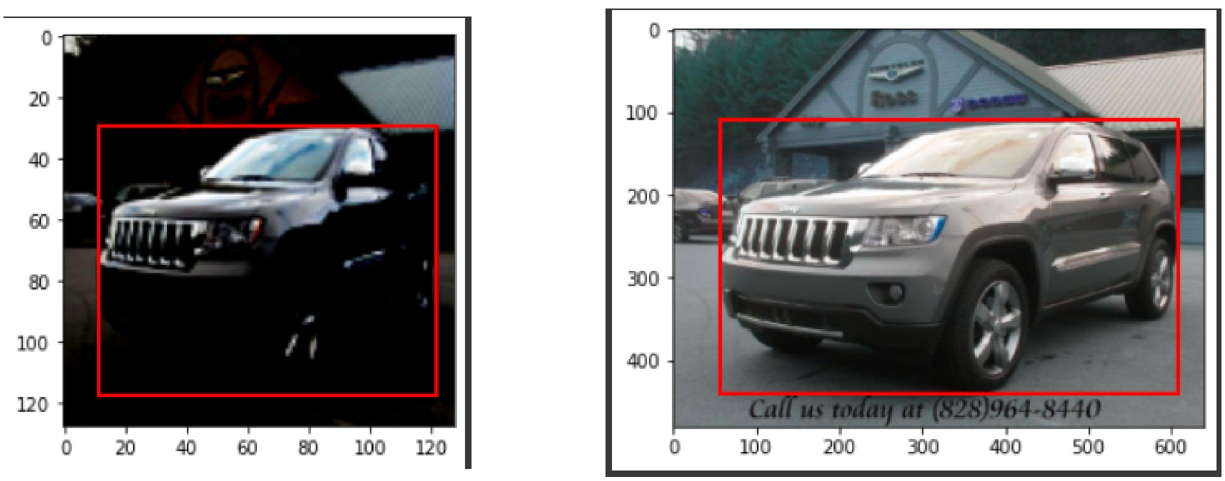
## 7. Data Pre-processing

### 7.1 Scaling the images

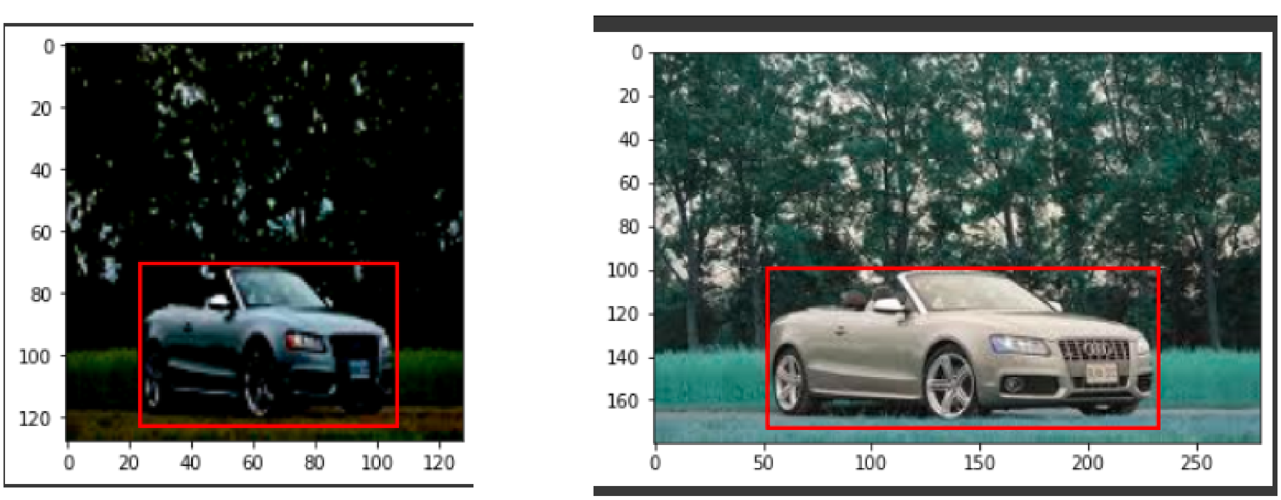
As mentioned in the literature review, MobileNet is used to build the model initially.

- Since the images are of different, they have to be scaled / normalized
- From the Mobilenet options (128, 160, 192, 224), image size of 128 is taken
- All train and test images are scaled to 128 x 128 size
- Random check (scaled vs unscaled) is done for any of the train & test images

Train image (scaled vs unscaled)




Test image (scaled vs unscaled)



### 7.2 Split train and test data into input & output variables

Important prerequisite to model building is splitting the train & test data in to input and output variables.

- RGB values of each image are saved as an Ndarray (X or input variable)
- The image class and bounding box details are saved as Ndarray (Y or output variable)

Train variables

```
print(X_train_sc.shape)
print(y_train_sc.shape)
```

```
(8144, 128, 128, 3)
(8144, 5)
```

Test variables

```
print (y_test_sc.shape)
print(X_test_sc.shape)
```

```
(8041, 5)
(8041, 128, 128, 3)
```

## 8. Model Building

From Tensorflow. Keras, MobileNet library is imported. Hyper parameter ALPHA is set as 1

```
[ ] # Model Creation (is in progress)
    ALPHA = 1.0 # Width hyper parameter for MobileNet (0.25, 0.5, 0.75, 1.0). Higher width means more accurate but slower

    def create_model(trainable=True):
        model = MobileNet(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3), include_top=True, alpha=ALPHA) # Load pre-trained mobilenet

        for layer in model.layers:
            layer.trainable = trainable


        x0 = model.layers[-1].output
        x1 = Conv2D(4, kernel_size=4, name="coords")(x0)

        x2 = Reshape((5,))(x1)

        return Model(inputs=model.input, outputs=x2)
```

Time taken for the initial run is close to 30 minutes. Further iterations to be made and metrics to be calculated.

## 9. Milestone-1 Summary

- Dataset contains 8144 train and 8041 test images.
- Annotation file contains bounding box origin and geometry details
- Total number of image classes: 196
- Random images are displayed and mapped to its respective classes
- From EDA, we see maximum and minimum of image classes belong to "GMC Savanna Van 2012 (68 images) and "Hyundai Accent Sedan 2012" (24 images) respectively. Further, total number of car manufacturers considered in this dataset are 49, with maximum image classes from Chevrolet (22 classes)
- To begin with, MobileNet is chosen as the model. Accordingly all images are scaled to 128 x 128 image size.
- From train & test data, image details stored in Ndarray is taken as input (X) & bounding box details, image class taken as output (Y) variables.
- Model building is in progress.