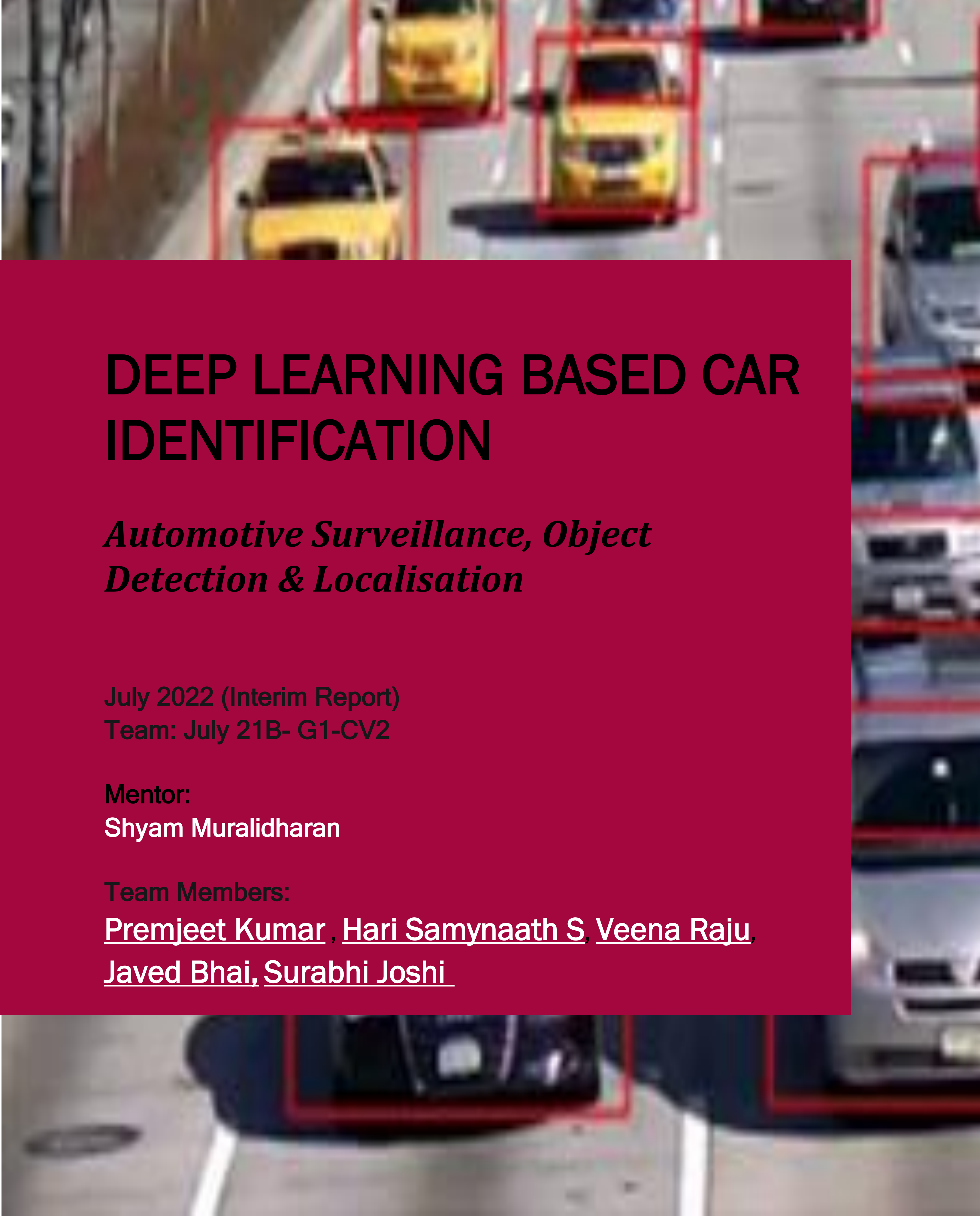


DEEP LEARNING BASED CAR IDENTIFICATION

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

1.Abstract

This interim report focusses on design and deployment of a multiclass object detection model for cars which enables identification of 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 using Stanford Car Database. 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 was aimed to understand the biases in the training data along with data preprocessing which included image resizing and bounding box creation. The report further details steps for selection, design and implementation of a preliminary car detection model, identifies few shortcomings and delineates 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 to be deployed

2.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 overfit and would need drastic performance improvements to reach deployable stage reliable model. Spaces of performance improvement are identified along 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.

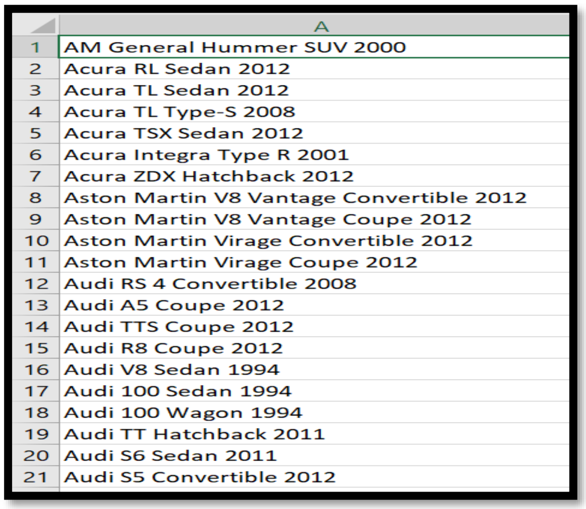
<u>Data description:</u>

Train Images: Real images of cars as per the make and year of the car.
Test Images: 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.

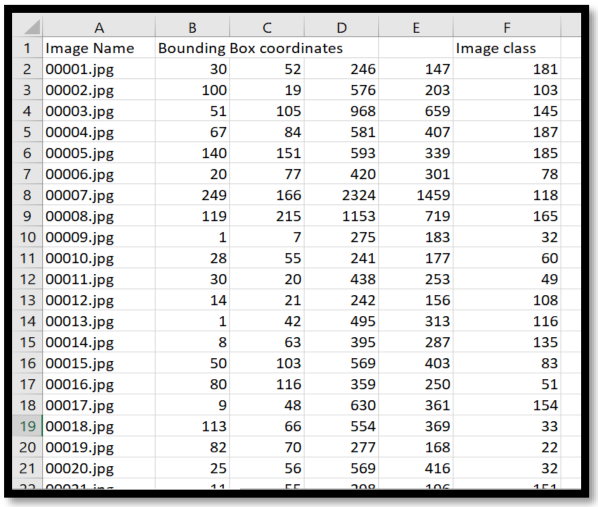
S No.	File name	Format	Size	Details
1	Car Images	. zip		The .zip folder includes two sub folders Train and Test. Each of these subfolders have 196 class folder with .jpeg images of folders
2	Annotations	. zip	171kb	CSV files in two folders train and test with bounding box coordinates
3	Car names & make	.CSV	6 kb	Single CSV file with car name and makes indeed with 196 car classes

A snapshot of the data through various available files provides following insights



	A
1	AM General Hummer SUV 2000
2	Acura RL Sedan 2012
3	Acura TL Sedan 2012
4	Acura TL Type-S 2008
5	Acura TSX Sedan 2012
6	Acura Integra Type R 2001
7	Acura ZDX Hatchback 2012
8	Aston Martin V8 Vantage Convertible 2012
9	Aston Martin V8 Vantage Coupe 2012
10	Aston Martin Virage Convertible 2012
11	Aston Martin Virage Coupe 2012
12	Audi RS 4 Convertible 2008
13	Audi A5 Coupe 2012
14	Audi TTS Coupe 2012
15	Audi R8 Coupe 2012
16	Audi V8 Sedan 1994
17	Audi 100 Sedan 1994
18	Audi 100 Wagon 1994
19	Audi TT Hatchback 2011
20	Audi S6 Sedan 2011
21	Audi S5 Convertible 2012

Car Make & Name



	A	B	C	D	E	F
1	Image Name	Bounding Box coordinates				Image class
2	00001.jpg	30	52	246	147	181
3	00002.jpg	100	19	576	203	103
4	00003.jpg	51	105	968	659	145
5	00004.jpg	67	84	581	407	187
6	00005.jpg	140	151	593	339	185
7	00006.jpg	20	77	420	301	78
8	00007.jpg	249	166	2324	1459	118
9	00008.jpg	119	215	1153	719	165
10	00009.jpg	1	7	275	183	32
11	00010.jpg	28	55	241	177	60
12	00011.jpg	30	20	438	253	49
13	00012.jpg	14	21	242	156	108
14	00013.jpg	1	42	495	313	116
15	00014.jpg	8	63	395	287	135
16	00015.jpg	50	103	569	403	83
17	00016.jpg	80	116	359	250	51
18	00017.jpg	9	48	630	361	154
19	00018.jpg	113	66	554	369	33
20	00019.jpg	82	70	277	168	22
21	00020.jpg	25	56	569	416	32

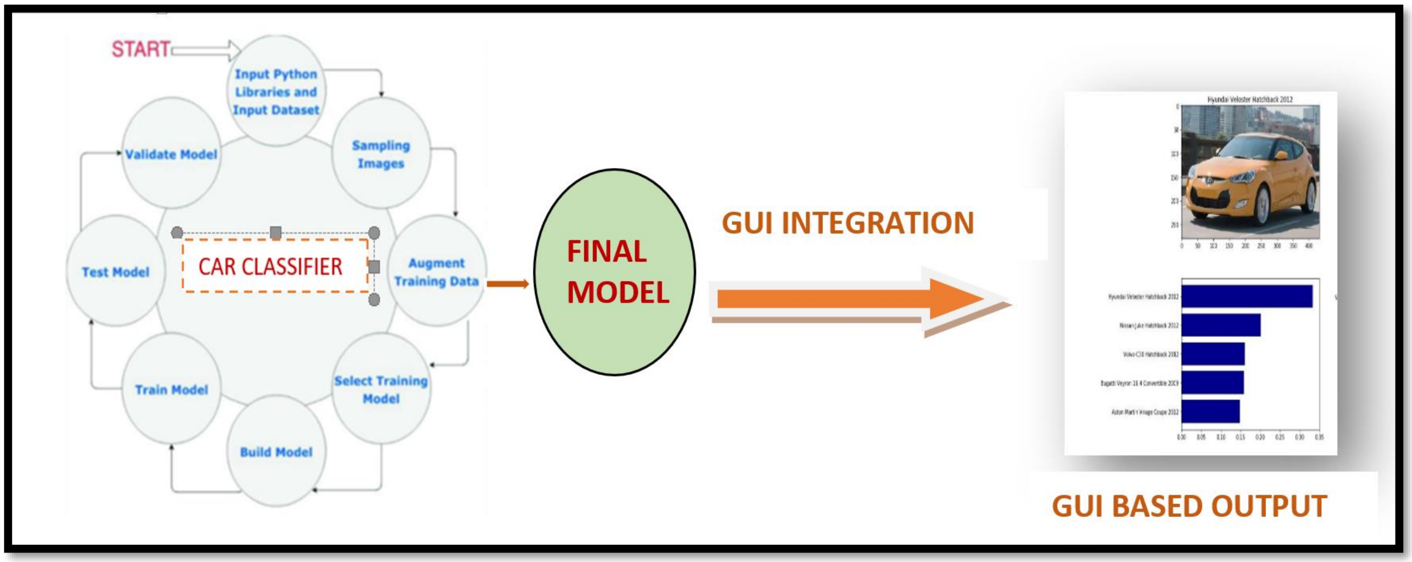
Annotations

The car make & Name folder provides indeed names for 196 different car models for which image has been provided. The model name is made available in a single columns and contains usable input feature information which would need to be suitably extracted i.e. Car OEM, car type, car model and year of make through processing before being used for neural network training. The index column of Car make & name corresponds with column F (Image class) of the annotations data.

The annotations data is provided in two folders train and test. The individual image in this data is mapped with its bounding box coordinates and class of the car. Its to be noted that bounding box coordinates show a wide range of values pointing to images with various different resolutions and backgrounds. Further, there is no chronological sequence or significance of image name with image class

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 through creation of a GUI.

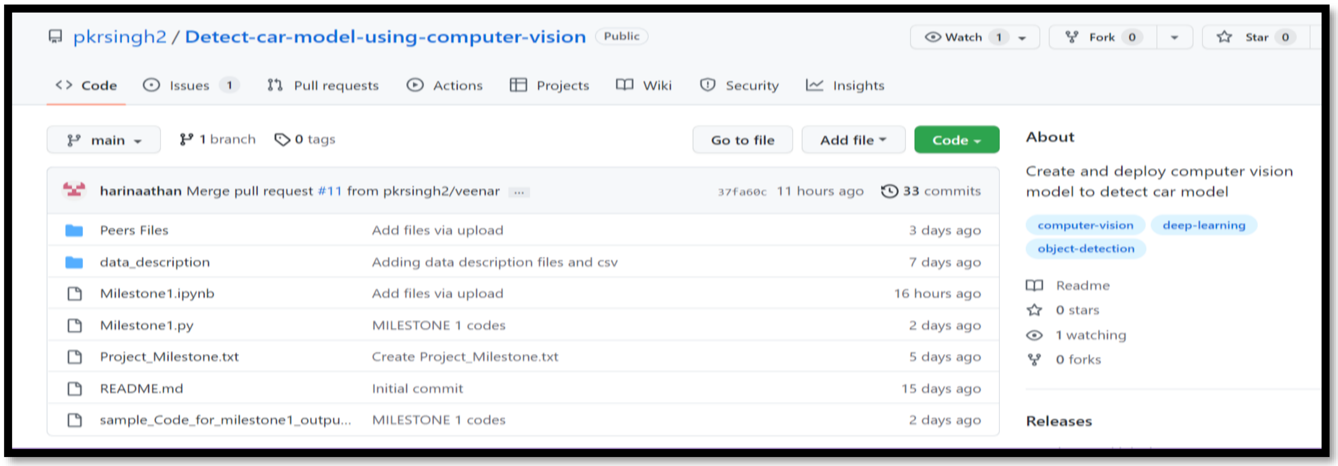


Prediction model for car classification will be deployed using publicly hosted application based on Streamlit by creating a GUI . This application will enable users to interact with trained model. User can select any of the images used for training, Validation and testing and see how the model would classify it and also provides necessary information like car model, make and year.

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


pkrsingh2 / Detect-car-model-using-computer-vision Public

harinaathan Merge pull request #11 from pkrsingh2/veenar ... 37fa60c 11 hours ago 33 commits

Peers Files	Add files via upload	3 days ago
data_description	Adding data description files and csv	7 days ago
Milestone1.ipynb	Add files via upload	16 hours ago
Milestone1.py	MILESTONE 1 codes	2 days ago
Project_Milestone.txt	Create Project_Milestone.txt	5 days ago
README.md	Initial commit	15 days ago
sample_Code_for_milestone1_outpu...	MILESTONE 1 codes	2 days ago

About

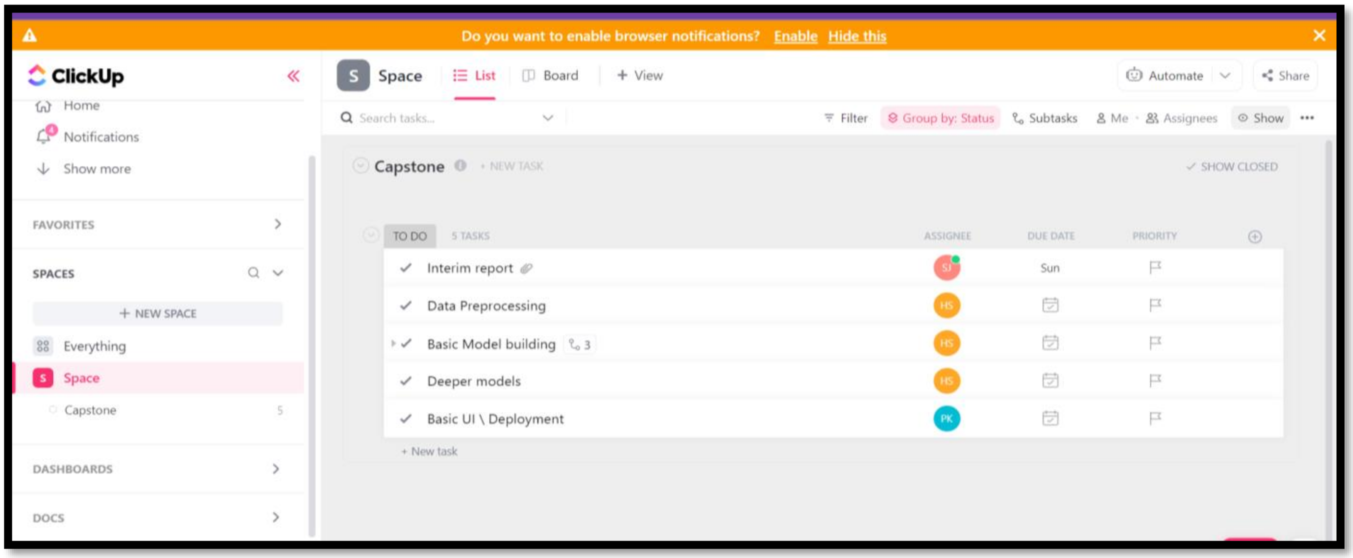
Create and deploy computer vision model to detect car model

computer-vision deep-learning object-detection

Readme
0 stars
1 watching
0 forks

Releases

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


Do you want to enable browser notifications? Enable Hide this

Capstone + NEW TASK ✓ SHOW CLOSED

TO DO 5 TASKS	ASSIGNEE	DUE DATE	PRIORITY
Interim report	SJ	Sun	
Data Preprocessing	HS		
Basic Model building 3	HS		
Deeper models	HS		
Basic UI \ Deployment	PK		

+ New task

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**

3 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 size of images and 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 make year 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

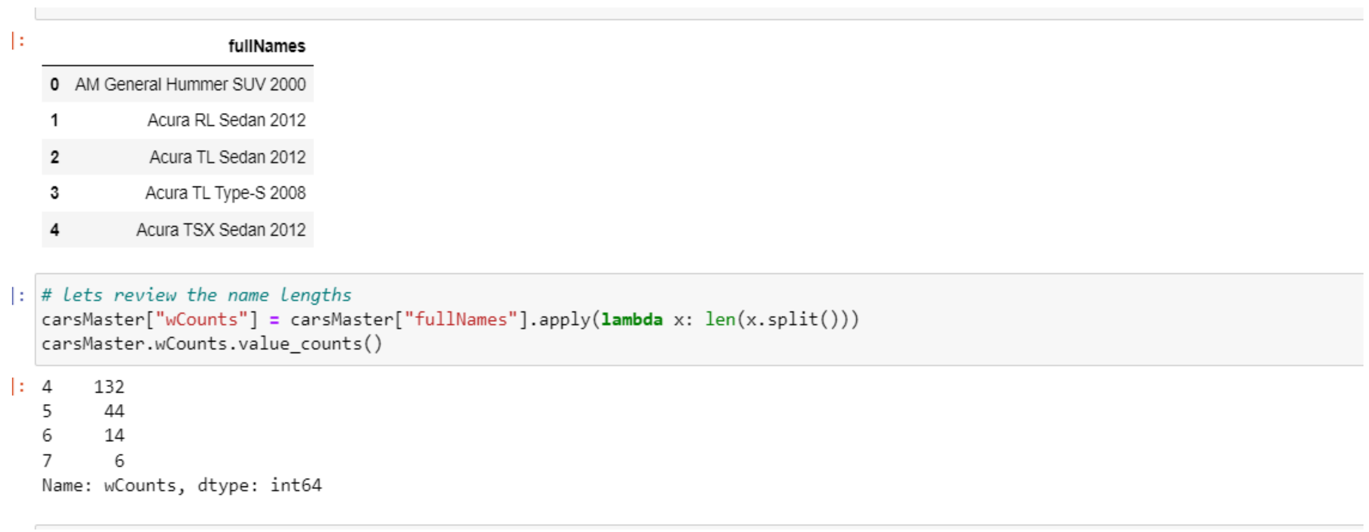
The dataset is neither homogenous in car instances nor consistent with respect to image size therefore would be requiring significant preprocessing and improvements

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

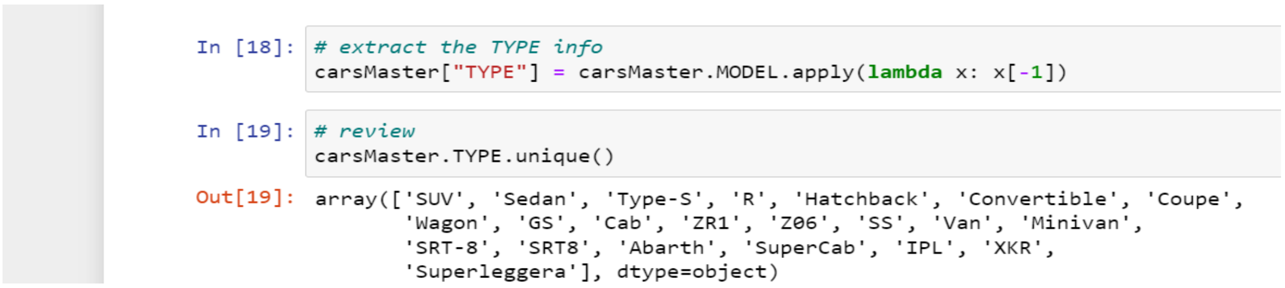
```
# lets review the name lengths
carsMaster["wCounts"] = carsMaster["fullNames"].apply(lambda x: len(x.split()))
carsMaster.wCounts.value_counts()
```

```
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

The full name length has to be now processed to get appropriate variables for the inputs. We find inconsistencies in the given full names for the cars.

We find that position of car type and Car model number are same for many given names which created issues while sorting as car type

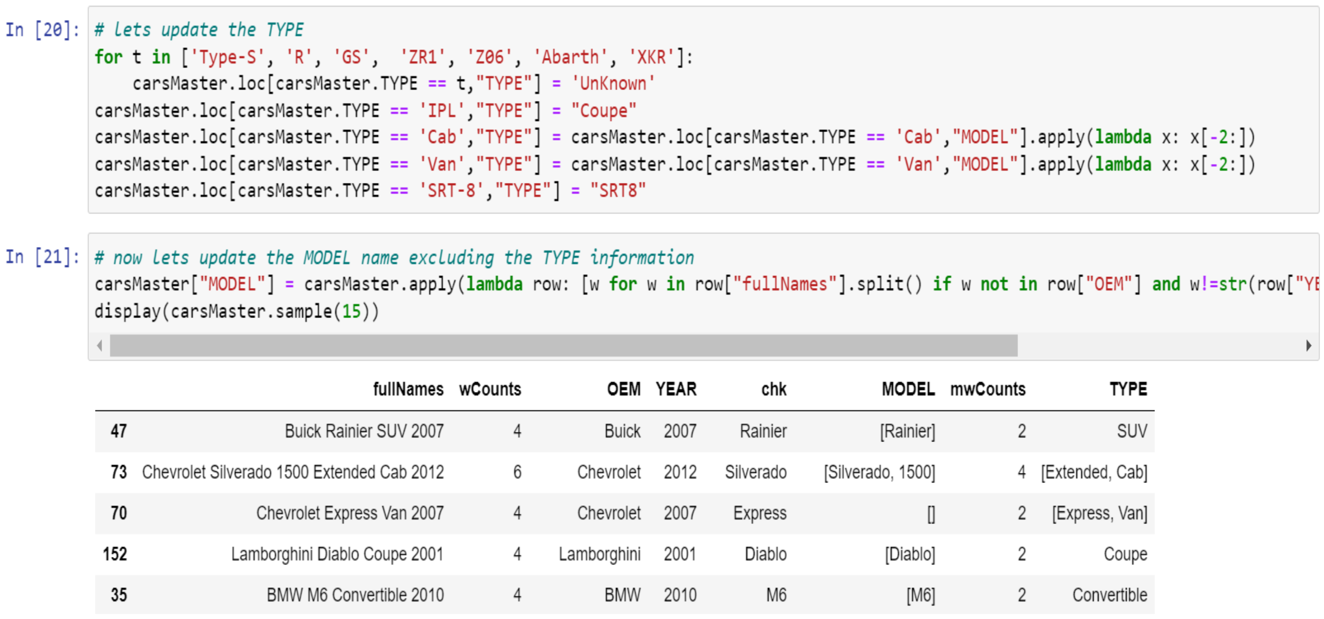


```
In [18]: # extract the TYPE info
         carsMaster["TYPE"] = carsMaster.MODEL.apply(lambda x: x[-1])

In [19]: # review
         carsMaster.TYPE.unique()

Out[19]: array(['SUV', 'Sedan', 'Type-S', 'R', 'Hatchback', 'Convertible', 'Coupe',
                'Wagon', 'GS', 'Cab', 'ZR1', 'Z06', 'SS', 'Van', 'Minivan',
                'SRT-8', 'SRT8', 'Abarth', 'SuperCab', 'IPL', 'XKR',
                'Superleggera'], dtype=object)
```

We find that

- Type IPL hides Coupe type before it.
- Type-S', 'R', 'GS', 'ZR1', 'Z06', 'Abarth', 'XKR' types are not coach types, hence to be marked as unknown
- 'SS', 'SRT-8', 'SRT8' could be considered as car type (though not coach type) as they are technology/class of car

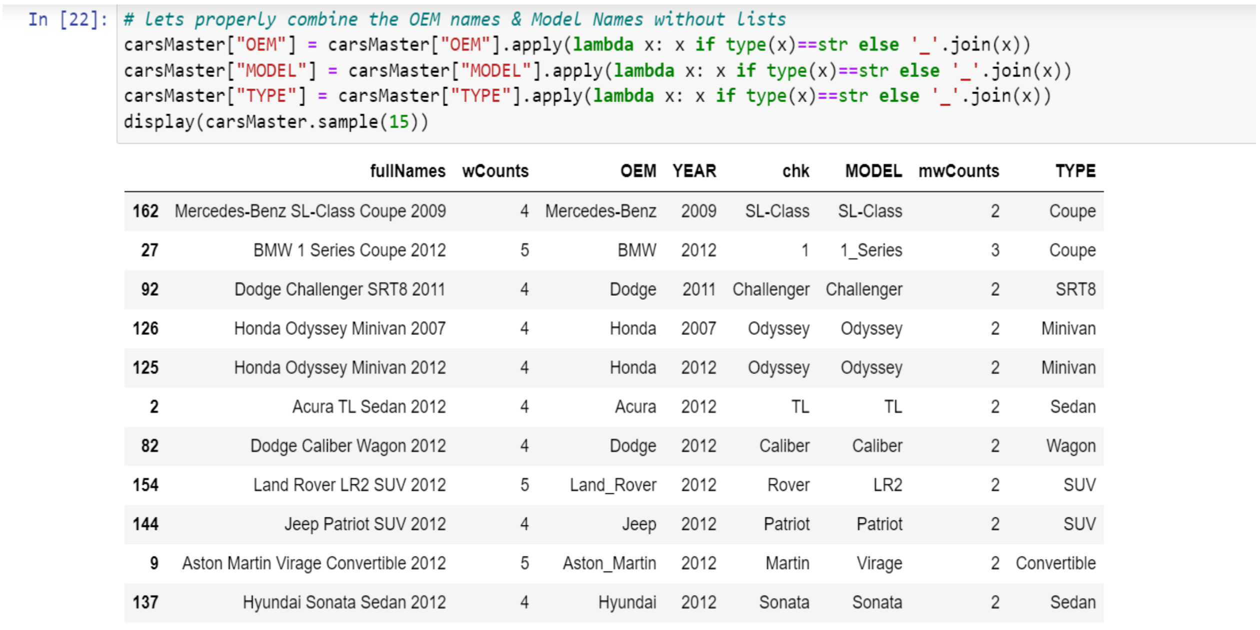
We manually define the types to get the data consistency



```
In [20]: # lets update the TYPE
         for t in ['Type-S', 'R', 'GS',  'ZR1', 'Z06', 'Abarth', 'XKR']:
             carsMaster.loc[carsMaster.TYPE == t,"TYPE"] = 'UnKnown'
         carsMaster.loc[carsMaster.TYPE == 'IPL',"TYPE"] = "Coupe"
         carsMaster.loc[carsMaster.TYPE == 'Cab',"TYPE"] = carsMaster.loc[carsMaster.TYPE == 'Cab',"MODEL"].apply(lambda x: x[-2:])
         carsMaster.loc[carsMaster.TYPE == 'Van',"TYPE"] = carsMaster.loc[carsMaster.TYPE == 'Van',"MODEL"].apply(lambda x: x[-2:])
         carsMaster.loc[carsMaster.TYPE == 'SRT-8',"TYPE"] = "SRT8"

In [21]: # now lets update the MODEL name excluding the TYPE information
         carsMaster["MODEL"] = carsMaster.apply(lambda row: [w for w in row["fullNames"].split() if w not in row["OEM"] and w!=str(row["YE
         display(carsMaster.sample(15))
```

	fullNames	wCounts	OEM	YEAR	chk	MODEL	mwCounts	TYPE
47	Buick Rainier SUV 2007	4	Buick	2007	Rainier	[Rainier]	2	SUV
73	Chevrolet Silverado 1500 Extended Cab 2012	6	Chevrolet	2012	Silverado	[Silverado, 1500]	4	[Extended, Cab]
70	Chevrolet Express Van 2007	4	Chevrolet	2007	Express	[]	2	[Express, Van]
152	Lamborghini Diablo Coupe 2001	4	Lamborghini	2001	Diablo	[Diablo]	2	Coupe
35	BMW M6 Convertible 2010	4	BMW	2010	M6	[M6]	2	Convertible

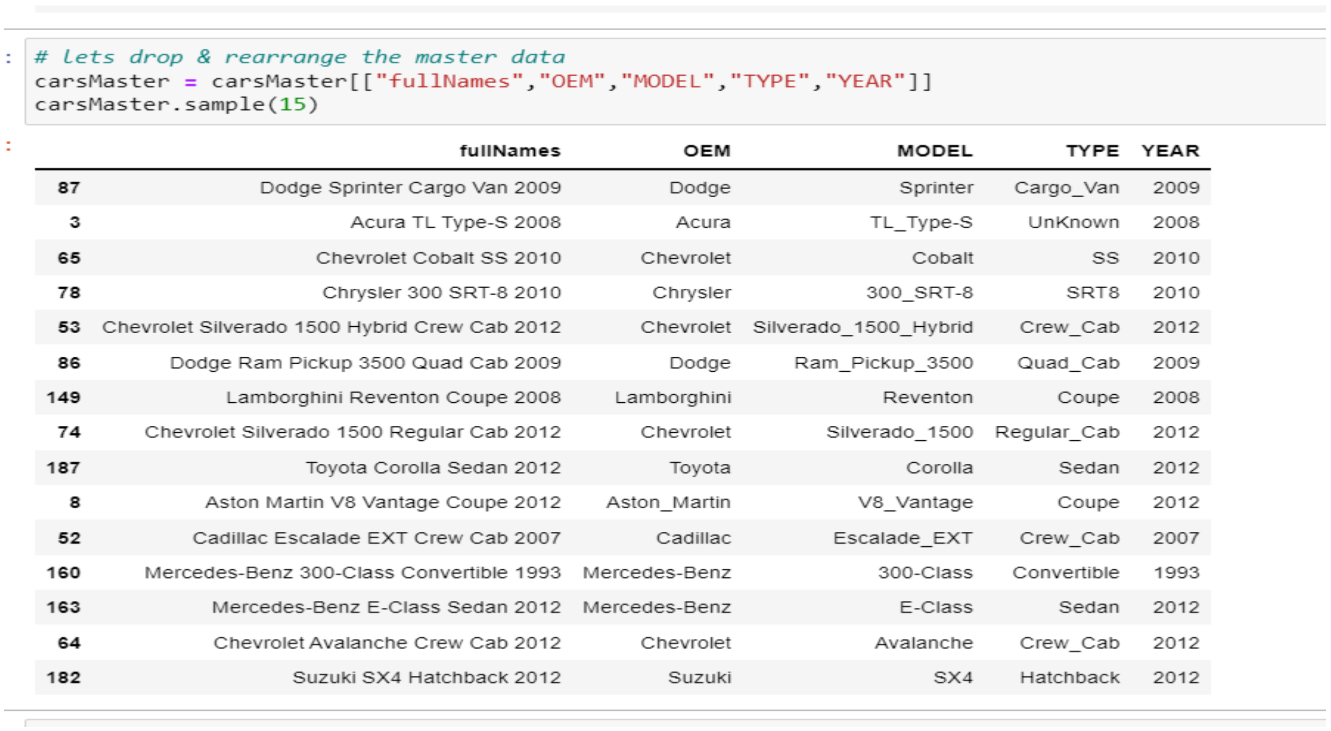
Now the OEM names and Model names are combined without list



```
In [22]: # lets properly combine the OEM names & Model Names without lists
         carsMaster["OEM"] = carsMaster["OEM"].apply(lambda x: x if type(x)==str else '_'.join(x))
         carsMaster["MODEL"] = carsMaster["MODEL"].apply(lambda x: x if type(x)==str else '_'.join(x))
         carsMaster["TYPE"] = carsMaster["TYPE"].apply(lambda x: x if type(x)==str else '_'.join(x))
         display(carsMaster.sample(15))
```

	fullNames	wCounts	OEM	YEAR	chk	MODEL	mwCounts	TYPE
162	Mercedes-Benz SL-Class Coupe 2009	4	Mercedes-Benz	2009	SL-Class	SL-Class	2	Coupe
27	BMW 1 Series Coupe 2012	5	BMW	2012	1	1_Series	3	Coupe
92	Dodge Challenger SRT8 2011	4	Dodge	2011	Challenger	Challenger	2	SRT8
126	Honda Odyssey Minivan 2007	4	Honda	2007	Odyssey	Odyssey	2	Minivan
125	Honda Odyssey Minivan 2012	4	Honda	2012	Odyssey	Odyssey	2	Minivan
2	Acura TL Sedan 2012	4	Acura	2012	TL	TL	2	Sedan
82	Dodge Caliber Wagon 2012	4	Dodge	2012	Caliber	Caliber	2	Wagon
154	Land Rover LR2 SUV 2012	5	Land_Rover	2012	Rover	LR2	2	SUV
144	Jeep Patriot SUV 2012	4	Jeep	2012	Patriot	Patriot	2	SUV
9	Aston Martin Virage Convertible 2012	5	Aston_Martin	2012	Martin	Virage	2	Convertible
137	Hyundai Sonata Sedan 2012	4	Hyundai	2012	Sonata	Sonata	2	Sedan

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

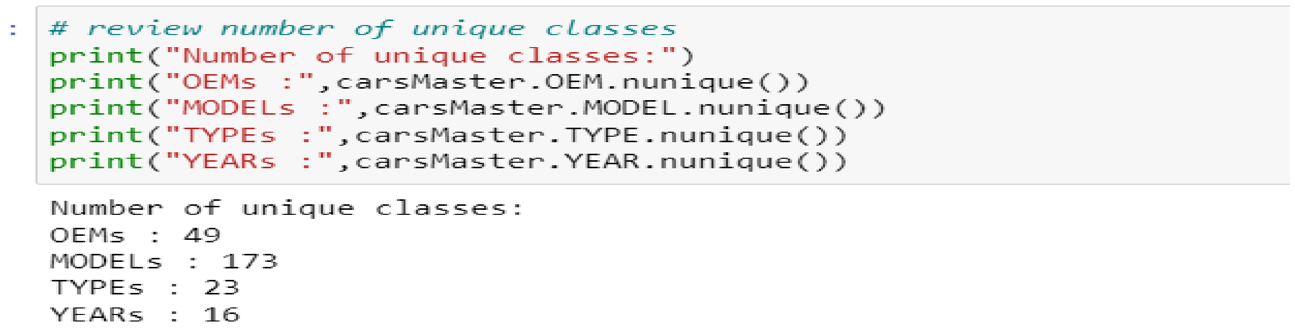


```
: # lets drop & rearrange the master data
  carsMaster = carsMaster[["fullNames","OEM","MODEL","TYPE","YEAR"]]
  carsMaster.sample(15)
```

	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



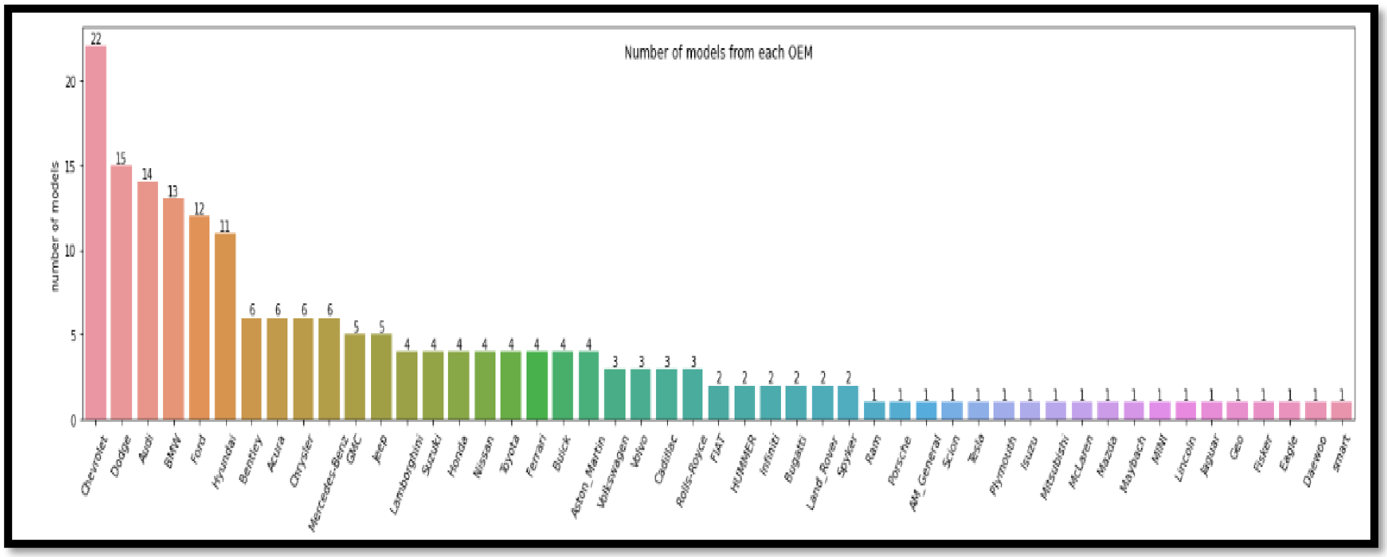
```
# review number of unique classes
print("Number of unique classes:")
print("OEMs :",carsMaster.OEM.nunique())
print("MODELs :",carsMaster.MODEL.nunique())
print("TYPEs :",carsMaster.TYPE.nunique())
print("YEARs :",carsMaster.YEAR.nunique())
```

```
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

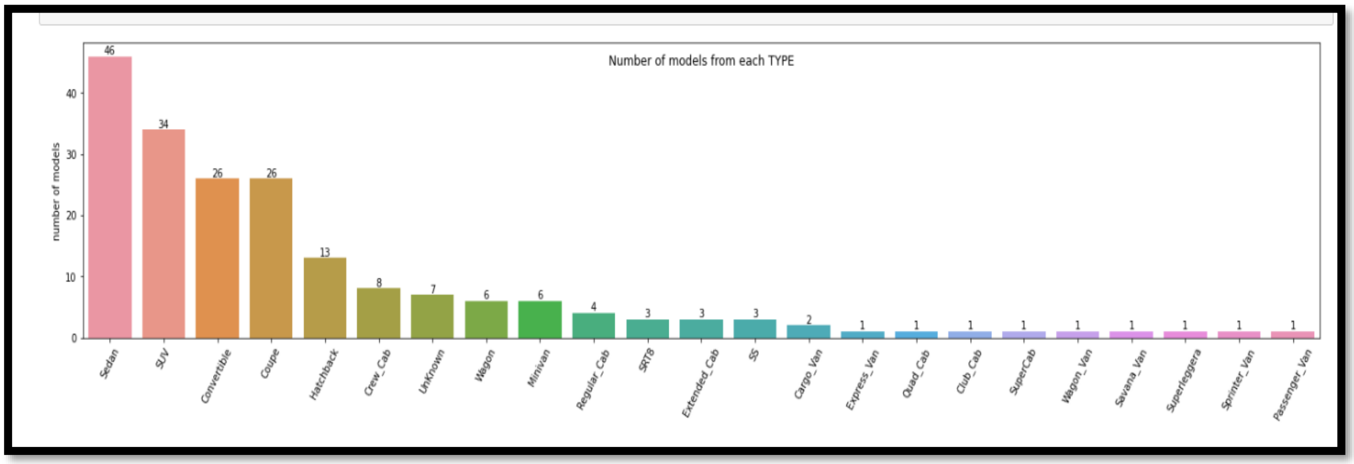
We now explore the frequency distribution and data distribution of various categories of cars WRT to OEM's, Type of cars, and year of manufacturing and plot the frequencies

Number of Models for each Car Manufacturer (OEM)



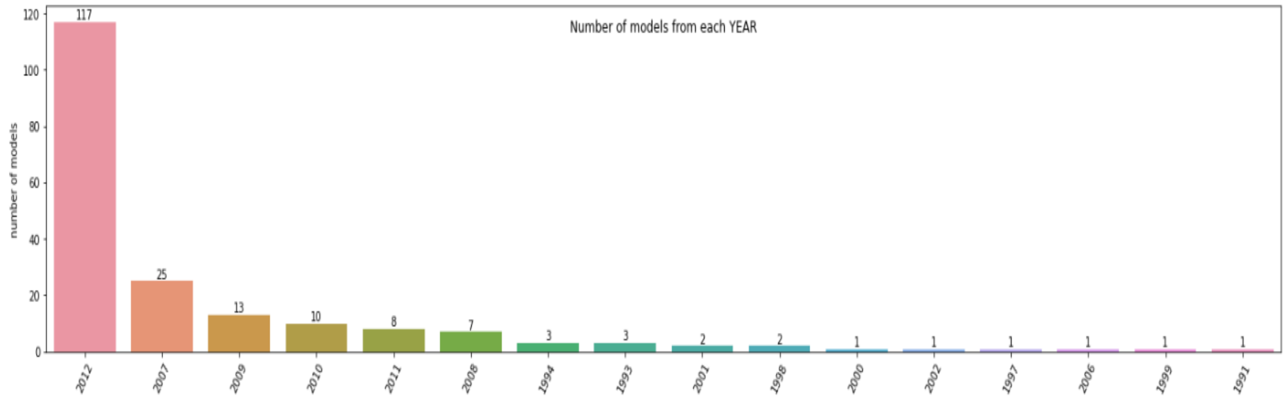
The data consists of 22 models for Chevrolet followed by Dodge (15), Audi(14), BMW(12) , Ford(12) and Hundai (11) which have more then 10 models in the data thus higher representation that significantly higher RAM , MADA etc which have only one model in the data

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



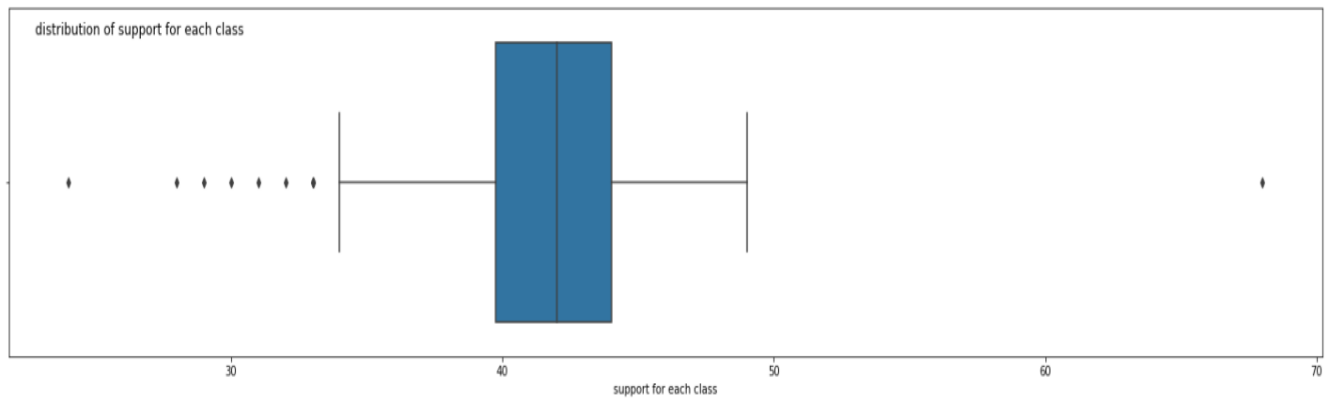
From the perspective of data distribution highest data frequency for make year in 117, which emerges as a predominant bias in the existing dataset . From the above three parameters we get an insight that there is a high probability of intrinsic biases in terms of predominant OEM, type and year of manufacturing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

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

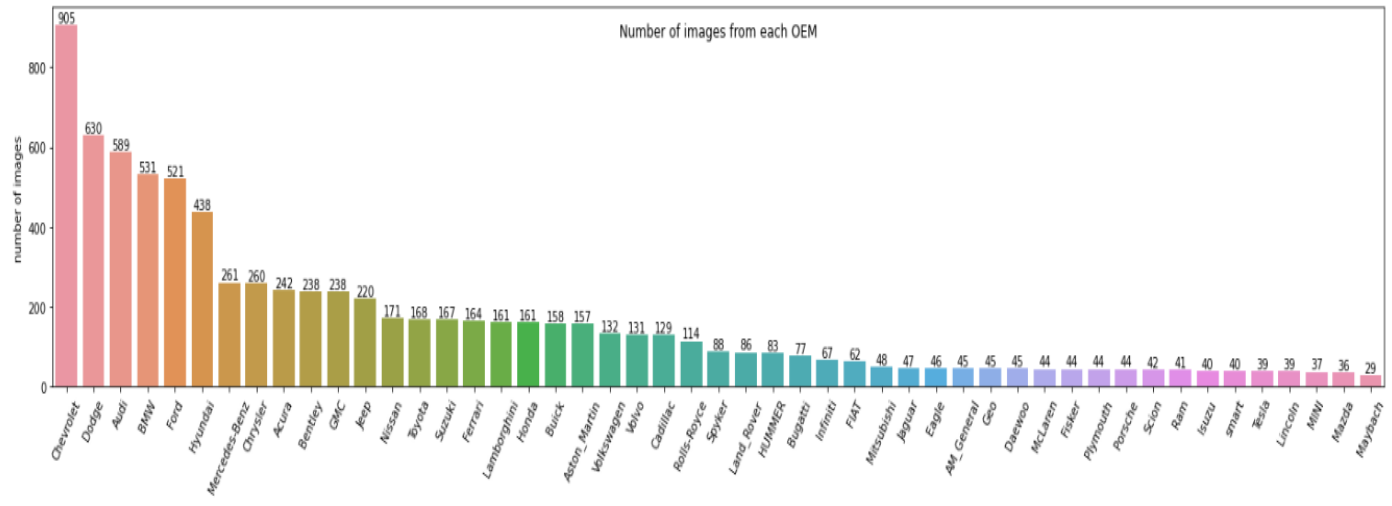
Data Distribution in terms of individual image counts

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

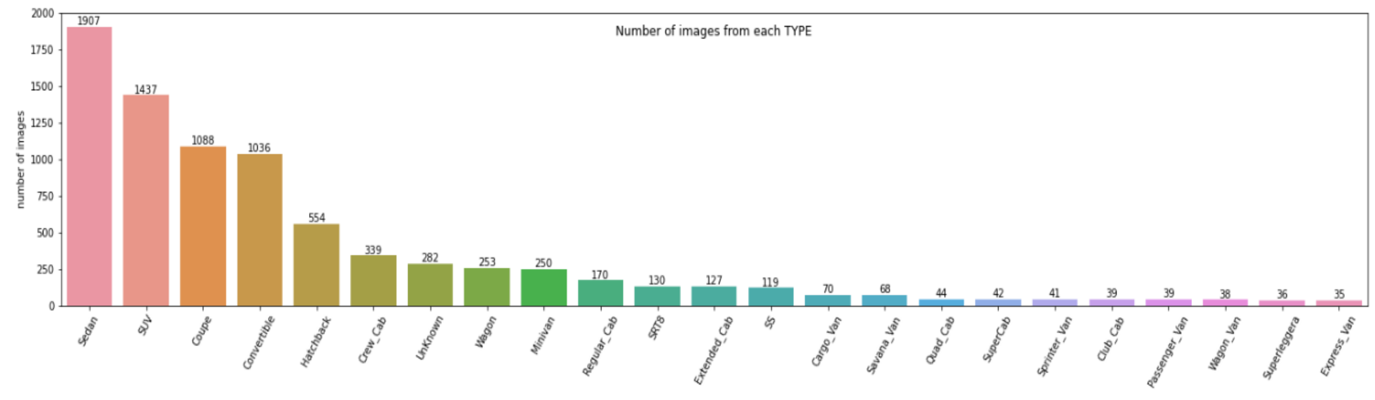
We determine the distribution support of each class and find that for most classes the total images available lie between 40-50 instances for training



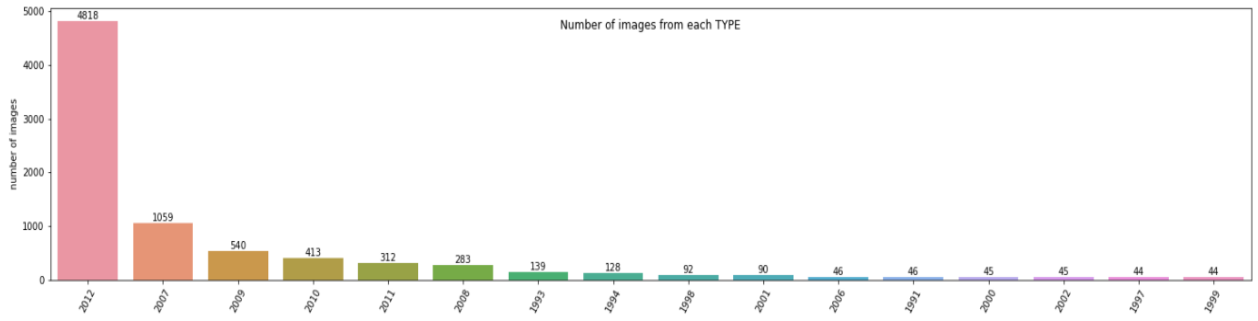
Number of images in each OEM



Number of images in each OEM



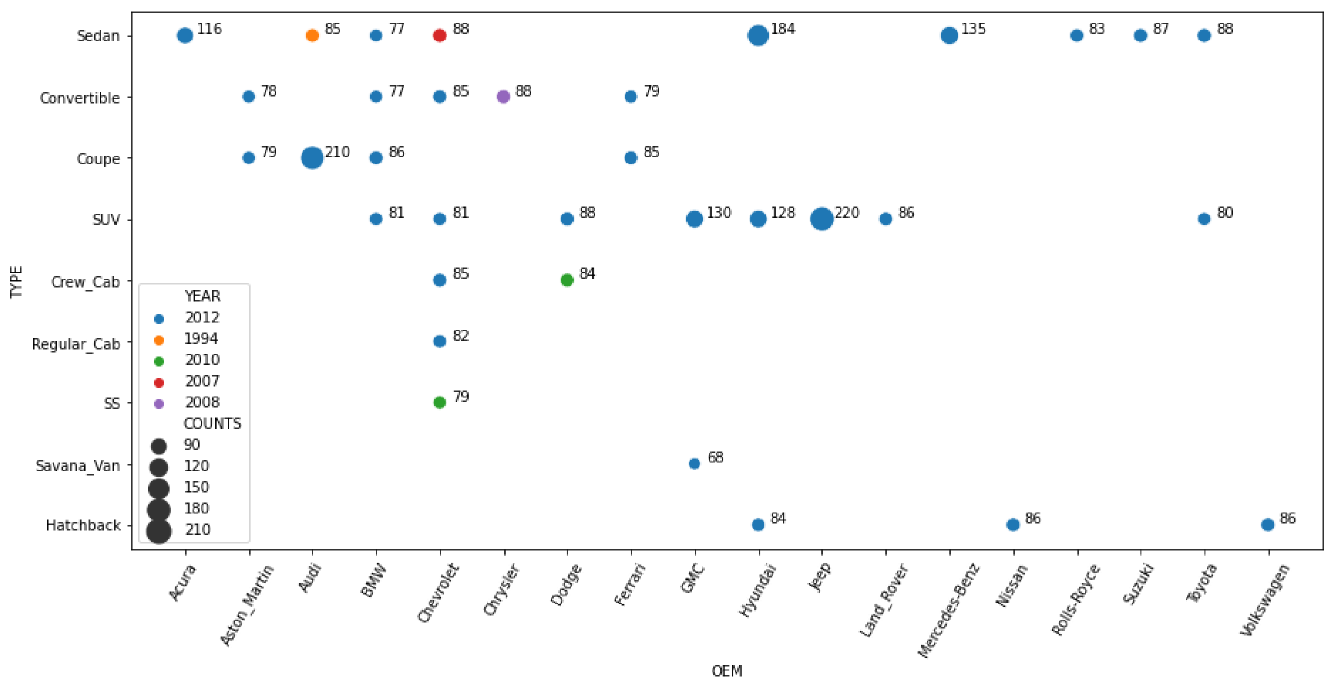
Number of images in each make year



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

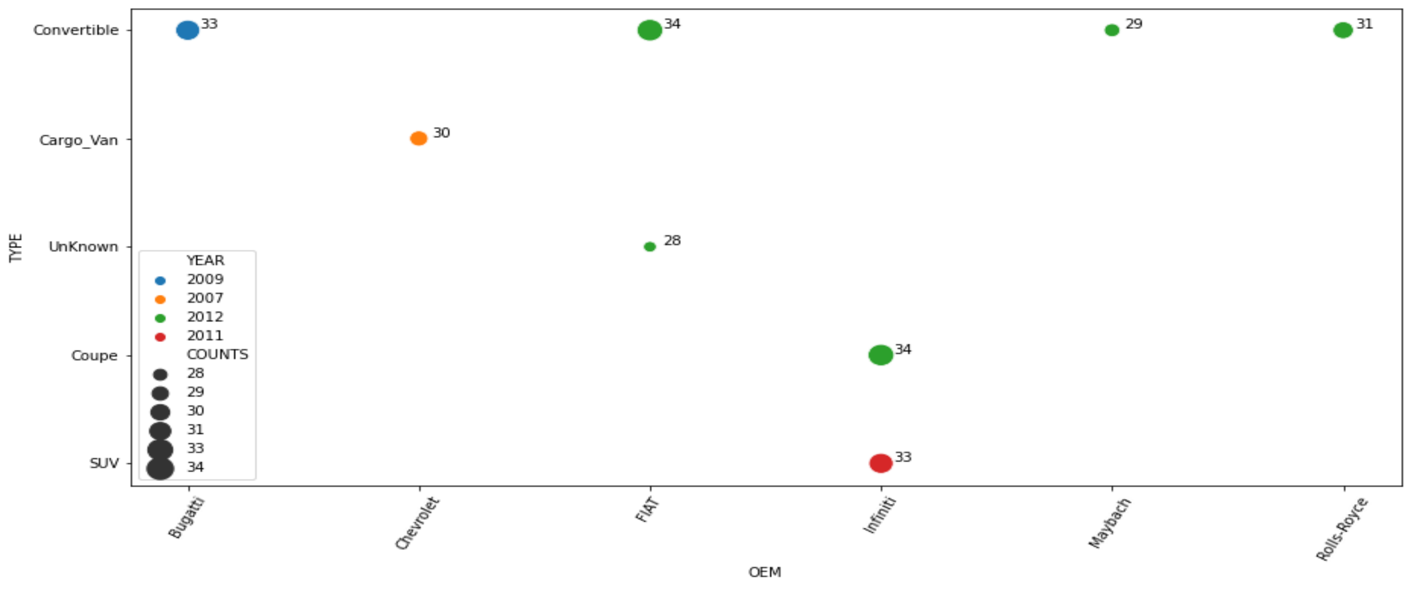
We further evaluate combined frequencies for various models as the inputs would be going in terms of 196 training classes (multiclass training)

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



We find that although Chevrolet has maximum 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 there are inherent biases in terms of car make year so the model that would be design would be appropriate for lower range in terms of make year. Further, the data for OEM's and car type also have biases which would need to be dealt with through data augmentation to design a generalized model.

Data Preprocessing

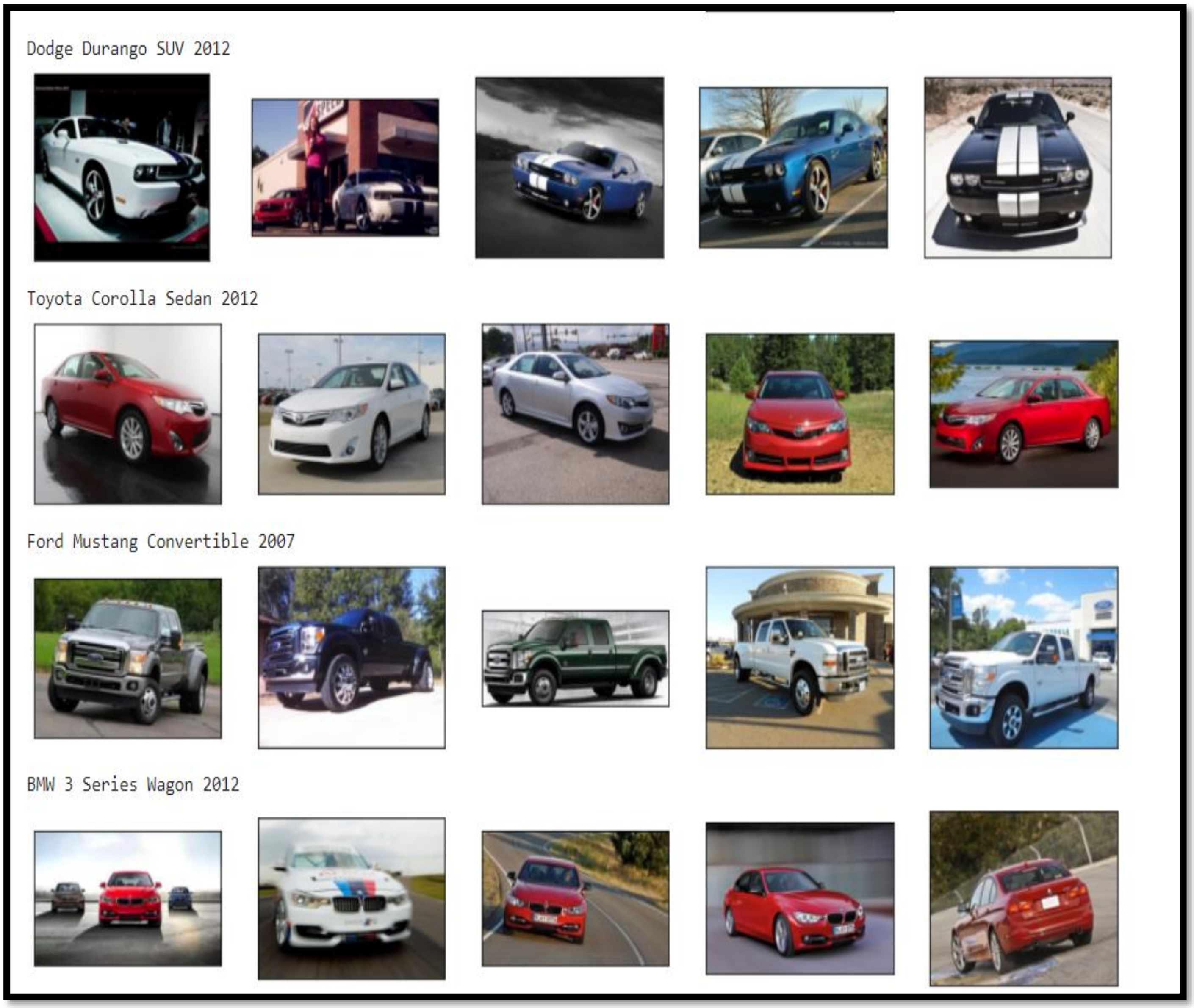
The acquired data from the real world is usually messy and come from different sources. To feed them to the machine learning or neural network-based model, they need to be standardized and cleaned up.

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

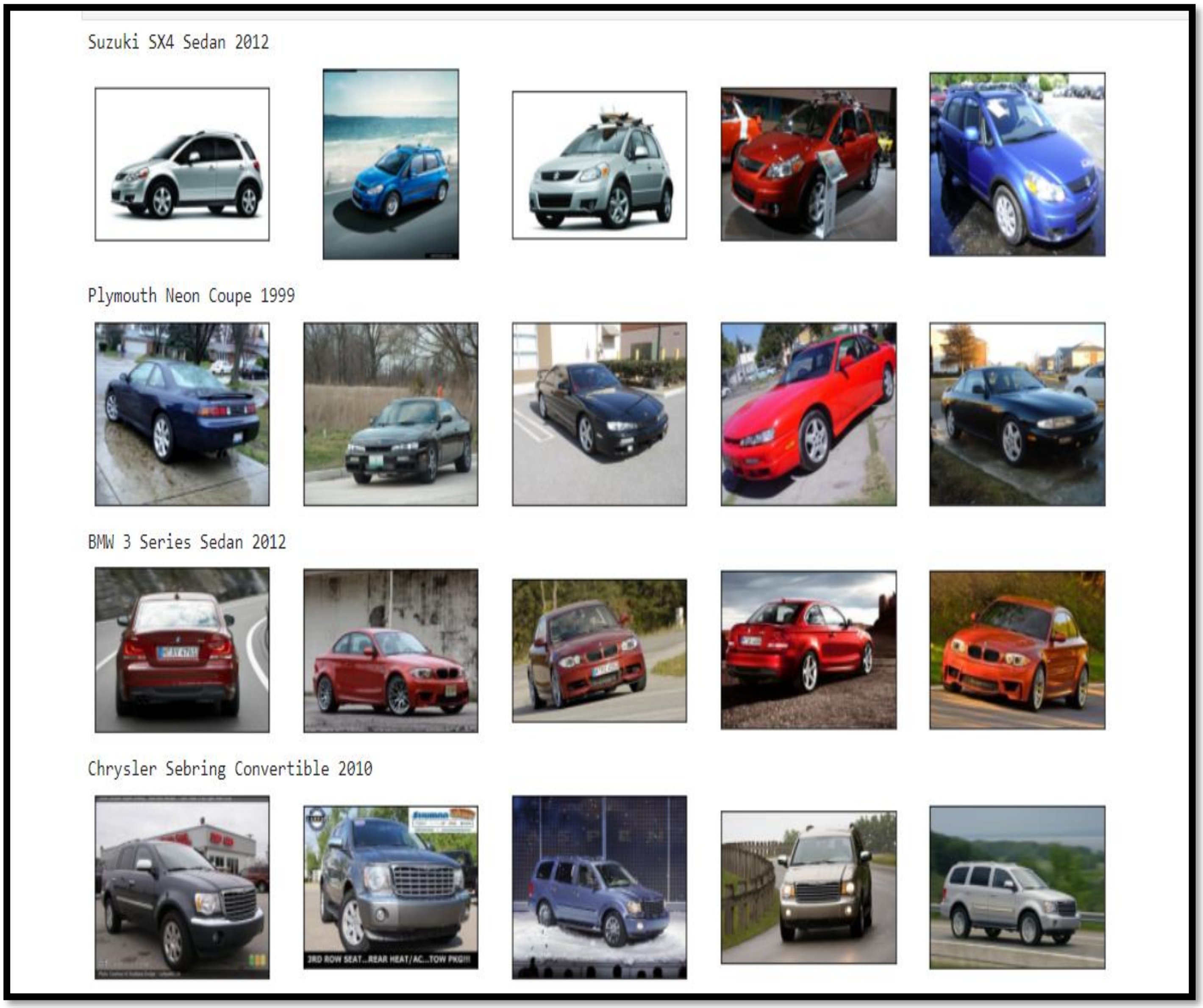
in which an image is taken would be very cumbersome and resource constraining thus, when an image is acquired, its first converted into a form that allows a general algorithm to solve it.

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


Dodge Durango SUV 2012
Toyota Corolla Sedan 2012
Ford Mustang Convertible 2007
BMW 3 Series Wagon 2012

Snap shot Training Images

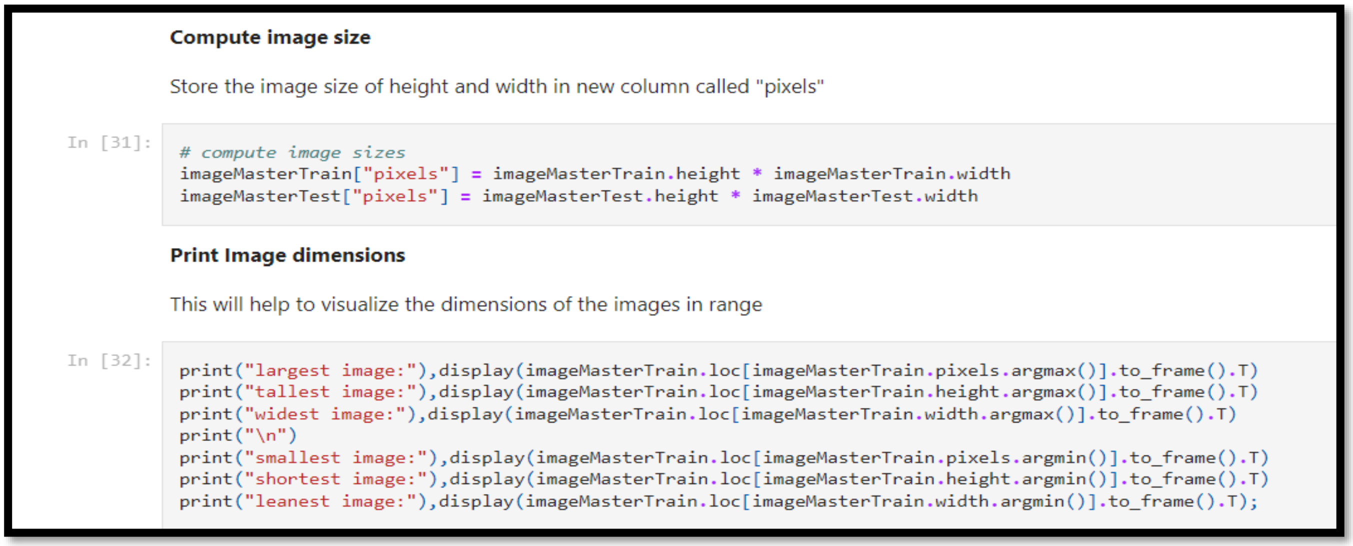


The image data preprocessing included

1. Image size resizing
2. Augmentation of the bounding boxes

Image resizing

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



Compute image size

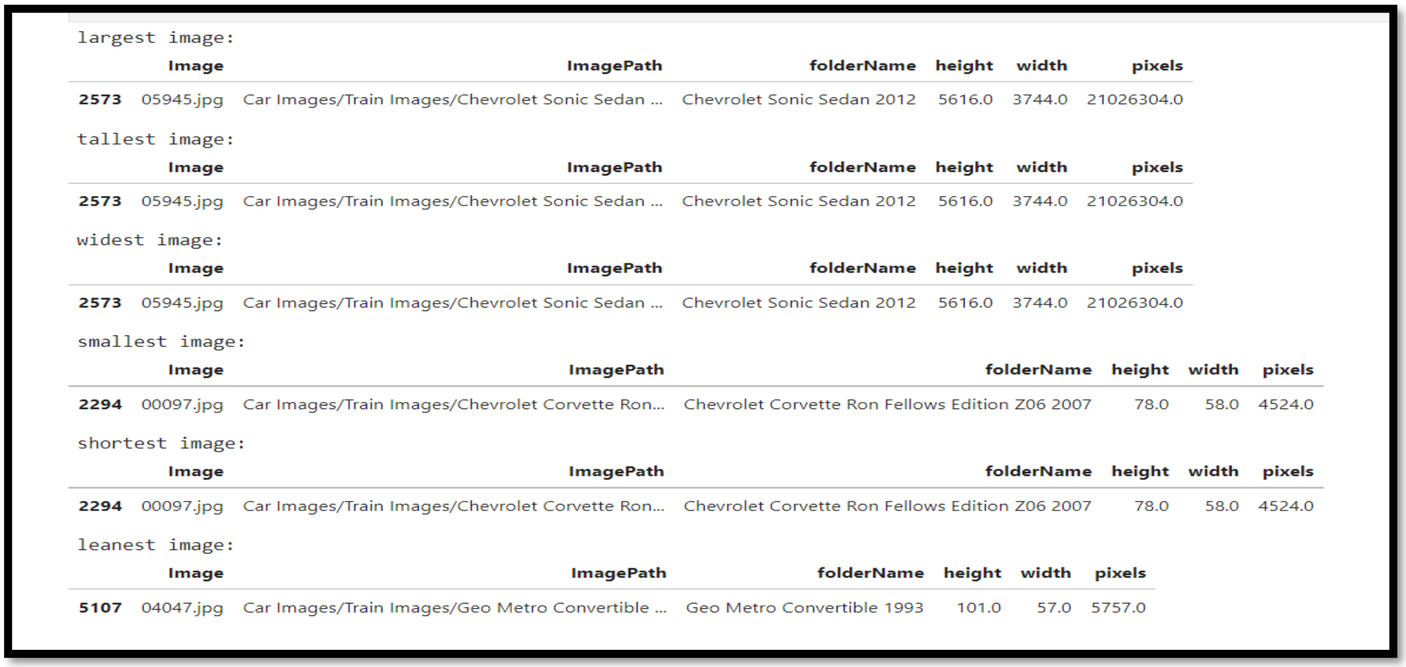
Store the image size of height and width in new column called "pixels"

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

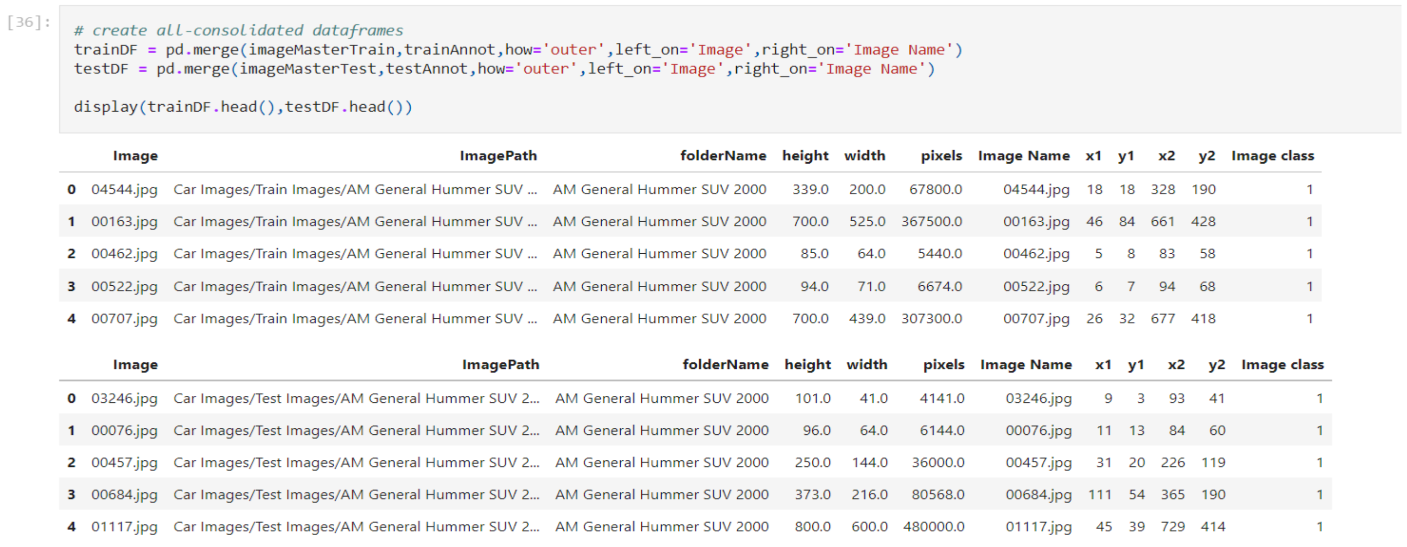
Based on above review, we shall restrict the image size fed to the network at 50x50 pixels, so as not to deteriorate lower resolution images and thus affect model capabilities. However, as a standard best practice image size will either be fixed as the image size median for the sample or specification as per the algorithm used will be applied in specific model testing e.g. VGG 16 takes in 224 x 224 , YOLO V3 416 x 416.

```
# 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 size we still have the images with different backgrounds and resolutions. We now use the already available annotations to create bounding boxes.

Adding Bounding Boxes

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

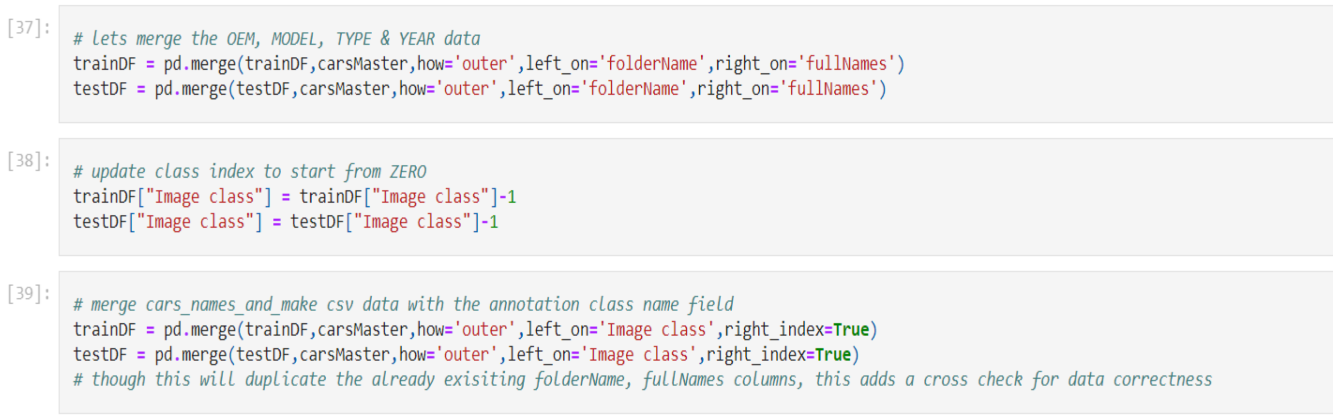


```
[36]: # create all-consolidated dataframes
      trainDF = pd.merge(imageMasterTrain,trainAnnot,how='outer',left_on='Image',right_on='Image Name')
      testDF = pd.merge(imageMasterTest,testAnnot,how='outer',left_on='Image',right_on='Image Name')

      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

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

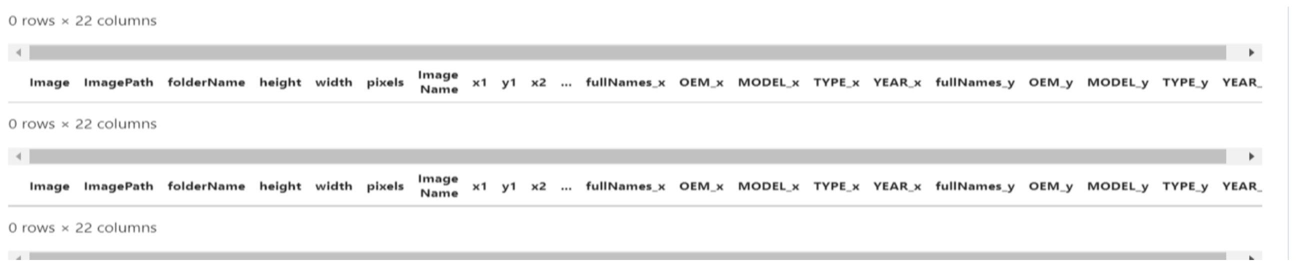


```
[37]: # lets merge the OEM, MODEL, TYPE & YEAR data
      trainDF = pd.merge(trainDF,carsMaster,how='outer',left_on='folderName',right_on='fullNames')
      testDF = pd.merge(testDF,carsMaster,how='outer',left_on='folderName',right_on='fullNames')

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

[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



0 rows × 22 columns

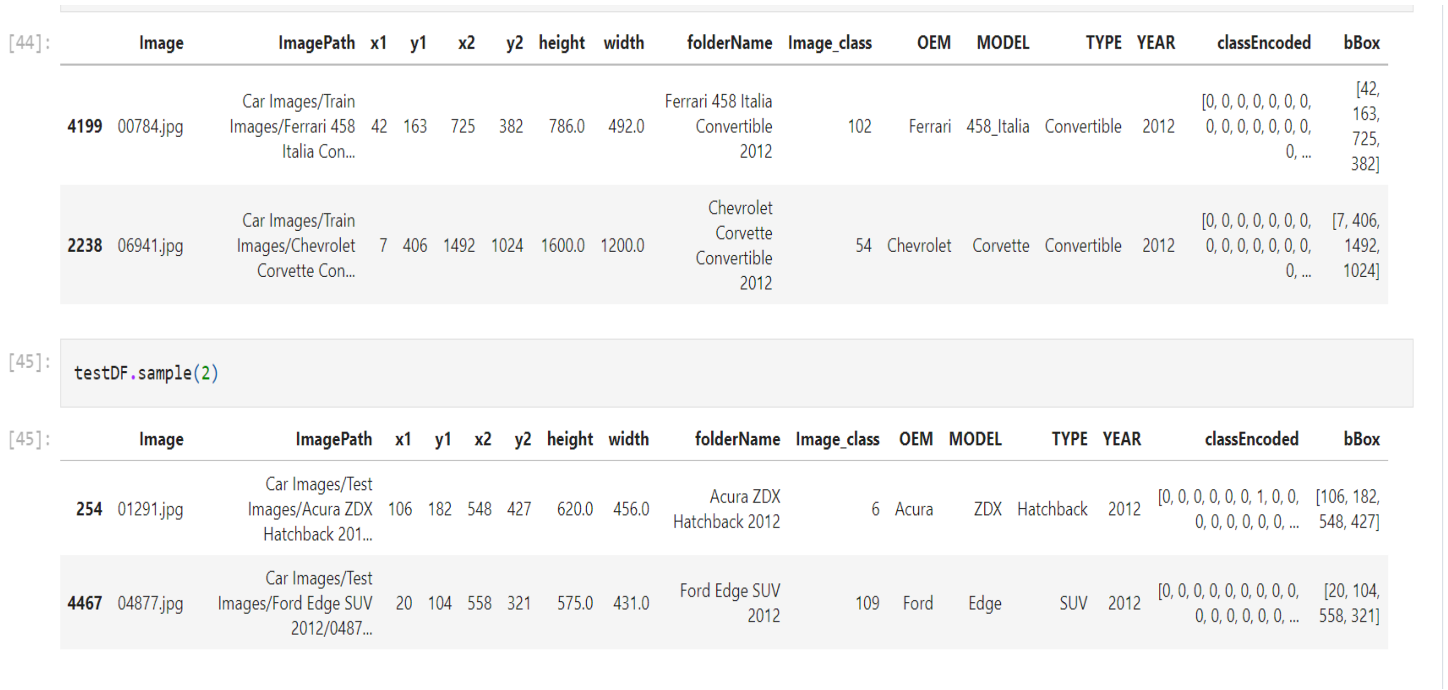
	Image	ImagePath	folderName	height	width	pixels	Image Name	x1	y1	x2	...	fullNames_x	OEM_x	MODEL_x	TYPE_x	YEAR_x	fullNames_y	OEM_y	MODEL_y	TYPE_y	YEAR_

0 rows × 22 columns

	Image	ImagePath	folderName	height	width	pixels	Image Name	x1	y1	x2	...	fullNames_x	OEM_x	MODEL_x	TYPE_x	YEAR_x	fullNames_y	OEM_y	MODEL_y	TYPE_y	YEAR_

0 rows × 22 columns

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



[44]:

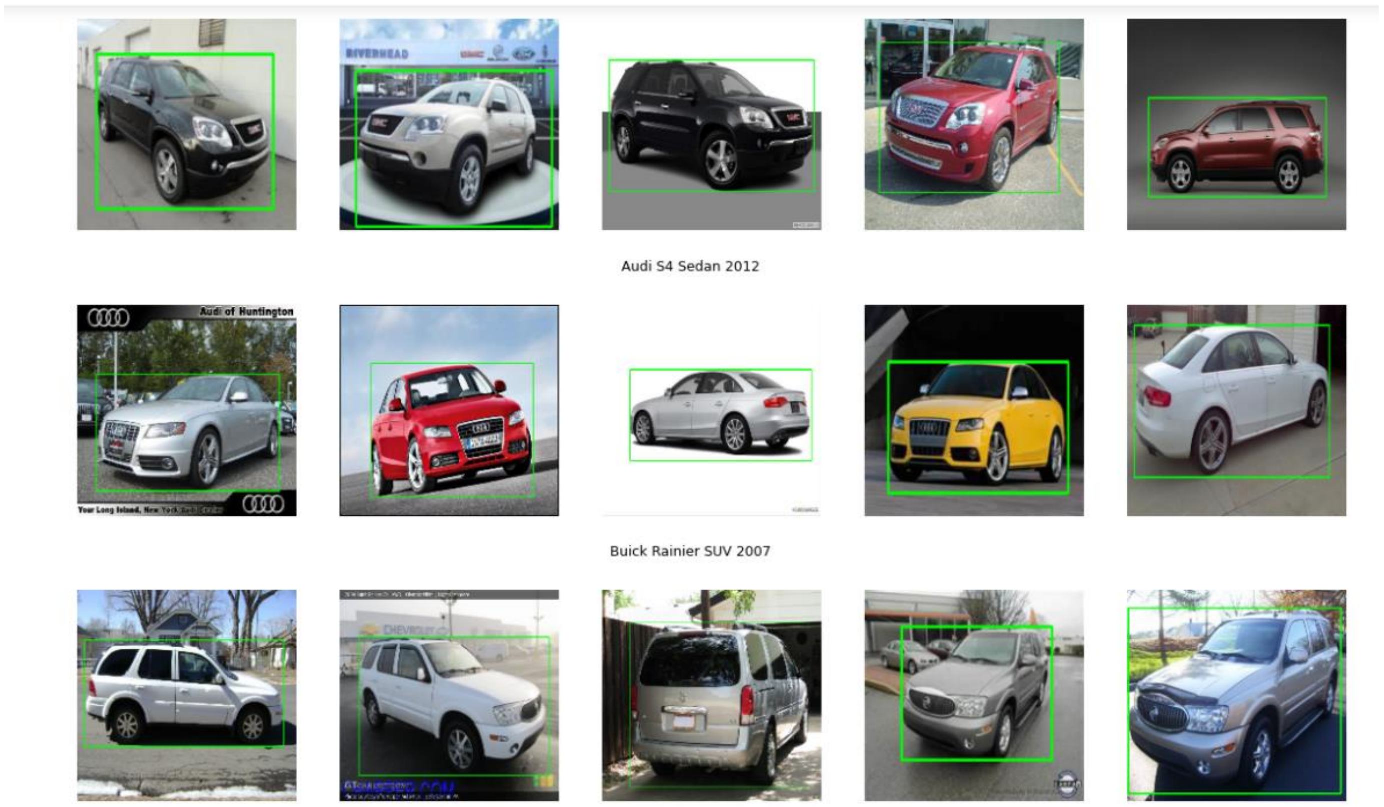
	Image	ImagePath	x1	y1	x2	y2	height	width	folderName	Image_class	OEM	MODEL	TYPE	YEAR	classEncoded	bBox
4199	00784.jpg	Car Images/Train Images/Ferrari 458 Italia Con...	42	163	725	382	786.0	492.0	Ferrari 458 Italia Convertible 2012	102	Ferrari	458_Italia	Convertible	2012	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[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, ...	[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, ...	[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, ...	[20, 104, 558, 321]

Null values are checked and visualization of images with bounding boxes



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 w.r.t to problem in hand was performed . Further two very simple trial models Mode & Median and Random model which were developed from scratch along with two established algorithms mobileNet & VGG16 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 17 different computer vision models from conventional to state of art with respect to their mechanism, efficiency advantages and disadvantages (annexure 1). The table below delineates the key models explored for their appropriateness for model building. We will be trying as many models from this as possible within the existing constrains of computational band width and free availability of the model for download and use

S. No	Model Name	S. No	Model Name
1	VGG 16	10	Inception Net
2	Mobile Net	11	Faster RCNN
3	VGG 19	12	Google Net V2
4	Mask RCNN	13	Alex Net
5	Single Shot Detector (SSD)	14	Detectron
6	YOLO	15	Efficient Det D7
7	YOLO 5	16	DETR/Deformable DETR
8	SPPNET	17	Varifocal Net
9	Res NET	18	YOLO 6

The initial runs have been performed with simple deep learning models. The following basic rubric has been finalized to evaluate multiple models for the final model selection process

I. The final model should be computationally light but high accuracy results from the given dataset
II. As the model will be GUI integrated speed of processing should be lowest without impacting the accuracy

III. Provides easy Integration with GUI and easily expandable for multiple business cases
IV. Can be built on top of any pretrained model (transfer model) to bring better accuracy and performance.
V. The evaluation matrix would include accuracy, precision , recall , loss, F1, IOU,

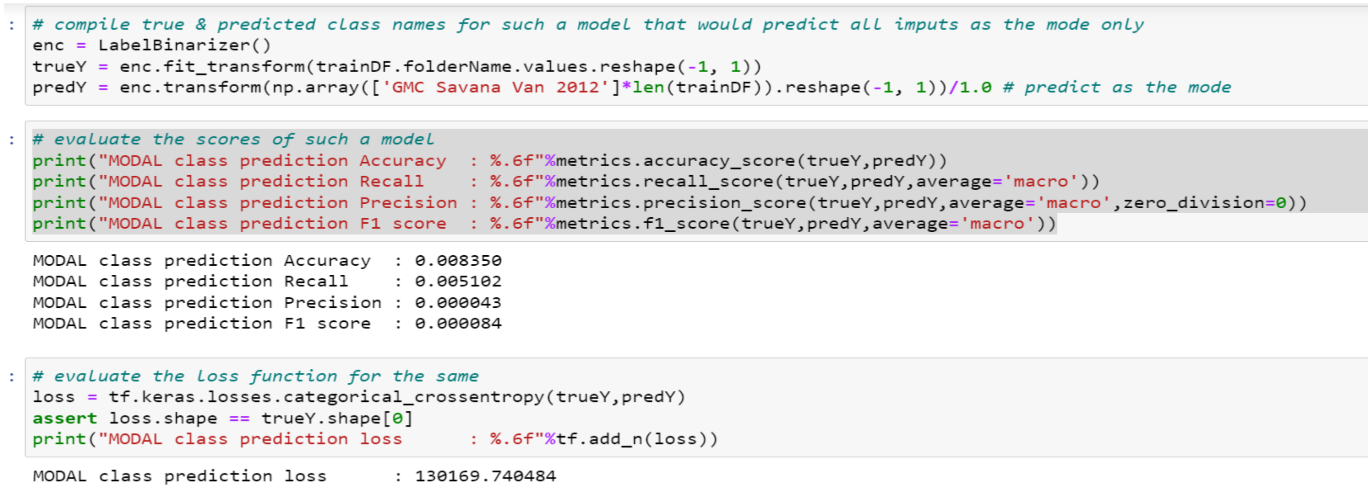
In the initial phase we chose simple models to be run through model pipeline to identify key spaces of lacuna that can be filled by specialized models in future work

Working with the initial model choices

We used four models

Trail Model 1 Median and Mode

This model was developed to predict the mode of training data compare any new model training against this



```
# compile true & predicted class names for such a model that would predict all imputs as the mode only
enc = LabelBinarizer()
trueY = enc.fit_transform(trainDF.folderName.values.reshape(-1, 1))
predY = enc.transform(np.array(['GMC Savana Van 2012']*len(trainDF)).reshape(-1, 1))/1.0 # predict as the mode
```

```
# evaluate the scores of such a model
print("MODAL class prediction Accuracy  : %.6f"%metrics.accuracy_score(trueY,predY))
print("MODAL class prediction Recall    : %.6f"%metrics.recall_score(trueY,predY,average='macro'))
print("MODAL class prediction Precision : %.6f"%metrics.precision_score(trueY,predY,average='macro',zero_division=0))
print("MODAL class prediction F1 score  : %.6f"%metrics.f1_score(trueY,predY,average='macro'))
```

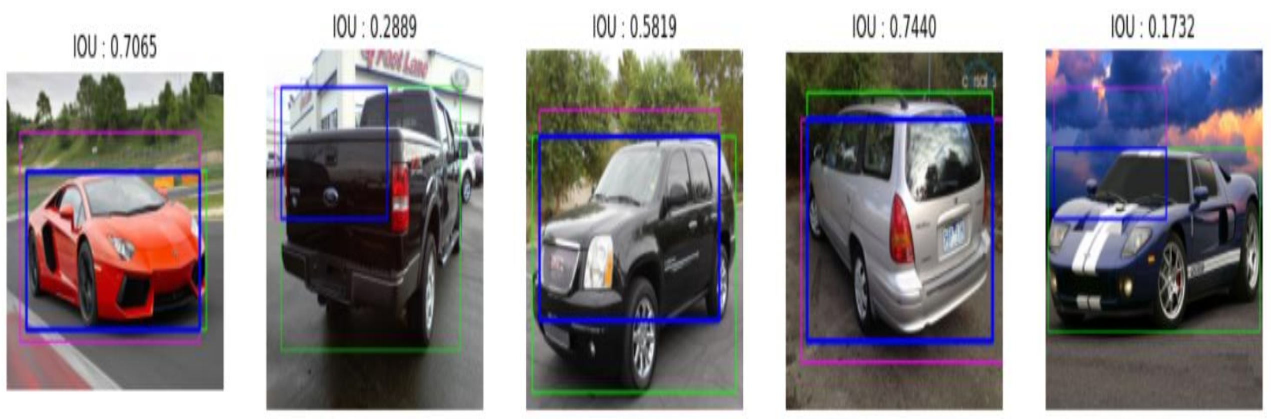
```
MODAL class prediction Accuracy  : 0.008350
MODAL class prediction Recall    : 0.005102
MODAL class prediction Precision : 0.000043
MODAL class prediction F1 score  : 0.000084
```

```
# evaluate the loss function for the same
loss = tf.keras.losses.categorical_crossentropy(trueY,predY)
assert loss.shape == trueY.shape[0]
print("MODAL class prediction loss       : %.6f"%tf.add_n(loss))
```

```
MODAL class prediction loss      : 130169.740484
```

```
: # compute coordinates of intersection of bounding boxes
  def inter(max_dims,y_true,y_pred):
      # ensure dimensions
      y_true = y_true.copy().reshape(1,-1) if len(y_true.shape)==1 else y_true
      y_pred = y_pred.copy().reshape(1,-1) if len(y_pred.shape)==1 else y_pred
      max_dims = max_dims.copy().reshape(1,-1) if len(max_dims.shape)==1 else max_dims
      # intersection coordinates (x1,y1)
      x1 = np.maximum(y_true[:,0],y_pred[:,0])
      x1 = np.maximum(np.zeros(y_true.shape[0],dtype=int),x1) # maximum within image
      y1 = np.maximum(y_true[:,1],y_pred[:,1])
      y1 = np.maximum(np.zeros(y_true.shape[0],dtype=int),y1) # maximum within image
      # intersection coordinates (x2,y2)
      x2 = np.minimum(y_true[:,2],y_pred[:,2])
      x2 = np.minimum(max_dims[:,0],x2) # minimum within image
      y2 = np.minimum(y_true[:,3],y_pred[:,3])
      y2 = np.minimum(max_dims[:,1],y2) # minimum within image

      return (x1,y1,x2,y2)
```

```
In [14]: # lets display a few images with the bounding boxes, intersection & IOU
         indices = np.random.choice(range(len(images)),5)
         #indices = np.array([4066, 3122, 6133, 2252, 7679])
         plt.figure(figsize=(20,10))
         tfi = tf.keras.preprocessing.image
         for i,index in enumerate(indices):
             plt.subplot(1,5,i+1)
             img = tfi.load_img(images[index][0]) # read image
             max_dims = np.array(img.size)
             img = tfi.img_to_array(img)
             cv2.rectangle(img,tuple(trueYbox[index][:2]),tuple(trueYbox[index][2:]),(0,255,0),2); # embed true bBox
             cv2.rectangle(img,tuple(predYbox[index][:2]),tuple(predYbox[index][2:]),(255,0,255),2); # embed predicted bBox
             Ix1,Iy1,Ix2,Iy2 = inter(max_dims,trueYbox[index],predYbox[index]) # calculate intersection
             cv2.rectangle(img,(Ix1[0],Iy1[0]),(Ix2[0],Iy2[0]),(0,0,255),5); # highlight Union
             img = tfi.array_to_img(img)
             plt.axis('off')
             iou = IOU(max_dims,trueYbox[index],predYbox[index]) # calculate IOU metric
             plt.title("IOU : %.4f"%iou)
             plt.imshow(img);
         plt.show()
```



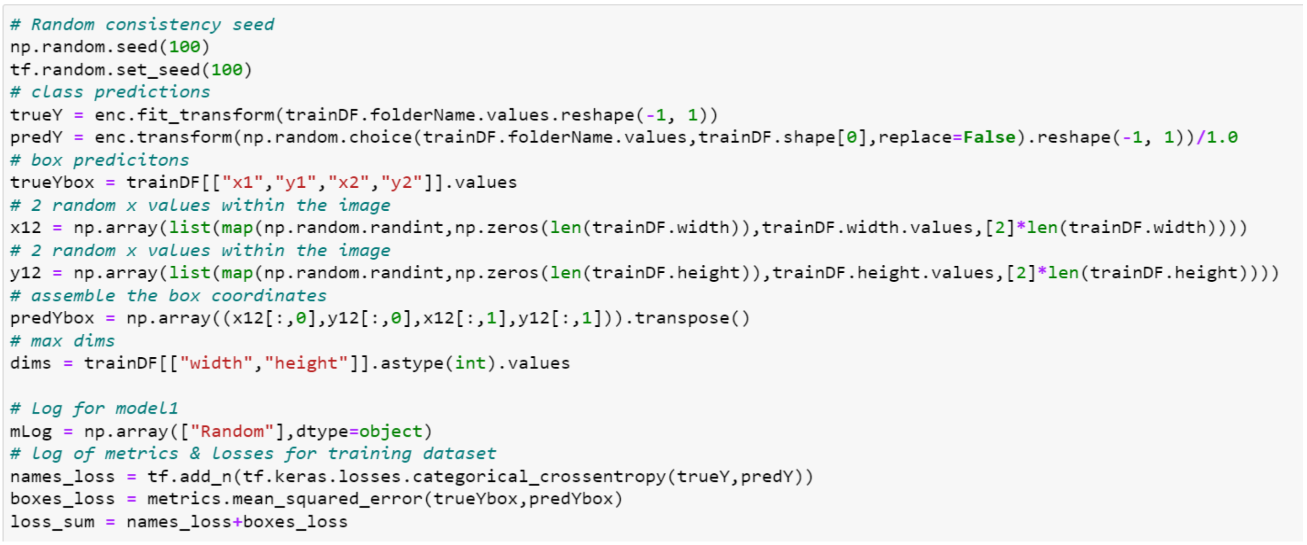
Observations:

- the central tendencies of the input dataset is referred as the generic model output
- the results has extremely huge losses, yet the bounding box IOU has achived a decent 0.26 for an unlearned model

- class prediction accuracies are less than 1%, and its understandable for the 196 possibilities

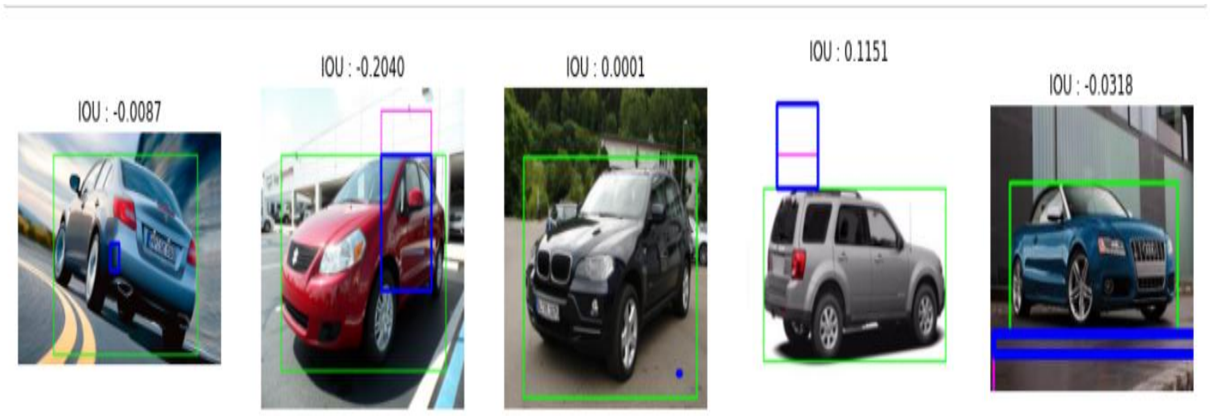
Trial Model 2 Random Model

In this model we do not take any consideration of distributions as we did in the previous model but try random sies (height and Width) for bounding boes and random selection of classes .



```
# Random consistency seed
np.random.seed(100)
tf.random.set_seed(100)
# class predictions
trueY = enc.fit_transform(trainDF.folderName.values.reshape(-1, 1))
predY = enc.transform(np.random.choice(trainDF.folderName.values,trainDF.shape[0],replace=False).reshape(-1, 1))/1.0
# box predicitons
trueYbox = trainDF[["x1","y1","x2","y2"]].values
# 2 random x values within the image
x12 = np.array(list(map(np.random.randint,np.zeros(len(trainDF.width)),trainDF.width.values,[2]*len(trainDF.width))))
# 2 random x values within the image
y12 = np.array(list(map(np.random.randint,np.zeros(len(trainDF.height)),trainDF.height.values,[2]*len(trainDF.height))))
# assemble the box coordinates
predYbox = np.array((x12[:,0],y12[:,0],x12[:,1],y12[:,1])).transpose()
# max dims
dims = trainDF[["width","height"]].astype(int).values

# Log for model1
mLog = np.array(["Random"],dtype=object)
# log of metrics & losses for training dataset
names_loss = tf.add_n(tf.keras.losses.categorical_crossentropy(trueY,predY))
boxes_loss = metrics.mean_squared_error(trueYbox,predYbox)
loss_sum = names_loss+boxes_loss
```

*please refer to full code in the submitted code book. The figures below provide the performance of the model with respect to IOU values of bounding boxes



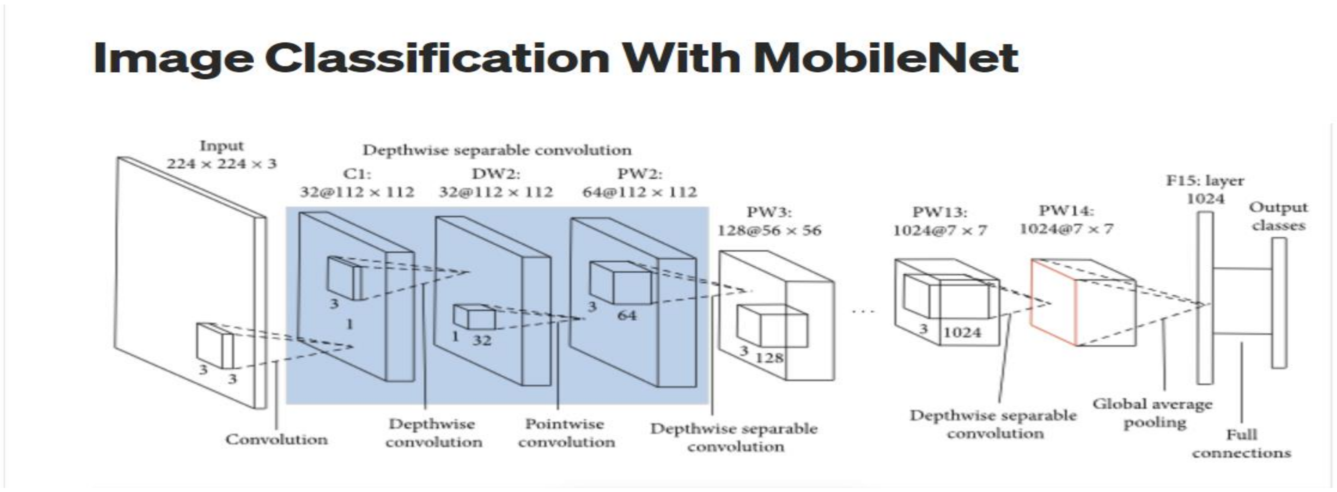
Observations: Ignoring any information about the input distributions, random values from the target space was considered as the prediction

- huge loses witnessed again, and the random bounding box seems to cover IOU of 0.37 in testing dataset
- class prediction accuracies are less than 1%

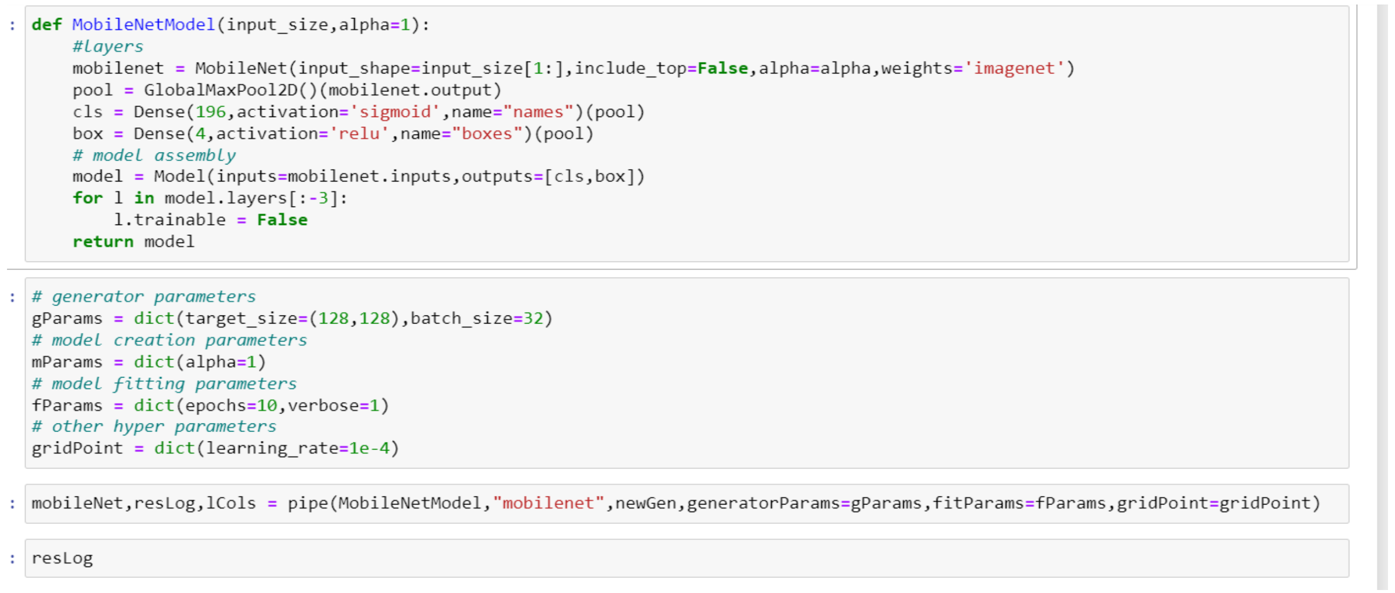
These two preliminary models provide us a significant target for the trained models. We train a Mobile Net and VGG 16 to start with

Trial Model 3 Mobile Net

We did an initial run using mobile net for *car classification*. 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.
Reduced network size - 17MB.
Reduced number of parameters - 4.2 million.
Faster in performance and are useful for mobile applications.
Small, low-latency convolutional neural network

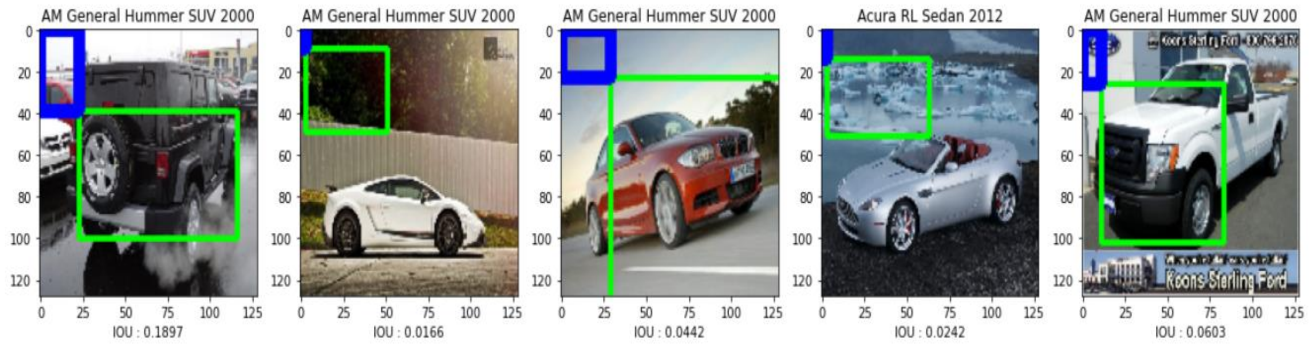


```
def MobileNetModel(input_size,alpha=1):
    #layers
    mobilenet = MobileNet(input_shape=input_size[1:],include_top=False,alpha=alpha,weights='imagenet')
    pool = GlobalMaxPool2D()(mobilenet.output)
    cls = Dense(196,activation='sigmoid',name="names")(pool)
    box = Dense(4,activation='relu',name="boxes")(pool)
    # model assembly
    model = Model(inputs=mobilenet.inputs,outputs=[cls,box])
    for l in model.layers[:-3]:
        l.trainable = False
    return model
```

```
# generator parameters
gParams = dict(target_size=(128,128),batch_size=32)
# model creation parameters
mParams = dict(alpha=1)
# model fitting parameters
fParams = dict(epochs=10,verbose=1)
# other hyper parameters
gridPoint = dict(learning_rate=1e-4)
```

```
mobileNet,resLog,lCols = pipe(MobileNetModel,"mobilenet",newGen,generatorParams=gParams,fitParams=fParams,gridPoint=gridPoint)
```

```
resLog
```


AM General Hummer SUV 2000
IOU : 0.1897
AM General Hummer SUV 2000
IOU : 0.0166
AM General Hummer SUV 2000
IOU : 0.0442
Acura RL Sedan 2012
IOU : 0.0242
AM General Hummer SUV 2000
IOU : 0.0603

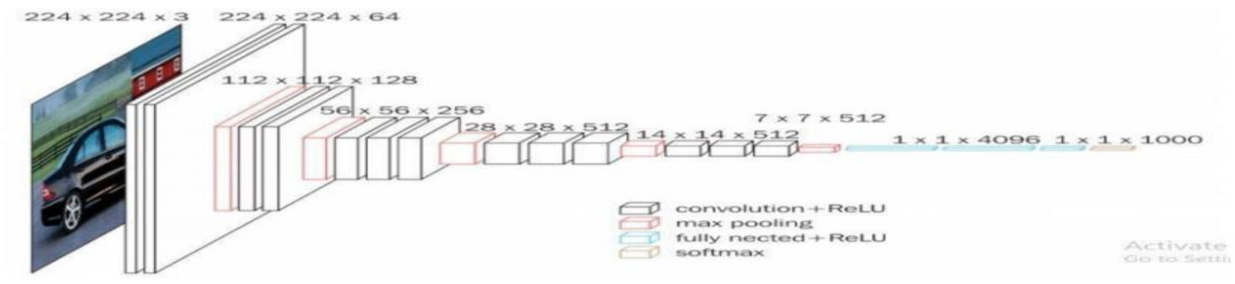
Observations:

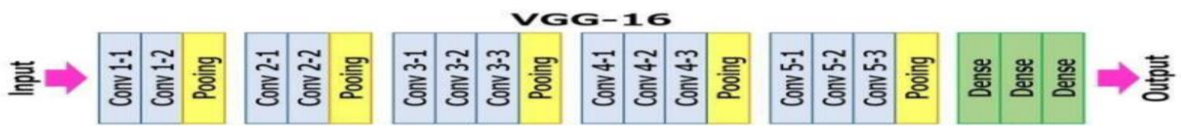
- transfer learning of MobileNet was implemented, with its layers frozen
- the losses are far less (possibly cause by scaling of input data between 0 & 1), hence not directly comparable with the previous two models
- The class prediction accuracy are shut to ZERO, and needs to explored before hyperparameter tuning
- the class recall score shows valuable growth in performance (0.688 in testing data)

- the bounding box IOU has secured only 0.06 in testing data, which infers that there is not much learning. the same is substantiated with the negative IOU in training data

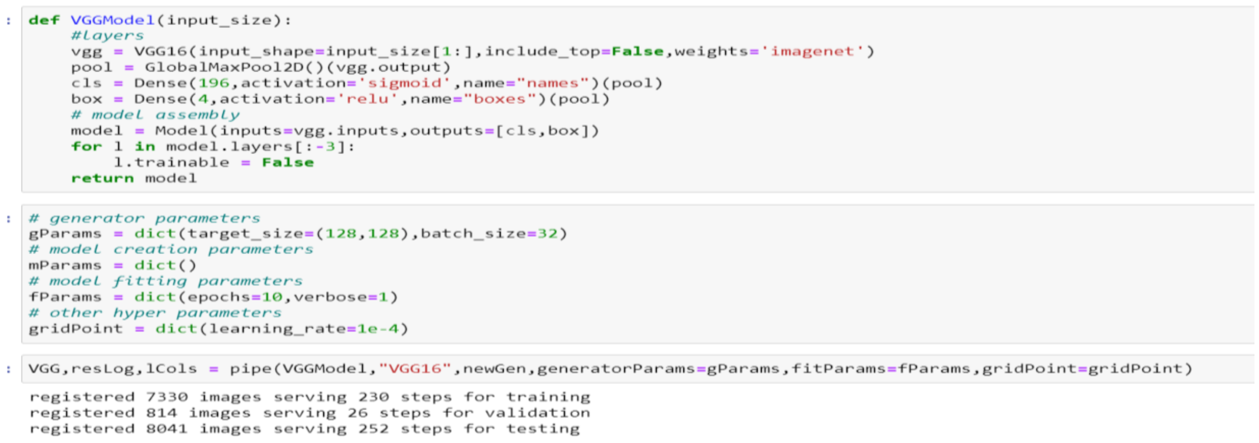
Trial Model 4 VGG 16

VGG16 is object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.





The following model was run and prediction carried out

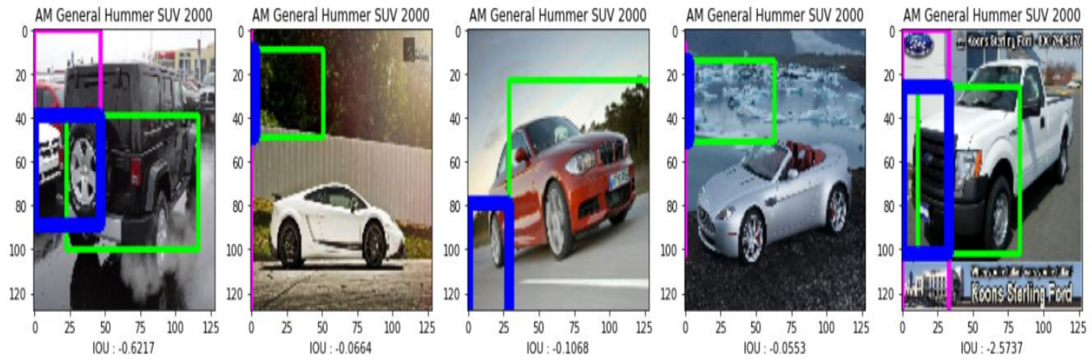


```
: def VGGModel(input_size):
      #layers
      vgg = VGG16(input_shape=input_size[1:],include_top=False,weights='imagenet')
      pool = GlobalMaxPool2D()(vgg.output)
      cls = Dense(196,activation='sigmoid',name="names")(pool)
      box = Dense(4,activation='relu',name="boxes")(pool)
      # model assembly
      model = Model(inputs=vgg.inputs,outputs=[cls,box])
      for l in model.layers[:-3]:
          l.trainable = False
      return model
```

```
: # generator parameters
  gParams = dict(target_size=(128,128),batch_size=32)
  # model creation parameters
  mParams = dict()
  # model fitting parameters
  fParams = dict(epochs=10,verbose=1)
  # other hyper parameters
  gridPoint = dict(learning_rate=1e-4)
```

```
: VGG,resLog,lCols = pipe(VGGModel,"VGG16",newGen,generatorParams=gParams,fitParams=fParams,gridPoint=gridPoint)

  registered 7330 images serving 230 steps for training
  registered 814 images serving 26 steps for validation
  registered 8041 images serving 252 steps for testing
```

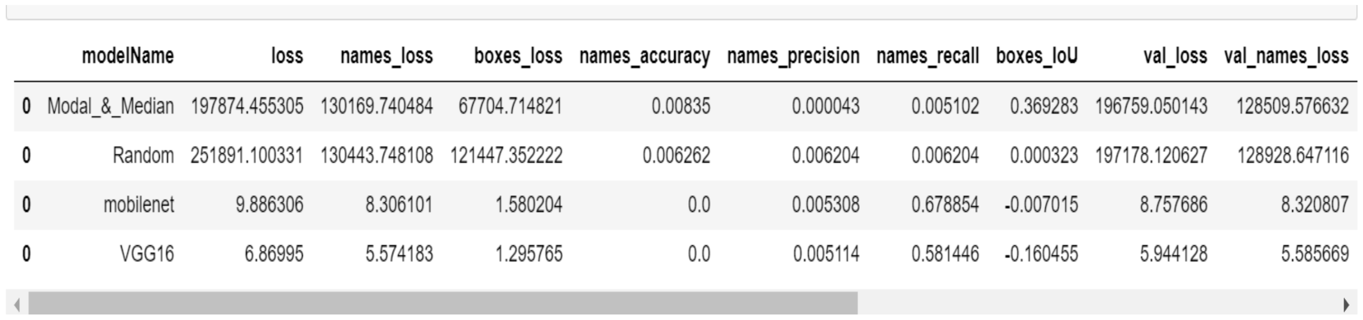
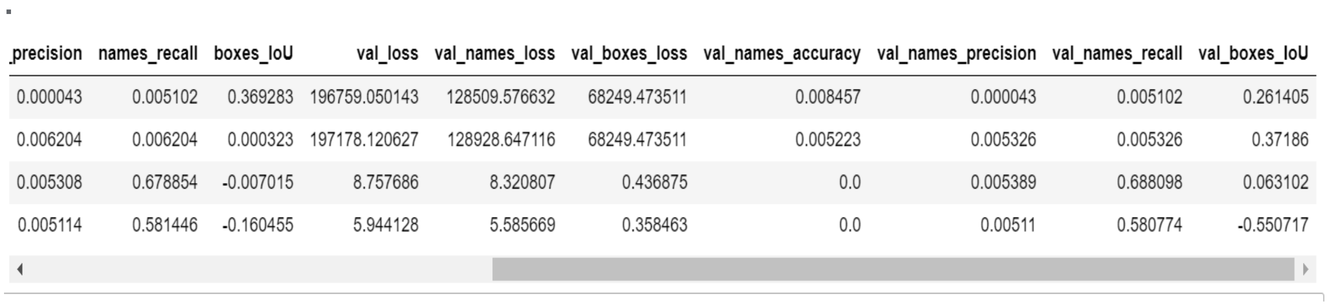
IOU values were evaluated as illustrated below


AM General Hummer SUV 2000
IOU : -0.6217
AM General Hummer SUV 2000
IOU : -0.0664
AM General Hummer SUV 2000
IOU : -0.1068
AM General Hummer SUV 2000
IOU : -0.0553
AM General Hummer SUV 2000
IOU : -2.5737

Observations :

- transfer learning of VGG16 was implemented, with its layers frozen
- the losses are notionally lesser in comparison with MobileNet
- the bounding box predictions are also inferring that not much of a learning has happened

Overall Performance Table for All Models

	modelName	loss	names_loss	boxes_loss	names_accuracy	names_precision	names_recall	boxes_IoU	val_loss	val_names_loss	val_boxes_loss	val_names_accuracy	val_names_precision	val_names_recall	val_boxes_IoU
0	Modal_&_Median	197874.455305	130169.740484	67704.714821	0.00835	0.000043	0.005102	0.369283	196759.050143	128509.576632	68249.473511	0.008457	0.000043	0.005102	0.261405
0	Random	251891.100331	130443.748108	121447.352222	0.006262	0.006204	0.006204	0.000323	197178.120627	128928.647116	68249.473511	0.005223	0.005326	0.005326	0.37186
0	mobilenet	9.886306	8.306101	1.580204	0.0	0.005308	0.678854	-0.007015	8.757686	8.320807	0.436875	0.0	0.005389	0.688098	0.063102
0	VGG16	6.86995	5.574183	1.295765	0.0	0.005114	0.581446	-0.160455	5.944128	5.585669	0.358463	0.0	0.00511	0.580774	-0.550717

The results reveal that the tested models are not reliable enough and more detailed configuration with hyperparameter training and preprocessing would be required to design a sufficiently reliable and generalized classification model for deployment. The next section provides details and insights for model performance improvements.

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

.

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 However these choices are easy to implement and computationally light *, we find that* they do not *show a satisfactory* perf*ormance at* image classification. 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

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

Class imbalance will be reduced by following few methods

Change the Performance Metric

Rather than just focusing on accuracy score, other performance metrics like confusion matrics, precision, recall, F1 score, and Area Under ROC Curve will be considered for evaluation. Other option can be balanced_accuracy_score that computes balanced accuracy that can deal with imbalanced dataset for both binary or multiclass classification problems. **Random Resampling:**

It consists of oversampling of the minority class and the under sampling of the majority class. We can use custom image generators to increase the number of images with bounding boxes to make the data balanced.

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

Dealing with Multiple spatial scales and aspect ratios:

In our cars dataset the image size varies in height, width and pixels. eg;
There are images of width 5616 px, which is 72 times in width that the image with smallest width of 78p. We have images of height 3744 px, which is 65 times in height that the image with smallest height of 57px. We have images of pixels 21026304 px, which is 4647 times high in dimension than the image with least pixel value of 4524 px. There is also the problem of not having same dimensions of height and width for the images (rectangular images). To address this issue, we will be evaluating multiple techniques like bilinear interpolation, use of padding to balance the dimension of height and width, models with FPN (Feature pyramid network) to increase detection accuracy when there are objects with various scales in the image while not changing detection speed.

Future Work

Future work will pick up from the existing exercise of preliminary 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

Annexure

Evaluating 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	Transfer network , pretrained weights avaialble	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	**Inception v2** is the second generation of Inception convolutional neural network architectures which notably uses batch normalization. Other changes include dropping dropout and	

			removing local response normalization, due to the benefits of batch normalization.	
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	1. Detectron2 makes it easier to experiment with different

				hyperparameters 2. Detectron2 is Facebook AI Research’s next-generation software system that implements state-of-the-art object detection algorithms 3. Provides a very easy API to extract scoring results 4. Panoptic segmentation 5. Flexible and fast training on single or multiple GPU servers 6. Built using Pytorch, which has a very active community and continuous up-gradation & bug fixes
8	Efficient Det 7	Efficient Det has been developed by google	The EfficientDet family of models achieves better accuracy and efficiency than previous detectors across a wide range of accuracy levels and resource constraints. It uses compound scaling so performance of the model in terms of both accuracy and efficiency has been improved	
9	Google Net V2	GoogLeNet is 22-layer deep convolutional neural network thats variant of the inception Net		
10	Alex Net	AlexNet was the first major CNN model that used GPU’s for training.	This lead to faster training of models.	Alex Net is much slower that state of art models used today

			The ReLu activation function used in this network has 2 advantages. It does not limit the output unlike other activation functions. This means there isn't too much loss of features.	
11	Mobile Net	In 2017, **Google** introduced MobileNets, a family of computer vision models based on TensorFlow	MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases.	There efficiencies are usually lower than other models
12	VGG 16	VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University	VGG brought with it a massive improvement in accuracy and an improvement in speed as well. The increase in the number of layers with smaller kernels saw an increase in non-linearity which is always a positive in deep learning.These were the first pretrainable models introduced.	model experiences the vanishing gradient problem VGG is slower than the newer ResNet architecture
13	Mask RCNN	Mask R-CNN was proposed by Kaiming He et al. in 2017. It is very similar to Faster R-CNN except there is another layer to predict segmented	Mask R-CNN is a Convolutional Neural Network (CNN) and state-of-the-art in terms of image segmentation. Mask R-CNN is an extension of Faster R-CNN and works by adding a branch for predicting an object mask (Region of Interest) in parallel with the existing branch for bounding box recognition.	
14	Single Shot Detector	By using SSD, we only need to **take one single shot to detect multiple objects within the image**, while regional proposal network (RPN)	Thus, SSD is much faster compared with two-shot RPN-based approaches.	

		based approaches such as R-CNN series that need two shots,		
15	SPPNET	Spatial Pyramid Pooling layer was introduced in SPPNET , which makes the CNN model agnostic of input image size.	They provide 24 to 64 times speed up in the model as compared to RCNN	
16	DETR/Deformable DETR	1. from Facebook AI released in May 2020 2. Uses an intersection of NLP(Natural Language Processing) and Computer Vision to accomplish the object detection task.	1. demonstrates accuracy and run-time performance on par with the well-established and highly-optimized Faster RCNN 2. Can be easily generalized to produce panoptic segmentation in a unified manner	1. it suffers from slow convergence and limited feature spatial resolution, due to the limitation of Transformer attention modules in processing image feature maps 2. Deformable DETR can achieve better performance than DETR (especially on small objects) with 10 times less training epochs
17	Varifocal Net	1. Extensive experiments on MS COCO show that our VFNet consistently surpasses the strong baseline by ~2.0 AP with different backbones	1. It matches many SOTA object detection models like YoloV5, and EfficientDet and in some cases even outperforming them 2. VarifocalNet uses the varifocal loss to predict the IoU-aware classification score for each image. The varifocal loss is inspired by the focal loss	
18	YOLO V5.V6	Leading object detection model and open source repository.	With a focus on the new YOLOV5 P5 and P6 nano models, reducing the model size and inference speed footprint of previous models. The new micro models are small enough that they can be run on mobile and CPU.	