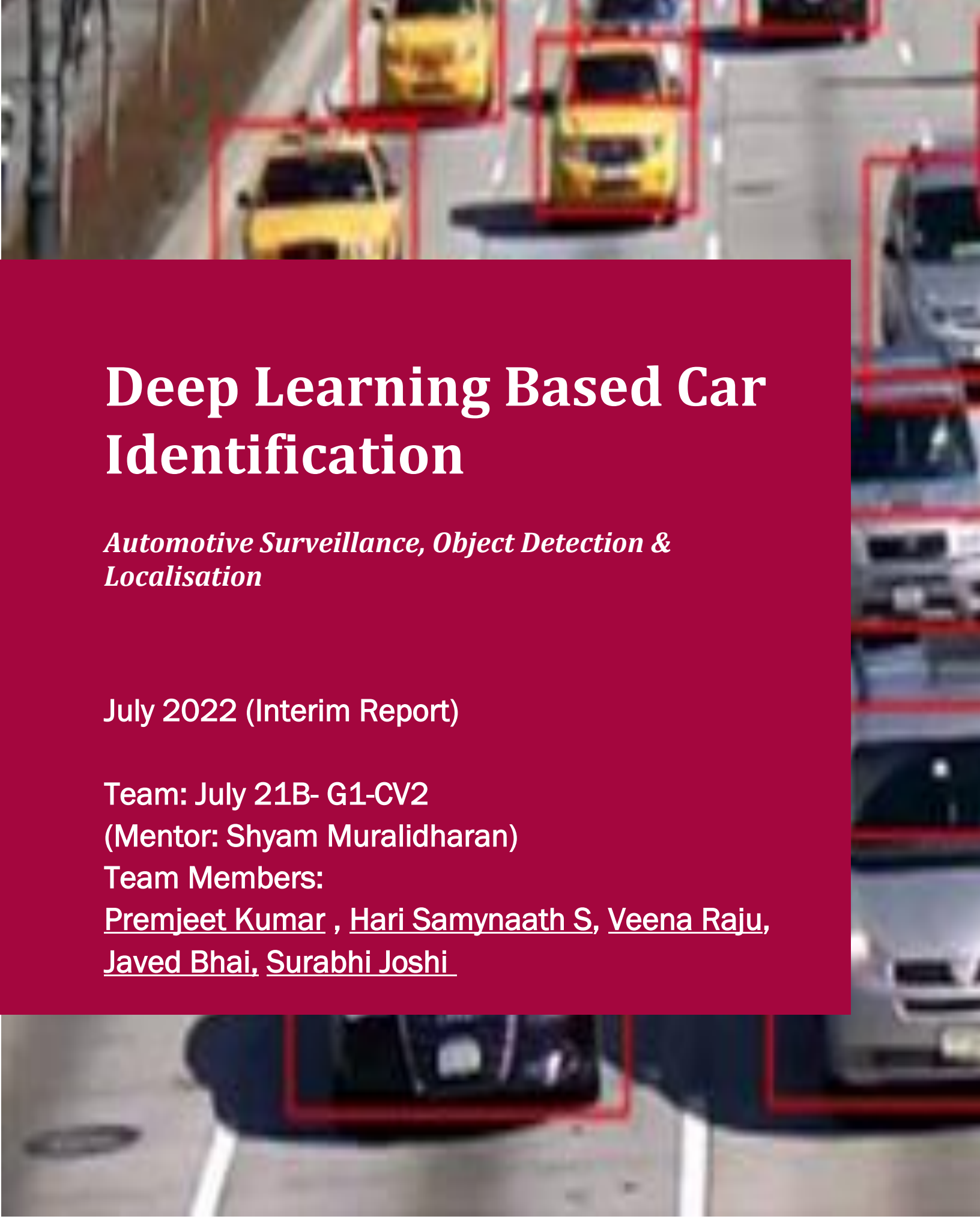


Deep Learning Based Car Identification

Automotive Surveillance, Object Detection & Localisation

July 2022 (Interim Report)

Team: July 21B- G1-CV2
(Mentor: Shyam Muralidharan)
Team Members:
Premjeet Kumar , Hari Samynaath S, Veena Raju, Javed Bhai, Surabhi Joshi

Table of Contents

Abstract

This interim report focusses on design and deployment of a multiclass object detection model for cars which enables identification moving cars on the road by a camera as make, type, model and OEM. The aim is to build a deployable deep learning model for car classification and prediction.
This report primarily focusses on initial data understanding, processing and trial runs towards model design. The content includes results from Exploratory Data Analysis (EDA) and Data preprocessing. The EDA to understand the biases in the training data along with data preprocessing including image resizing and bounding box creation was performed on available Stanford Car database. The report further details steps for selection, design and implementation of a preliminary car detection model, its shortcomings and future work to be accomplished in final report. There are four key finding from the report
1) The training data has intrinsic biases in terms of frequency distribution for OEM, Car Type and Car make year
2) The size and number of input images are not consistent and would require resizing and augmentation
3) Images vary in background and resolution so bounding boxes are essential
4) Simple models like mobile net would not be sufficient for creating an effective model, so more advanced and comprehensive model would need to designed for achieving reliable performance from the model

1 INTRODUTION

Summary of problem statement, data and findings

This project is aimed at creating multiclass object detection model for car detection and classification. The Stanford car dataset is used for the model building. Full lifecycle of model building including image sampling and augmentation, model selection, training, testing and validation is performed by using a preliminary model. We find that the preliminary model has much higher training accuracy than testing and would need drastic performance improvements. Spaces of improvement are identified for improving performance of model with setting up of pipeline to test multiple algorithms efficiently.

Introduction

Object detection is a computer vision technique in which a software system can detect, locate, and trace the object from a given image or video. The special attribute about object detection is that it identifies the class of object (person, table, chair, etc.) and their location-specific coordinates in the given image. The location is pointed out by drawing a bounding box around the object. The bounding box may or may not accurately locate the position of the object. The ability to locate the object inside an image defines the performance of the algorithm used for detection.

Problem Statement:

Computer vision-based models can be effectively used for object detection in real time. Ability to easily identify a moving vehicle on road through camera can go a long way in automating road supervision and surveillance for various business and law enforcement purposes. Further, these models can be integrated with other network systems for generation of appropriate actions triggers. Designing and building a computer vision model which can be used as vehicle recognition predictive models or car classification models can provide an effective solution for various vehicle detection application.

Objective

To design a deep learning-based car identification model that can be deployed to automate detection, identification and surveillance of cars on road for various business and law enforcement purposes. The model will enable identification of car moving on the road by a camera as make, type, model and OEM.

Data sources

The car detection model will be prepared using The Stanford Cars dataset, which is developed by Stanford University AI Lab specifically to create models for differentiating car types from each other.

The Cars dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. Classes are typically at the level of Make, Model, Year, e.g. 2012 Tesla Model S or 2012 BMW M3 coupe.

Data description:

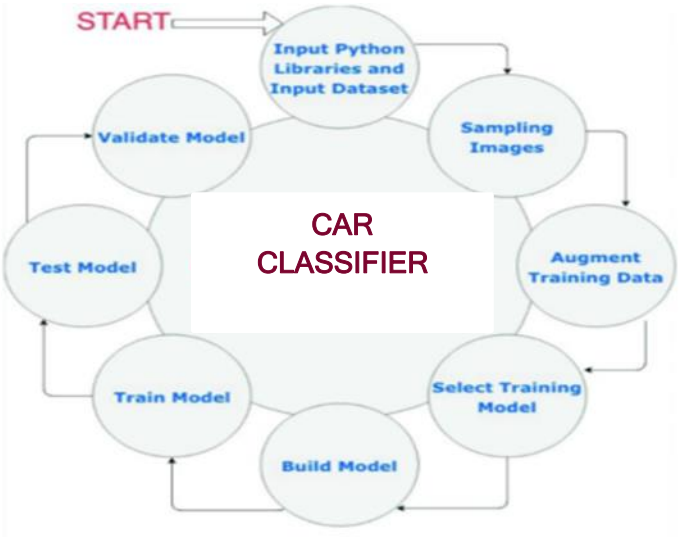
Train Images: Consists of real images of cars as per the make and year of the car.
Test Images: Consists of real images of cars as per the make and year of the car.
Train Annotation: Consists of bounding box region for training images.
Test Annotation: Consists of bounding box region for testing images.

The useful data to create the model is available in three files including two zipped folder.

S No.	File name	Format	Size	Details
1	Car Images	. zip		The .zip folder includes two sub folders Train and Test. Each of these subfolders have 196 class folder with .jepg images of folders
2	Annotations	. zip		CSV files in two folders train and test with bounding box coordinates
3	Car names & make	.CSV		Single CSV file with car name and makes indeed with 196 car classes

Problem Approach

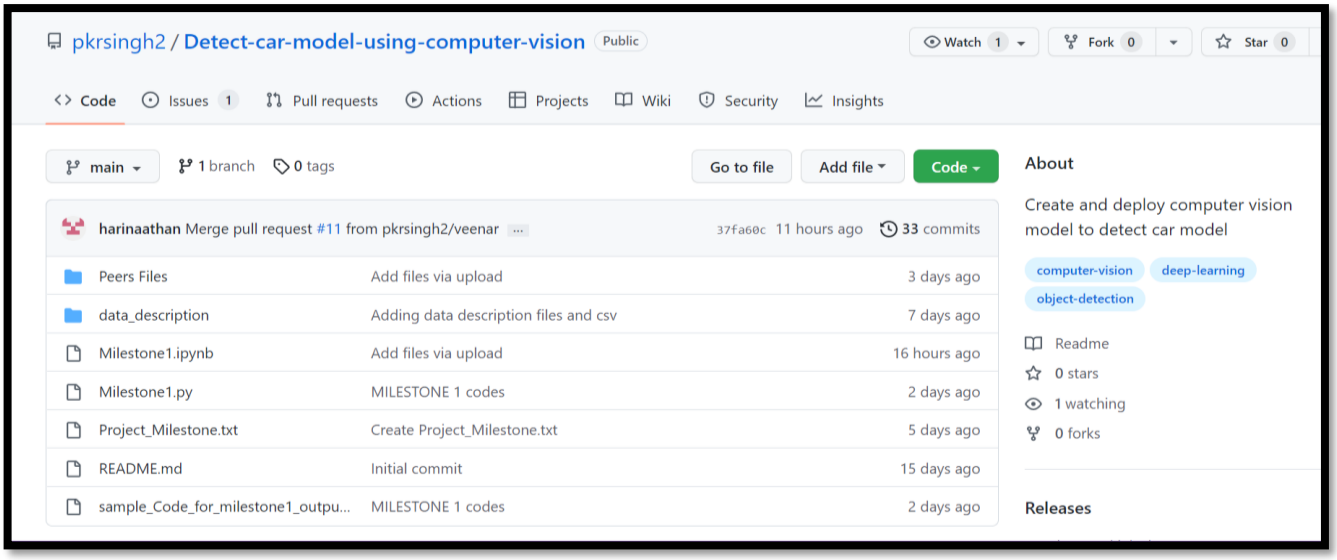
The key steps for project implementation includes setting up a pipeline for data extraction , exploratory data analysis to understand the available data, preprocessing , model building , model assessments , model improvement for best model selection followed by deployment



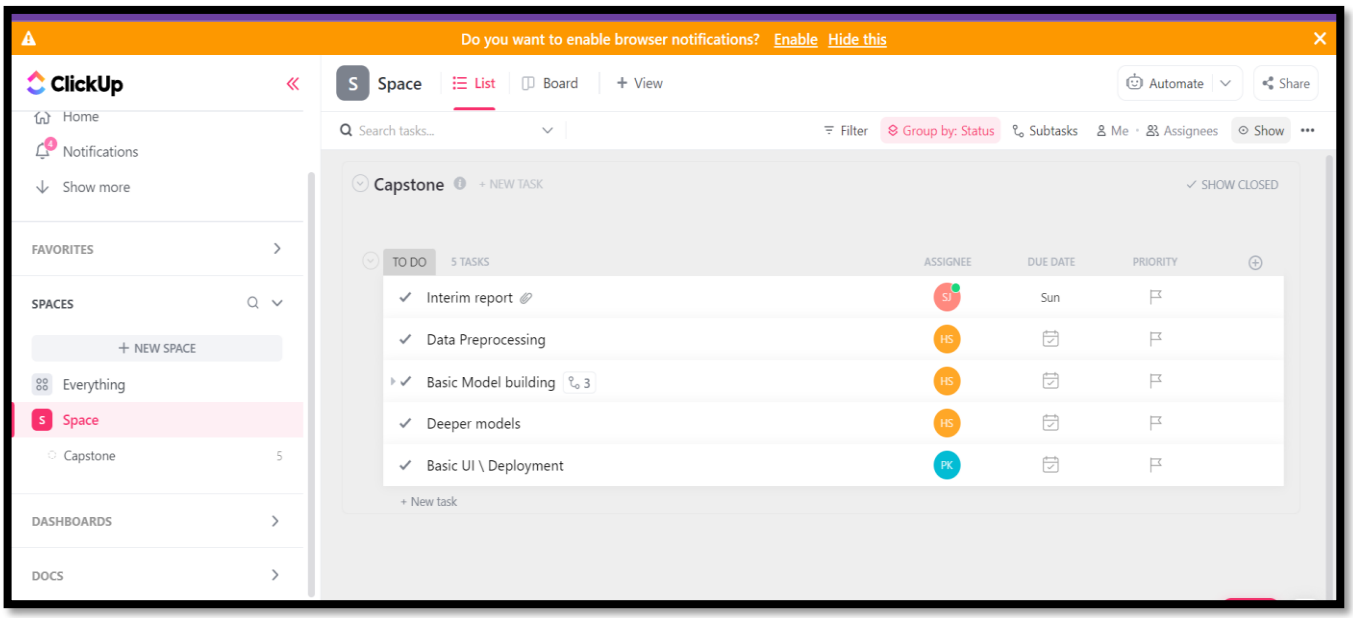
Project Management and managing team work flow

This project requires a constant collaboration and sharing along with task differentiation and allocation amongst the team members for timely execution and delivery of interim and final targets . The team under the able guidance of mentor Mr. Shyam Muralidharan , has created a GitHub repository (https://github.com/pkrsingh2/Detect-car-model-using-computer-vision) .

Github repository for collaborative sharing and managing the work flow



In Click Up main activity dashboard created and activities and task progress monitored.



Further the team communicates and coordinates on daily basis for effective sharing and learning over the entire project life cycle and managing the work flow.

In the next section we detail the steps and insights for data extraction and exploration

2 EDA & Pre-processing

Summary of the Approach to EDA and Pre-processing

The EDA was performed on the data set. Visual review revealed there are no missing data but inconsistencies in terms of number of images for each car type. The frequency distribution was initially performed with respect to distribution of classes and number of images for OEM, car type and year of make. This was followed by combined analysis of OEM, type, make year Following insights were revealed

1) *Chevrolet has highest number of classes in the data set*
2) *Sedan has highest number of classes in car type*
3) *2012 is the predominant make year available in the data.*
4) *A combined analysis for car make, type and year together revealed a very different scenarios*
5) *Number of training images are highest in case of GMC, Hyundai and Jeep reaching as high as four times the median value centered near 50.*
6) *Along with Sedans SUVs also have class instances with much higher number of training images as compared to median*
7) *2012 car year make dominates even in combined analysis emerging as the strongest bias in the data*
8) *Further image vary from as large as 210 Mega **pixels** to 4.5 kilo pixels requiring image resizing.*
9) *All the images also have different back ground and resolution making bounding box for training the model essential and was performed for the data*

Data Extraction and Exploration

Necessary libraries were imported for analysis and the data files were imported using python note book. The data extraction included three steps

1.Reading the provided .CSV files and image files
2.Reviewing the name lengths and format in the CSV files for identifying different features i.e. OEM, Car type, year of make
3.Checking for any inconsistencies and missing data

The car name & make CSV file provides names of 196 classes of cars for which images are available for model training indexed with class number. The name provides a combined information in four categories car manufacturer, model name , car type and Car make year. The names and name length are initially studied

	fullNames
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

```
# lets review the name lengths
carsMaster["wCounts"] = carsMaster["fullNames"].apply(lambda x: len(x.split()))
carsMaster.wCounts.value_counts()
```

```
4    132
5     44
6     14
7      6
Name: wCounts, dtype: int64
```

We find that full name length range between 4 to 7 words in the data base with information provided for the four features OEM, Type, Model and make year in the respective order . Through multiple steps of data sorting we could segregate the features of Car entries in four different columns

As the key feature categories were extracted the data became comprehensible to carry a detailed exploratory data analysis

```
# lets drop & rearrange the master data
carsMaster = carsMaster[["fullNames","OEM","MODEL","TYPE","YEAR"]]
carsMaster.sample(15)
```

	fullNames	OEM	MODEL	TYPE	YEAR
87	Dodge Sprinter Cargo Van 2009	Dodge	Sprinter	Cargo_Van	2009
3	Acura TL Type-S 2008	Acura	TL_Type-S	UnKnown	2008
65	Chevrolet Cobalt SS 2010	Chevrolet	Cobalt	SS	2010
78	Chrysler 300 SRT-8 2010	Chrysler	300_SRT-8	SRT8	2010
53	Chevrolet Silverado 1500 Hybrid Crew Cab 2012	Chevrolet	Silverado_1500_Hybrid	Crew_Cab	2012
86	Dodge Ram Pickup 3500 Quad Cab 2009	Dodge	Ram_Pickup_3500	Quad_Cab	2009
149	Lamborghini Reventon Coupe 2008	Lamborghini	Reventon	Coupe	2008
74	Chevrolet Silverado 1500 Regular Cab 2012	Chevrolet	Silverado_1500	Regular_Cab	2012
187	Toyota Corolla Sedan 2012	Toyota	Corolla	Sedan	2012
8	Aston Martin V8 Vantage Coupe 2012	Aston_Martin	V8_Vantage	Coupe	2012
52	Cadillac Escalade EXT Crew Cab 2007	Cadillac	Escalade_EXT	Crew_Cab	2007
160	Mercedes-Benz 300-Class Convertible 1993	Mercedes-Benz	300-Class	Convertible	1993
163	Mercedes-Benz E-Class Sedan 2012	Mercedes-Benz	E-Class	Sedan	2012
64	Chevrolet Avalanche Crew Cab 2012	Chevrolet	Avalanche	Crew_Cab	2012
182	Suzuki SX4 Hatchback 2012	Suzuki	SX4	Hatchback	2012

EXPLORATORY DATA ANALYSIS

We do an overall review of heterogeneity in the data by understanding number of unique values for each variable

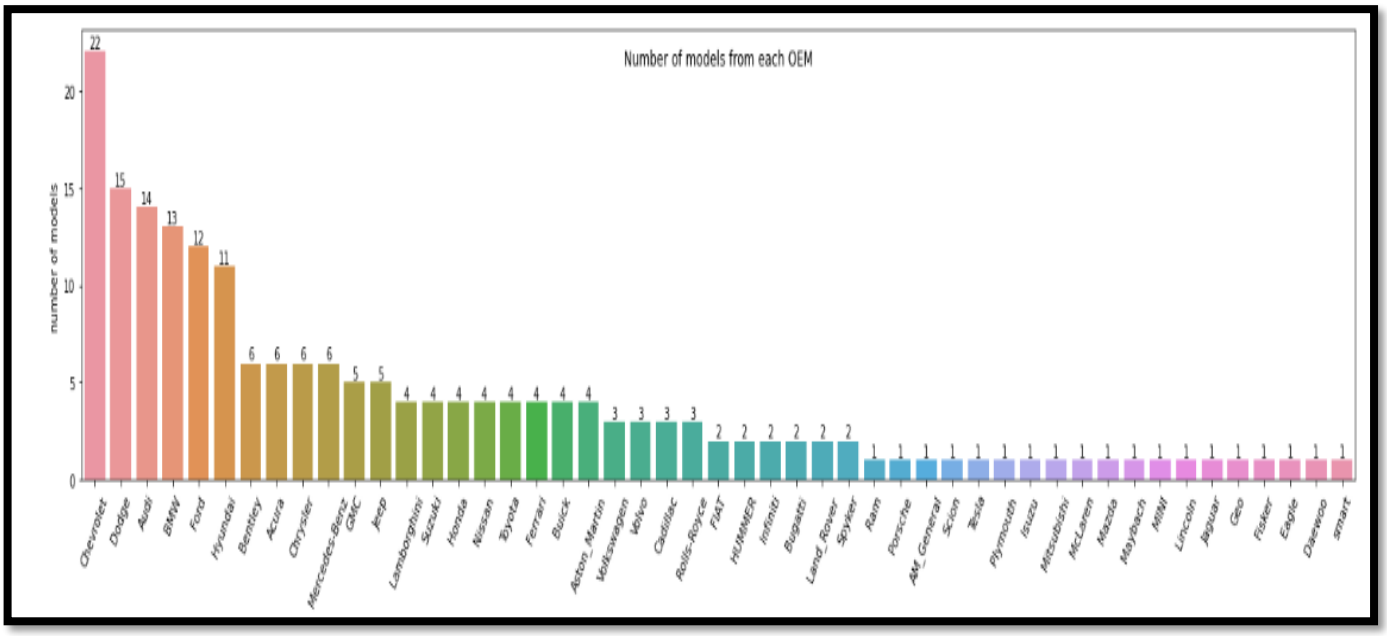
```
# review number of unique classes
print("Number of unique classes:")
print("OEMs :",carsMaster.OEM.nunique())
print("MODELs :",carsMaster.MODEL.nunique())
print("TYPEs :",carsMaster.TYPE.nunique())
print("YEARs :",carsMaster.YEAR.nunique())
```

```
Number of unique classes:
OEMs : 49
MODELs : 173
TYPEs : 23
YEARs : 16
```

We find that data includes 49 distinct OEM's with 173 different car models of 23 different type of cars. The data spans for 16 unique time periods

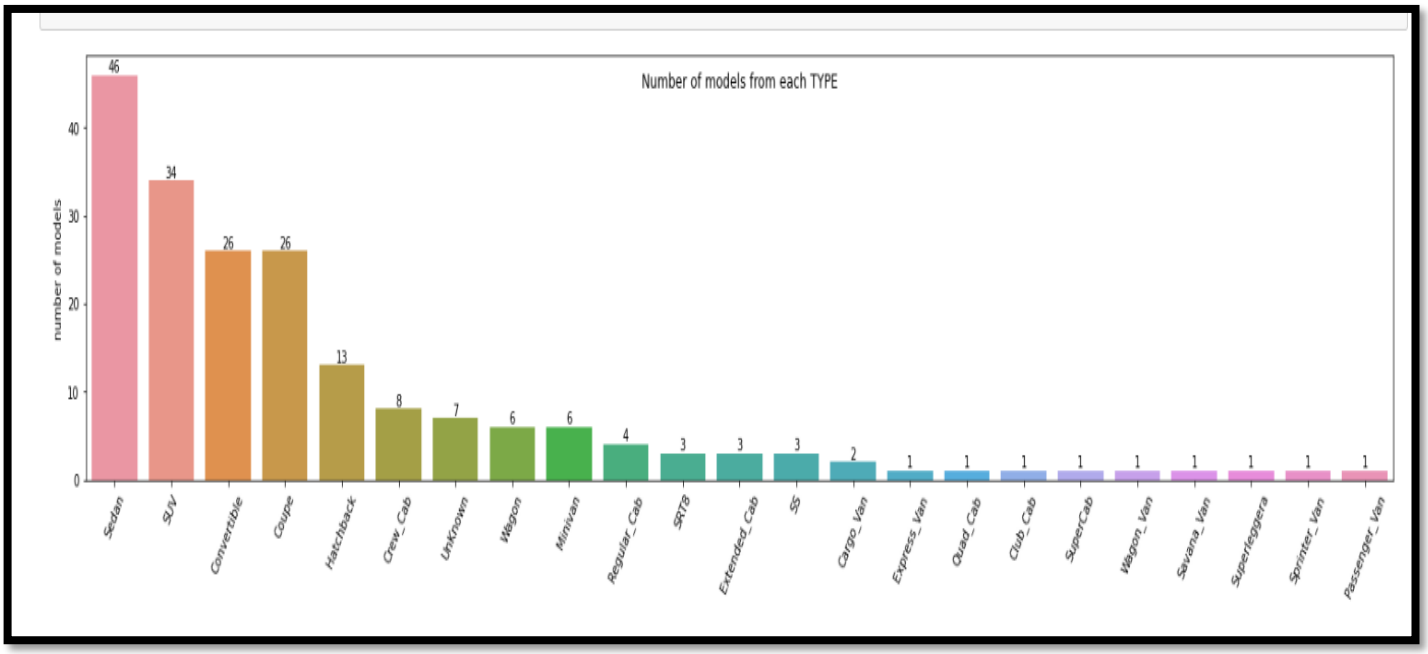
We now explore the frequency distribution and data distribution of various categories of cars WRT to OEM's , Type of cars , and year of manufacturing and plot the frequencies

Number of Models for each Car Manufacturer (OEM)



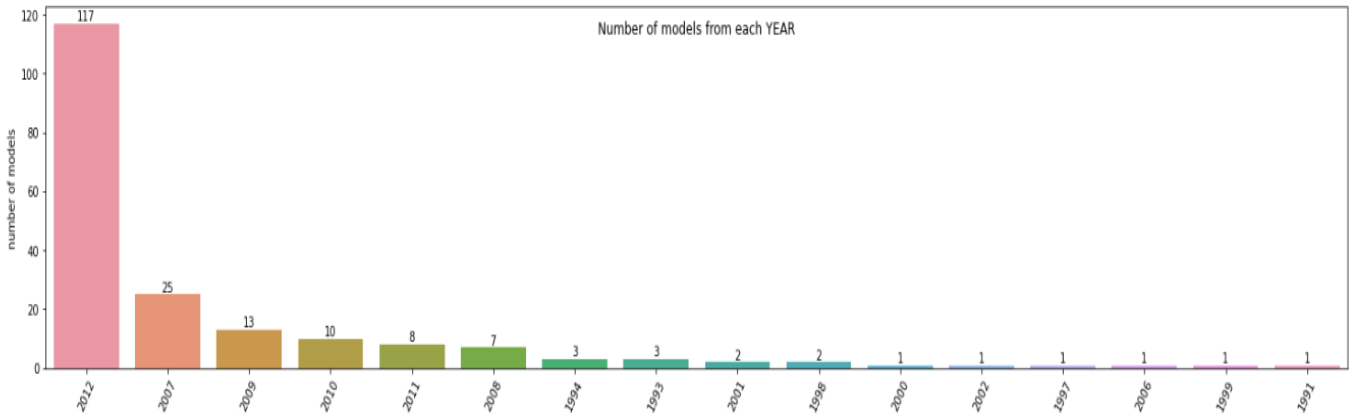
The data consists of 22 models for Chevrolet followed by Dodge (15), Audi(14), BMW(12) , Ford(12) and Hundai (11) which have more then 10 models in the data thus higher representation that significantly higher RAM , MADA etc which have only one model in the data

Number of Models for each Car Type



Most represented car type in data are Sedans which have 46 models in the current dataset

Number of Models for each make year



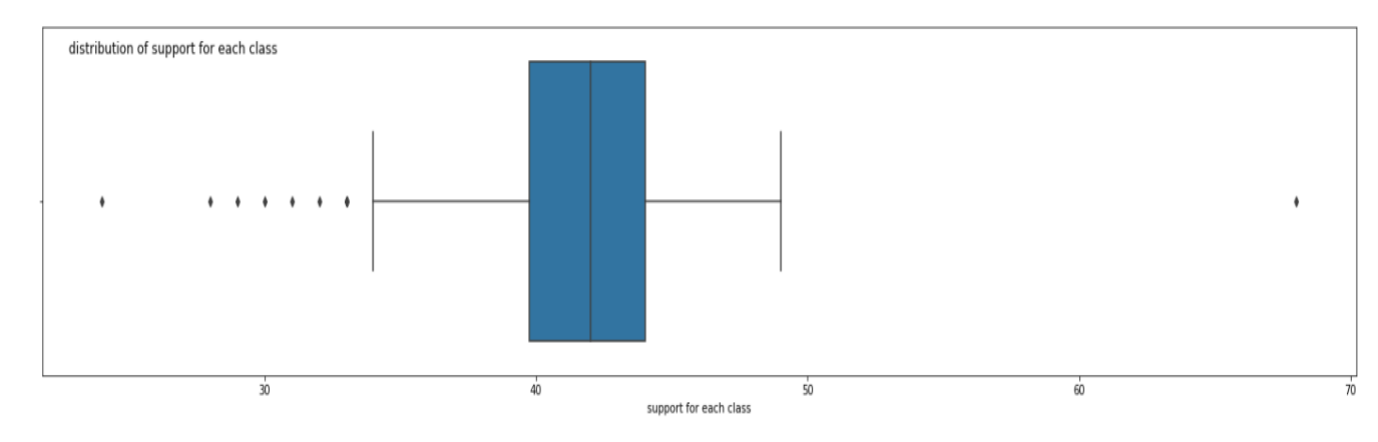
From the perspective of data distribution highest data frequency for make year in 117 , which emerges as a predominant bias in the existing dataset . From the above three parameters we get an insight that there is a high probability of instrinsic biases in terms of predominant OEM, type and year of manufacturing for the data .

We further explore this findings by going deeper into understanding the data distribution interms of available images for each category 196 classes of cars in the data set . For doing we do analysis in two steps .

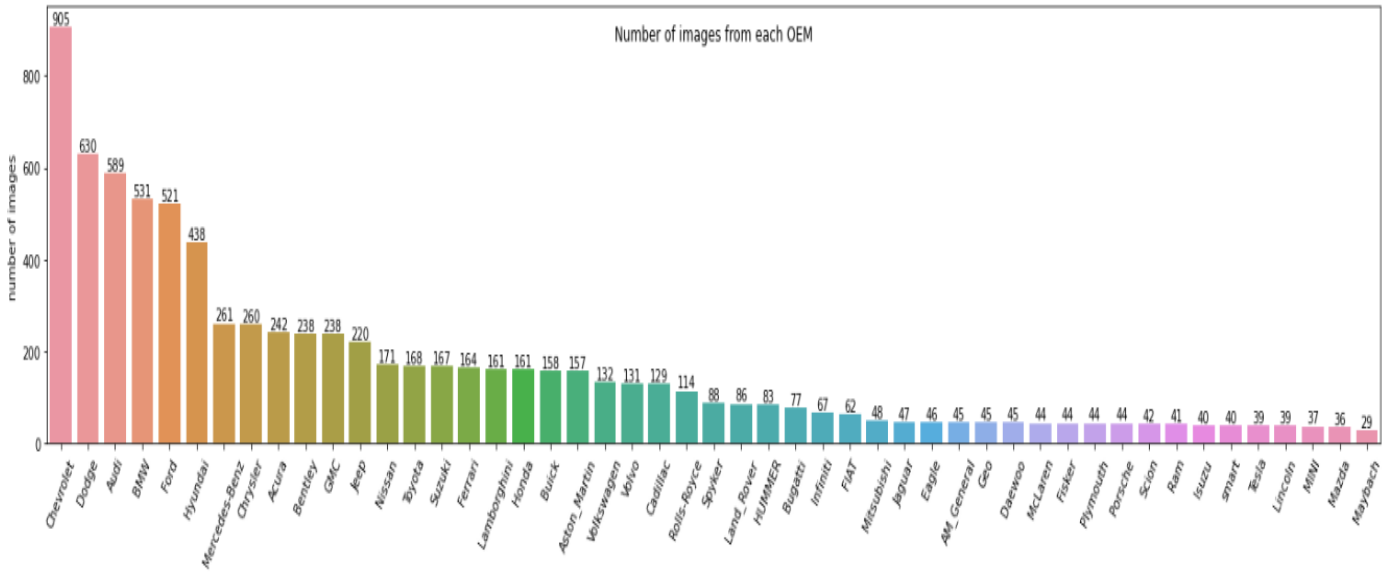
Data Distribution in terms of individual image counts

1) Step one we try to evaluate an median counts for images for training for each class as that will be crucial in determining the training biases for the model

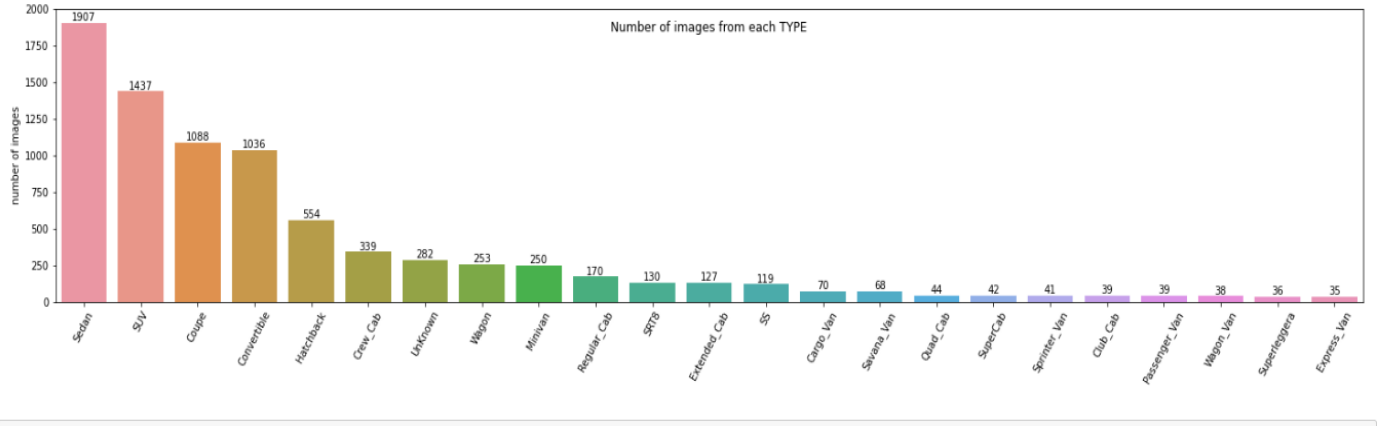
We determine the distribution support of each class and find that for most classes the total images available lie between 40-50 instances for training



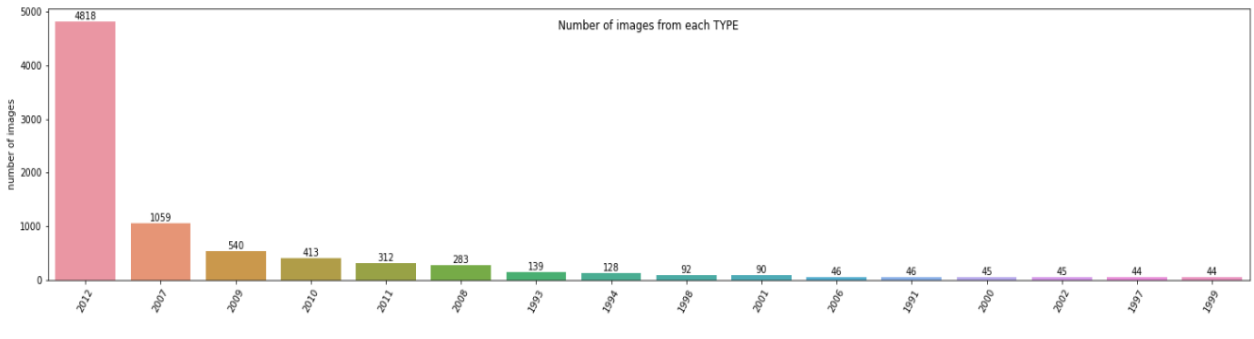
Number of images in each OEM



Number of images in each OEM



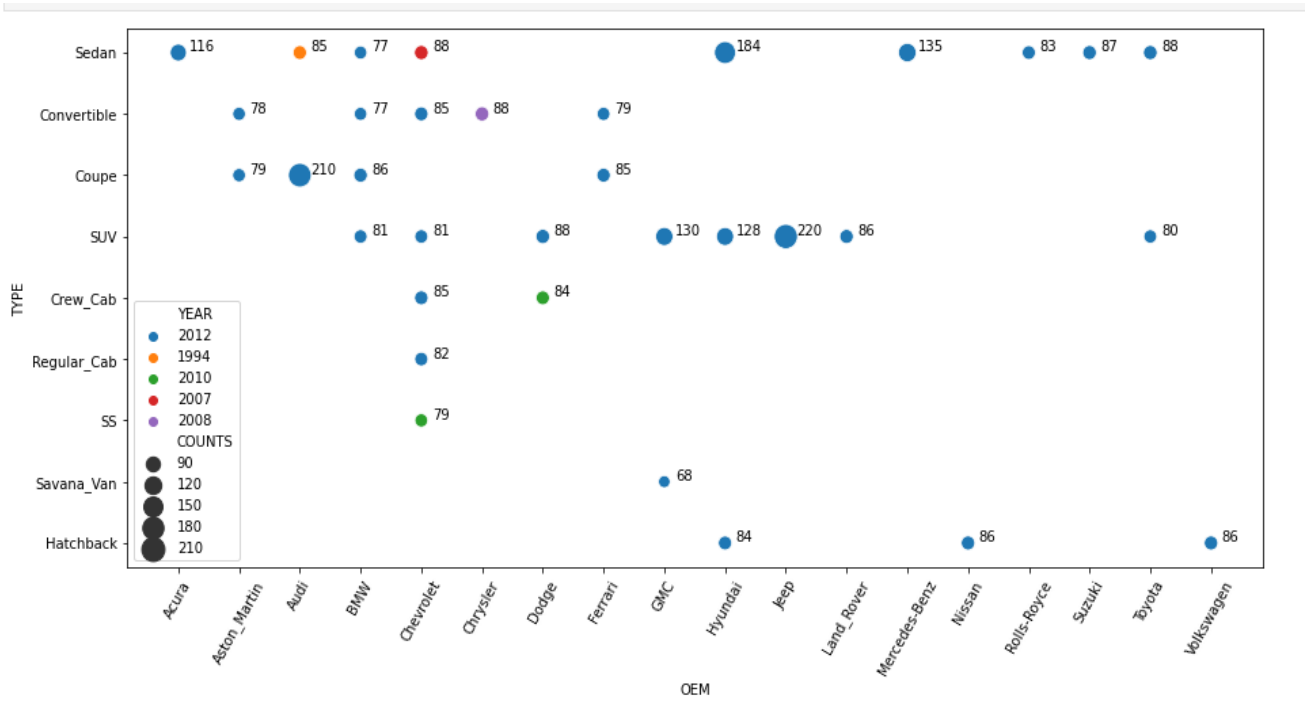
Number of images in each make year



In the above analysis the frequency distribution of images closely follows the class distribution with Chevrolet, sedan and 2012 make year emerging as the predominant data biases

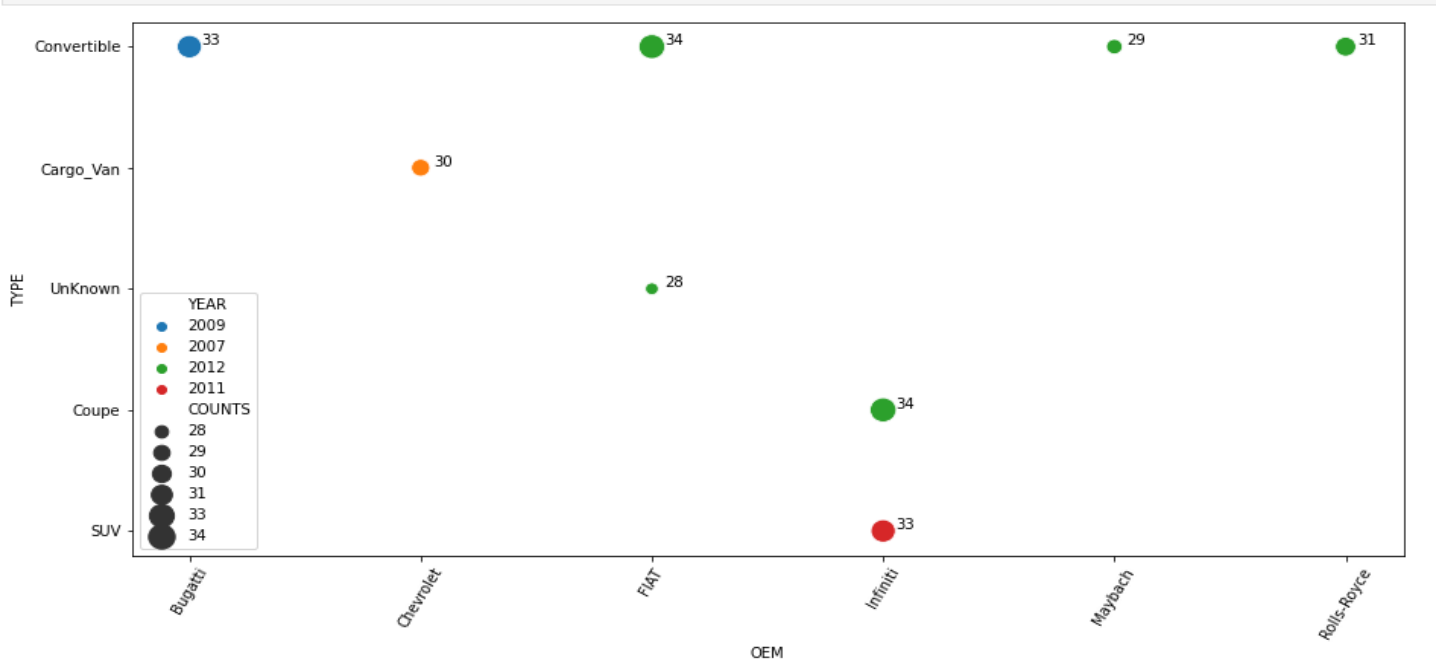
We further evaluate combined frequencies for various models as the inputs would be going in terms of 196 training classes (multiclass training)

Combined distribution of images wrt car type , OEM and car make year



We find that although Chervelotee has maimum images highest number of images for training are available for other OEMs like GMC, Hyundai , Jeep etc . Further highest number of training images are available SUVs with in a class thus Sedan alone do not form predominant image class at class level . Further the 2012 emerges as the predominant make year even in this combination analysis .

Further we try to understand distribution for lesser represented classes in terms of images



We find that lesser instances are found for cars like FIAT, May Batch , Rolls Royce but least image number in 28 which is for unknown type cars .

The above combination analysis highlight that

1. There are inherent biases in terms of car make year so the model that would design would be appropriate for lower range in terms of make year in terms of efficiency

Data Preprocessing

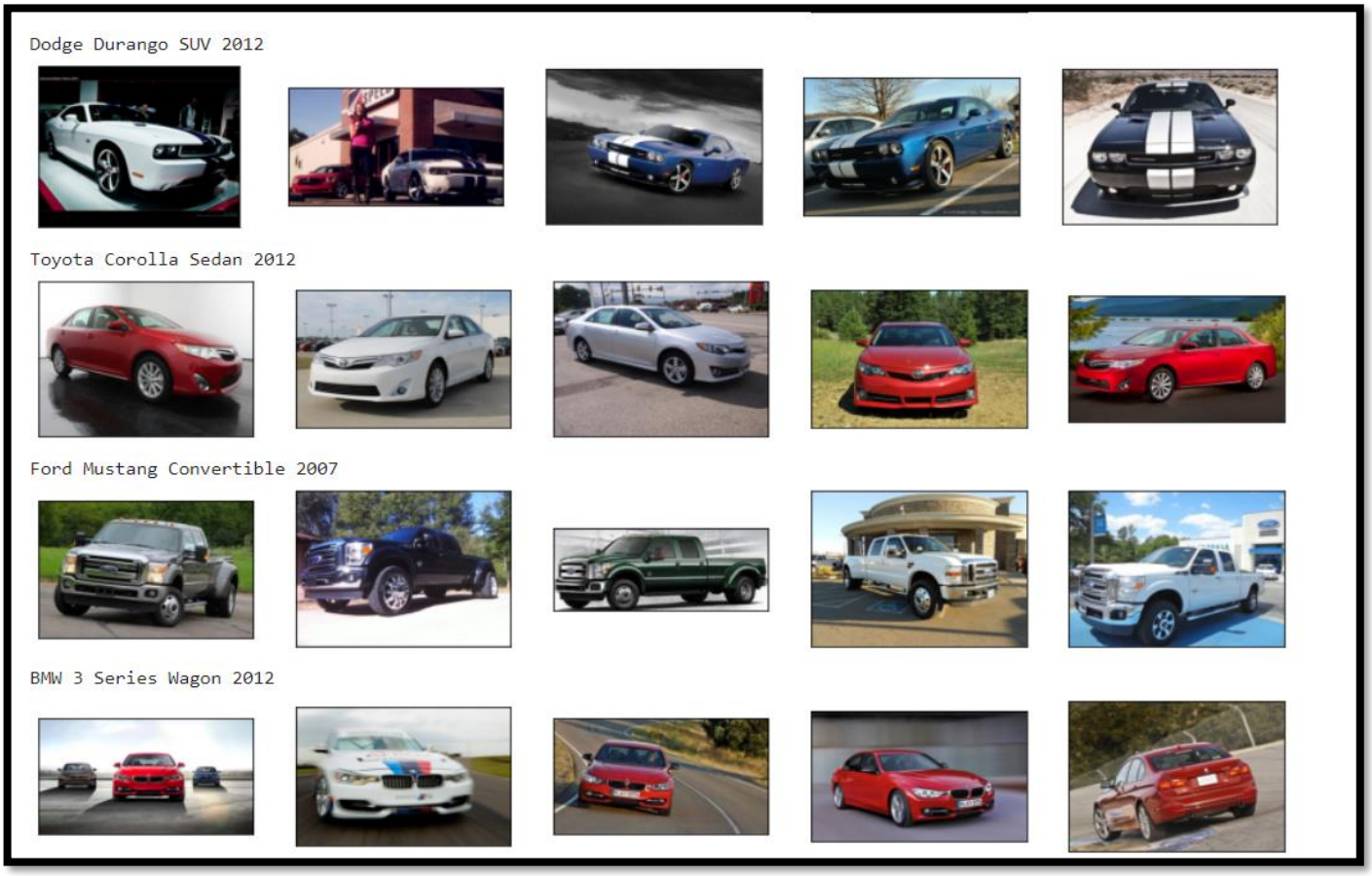
The acquired data from the real world is usually messy and come from different sources. To feed them to the machine learning or neural network-based model, they need to be standardized and cleaned up.

Preprocessing is more often used to conduct steps that reduce the complexity and increase the accuracy of the applied algorithm. As writing a unique algorithm for each of the condition

in which an image is taken would be very cumbersome and resource constraining thus, when an image is acquired, its first converted into a form that allows a general algorithm to solve it.

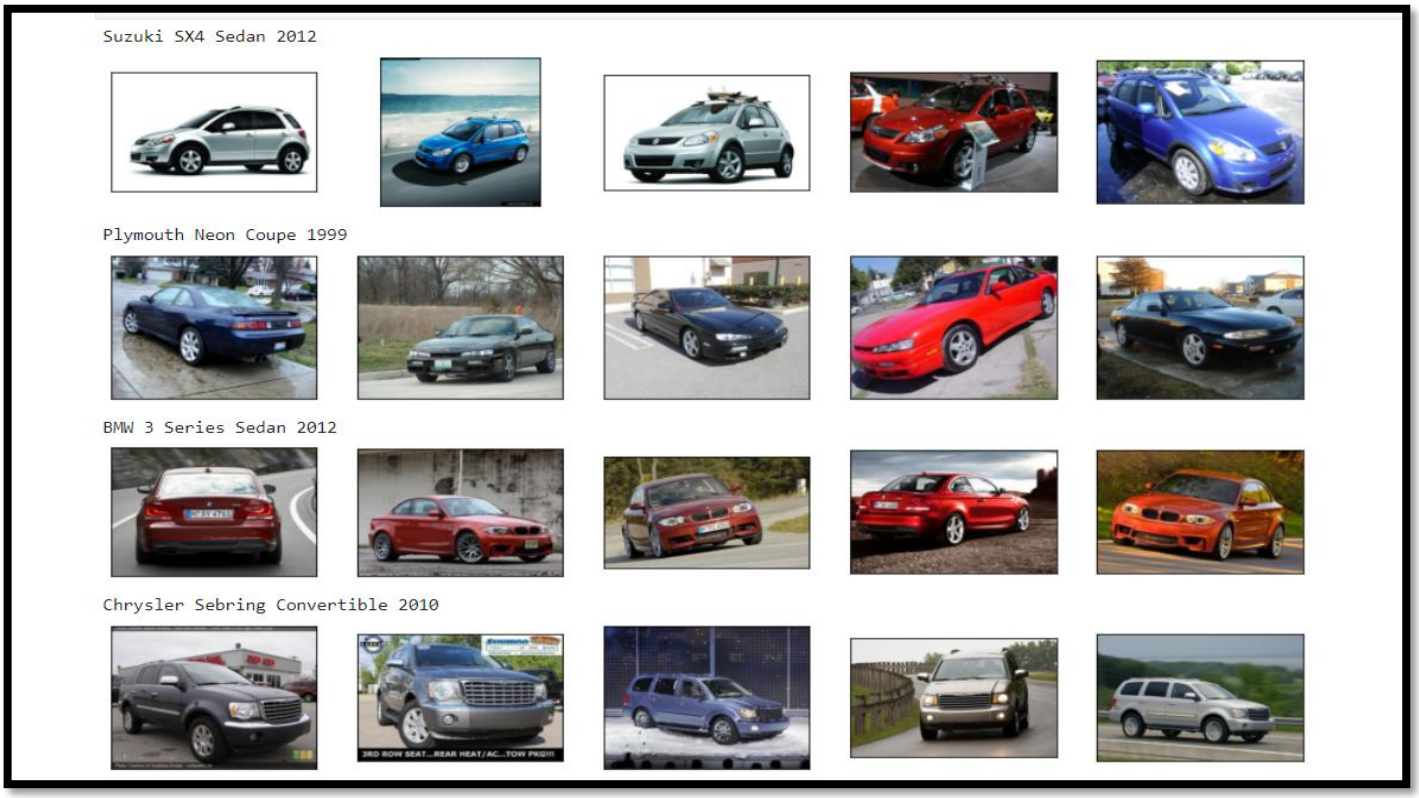
A manual survey of training and test images reveal that images have different backgrounds and resolution. Therefore, use of bounding boxes and image size normalization becomes essential

Snap shot Training Images



Snap shot Training Images

Snap shot Training Images



The data preprocessing included

1. Image size Normalization
2. Augmentation of the Bounding boxes

Image size Normalization

We initially compute image size and print image dimension in a separate column

Compute image size

Store the image size of height and width in new column called "pixels"

In [31]:
```
# compute image sizes
imageMasterTrain["pixels"] = imageMasterTrain.height * imageMasterTrain.width
imageMasterTest["pixels"] = imageMasterTest.height * imageMasterTest.width
```

Print Image dimensions

This will help to visualize the dimensions of the images in range

In [32]:
```
print("largest image:"),display(imageMasterTrain.loc[imageMasterTrain.pixels.argmax()].to_frame().T)
print("tallest image:"),display(imageMasterTrain.loc[imageMasterTrain.height.argmax()].to_frame().T)
print("widest image:"),display(imageMasterTrain.loc[imageMasterTrain.width.argmax()].to_frame().T)
print("\n")
print("smallest image:"),display(imageMasterTrain.loc[imageMasterTrain.pixels.argmin()].to_frame().T)
print("shortest image:"),display(imageMasterTrain.loc[imageMasterTrain.height.argmin()].to_frame().T)
print("leanest image:"),display(imageMasterTrain.loc[imageMasterTrain.width.argmin()].to_frame().T);
```

largest image:

	Image	ImagePath	folderName	height	width	pixels
2573	05945.jpg	Car Images/Train Images/Chevrolet Sonic Sedan ...	Chevrolet Sonic Sedan 2012	5616.0	3744.0	21026304.0

tallest image:

	Image	ImagePath	folderName	height	width	pixels
2573	05945.jpg	Car Images/Train Images/Chevrolet Sonic Sedan ...	Chevrolet Sonic Sedan 2012	5616.0	3744.0	21026304.0

widest image:

	Image	ImagePath	folderName	height	width	pixels
2573	05945.jpg	Car Images/Train Images/Chevrolet Sonic Sedan ...	Chevrolet Sonic Sedan 2012	5616.0	3744.0	21026304.0

smallest image:

	Image	ImagePath	folderName	height	width	pixels
2294	00097.jpg	Car Images/Train Images/Chevrolet Corvette Ron...	Chevrolet Corvette Ron Fellows Edition Z06 2007	78.0	58.0	4524.0

shortest image:

	Image	ImagePath	folderName	height	width	pixels
2294	00097.jpg	Car Images/Train Images/Chevrolet Corvette Ron...	Chevrolet Corvette Ron Fellows Edition Z06 2007	78.0	58.0	4524.0

leanest image:

	Image	ImagePath	folderName	height	width	pixels
5107	04047.jpg	Car Images/Train Images/Geo Metro Convertible ...	Geo Metro Convertible 1993	101.0	57.0	5757.0

The size of the images vary from as large as 210 Mega pixels to 4.5 kilo pixels

Resizing Images : Resizing images is a critical preprocessing step in computer vision. Machine Learning models train faster on smaller images and they need images of same size as input.

Some of the Best Practices

1. To decide on what should be the size of the images, a good strategy is to employ progressive resizing. eg; we can start with all images resized to the smallest one.

2. Progressive resizing will train an initial model with very small input images and gauge performance. We can use those weights as the starting point for the next model with larger input images.

3. Downsizing larger images to match the size of smaller images is often a better bet than increasing the size of small images to be larger.

4. In general, it is safer to maintain the raw image aspect ratio and resize proportionally.

5. Make use of image resizing methods like interpolation so that the resized images do not lose much of their perceptual character.

Initial Image Size

Based on above review, we shall restrict the image size fed to the network at 50x50 pixels, so as not to deteriorate lower resolution images and thus affect model capabilities

```
# display 5 random images of 5 random classes
classes = np.random.choice(imageMasterTrain.folderName.unique(),5,replace=False)
for cls in classes:
    dtmp = imageMasterTrain.loc[imageMasterTrain.folderName == cls]
    images = np.random.choice(dtmp.ImagePath.values,5,replace=False)
    plt.figure(figsize=(20,4))
    plt.suptitle(cls)
    for i,img in enumerate(images):
        img = Image.open(img).resize((200,200))
        plt.subplot(1,5,i+1)
        plt.imshow(img)
        plt.axis('off')
    plt.show()
```

After converting all images to same sie we still have the images with different backgrounds and resolutions . We now use the already available annotations to create bounding boxes.

Adding Bounding Boxes

i. Step 1 : Merge all the information of images, annotations, bounding box to single Data Frame

[36]:
```
# create all-consolidated dataframes
trainDF = pd.merge(imageMasterTrain,trainAnnot,how='outer',left_on='Image',right_on='Image Name')
testDF = pd.merge(imageMasterTest,testAnnot,how='outer',left_on='Image',right_on='Image Name')

display(trainDF.head(),testDF.head())
```

	Image	ImagePath	folderName	height	width	pixels	Image Name	x1	y1	x2	y2	Image class
0	04544.jpg	Car Images/Train Images/AM General Hummer SUV ...	AM General Hummer SUV 2000	339.0	200.0	67800.0	04544.jpg	18	18	328	190	1
1	00163.jpg	Car Images/Train Images/AM General Hummer SUV ...	AM General Hummer SUV 2000	700.0	525.0	367500.0	00163.jpg	46	84	661	428	1
2	00462.jpg	Car Images/Train Images/AM General Hummer SUV ...	AM General Hummer SUV 2000	85.0	64.0	5440.0	00462.jpg	5	8	83	58	1
3	00522.jpg	Car Images/Train Images/AM General Hummer SUV ...	AM General Hummer SUV 2000	94.0	71.0	6674.0	00522.jpg	6	7	94	68	1
4	00707.jpg	Car Images/Train Images/AM General Hummer SUV ...	AM General Hummer SUV 2000	700.0	439.0	307300.0	00707.jpg	26	32	677	418	1

	Image	ImagePath	folderName	height	width	pixels	Image Name	x1	y1	x2	y2	Image class
0	03246.jpg	Car Images/Test Images/AM General Hummer SUV 2...	AM General Hummer SUV 2000	101.0	41.0	4141.0	03246.jpg	9	3	93	41	1
1	00076.jpg	Car Images/Test Images/AM General Hummer SUV 2...	AM General Hummer SUV 2000	96.0	64.0	6144.0	00076.jpg	11	13	84	60	1
2	00457.jpg	Car Images/Test Images/AM General Hummer SUV 2...	AM General Hummer SUV 2000	250.0	144.0	36000.0	00457.jpg	31	20	226	119	1
3	00684.jpg	Car Images/Test Images/AM General Hummer SUV 2...	AM General Hummer SUV 2000	373.0	216.0	80568.0	00684.jpg	111	54	365	190	1
4	01117.jpg	Car Images/Test Images/AM General Hummer SUV 2...	AM General Hummer SUV 2000	800.0	600.0	480000.0	01117.jpg	45	39	729	414	1

ii. Step 2: Merge OEM, MODEL, Type, Year with the above data frame

In [37]:
```
# lets merge the OEM, MODEL, TYPE & YEAR data
trainDF = pd.merge(trainDF,carsMaster,how='outer',left_on='folderName',right_on='fullNames')
testDF = pd.merge(testDF,carsMaster,how='outer',left_on='folderName',right_on='fullNames')
```

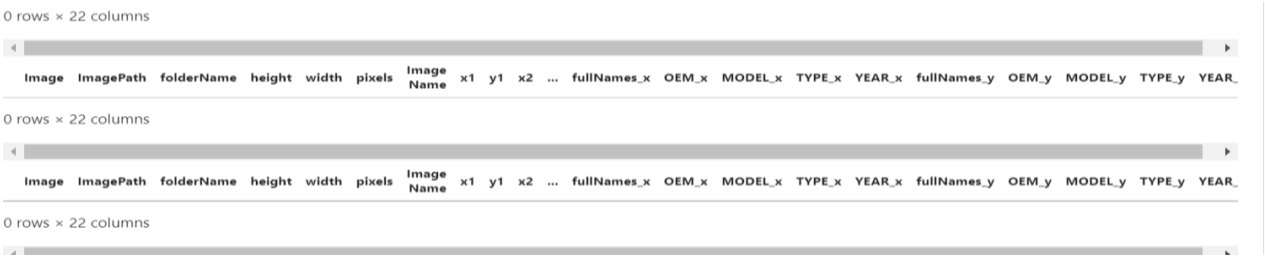
In [38]:
```
# update class index to start from ZERO
trainDF["Image class"] = trainDF["Image class"]-1
testDF["Image class"] = testDF["Image class"]-1
```

In [39]:
```
# merge cars_names_and_make csv data with the annotation class name field
trainDF = pd.merge(trainDF,carsMaster,how='outer',left_on='Image class',right_index=True)
testDF = pd.merge(testDF,carsMaster,how='outer',left_on='Image class',right_index=True)
# though this will duplicate the already exisiting folderName, fullNames columns, this adds a cross check for data correctness
```

Step 3: Validate data for any mismatch during merging

After doing the cross merged and synced with "Train/Test Annotations.csv", "Car names and make.csv" and the images in the "Train/Test images folders", it is found to have no mismatch of information

There are now 22 columns in the data frame



The data frame is now cleaned up, label encoding is done and a column to store the bounding box coordinates together **is created**

[44]:

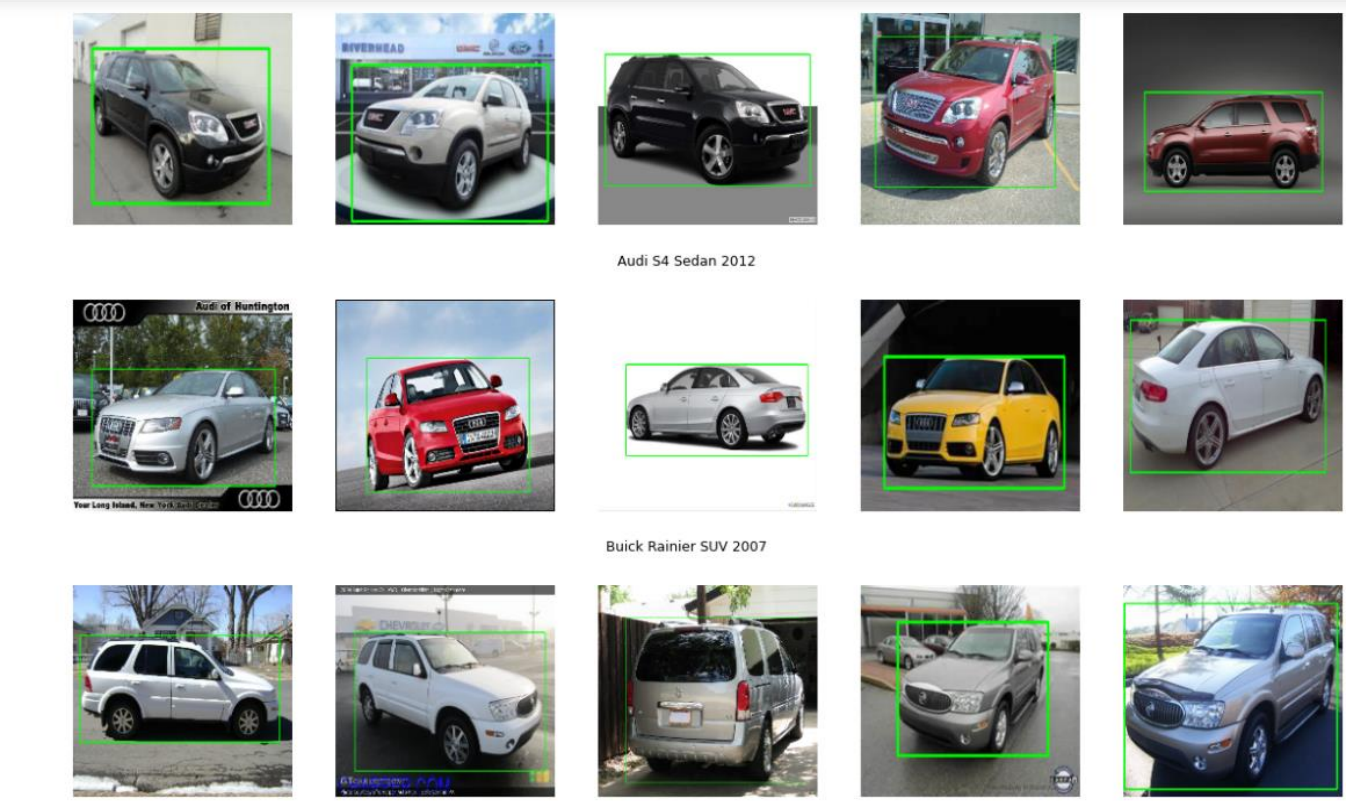
	Image	ImagePath	x1	y1	x2	y2	height	width	folderName	Image_class	OEM	MODEL	TYPE	YEAR	classEncoded	bBox
4199	00784.jpg	Car Images/Train Images/Ferrari 458 Italia Con...	42	163	725	382	786.0	492.0	Ferrari 458 Italia Convertible 2012	102	Ferrari	458_Italia	Convertible	2012	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[42, 163, 725, 382]
2238	06941.jpg	Car Images/Train Images/Chevrolet Corvette Con...	7	406	1492	1024	1600.0	1200.0	Chevrolet Corvette Convertible 2012	54	Chevrolet	Corvette	Convertible	2012	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[7, 406, 1492, 1024]

[45]:
```
testDF.sample(2)
```

[45]:

	Image	ImagePath	x1	y1	x2	y2	height	width	folderName	Image_class	OEM	MODEL	TYPE	YEAR	classEncoded	bBox
254	01291.jpg	Car Images/Test Images/Acura ZDX Hatchback 201...	106	182	548	427	620.0	456.0	Acura ZDX Hatchback 2012	6	Acura	ZDX	Hatchback	2012	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[106, 182, 548, 427]
4467	04877.jpg	Car Images/Test Images/Ford Edge SUV 2012/0487...	20	104	558	321	575.0	431.0	Ford Edge SUV 2012	109	Ford	Edge	SUV	2012	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[20, 104, 558, 321]

Null values are checked and visualization of images with bounding boxes in



The data is now preprocessed with all the images of comparable size and augmented by bounding boxes for training through various models and algorithms

3. Deciding model & Model Building

An extensive review to understand various different type of computer vision models and there pros and cons wrt to problem in hand was performed . Further a very simple preliminary model using mobileNet & ...(TBD)..... was build and tested to understand the accuracies and shortcomings of the model

Model Building

The section details the steps for evaluating the appropriate model choices and preliminary results and insights from the initial runs

Choosing an appropriate model algorithm

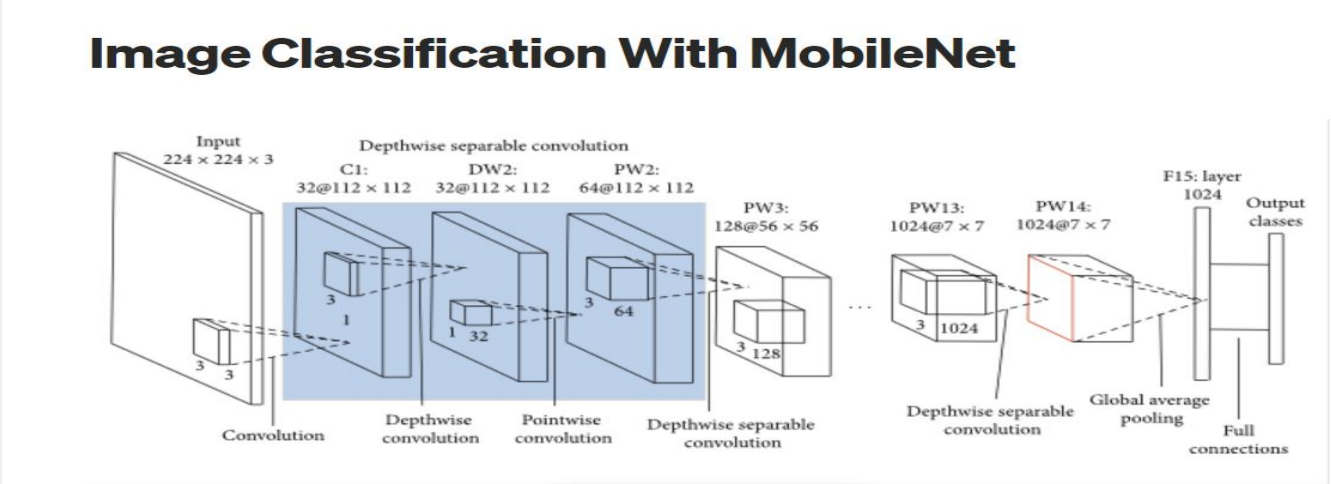
Coming up with a best model algorithm for the problem entailed an extensive exploratory study of various different models available and their appropriateness for being used for the problem in hand. We studied over 15 different computer vision models with respect to their mechanism, efficiency and appropriateness (annexure 1) The table below summarises the key features and pros and cons of the models for the problem in hand

In Progress

S. No	Model Name	Computation Requirement	Findings
1	VGG 16		
2	Mobile Net		
3	VGG 19		
4	Mask RCNN		
5	Single Shot Detector (SSD)		
6	YOLO		
7	YOLO 5		
8	SPPNET		
9	Res NET		
10	Inception Net		
11	Faster RCNN		
12	Google Net V2		
13	Alex Net		
14	Detectron		
15	Efficient Det D7		

Working with the initial model choices (Mobile Net)

We did an initial run using mobile net for classifcation. The screen shot below shares the configuration and the details of the model used. Mobile Net is a type of convolutional neural network designed for mobile and embedded vision applications. They are based on a streamlined architecture that uses depth wise separable convolutions to build lightweight deep neural networks that can have low latency for mobile and embedded devices.



Following are the advantages of using MobileNet over other state-of-the-art deep learning models.

i. Reduced network size - 17MB.
ii. Reduced number of parameters - 4.2 million.
iii. Faster in performance and are useful for mobile applications.
iv. Small, low-latency convolutional neural network

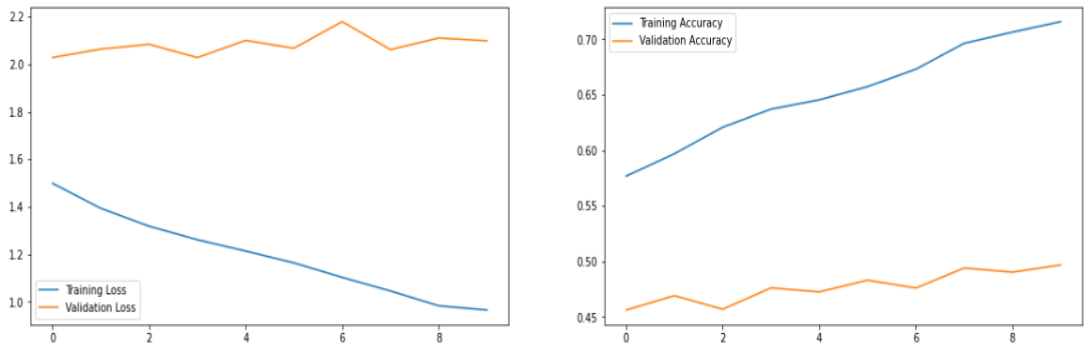
Initial Model Training

Initial model training was performed using the mobilenet .

i. Plain model training with existing training data and testing through test The results revealed data overfitting with training accuracy of about 50%

Out[49]:

```
<matplotlib.legend.Legend at 0x137b4c790>
```



Training accuracy increased to 71% and test accuracy increased to almost 50% There is no significant improvement in the test accuracy, the accuracy tends to saturate at 50%

Training accuracy increased to 71% and test accuracy increased to almost 50%

Model 2 TBD by Saturday July 9

4 Improving model performance & Future Work

This section provides an overview of various strategies to improve model performance and key future work in terms of deployment

Improving the Model Performance

On the basis of data EDA and preprocessing a number of strategies for model performance improvement will be attempted to achieve best performing model for deployment

1. Testing different model to come up with best performing model

At the model the algorithms used to build a model and simple but easy to implement like mobile net and VGG (TBD). However these choices are easy to implement and computationally light they do not perfume a very good task at image classification in Test data . We would be working with an array of state of art deep learning models to come up with best performing deployable model. The pipeline for the project has already been build to facilitate the process of model selection

2. Data Augmentation for getting balanced data set

The EDA reveals that image data obtained is imbalanced with some classes of cars having disproportionately high numbers of images when compared to mean. Further, the model building step also performed a very simple data augmentation leading to better results while training the model . A detailed data agumentation to reduce biases and better generalization of the model will be done for improving the model performance

3. Hyperparameter Tunning

Hyperparameter tunning would also be performed to improve the performance of selected models . Signs of *underfitting* or *overfitting* of the test or validation loss early in the training process are useful for tuning the hyper-parameters. Various Hyper parameters like learning rate , batch size, momentum and weight decay are commonly used to tune the hyperparameters for the best performance

Future Work

Future work will pick up from the existing exercise of preliminiary model design and experimenting with alternate model algorithm and improvisation of model through various strategies followed by final deployment . The key tasks are delineated below

i. Evaluating performance of multiple models and algorithms to compare and come up with best performing model
ii. Improving model performance by Upscaling and downscaling the image data for balanced dataset, fine tuning parameters and hyperparameters of best algorithms
iii. Optimizing and normalizing the image size to representative and efficient size
iv. Finalizing and integrating the GUI based interface
v. Overview of business case for the final model
vi. Advantages and limitations of the model

Business Models

1. On road car classification
2. On road car surveillance
3. City and town planning Designing and allocation of parking spaces
4. Container capacity logistics to determine how many cars of different model can be shipped in the fixed container size

Annexure 1. In progress

Evaluating pros and cons of various computer vision models

S.NO	Model Name	Features	Advantages	Disadvantages	
1	VGG 19	1. Transfer learning 2. Introduced by Simonyan and Zisserman in their 2014 paper		1. Slow while training	
2	Faster RCNN	1. Faster R-CNN was introduced in 2015 by k He et al. 2. Works on region proposal network Detection Time : 0.2 (with VGG), 0.059 (with ZF) 3. Uses cross-entropy for foreground and background loss	1. Detects small objects and objects that are next to each other	1. Faster RCNN is slower than YOLO	
3	Inception V2	1. Focuses on computational cost			
4	ResNet50	1. ResNet-50 is a convolutional neural network that is 50 layers deep 2. network-in-network architectures 3. Focuses on computational accuracy			
5	YOLO	1. Predicts the bounding box per grid cell 2. Uses bounding box regression 3. Classification and bounding box regression occur simultaneously 4. 98 bounding boxes per image 5. Two anchors per grid cell 6. Inspired from GoogleNet and implemented using DarkNet framework 7. YOLO used l2 loss	1. Fast algorithm	1. It cannot detect small objects and objects that are next to each other	

6	YOLO5	1. The YOLOv5 implementation has been done in Pytorch, so easy to train 2. Uses Cross Stage Partial Network (CSP) to reduce computation cost	1. Good performance than previous versions 2. Detection of small or far away objects 3. Little to no overlapping boxes 4. Detection of Crowded objects 5. YOLOv5 uses lesser resources compared to Detectron2 6. YOLOv5 has a much smaller model size compared to Detectron2	1. Use YOLOv5 only if your classes have unique features that are easily distinguishable 2. YOLOv5 should be the model of choice if you have enough training images	
7	Detectron		1. Detectron2 makes it easier to experiment with different hyperparameters		
8	Efficient Det 7		State of art		
9	Google Net V2				
10	Alex Net				
11	Mobile Net				
12	VGG 16				
13	Mask RCNN				
14	Single Shot Detector				
15	SPPNET				

Annexure 2 Model -1 Configuration & Final Model topology to be Added here (TBD) (Mobile net (Base data)

Model: "model_1"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv1_relu (ReLU)	(None, 112, 112, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2048
conv_pw_1_bn (BatchNormalization)	(None, 112, 112, 64)	256
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576
conv_dw_2_bn (BatchNormalization)	(None, 56, 56, 64)	256
conv_dw_2_relu (ReLU)	(None, 56, 56, 64)	0
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	8192
conv_pw_2_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_pw_2_relu (ReLU)	(None, 56, 56, 128)	0
conv_dw_3 (DepthwiseConv2D)	(None, 56, 56, 128)	1152
conv_dw_3_bn (BatchNormalization)	(None, 56, 56, 128)	512

```
conv_dw_3_relu (ReLU)        (None, 56, 56, 128)       0

conv_pw_3 (Conv2D)           (None, 56, 56, 128)       16384

conv_pw_3_bn (BatchNormaliz  (None, 56, 56, 128)       512
ation)

conv_pw_3_relu (ReLU)        (None, 56, 56, 128)       0

conv_pad_4 (ZeroPadding2D)   (None, 57, 57, 128)       0

conv_dw_4 (DepthwiseConv2D)  (None, 28, 28, 128)       1152

conv_dw_4_bn (BatchNormaliz  (None, 28, 28, 128)       512
ation)

conv_dw_4_relu (ReLU)        (None, 28, 28, 128)       0

conv_pw_4 (Conv2D)           (None, 28, 28, 256)       32768

conv_pw_4_bn (BatchNormaliz  (None, 28, 28, 256)       1024
ation)

conv_pw_4_relu (ReLU)        (None, 28, 28, 256)       0

conv_dw_5 (DepthwiseConv2D)  (None, 28, 28, 256)       2304

conv_dw_5_bn (BatchNormaliz  (None, 28, 28, 256)       1024
ation)

conv_dw_5_relu (ReLU)        (None, 28, 28, 256)       0

conv_pw_5 (Conv2D)           (None, 28, 28, 256)       65536

conv_pw_5_bn (BatchNormaliz  (None, 28, 28, 256)       1024
ation)

conv_pw_5_relu (ReLU)        (None, 28, 28, 256)       0

conv_pad_6 (ZeroPadding2D)   (None, 29, 29, 256)       0

conv_dw_6 (DepthwiseConv2D)  (None, 14, 14, 256)       2304

conv_dw_6_bn (BatchNormaliz  (None, 14, 14, 256)       1024
ation)

conv_dw_6_relu (ReLU)        (None, 14, 14, 256)       0

conv_pw_6 (Conv2D)           (None, 14, 14, 512)       131072

conv_pw_6_bn (BatchNormaliz  (None, 14, 14, 512)       2048
ation)

conv_pw_6_relu (ReLU)        (None, 14, 14, 512)       0
```

```
conv_dw_7 (DepthwiseConv2D)  (None, 14, 14, 512)       4608

conv_dw_7_bn (BatchNormaliz  (None, 14, 14, 512)       2048
ation)

conv_dw_7_relu (ReLU)        (None, 14, 14, 512)       0

conv_pw_7 (Conv2D)           (None, 14, 14, 512)       262144

conv_pw_7_bn (BatchNormaliz  (None, 14, 14, 512)       2048
ation)

conv_pw_7_relu (ReLU)        (None, 14, 14, 512)       0

conv_dw_8 (DepthwiseConv2D)  (None, 14, 14, 512)       4608

conv_dw_8_bn (BatchNormaliz  (None, 14, 14, 512)       2048
ation)

conv_dw_8_relu (ReLU)        (None, 14, 14, 512)       0

conv_pw_8 (Conv2D)           (None, 14, 14, 512)       262144

conv_pw_8_bn (BatchNormaliz  (None, 14, 14, 512)       2048
ation)

conv_pw_8_relu (ReLU)        (None, 14, 14, 512)       0

conv_dw_9 (DepthwiseConv2D)  (None, 14, 14, 512)       4608

conv_dw_9_bn (BatchNormaliz  (None, 14, 14, 512)       2048
ation)

conv_dw_9_relu (ReLU)        (None, 14, 14, 512)       0

conv_pw_9 (Conv2D)           (None, 14, 14, 512)       262144

conv_pw_9_bn (BatchNormaliz  (None, 14, 14, 512)       2048
ation)

conv_pw_9_relu (ReLU)        (None, 14, 14, 512)       0

conv_dw_10 (DepthwiseConv2D  (None, 14, 14, 512)       4608
)

conv_dw_10_bn (BatchNormali  (None, 14, 14, 512)       2048
zation)

conv_dw_10_relu (ReLU)       (None, 14, 14, 512)       0

conv_pw_10 (Conv2D)          (None, 14, 14, 512)       262144

conv_pw_10_bn (BatchNormali  (None, 14, 14, 512)       2048
zation)

conv_pw_10_relu (ReLU)       (None, 14, 14, 512)       0
```

```
conv_dw_11 (DepthwiseConv2D  (None, 14, 14, 512)      4608
)

conv_dw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)

conv_dw_11_relu (ReLU)       (None, 14, 14, 512)      0

conv_pw_11 (Conv2D)          (None, 14, 14, 512)      262144

conv_pw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)

conv_pw_11_relu (ReLU)       (None, 14, 14, 512)      0

conv_pad_12 (ZeroPadding2D)  (None, 15, 15, 512)      0

conv_dw_12 (DepthwiseConv2D  (None, 7, 7, 512)        4608
)

conv_dw_12_bn (BatchNormali  (None, 7, 7, 512)        2048
zation)

conv_dw_12_relu (ReLU)       (None, 7, 7, 512)        0

conv_pw_12 (Conv2D)          (None, 7, 7, 1024)       524288

conv_pw_12_bn (BatchNormali  (None, 7, 7, 1024)       4096
zation)

conv_pw_12_relu (ReLU)       (None, 7, 7, 1024)       0

conv_dw_13 (DepthwiseConv2D  (None, 7, 7, 1024)       9216
)

conv_dw_13_bn (BatchNormali  (None, 7, 7, 1024)       4096
zation)

conv_dw_13_relu (ReLU)       (None, 7, 7, 1024)       0

conv_pw_13 (Conv2D)          (None, 7, 7, 1024)       1048576

conv_pw_13_bn (BatchNormali  (None, 7, 7, 1024)       4096
zation)

conv_pw_13_relu (ReLU)       (None, 7, 7, 1024)       0

global_average_pooling2d_1   (None, 1024)             0
(GlobalAveragePooling2D)

dense_3 (Dense)              (None, 1024)             1049600

dense_4 (Dense)              (None, 512)              524800

dense_5 (Dense)              (None, 196)              100548
```

```
=================================================================
Total params: 4,903,812
Trainable params: 1,674,948
Non-trainable params: 3,228,864
_________________________________________________________________
```