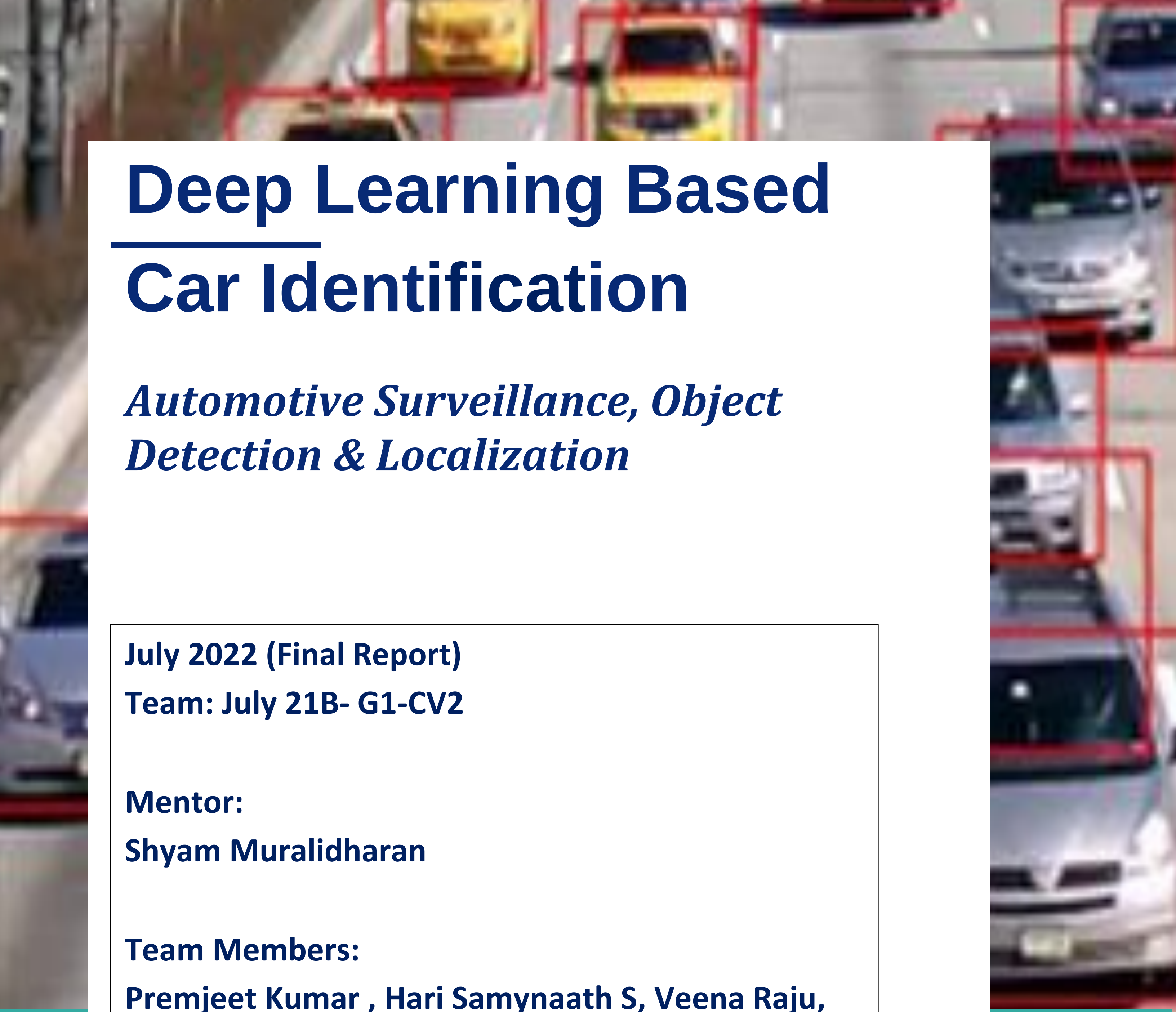


Deep Learning Based Car Identification

Automotive Surveillance, Object Detection & Localization

July 2022 (Final Report)

Team: July 21B- G1-CV2

Mentor:

Shyam Muralidharan

Team Members:

Premjeet Kumar , Hari Samynaath S, Veena Raju, Javed Bhai, Surabhi Joshi

GREAT LAKES INSTITUTE OF MANAGEMENT
Global Mindset - Indian Roots

TEXAS McCombs
The University of Texas at Austin
McCombs School of Business

Contents

1.Summary

This 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 simultaneously. The aim was to build a deployable deep learning model for car classification and prediction using Stanford Car Database. This report primarily focusses on delineating the journey from Exploratory Data Analysis, data preprocessing leading to model development, analysis and deployment with a GUI development. The work stands at the distinction of designing a model which simultaneously provides capability of both object classification and segmentation in a single model. Initially a CNN model (Mobile Net V3) for object classification was trained. Further, various different options for creating region proposal and segmentation were explored within the existing computing constrains of the team. A novel hybrid modelling configuration which provides capability of segmentation as in RCNN models and object classification through CNN was designed by coupling a new generator model developed by the team with a battery of 6 CNN models (VGG 16, VGG19, MobileNetV2, ResNet50V2, EfficientNetV2s, NasNetmobile). This development provided an advanced solution to the two-step modelling process for classification and segmentation to simultaneous detection and classification. We found that the ResNet50V2s hybrid as the best performing model. which was deployed by developing a GUI. The report finally underlines the limitations and reflections on entire project lifecycles along with key takeaways.

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 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.

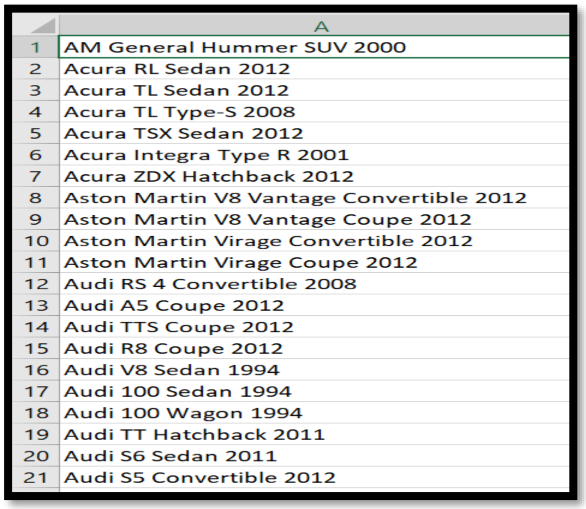
The Cars dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. Classes are typically at the level of Make, Model, Year, e.g., 2012 Tesla Model S or 2012 BMW M3 coupe.

Data description:
Train Images: 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.

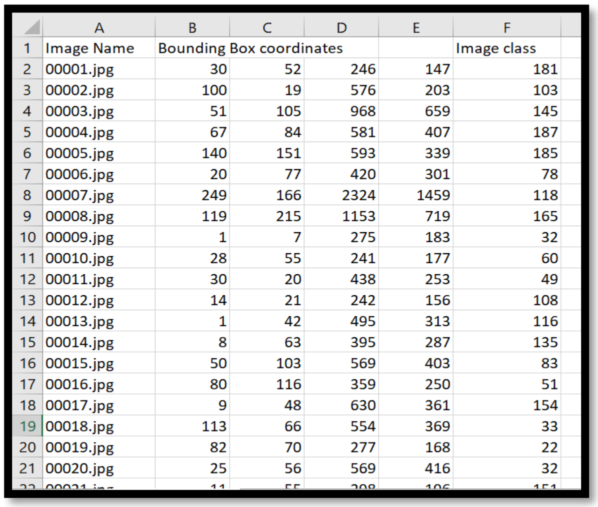
The useful data to create the model is available in three files including two zipped folders.

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 folders 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



	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

Car Make & Name **Annotations**

The car make & name folder provides names for 196 different car models for which image has been provided. The model's name is made available in a single column 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.

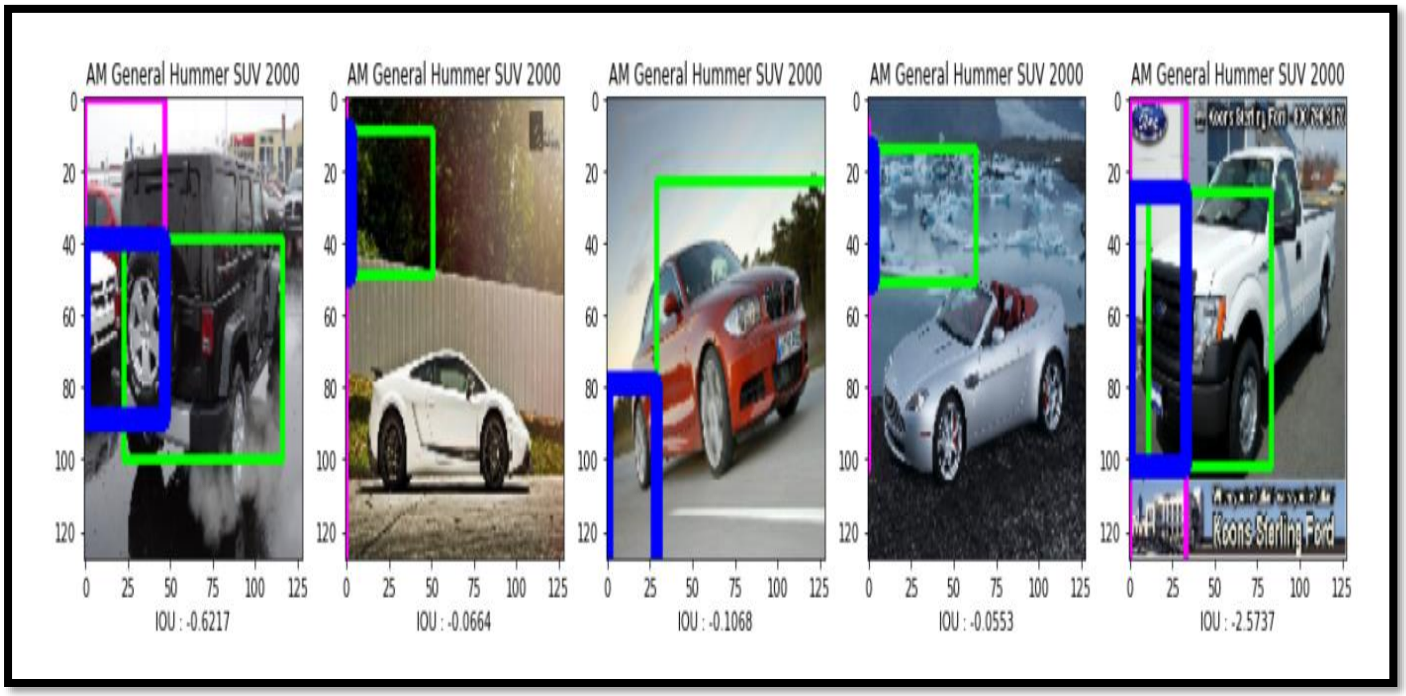
The annotations data is provided in two folders train and test. The individual image in this data is mapped with its bounding box coordinates and class of the car. It's to be noted that bounding box coordinates show a wide range of values pointing to images with various different resolutions and backgrounds.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

A novel generator model was prepared for segmentation through bounding box creation and works as alternate for region proposals implementation. This was to work around the limited computing abilities available with the team.

Findings and implications

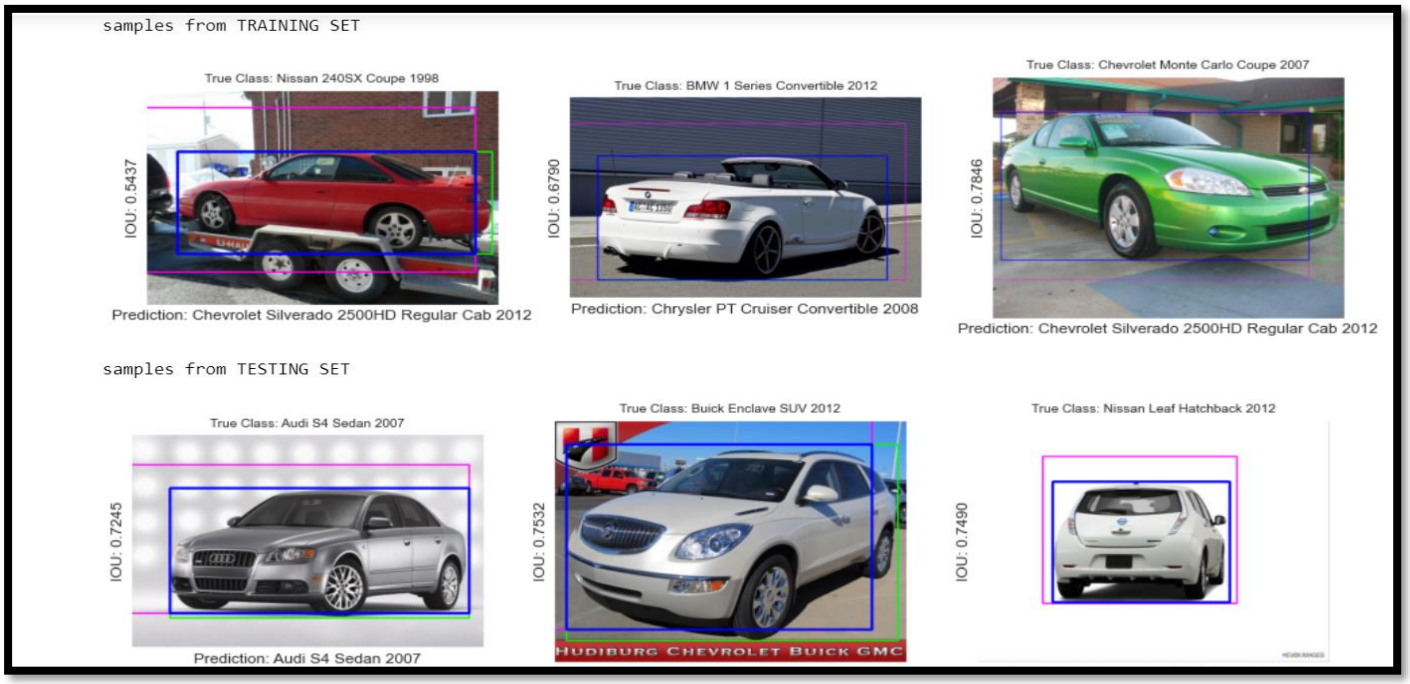
After testing through CNN models (MobileNet50) for name classification efficiency and generator coupled CNN models for object detection which simultaneously perform segmentation and classification we finalized RESNET50V2 based model hybrid for deployment to create the GUI. The final model efficiencies to object detection are much higher to the initial models presented in the interim report and provide a basis of a consistent model for car detection and classification.
Figure 1.1 and 1.2 compare the classification and detection efficiency of earlier model and the final model

Initial Models with training Predictions


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

The initial model built by the team had very low accuracy and lacked consistency. The team worked with finetuning and changing various different parameter along with development of a generator pipe for efficient segmentation of images along with creation of bounding boxes.

Final Model with predictions of both training and Validation set



```
samples from TRAINING SET
```

True Class: Nissan 240SX Coupe 1998
IOU: 0.5437
Prediction: Chevrolet Silverado 2500HD Regular Cab 2012
True Class: BMW 1 Series Convertible 2012
IOU: 0.6790
Prediction: Chrysler PT Cruiser Convertible 2008
True Class: Chevrolet Monte Carlo Coupe 2007
IOU: 0.7846
Prediction: Chevrolet Silverado 2500HD Regular Cab 2012

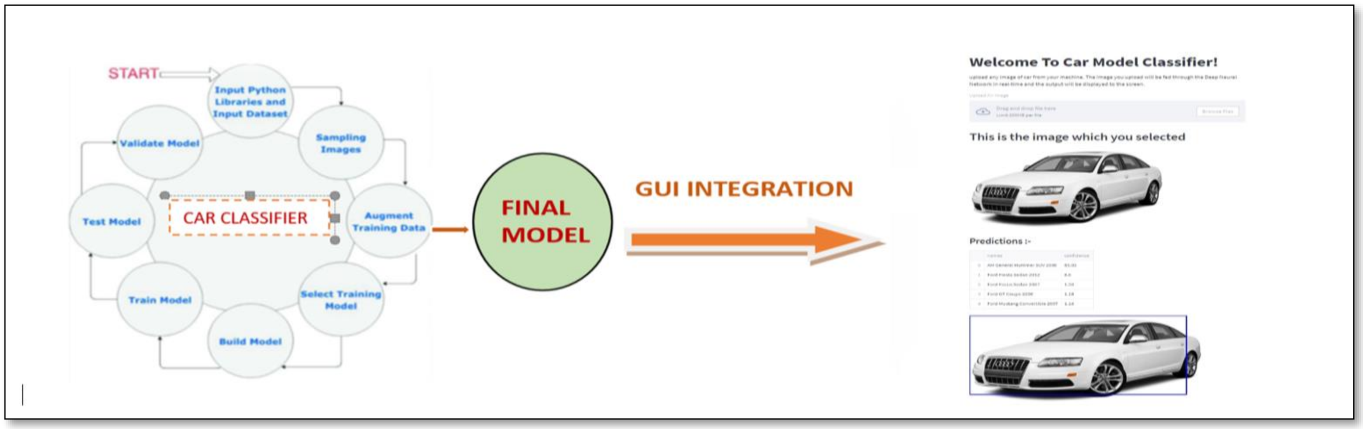
```
samples from TESTING SET
```

True Class: Audi S4 Sedan 2007
IOU: 0.7245
Prediction: Audi S4 Sedan 2007
True Class: Buick Enclave SUV 2012
IOU: 0.7532
HUDIBURG CHEVROLET BUICK GMC
True Class: Nissan Leaf Hatchback 2012
IOU: 0.7490

2.Overview of the final process

Problem Approach

The key steps for project implementation include 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.

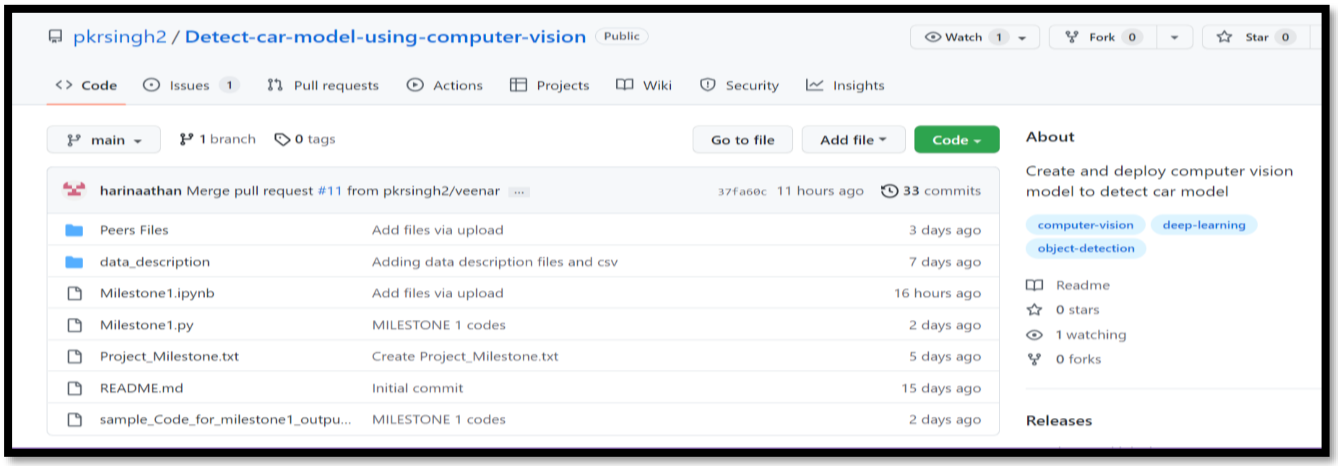


Prediction model for car classification will be deployed using publicly hosted application based on Streamlet by creating a GUI. This application will enable users to interact with trained model. User can select any of the images used for training, Validation and testing and see how the model would classify 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. Sham Muralitharan, 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

Watch 1 · Fork 0 · Star 0

Code · Issues 1 · Pull requests · Actions · Projects · Wiki · Security · Insights

main · 1 branch · 0 tags · Go to file · Add file · Code

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

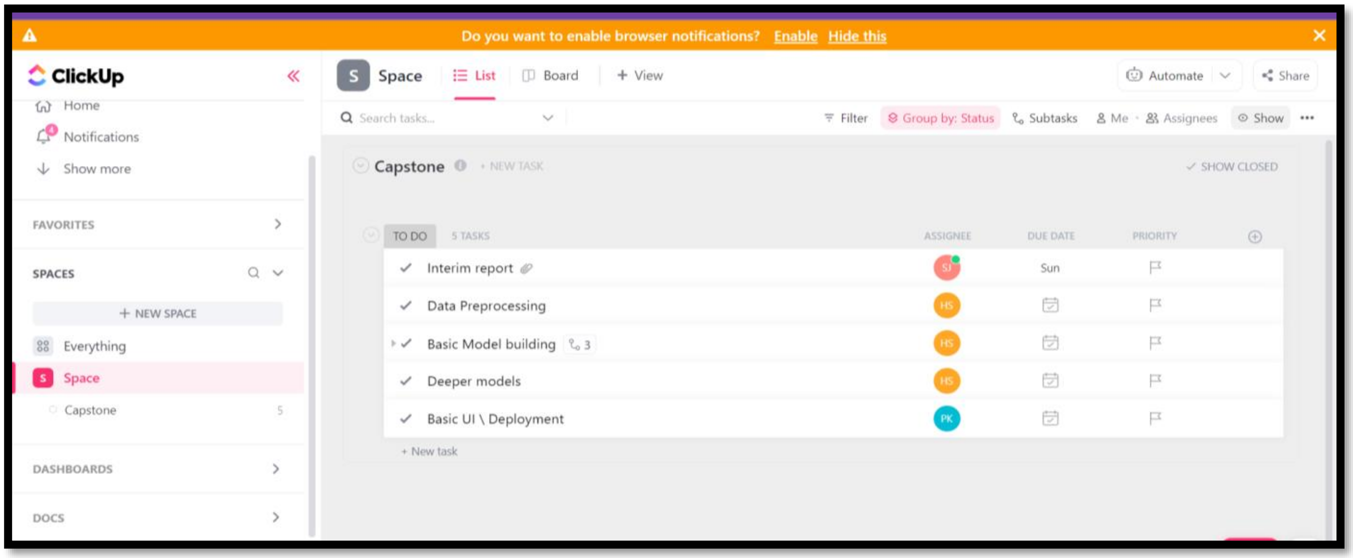
Create and deploy computer vision model to detect car model

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

ClickUp · Home · Notifications · Show more · FAVORITES · SPACES · + NEW SPACE · Everything · Space · Capstone 5 · DASHBOARDS · DOCS

Space · List · Board · + View · Automate · Share

Search tasks... · Filter · Group by: Status · Subtasks · Me · Assignees · Show

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

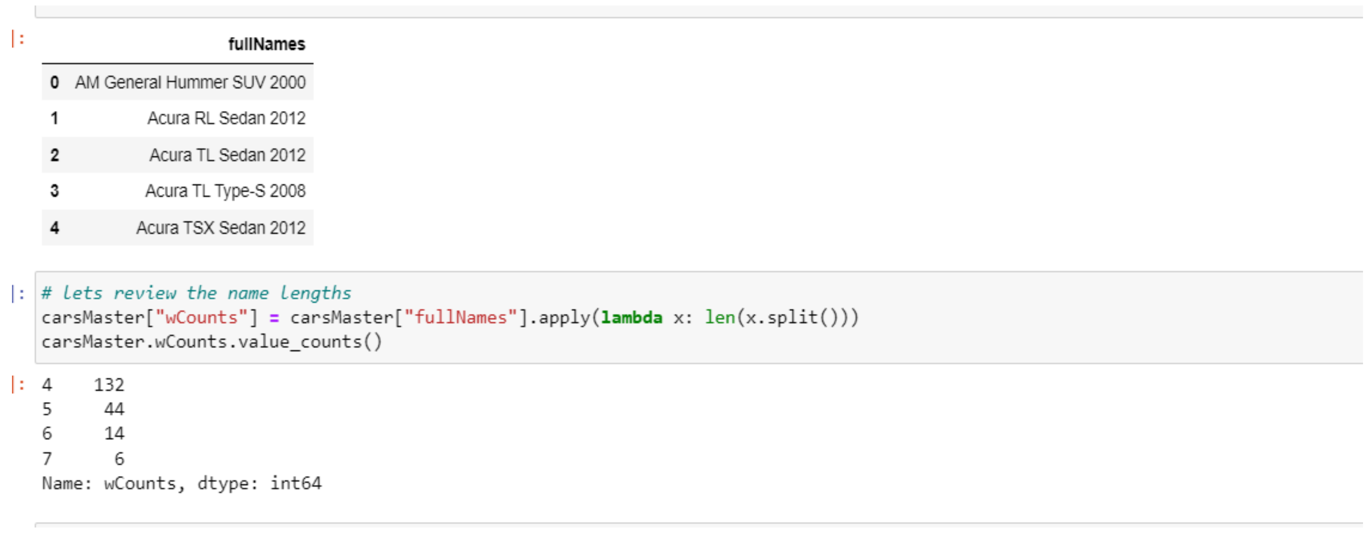
Data Extraction and Exploration

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

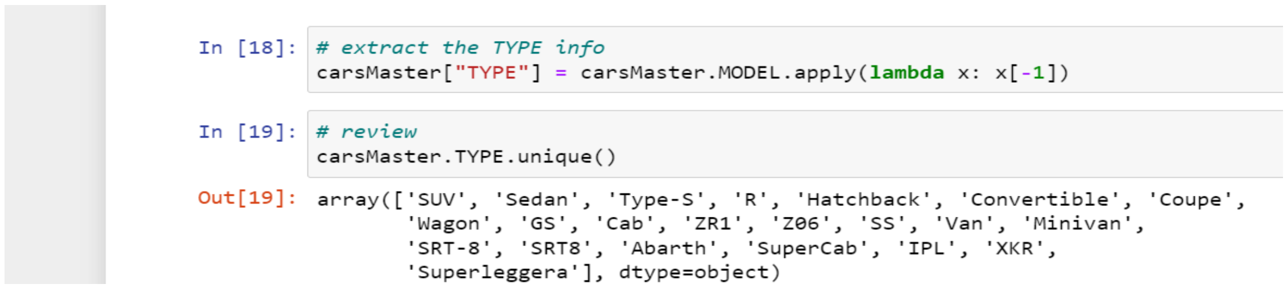
```
# lets review the name lengths
carsMaster["wCounts"] = carsMaster["fullNames"].apply(lambda x: len(x.split()))
carsMaster.wCounts.value_counts()
```

```
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

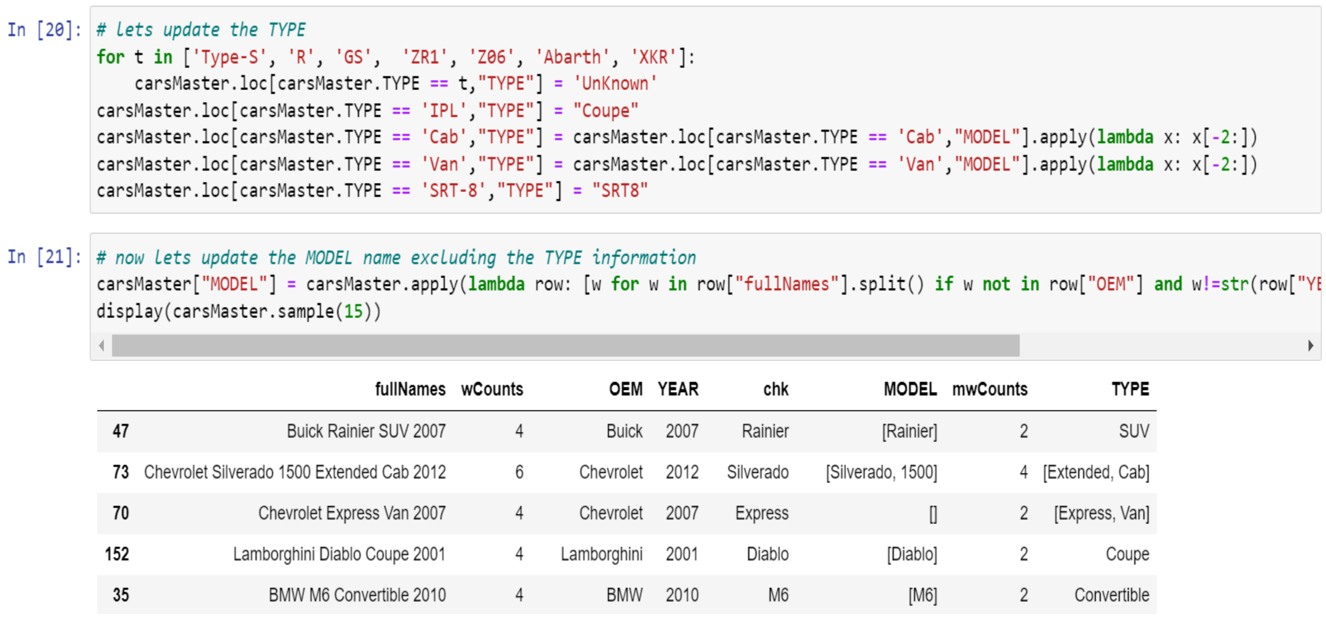


```
In [18]: # extract the TYPE info
         carsMaster["TYPE"] = carsMaster.MODEL.apply(lambda x: x[-1])

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

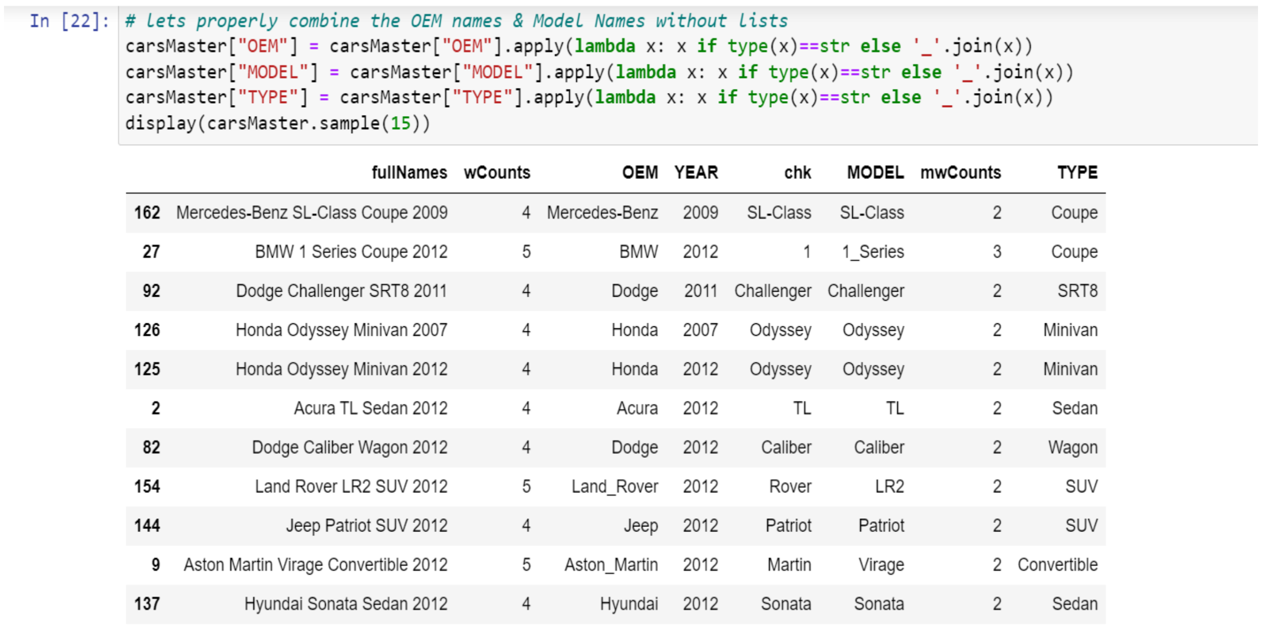
We manually define the types to get the data consistency



```
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

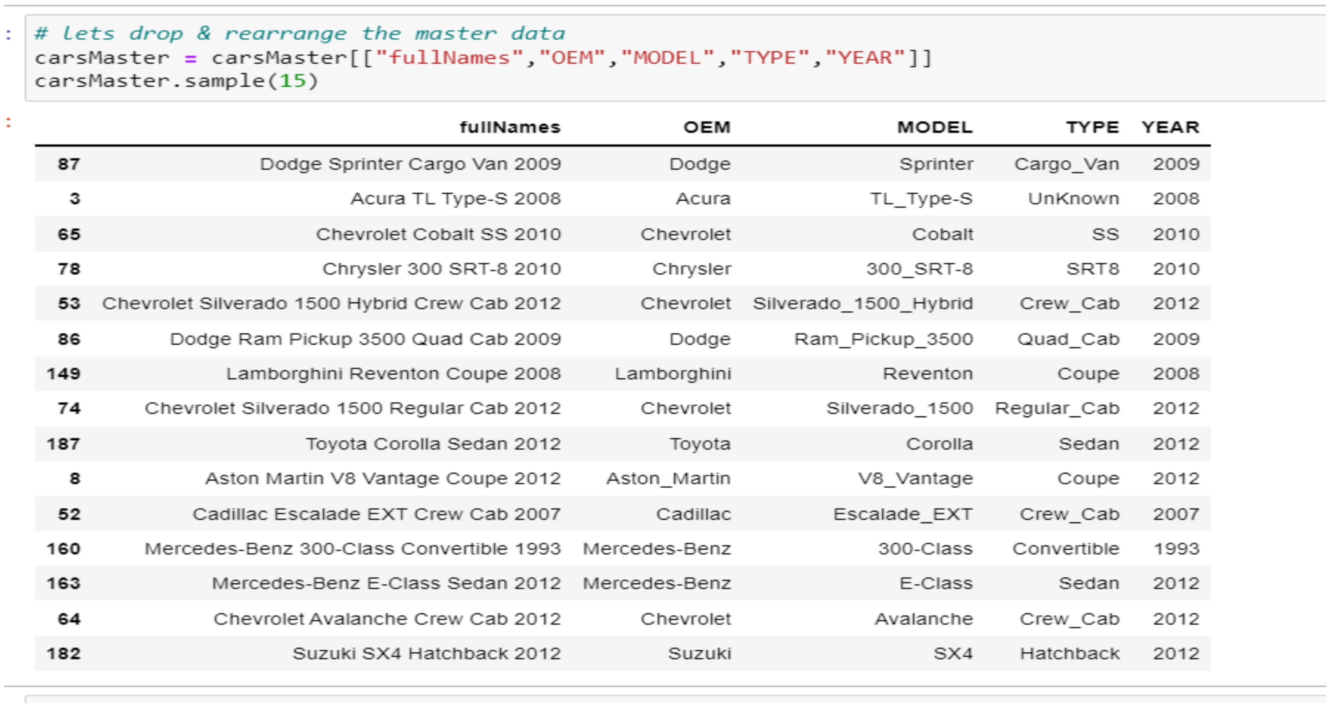
Now the OEM names and Model names are combined without list



```
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

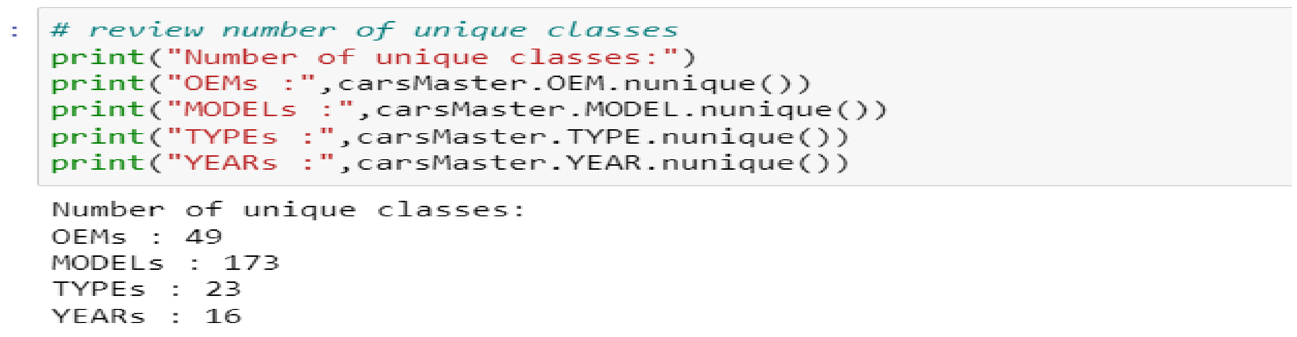


```
: # lets drop & rearrange the master data
  carsMaster = carsMaster[["fullNames","OEM","MODEL","TYPE","YEAR"]]
  carsMaster.sample(15)
```

	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

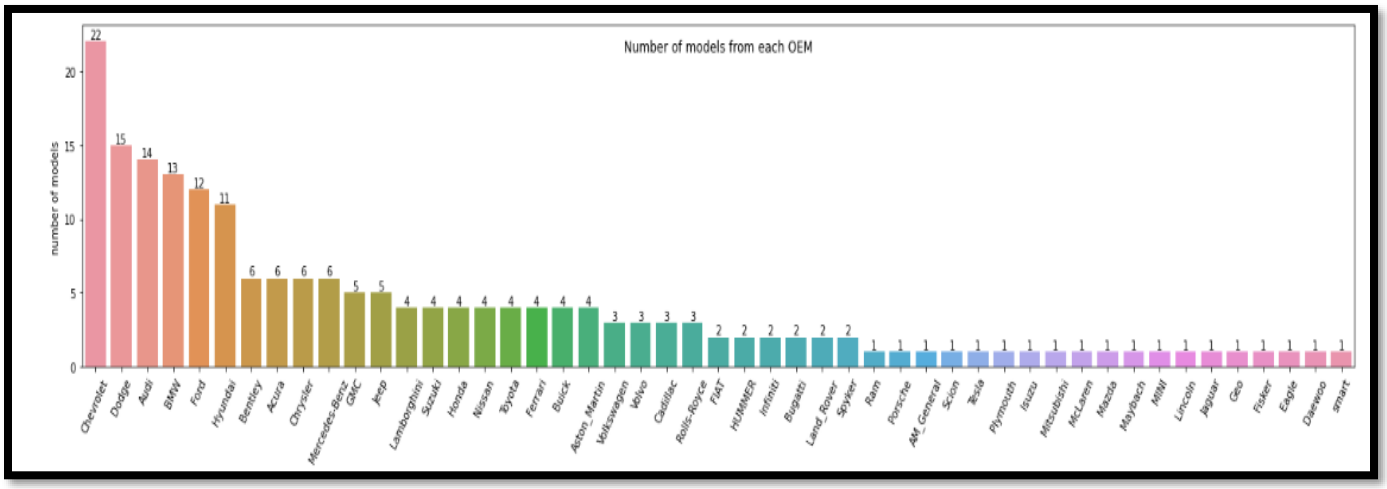


```
: # review number of unique classes
  print("Number of unique classes:")
  print("OEMs :",carsMaster.OEM.nunique())
  print("MODELs :",carsMaster.MODEL.nunique())
  print("TYPEs :",carsMaster.TYPE.nunique())
  print("YEARs :",carsMaster.YEAR.nunique())

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

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

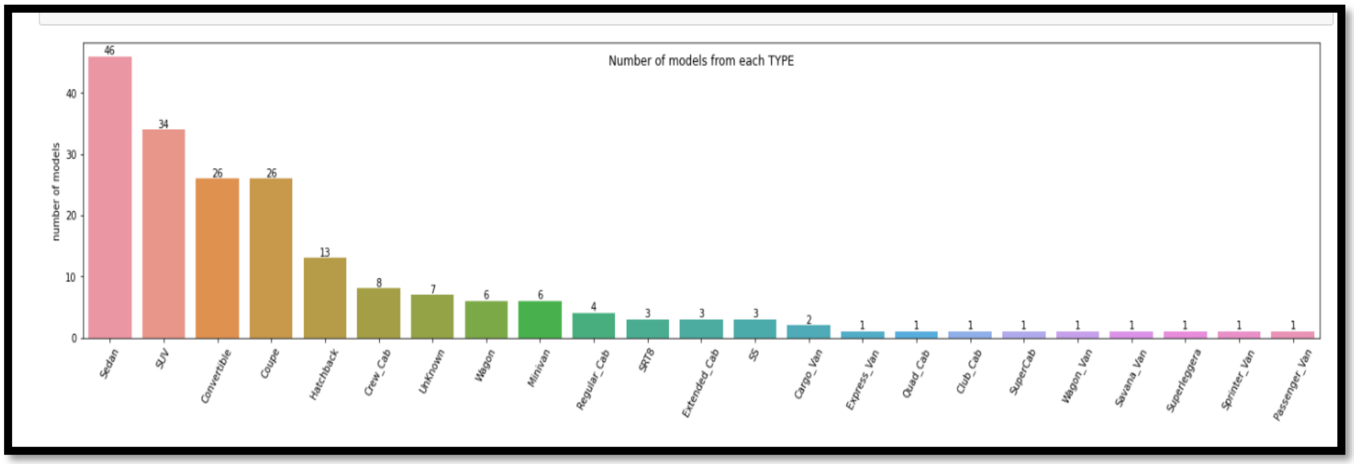
We now explore the frequency distribution and data distribution of various categories of cars WRT to OEM’s, Type of cars, and year of manufacturing and plot the frequencies

Number of Models for each Car Manufacturer (OEM)



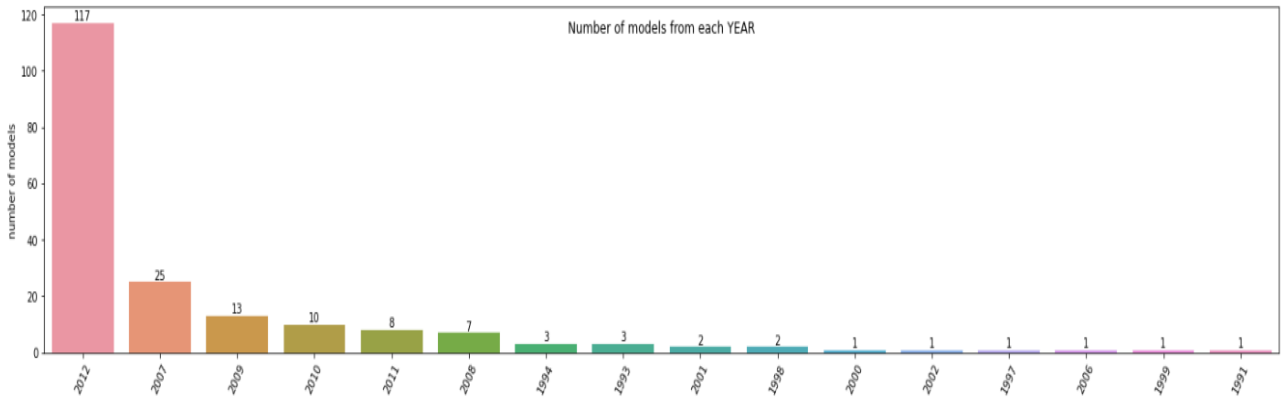
The data consists of 22 models for Chevrolet followed by Dodge (15), Audi (14), BMW (12), Ford (12) and Hyundai (11) which have more than 10 models in the data thus higher representation that significantly higher RAM, MADA etc. which have only one model in the data

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

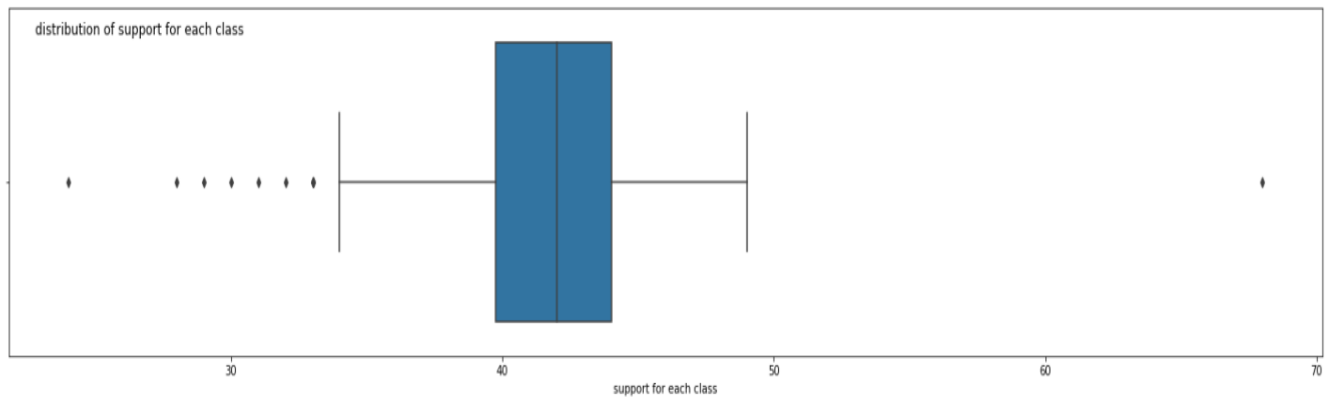


From the perspective of data distribution highest data frequency for make year in 117, which emerges as a predominant bias in the existing dataset. From the above three parameters we get an insight that there is a high probability of intrinsic biases in terms of predominant OEM, type and year of manufacturing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

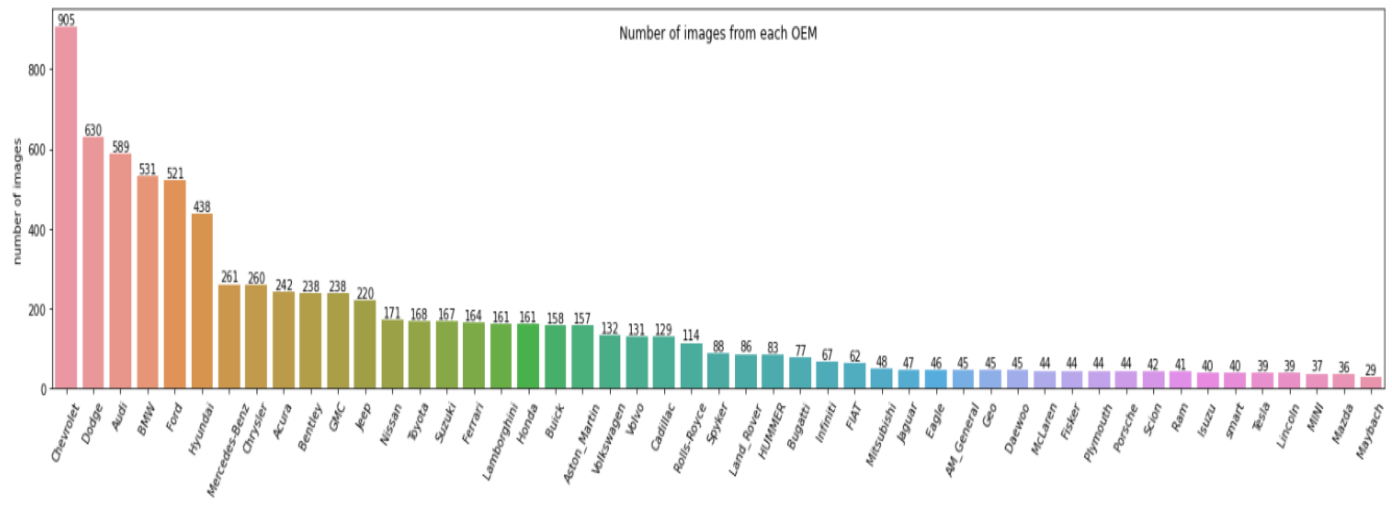
We further explore this finding by going deeper into understanding the data distribution in terms of available images for each category 196 classes of cars in the data set. We do analysis in two steps.

Data Distribution in terms of individual image counts
Step one we try to evaluate a median count for images for training for each class as that will be crucial in determining the training biases for the model

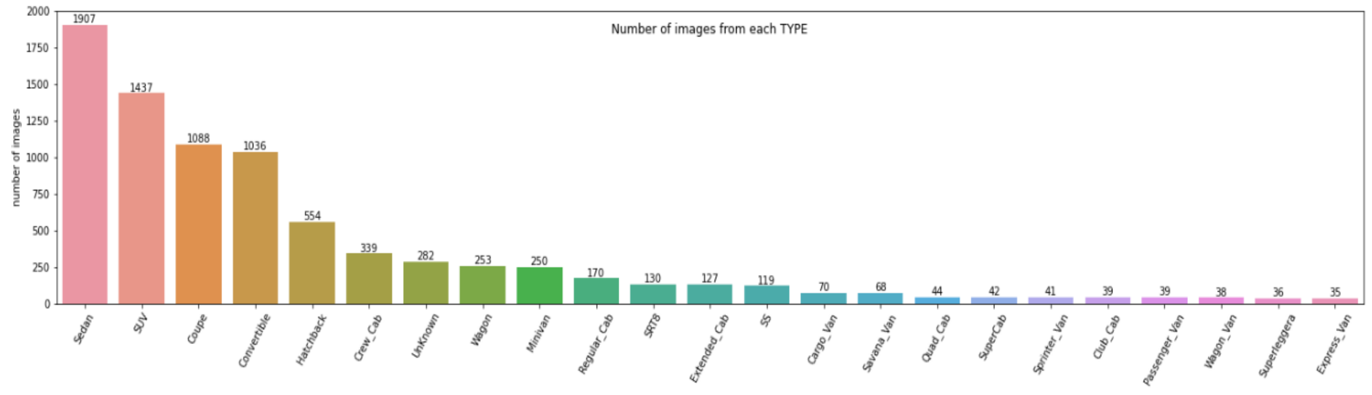
We determine the distribution support of each class and find that for most classes the total images available lie between 40-50 instances for training



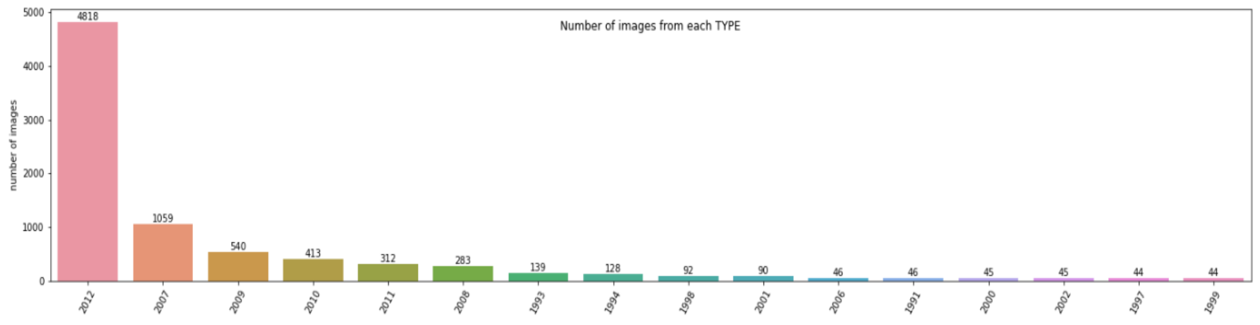
Number of images in each OEM



Number of images in each OEM



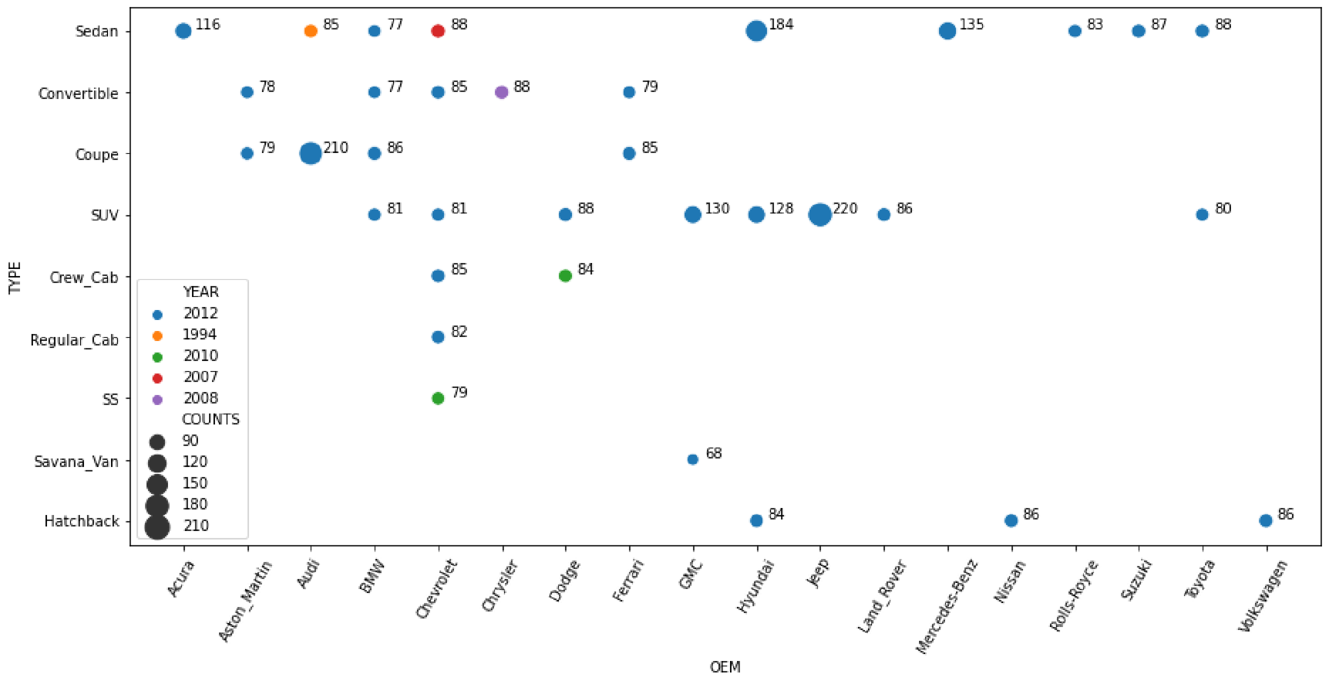
Number of images in each make year



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

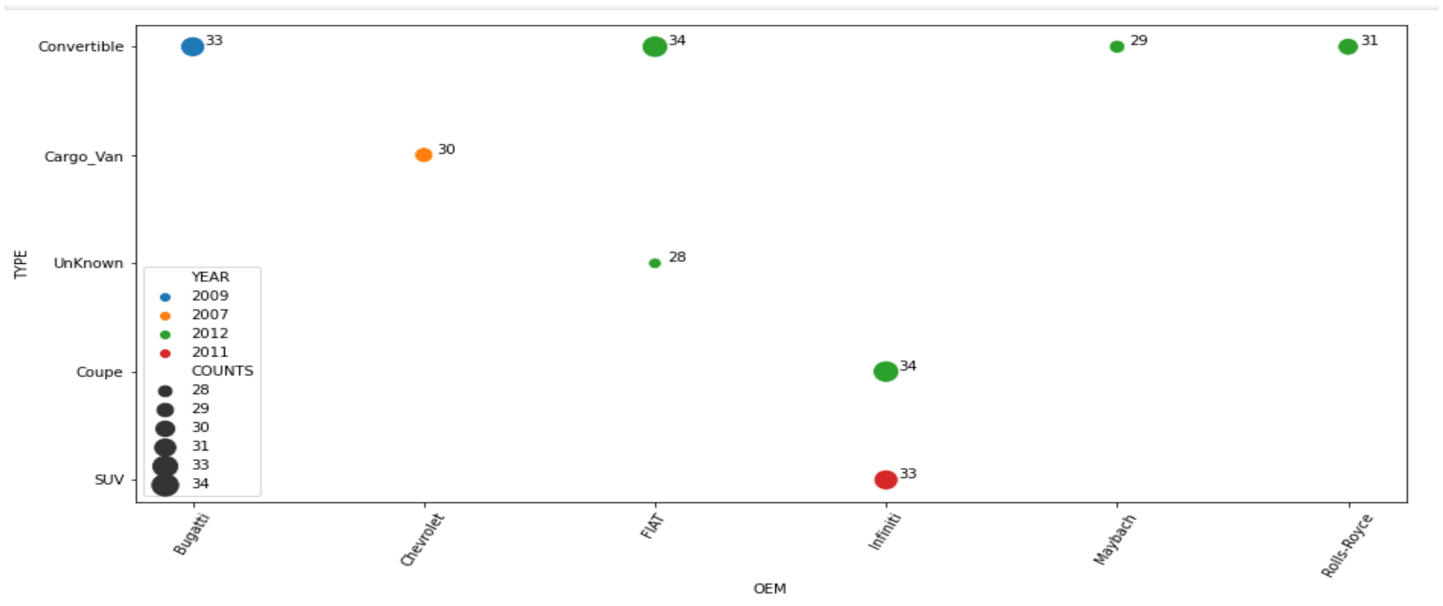
We further evaluate combined frequencies for various models as the inputs would be going in terms of 196 training classes (multiclass training)

Combined distribution of images writ 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 highlights 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

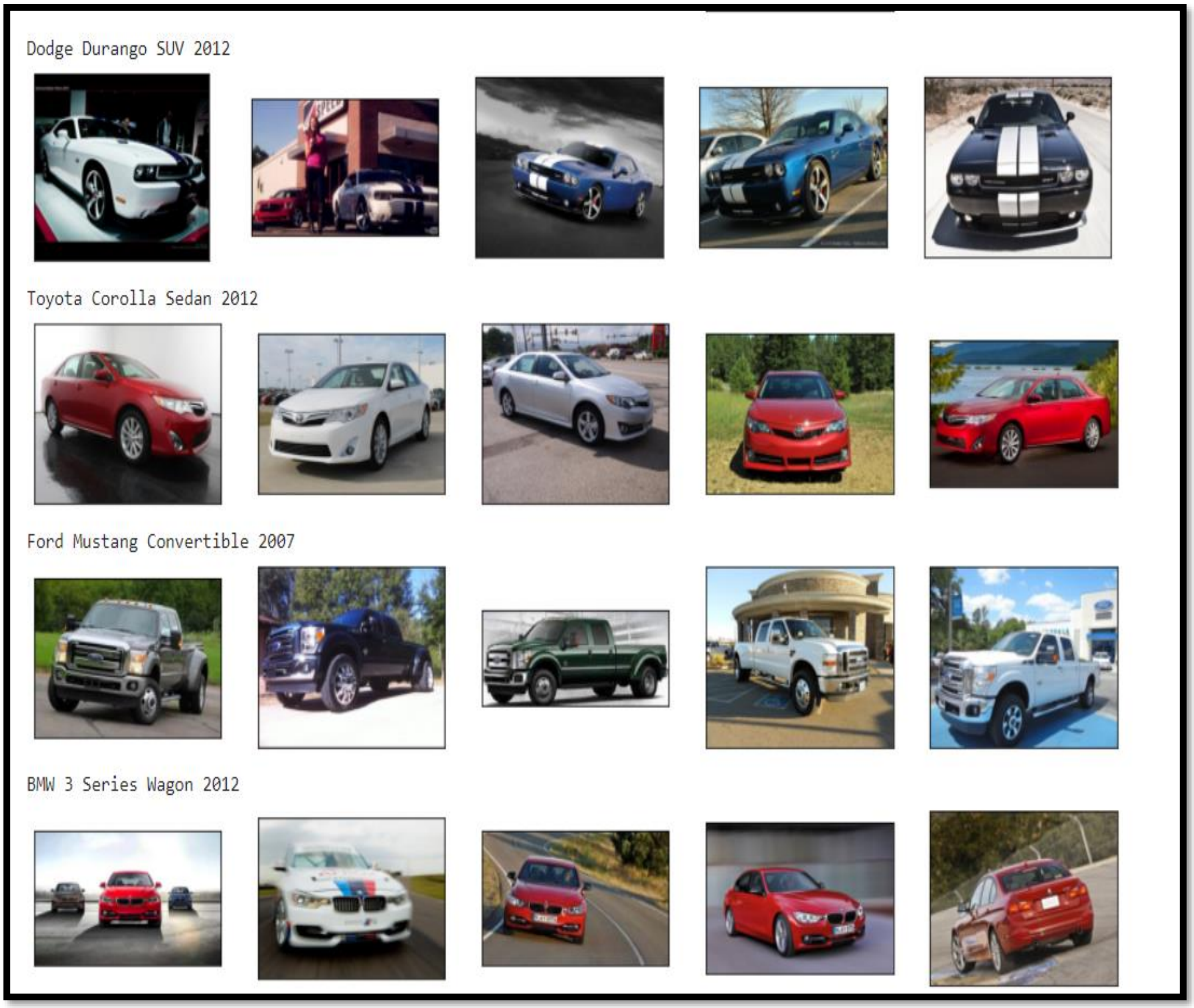
The acquired data from the real world is usually messy and come from different sources. To feed them to the machine learning or neural network-based model, they need to be standardized and cleaned up.

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

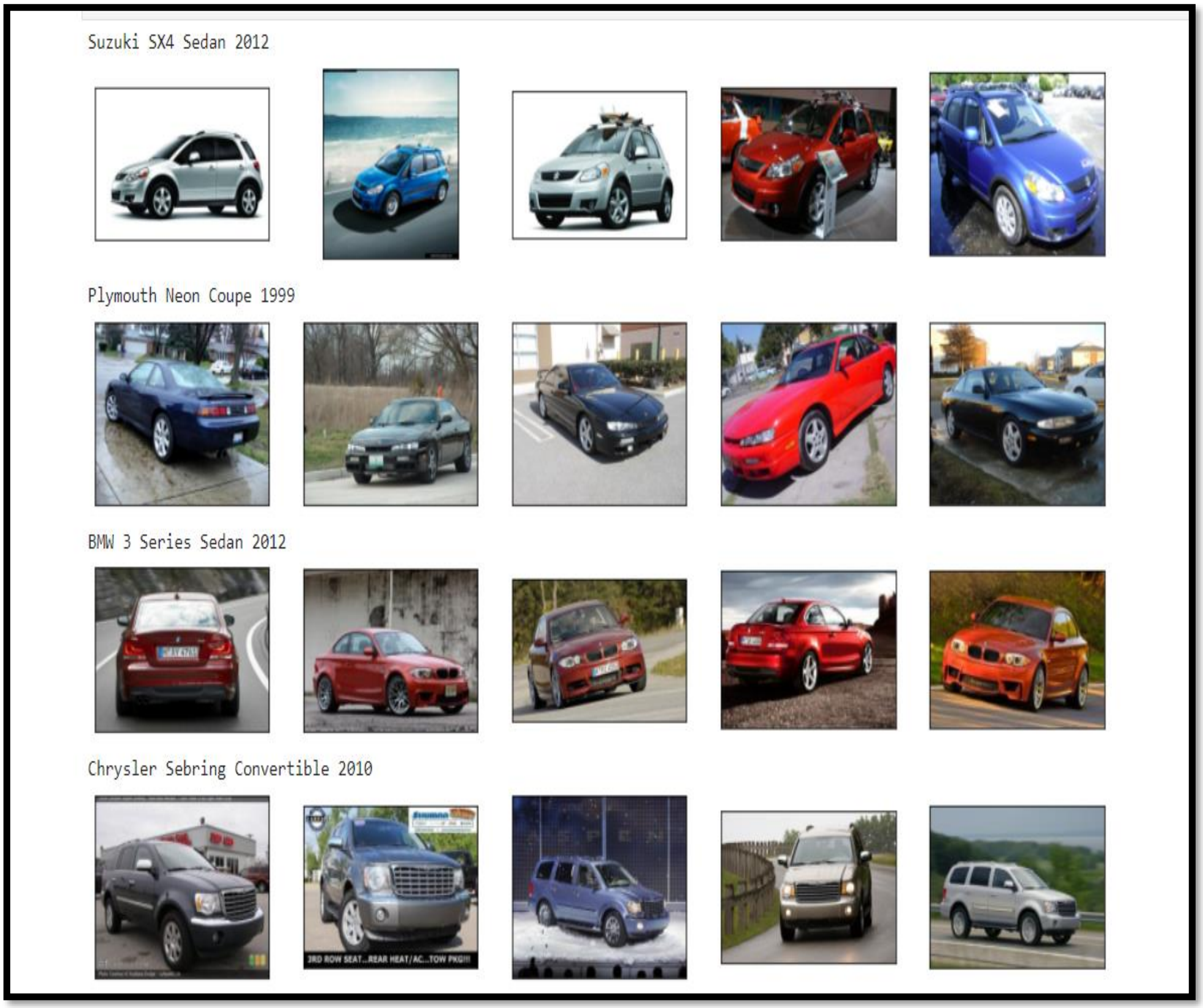
in which an image is taken would be very cumbersome and resource constraining thus, when an image is acquired, its first converted into a form that allows a general algorithm to solve it.

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

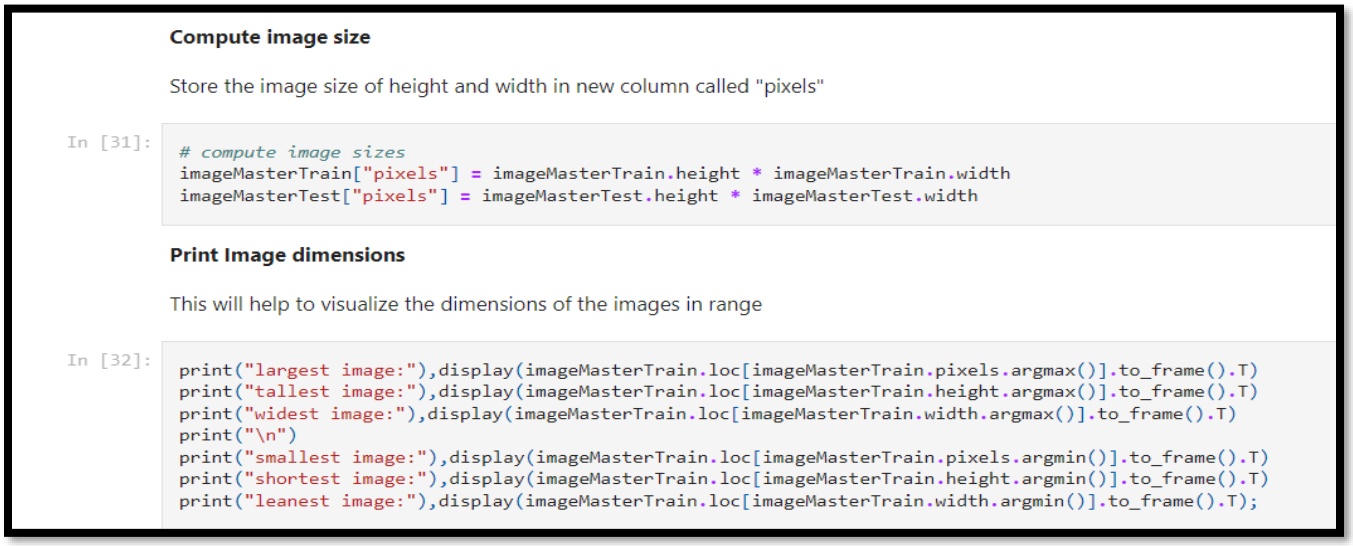


The image data preprocessing included

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

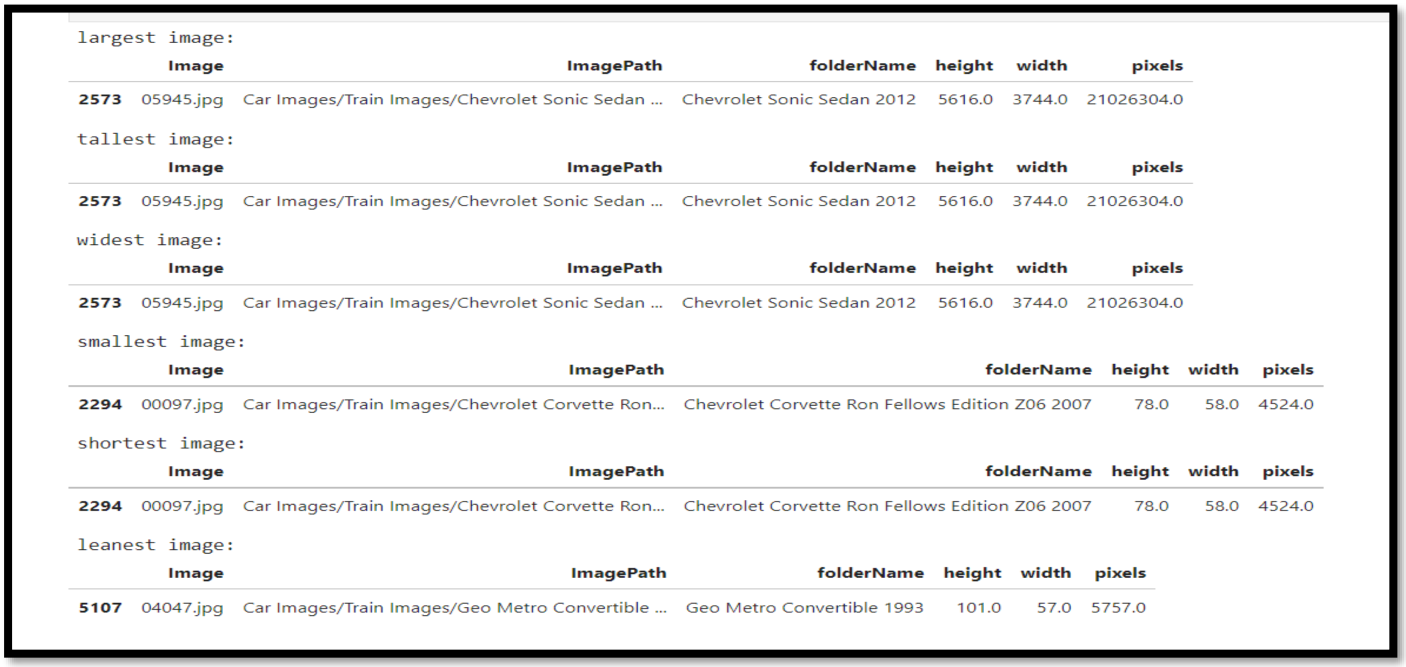
Store the image size of height and width in new column called "pixels"

```
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 varies 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.

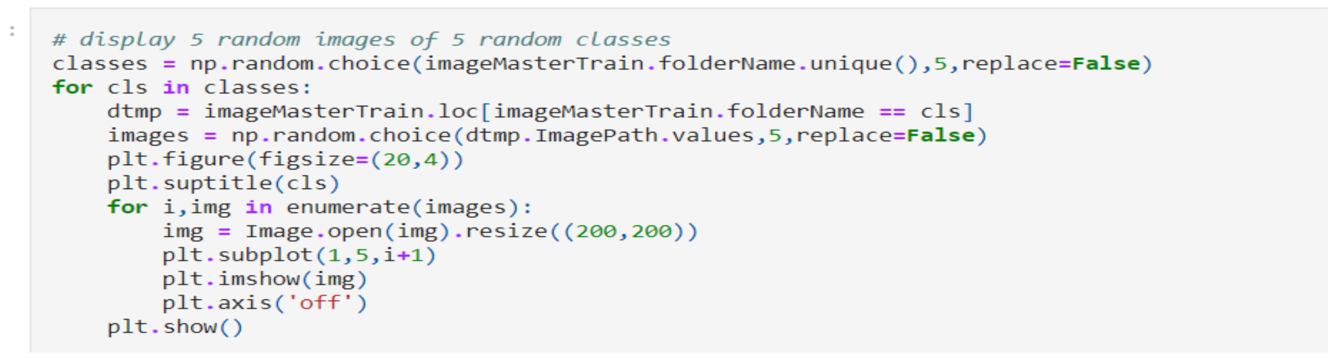
Some of the Best Practices

1. To decide on what should be the size of the images, a good strategy is to employ progressive resizing. egg; 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.

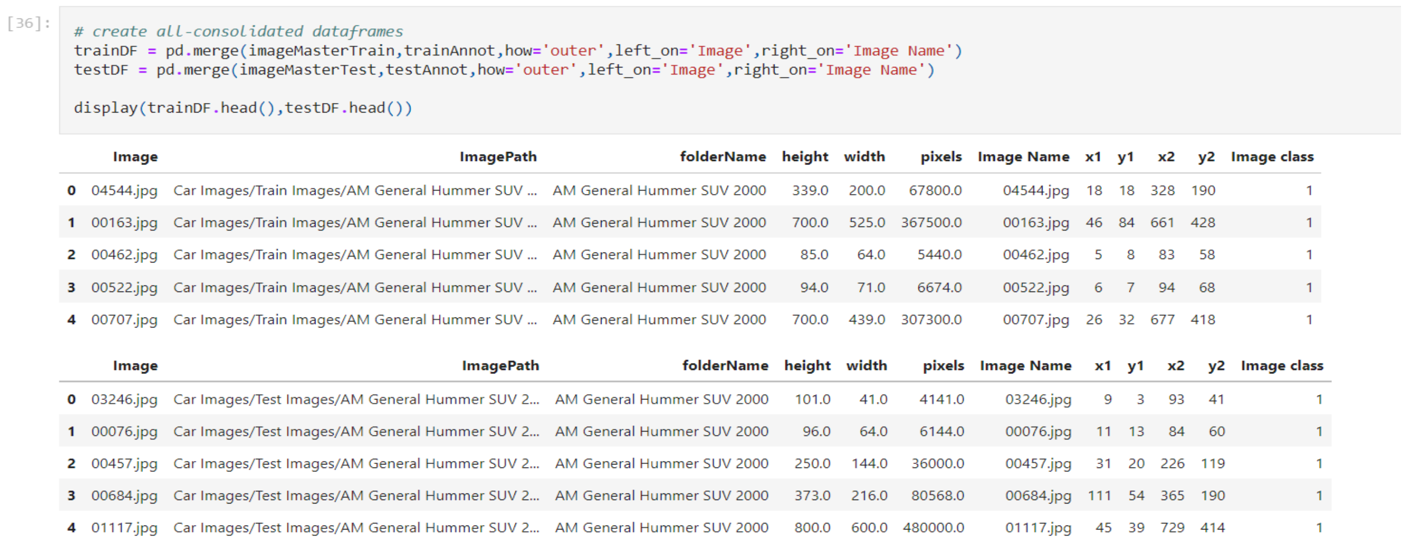


```
# display 5 random images of 5 random classes
classes = np.random.choice(imageMasterTrain.folderName.unique(),5,replace=False)
for cls in classes:
    dtmp = imageMasterTrain.loc[imageMasterTrain.folderName == cls]
    images = np.random.choice(dtmp.ImagePath.values,5,replace=False)
    plt.figure(figsize=(20,4))
    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')
```

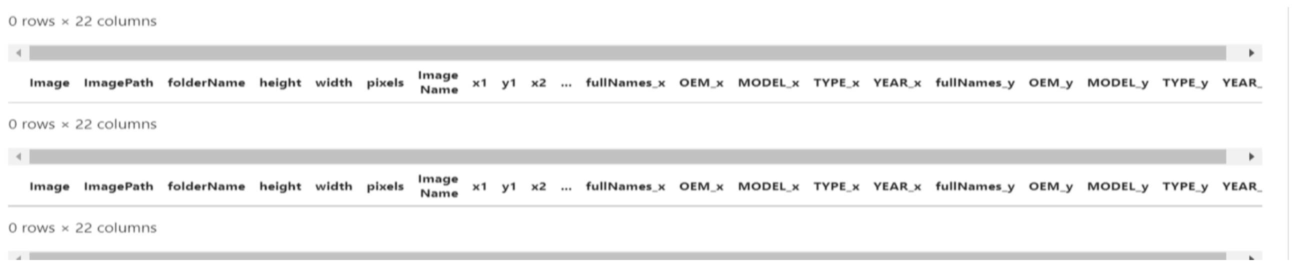
```
[38]:
# update class index to start from ZERO
trainDF["Image class"] = trainDF["Image class"]-1
testDF["Image class"] = testDF["Image class"]-1
```

```
[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

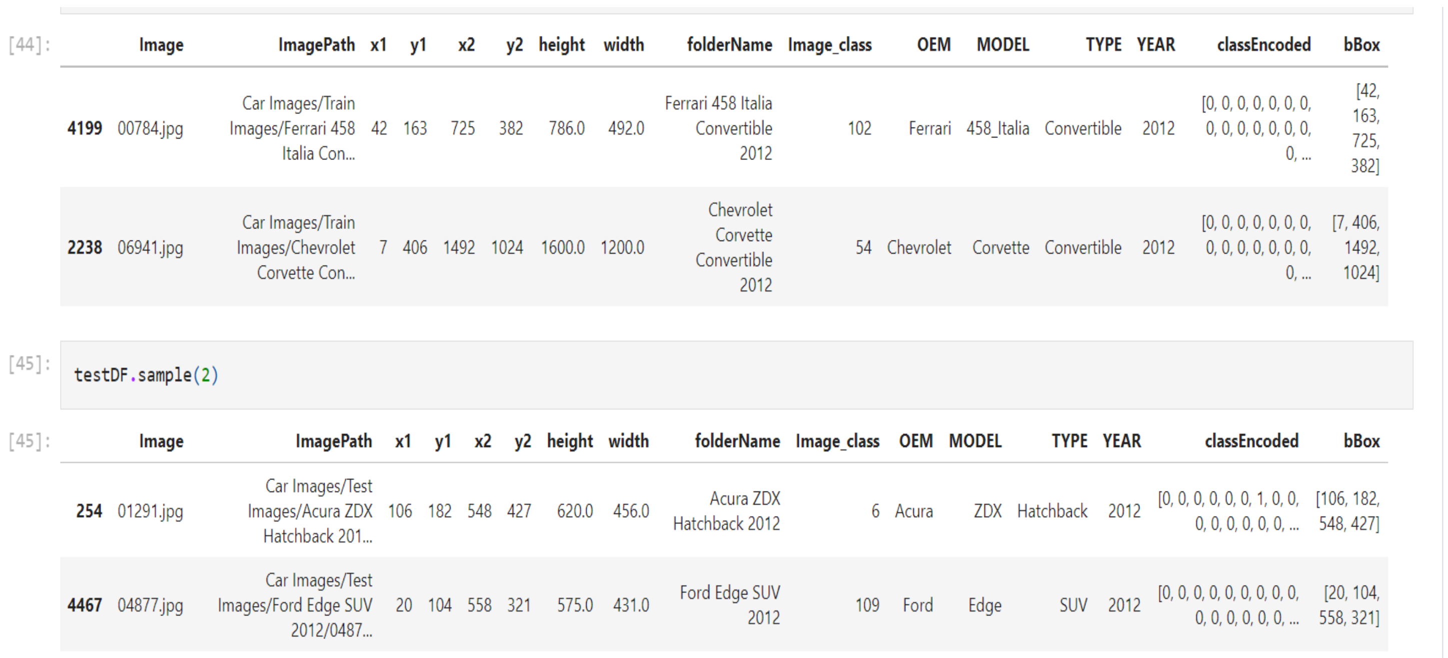
After doing the cross merged and synced with "Train/Test Annotations.csv", "Car names and make.csv" and the images in the "Train/Test images folders", it is found to have no mismatch of information

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]:

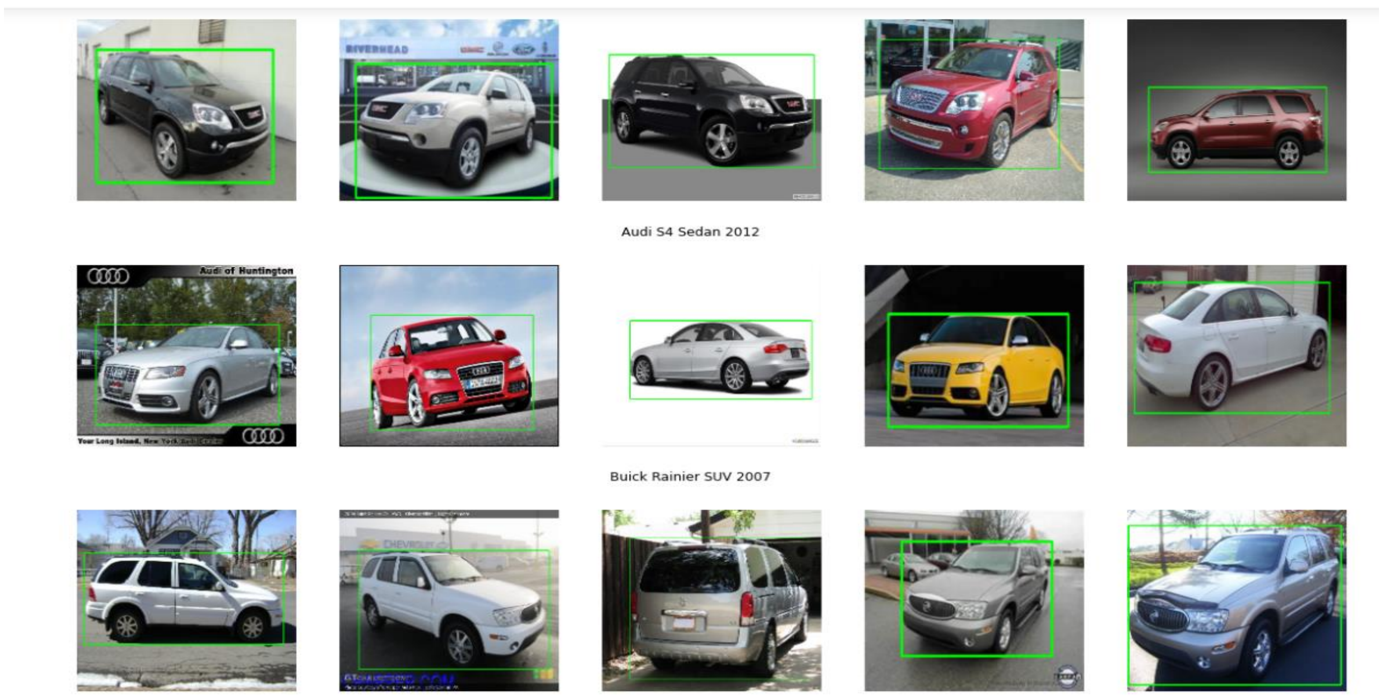
	Image	ImagePath	x1	y1	x2	y2	height	width	folderName	Image_class	OEM	MODEL	TYPE	YEAR	classEncoded	bBox
4199	00784.jpg	Car Images/Train Images/Ferrari 458 Italia Con...	42	163	725	382	786.0	492.0	Ferrari 458 Italia Convertible 2012	102	Ferrari	458_Italia	Convertible	2012	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,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

The aim is to build a deployable deep learning model for car classification and prediction using Stanford Car Database The steps to model development involved and initial preparation for creating processing for seamlessly testing multiple models in order to select the best model followed by hyper tunning and deployment with a GUI. The following steps were followed

1. Design and implementation of pipeline for preprocessing and model runs
2. Design and development of a generator
3. Design and Creation of custom call out
4. Model evaluation and selection
5. Model pickling
6. Design and development of GUI
7. Integration of pickled model with GUI
8. Refining the GUI and hosting on dedicated server

Creation of Pipeline

A machine learning pipeline helps automate machine learning workflows by processing and integrating data sets into a model, which can then be evaluated and delivered. A well-built pipeline helps in the flexibility of the model implementation. A pipeline in machine learning is a technical infrastructure that allows an organization to organize and automate machine learning operations.

There are various stages in a machine learning pipeline architecture, mainly- Data preprocessing, Model training, Model evaluation, and Model deployment. Each stage of the data pipeline passes processed data to the next step, i.e., it gives the output of one phase as input data into the next phase.

- **Data Preprocessing**- This step entails collecting raw and inconsistent data selected by a team of experts. The pipeline processes the raw data into an understandable format. Data processing techniques include feature extraction, feature selection, dimensionality reduction, sampling, etc. The final sample used for training and testing the model is the output of data preprocessing.
- **Model Training**- Selecting an appropriate machine learning algorithm for model training is crucial in a machine learning pipeline architecture. A mathematical algorithm specifies how a model will detect patterns in data.

- **Model Evaluation-** The sample models are trained and tested on historical data to make predictions and choose the best-performing model for the next step.
- **Model Deployment-** The final step is to deploy the machine learning model to the production line. Ultimately, the end-user can obtain predictions based on real-time data.

A pipeline for data preprocessing was constructed as goal one and has been detailed in the interim report and chapter 2 of this report. A separate pipeline for for ETL-to-Learning-to-Evaluation-&-Reporting was designed and implemented.

The pipeline include

1) **def IOU** which computes intersection over union and includes following arguments

   i. true: true bounding box coordinates as list

   ii. yapped: predicted bounding box coordinates as list

   iii. expected shape of arguments: (None,4)

   iv. returns: bounding box metric, Intersection-over-Union as list of shape (None,)

2) **definite** for initializing the model and consists of following arguments

   i. definite for: duct of wars to multigene (), positional arguments excluded

   ii. Madelung: method that will return an assembled model

   iii. model Name: representative name for the purpose of record & comparison

   iv. modulars: duct of arguments to Madelung, except input size

   v. weights: path of pretrained weights file

3) **def compiler " compile the model & other callbacks**" includes following arguments

   i. primiparas: duct of arguments to RMSprop, except learning rate

   ii. primiparas: duct of arguments to duct scheduler, except duct

   iii. stoppers: duct of arguments to stop callback instance

iv. stoppers: duct of arguments to layer Unfreeze callback

v. grid Point: duct of other hyperparameters

vi. best: option to save best weights

4) **def fit** defines the learner method and include the following arguments

I) grid Point: duct of hyperparameters of the model, must include learning rate

ii) diagrams: duct of arguments to model. Fit (), except dataset selection

5) **def visualize,** displays the model training history metrics

6) def evaluate(self), predict on train & test data using Prediger

7) def save(self) pickle each critical components to create a deployable model

8) **Evaluation & logging** includes results on the following 15 parameters

Model Name	**loss**	**names loss**	**boxes loss**
names accuracy	**names precision**	**names recall**	**boxes**
valyls	**valyls**	**valyls'**	**valyls**
valyls	**valyls**	**valyls**	

9) def report includes following arguments

i. name: model names as string
ii. train Images: list of training image paths
iii. outset: outputs for training dataset [outset]
iv. test Images: list of testing image paths
v. outset: outputs for testing dataset [outset]
vi. enc: fitted Label Binarizer() object
vii. note: all box coordinates to be rescaled to orignal aspect ratios & sizes
viii. returns: None

functionality:

i. updates resLog
ii. displays classification report & confusion stats
iii. displays sample results for training & testing datasets

10) def interBox (y_true,y_pred): includes the following arguments

i. y_true : true bounding box coordinates as list
ii. y_pred : predicted bounding box coordinates as list
iii. expected shape of arguments: (None,4)
iv. returns: coordinates of intersection of true & predicted bounding boxes (None,4)

11) def evalLog includes following arguments

i. mLog : array of current model metrics
ii. trueClass : binarized true labels (None,196)
iii. predClass : predicted logits (None,196)
iv. trueBox : true bounding box coordinates (None,4)
v. predBox : predicted bounding box coordinates (None,4)
vi. returns : updated array of current model metrics

12) def confPlot1 includes following arguments

i. labels : list of classes
ii. confTrain : multi-label confusion matrix for prediction on training set
iii. confTest : multi-label confusion matrix for prediction on test set
iv. functionality: displays subplots of confusion metrics

13) def confPlot2 includes the following arguments

i. labels : list of classes
ii. confTrain : multi-label confusion matrix for prediction on training set
iii. confTest : multi-label confusion matrix for prediction on test set
iv. functionality: displays overlayed confusion metrics in log scale

14) def sampleResult display 3 sample images with true & predicted targets

Includes arguments ----------

i. imPath : paths of images
ii. trueBox : true coordinates of bounding box
iii. predBox : predicted coordinates of bounding box
iv. trueClass : list of true class names (strings)
v. predClass : list of predicted class names (strings)

Creation of Generator

A typical tensor flow-based Image Data Generator does not suit the purpose for this project as multiple output (196 car classes) need to be mapped further custom augmentation and transformation of bounding boxes cannot be efficiently performed using commonly available tensor flow-based Image Data Generator

A Generator Model was thus created for catering to this multi-output models. The generator creates a pipeline and provide the following competencies:

1. def __init__ initializes bounding box creation and has following arguments

i. train : dataframe of training dataset
ii. test : dataframe of training dataset
iii. (note:both train & test dataframes to be of same structure)
iv. imagePath : column name contaning path of imagesclassName : column name containing true class names
v. bBox : list of columns containing bounding box coordinates [x1,y1,x2,y2]
vi. input_size : list of columns containing width & height of images
vii. target_size : expected size of image output, typically the input size of network that consumes the image
viii. batch_size : number of images to process for each batch of output
ix. normalize : boolean to opt for normalize or not
x. validation_split : ratio of validation vs training split
xi. seed : random consistency seed
xii. initial_epoch : used for retraining a model, to resume from an epoch

2.def __noSplit is a private helper function

3.def_unCode is a private helper function which predicts the predicted names and includes following arguments

i. names : array of predicted class names as predict_probable output and returns
ii. names : decoded names in strings

4.def __evalPrep is a private helper function

5.def __split is a private helper function

6. def_subset creates deep copy of the multiGen instance, orienting the generator towards requested subset and consists of following arguments

i. subset_name : one of 'training','validation','testing','evaluation on trainset', 'evaluation on testset'
ii. returns : a deepcopy of the current instance, prepared for generation

7. def on_epoch_end(self) override Sequence class method

8. def unCode decode the predicted names and include the following arguments

i. names : array of predicted class names as predict_proba output and returns
ii. names : decoded names in strings

9. **def on_epoch_end** overrides Sequence class method, shuffles dataset after each epoch, ensures consistent shuffling using randomizer seed

10. def __len__(self) override Sequence class method returns number of steps per

11. def __getitem__(self,step) generate a batch of data includes argument

i. step : current step/batch number and returns
ii. independant variables X & true target variables

12.def next(self):generate next batch and returns independant variables X & true target variables Y

13.def fetchBatch prepares a batch of images, names and bounding boxes arguments:

i. index : index of datapoints to feed to the current batch and returns

ii. X,Y as (array of images) and tupple of (names and bounding box coordinates)

14 .def __display : display 3 images in a subplot includes arguments

i. images : array of images as array (None,:,:,3)
ii. boxes : array of bounding box coordinates (None,4)
iii. names : array of class names (None,4)
iv. displays a subplot of images with bounding boxes augmented
v. x & y
vi. scales will be displayed to refer dimensions
vii. class name will be titled to the subplots

15. def _compare3 generate a batch of 3 images, and compare with original images and import numpy as np

16 .def_unSize scale the predicted bounding boxes to suit original images sizes and includes the following arguments:

i. boxes : array of predicted bounding box coordinates (None,4)
ii. index : index of original image used for prediction
iii. returns:
iv. boxes : array of bounding box coordinates mapped to original images sizes (None,4)

17. def unCode(self,names): decode the predicted names and include arguments

i. names : array of predicted class names as predict_proba output and returns
ii. names : decoded names in strings

Creation of custom call backs

The custom call backs have been specifically designed for hyperparameter tuning and include functionalities like gradually descend learning rate if the loss defined by the model is not descending at desired rate.

1. **def __init__** function includes the following arguments
   i. initial_learning_rate : the starting value of learning rate
   ii. patience : minimum number of epochs before manipulation
   iii. slope : loss gradient descent threshold
   iv. factor : multiplying factor for learning rate change
   v. lr_least : least value of learning rate to apply
   vi. verbose : status message display control flag
2. **def on_epoch_end** review learning rate & creates logs , includes following argument
   i. epoch : number of the epoch that ended
   ii. logs : learning evaluation logs
3. **For layer class layerUnFreeze** which controls the control layer trainable status over the epoch function **def __init__ includes following arguments**
   i. uncontrolled : number of layers (from the output end) to be left trainable always eg. -15 or -5 from output
   ii. schedule : dictionary of % of layers as values mapped to epochs as keys in ascending order
   iii. this defines the % of layers to be open for training from the respective epochs
   iv. this affects only the layes not marked by 'uncontrolled' eg. {20:0.25, 25:0.5, 30:0.75, 35:0.9}

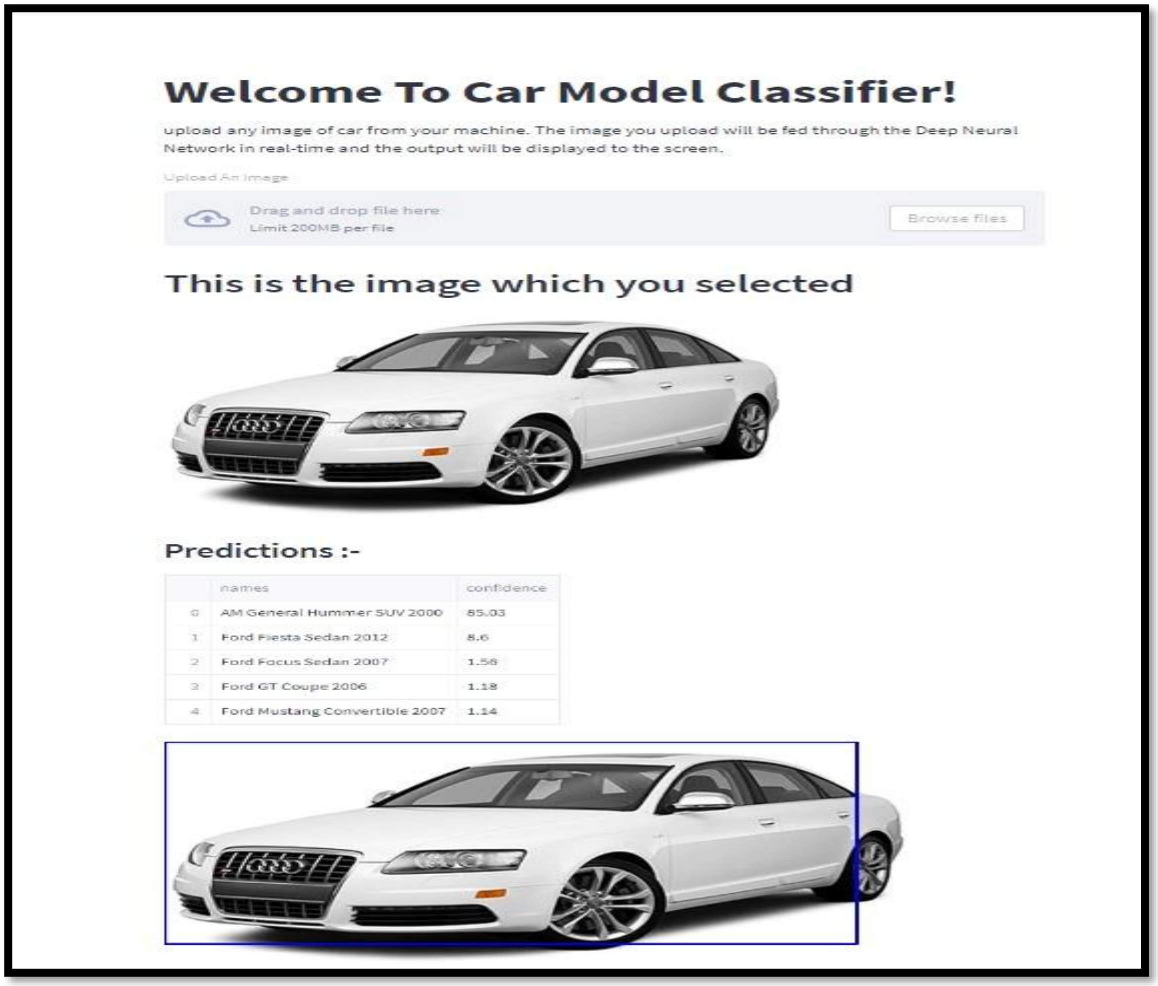
Creation of deployment block & GUI Creation

Creation of an interactive GUI has been the third milestone of the project . A free access to the model for detection and classification purposes was enabled by the GUI creation and hosting on a dedicated webpage. A pipeline for GUI creation was also developed. The GUI development makes use of streamlit application . Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom

web apps for machine learning and data science. Once an app has been created deployment, manage, and share can be done by cloud platform.
The GUI application for car classification which was created through Streamlit was deployed using Heroku which is a container-based cloud Platform as a Service (PaaS). Developers deploy, manage, and scale modern apps. The work flow for GUI creation

The streamlit model for the GUI creation included the pickled model , trained weights to perform a prediction on an image read through the GUI interface. The code further defines the attributed to be displayed along with the predicted output . A simple interface illustrated as image below was created by the team which allows upload of image to be predicted and prediction box with accuracy.

https://carmodelclassifier.herokuapp.com/



Welcome To Car Model Classifier!

upload any image of car from your machine. The image you upload will be fed through the Deep Neural Network in real-time and the output will be displayed to the screen.

Upload An Image

This is the image which you selected

Predictions :-

	names	confidence
0	AM General Hummer SUV 2000	85.03
1	Ford Fiesta Sedan 2012	8.6
2	Ford Focus Sedan 2007	1.58
3	Ford GT Coupe 2006	1.18
4	Ford Mustang Convertible 2007	1.14

4. Model evaluation

We evaluated a battery of 6 CNN based object detection models coupled with regional proposal mapping for our research problem. The models were a mix of conventional established models and latest high efficiency leaner models . The following models were tested for their accuracy in classification and segmentation for multiclass car classification and the best model in terms of performance was deployed through GUI development

1. **VGG 16**
2. **VGG 19**
3. **MobileNetV2**
4. **ResNet50V2**
5. **EfficientNetV2s**
6. **NASNetMobile**

The following section provides as overview of the deep learning models tested including their development, structure, salient features, known strength and weaknesses. The object detection models have come a long way from the initial Convolutional neural networks (CNNs) and have become the dominant machine learning approach for visual object recognition. Although they were originally introduced over 20 years ago , improvements in computer hardware and network structure have enabled the training of truly deep CNNs only recently. The original LeNet5 consisted of 5 layers, VGG featured 19 followed by ResNet and Highway getting significantly deeper.

VGG16 and VGG19

VGG stands for Visual Geometry Group; it is a standard deep Convolutional Neural Network (CNN) architecture with multiple layers. The “deep” refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers. The VGG architecture is the basis of ground-breaking object recognition models. Developed as a deep neural network, the VGGNet also surpasses baselines on many tasks and datasets beyond ImageNet. Moreover, it is now still one of the most popular image recognition architectures. VGG16 won the “2014 ILSVR (ImageNet)” competition

The VGG network is constructed with very small convolutional filters. The VGG-16 consists of 13 convolutional layers and three fully connected layers.

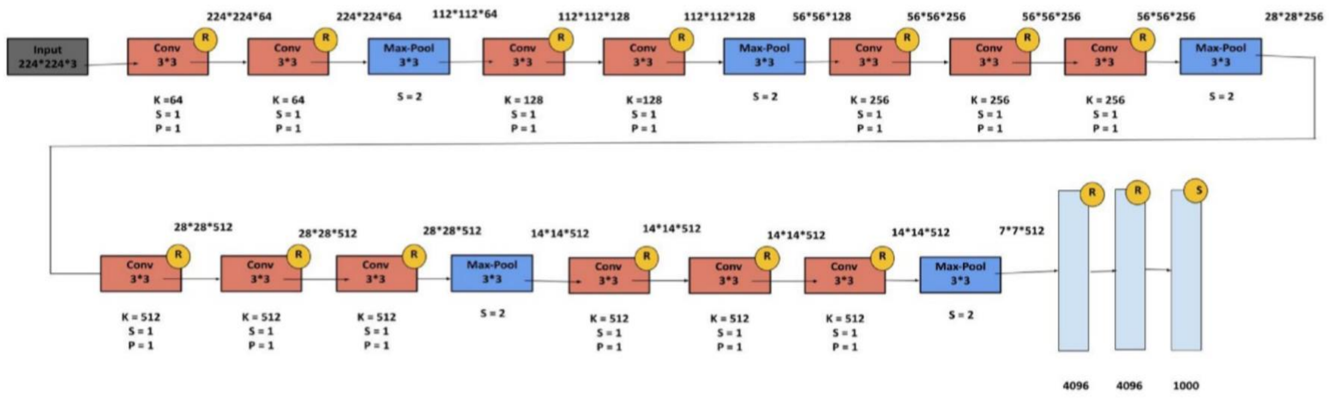
Input Layer: The VGGNet takes in an image input size of 224×224. For the ImageNet competition, the creators of the model cropped out the center 224×224 patch in each image to keep the input size of the image consistent.

Convolutional Layers: VGG’s convolutional layers leverage a minimal receptive field, i.e., 3×3, the smallest possible size that still captures up/down and left/right. Moreover, there are also 1×1 convolution filters acting as a linear transformation of the input. This is followed by a ReLU unit, which is a huge innovation from AlexNet that reduces training time. ReLU stands for rectified linear unit activation function; it is a piecewise linear function that will output the input if positive; otherwise, the output is zero. The convolution stride is fixed at 1 pixel to keep the spatial resolution preserved after convolution (stride is the number of pixel shifts over the input matrix).

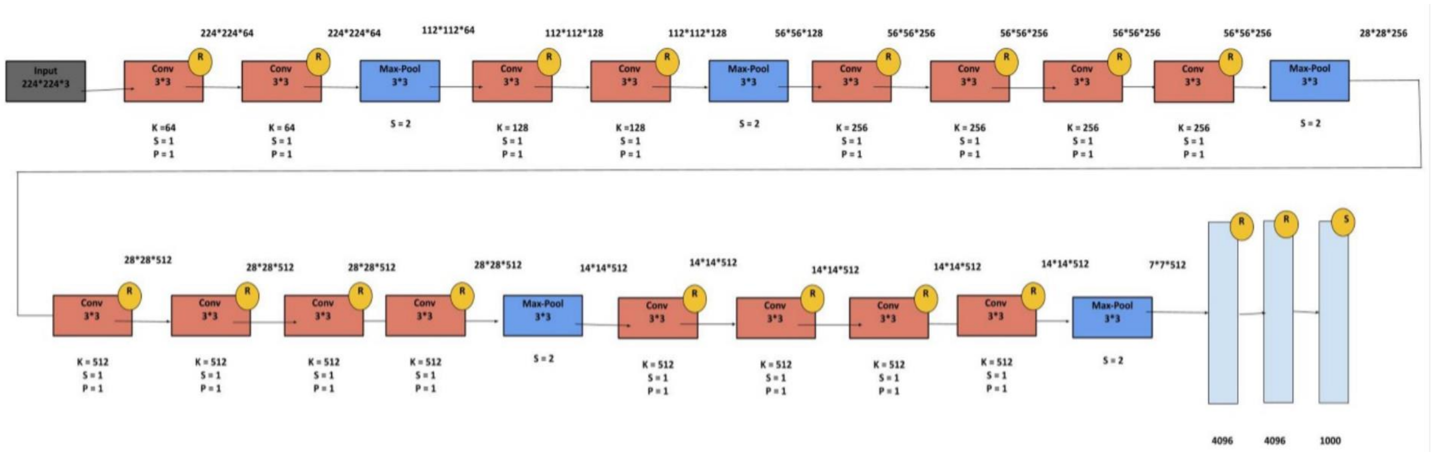
Hidden Layers: All the hidden layers in the VGG network use ReLU. VGG does not usually leverage Local Response Normalization (LRN) as it increases memory consumption and training time. Moreover, it makes no improvements to overall accuracy.

Fully-Connected Layers: The VGGNet has three fully connected layers. Out of the three layers, the first two have 4096 channels each, and the third has 1000 channels, 1 for each class.

VGG16 network configuration is a 224 by the 224-pixel image with three channels (R, G, and B). VGG16 has 16 layers. it follows this arrangement of 13 convolutional layers, 3 fully connected layers, and max- pooling layers that reduce volume size and softmax activation function, followed by the last fully connected layer. Instead of having a large number of hyper-parameters, VGG16 focuses on 3x3 filter convolution layers with stride 1 and always utilizes the same padding and MaxPool layer of a 2x2 filter with stride 2.



VGG19 architecture is a variant of the VGG model, consist of 16 convolutional neural networks, 3 FC layers, 5 MaxPool layers and 1 SoftMax layer. The fixed-size input images a 224 by 224 pixel with three channels (R, G, and B) which means that the matrix is of shape (224,224,3).



<u>**Advantages of Using VGG**</u>

i. Although derived from AlexNet, instead of using large receptive fields like in AlexNet (11x11 with a stride of 4), VGG uses very small receptive fields (3x3 with a stride of 1). Because there are now three ReLU units instead of just one, the decision function is more discriminative. There are also fewer parameters.

ii. VGG incorporates 1x1 convolutional layers to make the decision function more non-linear without changing the receptive fields.

iii. The small-size convolution filters allows VGG to have a large number of weight layers; of course, more layers leads to improved performance

iv. It is a very good architecture for benchmarking on a particular task.

v. Also, pre-trained networks for VGG are available freely on the internet, so it is commonly used out of the box for various applications

Shortcomings of VGGNet

i. It is very slow to train (the original VGG model was trained on Nvidia Titan GPU for 2-3 weeks).
ii. The size of VGG-16 trained imageNet weights is 528 MB. So, it takes quite a lot of disk space and bandwidth which makes it inefficient.
iii. 138 million parameters lead to exploding and also vanishing gradients problem

ResNet50V2

ResNet stands for Residual Network. It is an innovative neural network that was first introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their 2015 computer vision research paper titled 'Deep Residual Learning for Image Recognition'. Its ensemble won the top position at the ILSVRC 2015 classification competition with an error of only 3.57%. Additionally, it also came first in the ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation in the ILSVRC & COCO competitions of 2015. While the number of stacked layers can enrich the features of the model, a deeper network can show the issue of degradation. In other words, as the number of layers of the neural network increases, the accuracy levels may get saturated and slowly degrade after a point. As a result, the performance of the model deteriorates both on the training and testing data. This degradation is not a result of overfitting. Instead, it may result from the initialization of the network, optimization function, or, more importantly, the problem of vanishing or exploding gradients.

What is Deep Residual Learning used for?

ResNet was created with the aim of tackling this exact problem. Deep residual nets make use of residual blocks to improve the accuracy of the models. The concept of “skip connections,” which lies at the core of the residual blocks, is the strength of this type of neural network.

What are Skip Connections in ResNet?

These skip connections work in two ways. Firstly, they alleviate the issue of vanishing gradient by setting up an alternate shortcut for the gradient to pass through. In addition, they enable the model to learn an identity function. This ensures that the higher layers of the model do not perform any worse than the lower layers.

In short, the residual blocks make it considerably easier for the layers to learn identity functions. As a result, ResNet improves the efficiency of deep neural networks with more neural layers while minimizing the percentage of errors. In other words, the skip connections add the outputs from previous layers to the outputs of stacked layers, making it possible to train much deeper networks than previously possible.

ResNet35

The first ResNet architecture was the Resnet-34 [1]which involved the insertion of shortcut connections in turning a plain network into its residual network counterpart. In this case, the plain network was inspired by **VGG** neural networks (VGG-16, VGG-19), with the convolutional networks having 3×3 filters. However, compared to **VGGNets**, ResNets have fewer filters and lower complexity. The 34-layer ResNet achieves a performance of 3.6 bn FLOPs, compared to 1.8bn FLOPs of smaller 18-layer ResNets.

It also followed two simple design rules – the layers had the same number of filters for the same output feature map size, and the number of filters doubled in case the feature map size was halved in order to preserve the time complexity per layer. It consisted of 34 weighted layers.

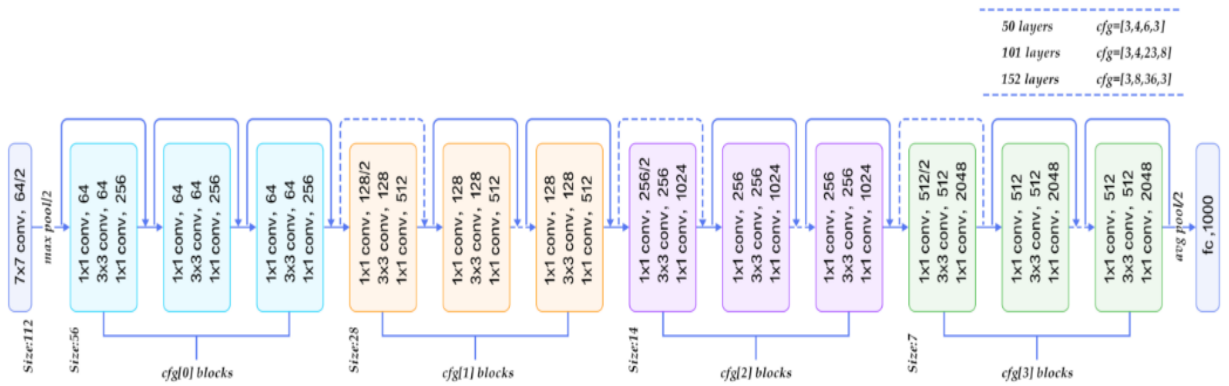
ResNet-50 Architecture

While the Resnet50 architecture is based on the above model, there is one major difference. In this case, the building block was modified into a bottleneck design due to concerns over the time taken to train the layers. This used a stack of 3 layers instead of the earlier 2.

[1]ResNet development , https://arxiv.org/pdf/1512.03385.pdf

Therefore, each of the 2-layer blocks in Resnet34 was replaced with a 3-layer bottleneck block, forming the Resnet 50 architecture. This has much higher accuracy than the 34-layer ResNet model. The 50-layer ResNet achieves a performance of 3.8 bn FLOPS.

ResNet50 is a variant of **ResNet model** which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer. It has 3.8 x 10^9 Floating points operations.



ResNet50 Architecture

Advantages of Using ResNet

i. Networks with large number (even thousands) of layers can be trained easily without increasing the training error percentage.

ii. ResNets help in tackling the vanishing gradient problem using identity mapping.

<u>Shortcoming of ResNet</u>

i. Increased complexity of architecture

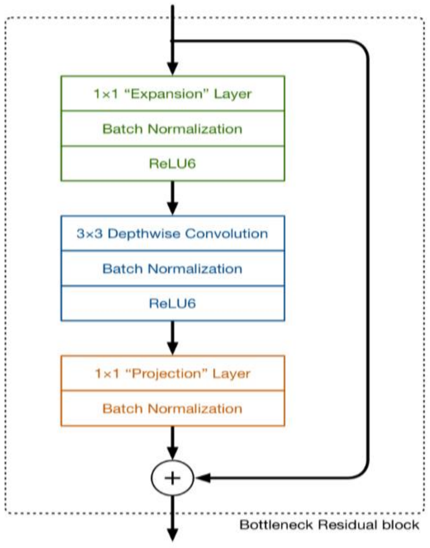
ii. Implementation of Batch normalization layers since ResNet heavily depends on it

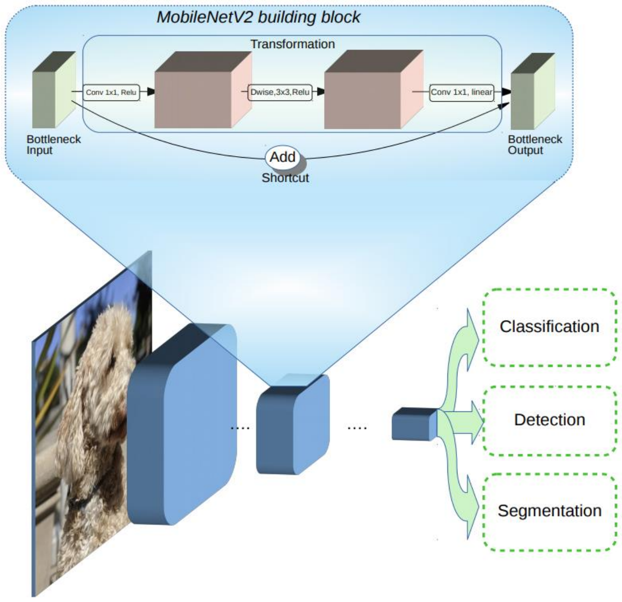
iii. Adding skip level connections for which you have take into account the dimensionality between the different layers

MobileNetV2

In 2017, Google introduced MobileNets[2], a family of computer vision models based on TensorFlow. The original paper was followed by the release of MobileNetV2 in April 2018 and MobileNetV3 in May 2019. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. We introduce two simple global hyper-parameters that efficiently trade off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem.

It is based on an inverted residual structure where the residual connections are between the bottleneck layers. The intermediate expansion layer uses lightweight depth wise convolutions to filter features as a source of non-linearity. As a whole, the architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers.





[2] **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,** https://arxiv.org/abs/1704.04861
https://machinethink.net/blog/mobilenet-v2/

The big idea behind MobileNet V1 is that convolutional layers, which are essential to computer vision tasks but are quite expensive to compute, can be replaced by so-called depthwise separable convolutions.The job of the convolution layer is split into two subtasks: first there is a depthwise convolution layer that filters the input, followed by a 1×1 (or pointwise) convolution layer that combines these filtered values to create new features.Together, the depthwise and pointwise convolutions form a “depthwise separable” convolution block. It does approximately the same thing as traditional convolution but is much faster.

In MobileNetV2 time there are three convolutional layers in the block. The last two are the ones we already know: a depthwise convolution that filters the inputs, followed by a 1×1 pointwise convolution layer. However, this 1×1 layer now has a different job. In V1 the pointwise convolution either kept the number of channels the same or doubled them. In V2 it does the opposite: it makes the number of channels smaller. This is why this layer is now known as the **projection layer** — it projects data with a high number of dimensions (channels) into a tensor with a much lower number of dimensions.

Advantages of MobileNet

i. They are computationally much lighter then conventional CNN models
ii. With depthwise separable convolutions, MobileNet has to do about 9 times less work than comparable neural nets with the same accuracy.
iii. They are easily deployable on mobile devices

Shortcomings of Mobile Net

i. There are trade off in terms of accuracy with conventional models like RESnets performing better than MobileNet

NASNetMobile

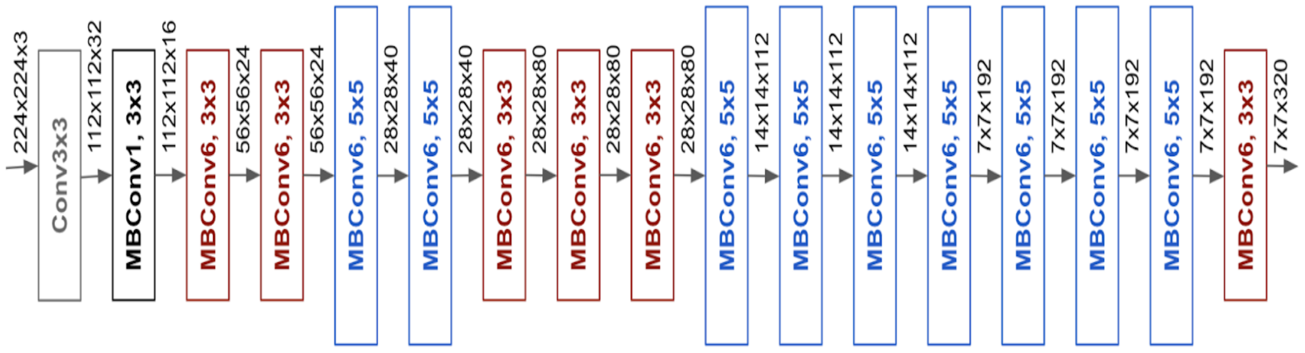
Equipped with abundance of computing power and engineering genius, Google introduced NASNet, which framed the problem of finding the best CNN architecture as a Reinforcement Learning problem. In NASNet, though the overall architecture is predefined as shown above, the blocks or cells are not predefined by authors. Instead, they are searched by reinforcement learning search method.

Basically the idea was to search the best combination of parameters of the given search space of filter sizes, output channels, strides, number of layers, etc. In this Reinforcement Learning setting, the reward after each search action was the accuracy for the searched architecture on the given dataset.
NASNet achieved state-of-the-art result in the ImageNet competition. However, the computation power required for NASNet was so big that only a handful of companies were able to make use of the same methodology.

EfficientNetV2S

EfficientNetV2S was launched in 2021 and is a type convolutional neural network that has faster training speed and better parameter efficiency than previous models. To develop these models, a combination of training-aware neural architecture search and scaling, to jointly optimize training speed. The models were searched from the search space enriched with new ops such as Fused-MBConv.
Architecturally the main differences are:
EfficientNetV2 extensively uses both MBConv and the newly added fused-MBConv in the early layers.
EfficientNetV2 prefers smaller expansion ratio for MBConv since smaller expansion ratios tend to have less memory access overhead.
EfficientNetV2 prefers smaller 3x3 kernel sizes, but it adds more layers to compensate the reduced receptive field resulted from the smaller kernel size.
EfficientNetV2 completely removes the last stride-1 stage in the original EfficientNet, perhaps due to its large parameter size and memory access overhead.

Efficient Net Architecture



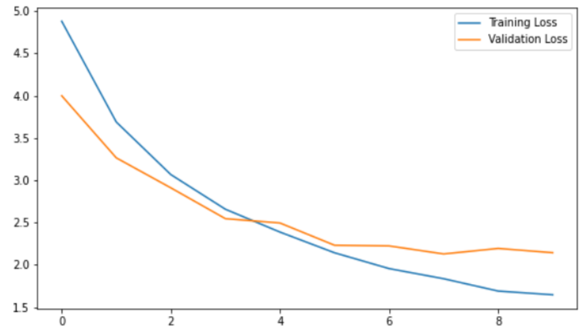
Model Evaluation Process

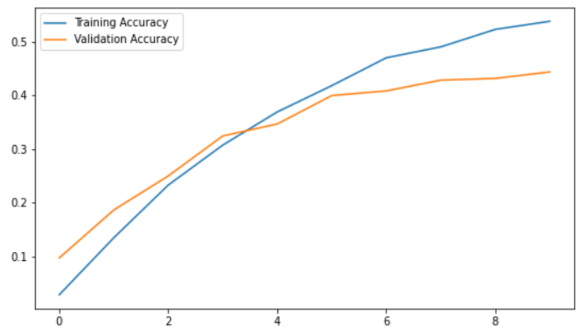
The following section details step-by-step process for final model evaluation.

Initial Car Classification with CNN Model

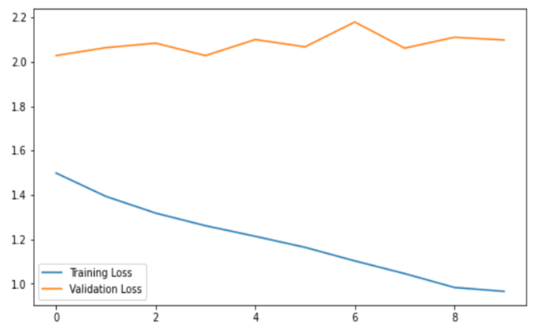
The initial model run was performed with CNN model for car name classification. We used MobileNetV2 for the evaluation and following results were achieved

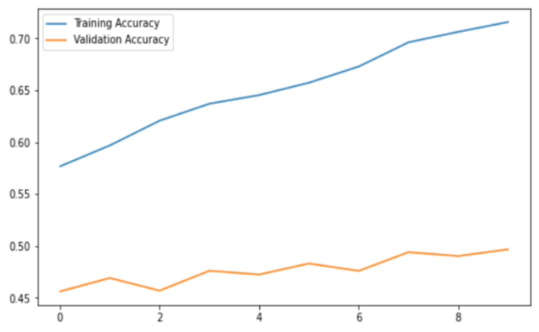
Initial Run





The training and test accuracy were in sync and for the initial runs we could get a training accuracy of 50% and validation accuracy of 40%. The data overfitting was reduced through data augmentation which led to improvement in classification accuracy for training to 70% and validation to 50%. **The validation accuracy saturated at 50% for 10 epochs.**





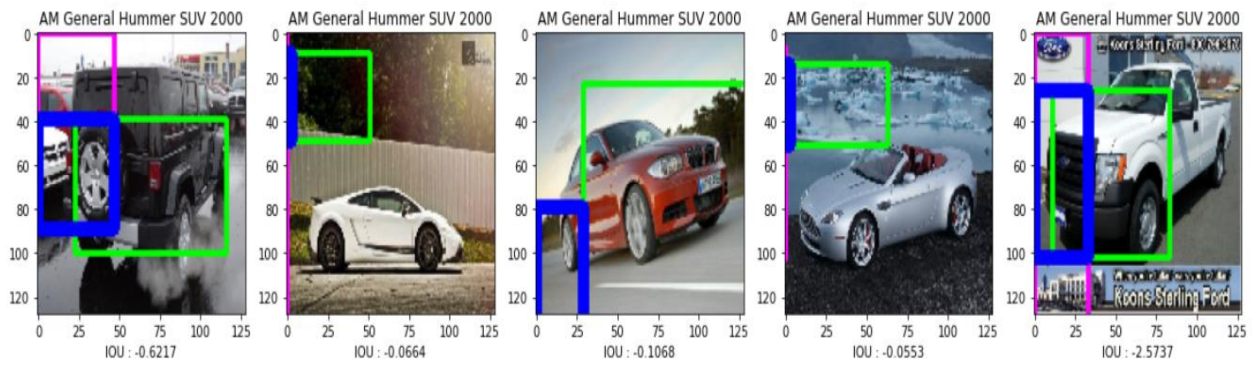
As the object detection goal includes both object classification and segmentation we developed a hybrid model which will allow both classification and segmentation simultaneously.

CNN models provide greater computational power and help in better classification of images as compared to initial shallow models.

Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Region-Based Convolutional Neural Networks, or R-CNNs, are a family of techniques for addressing object localization and recognition tasks and are designed for model performance. The models are computationally heavy and could not fit into the computing capabilities available with the group. The team developed a novel image classification and bounding box combo generator which simultaneously provides image classification and object localization function for the dataset.

CNN Models with Region Segmentation

As has been discussed earlier initial classification runs with segmentation by bounding boxes presented in the interim report had very low accuracy indicating that not much learning took place with then used code .



A series of iteration with development pipe and generator were performed to come up with an environment which provides consistent and explainable results. The initial

CNN models looked into were i.e VGG 16 and MobileNet for the interim report . Improvement on the performance were attributed to two three parameters

1. Selecting an appropriate optimizer : The initial runs were done with rmsprop with almost no learning for the model shifting to more advanced Adam optimizer caused a major breakthrough in terms of learning where the models with near zero training accuracy moved to 30%

2. Optimizing an appropriate image size input into model further improved the performance of the trained models

3. Image regularization was also performed which further improved the training efficiency of the models

After the initial consistency requirement of the models was fulfilled a battery of 6 CNN models coupled with region proposals or segmentation augmented through the generator model was performed .The details of network trained are delineated in **Table 4.1**

S.No	Pre trained Models	Deep Layer	Parameters	Size (MB)	Input Image Size
1	VGG 16	16	14.71 M	528	224x224x3
2	VGG 19	19	20.02 M	549	224x224x3
3	MobileNetV2	105	2.25M	14	224x224x3
4	ResNet50V2	50	23.5 M	98	224x224x3
5	EfficientNetV2	201	20.33M	88	224x224x3
6	NASNetMobile	389	4.2M	23	224x224x3

All the models were consistently trained. Table 4.2 provides the key parameters for training

S.No	PreTrained Models	Target Sie	Batch Sie	optimiser	patience	slope	factor	Learning Rate	Epochs
1	ResNet50V2	(224 , 224)	32	Adam	15	0.15	0.001	0.001	30
2	MobileNetV2	(224 , 224)	32	Adam	15	0.15	0.001	0.001	30
3	VGG16	(224 , 224)	32	Adam	15	0.15	0.001	0.001	30
4	NASNetMobile	(224 , 224)	32	Adam	15	0.15	0.001	0.001	30
5	VGG19	(224 , 224)	32	Adam	15	0.15	0.001	0.001	30
6	EffienctNetV2S	(224 , 224)	32	Adam	15	0.15	0.001	0.001	30

All the models have similar target and batch size. The learning rate was fixed at 0.001 for the initial training. All the models were run in as separate .py files calling in the pipeline and the generator model. This helped in parallel running of the models and optimizing the computational bandwidth available by the group. The overall accuracy for the model runs for both segmentation through bounding boxes and classification accuracy are delineated in table 4.3

S.No	PreTrained Models	Loss	Classification Accuracy	IOU	Val Loss	Name Accuracy	IOU
		Training	Training	Training	Validation	Validation	Validation
1	ResNet50V2	0.37	0.936	0.587	2.48	0.409	0.589
2	MobileNetV2	0.45	0.929	0.65	2.5	0.398	0.648
3	VGG16	0.84	0.859	0.646	2.86	0.327	0.651
4	NASNetMobile	1.2	0.729	0.667	2.81	0.305	0.665
5	VGG19	0.95	0.838	0.6182	3.03	0.304	0.62
6	EffienctNetV2S	4.9	0.037	0.511	5.24	0.016	0.515

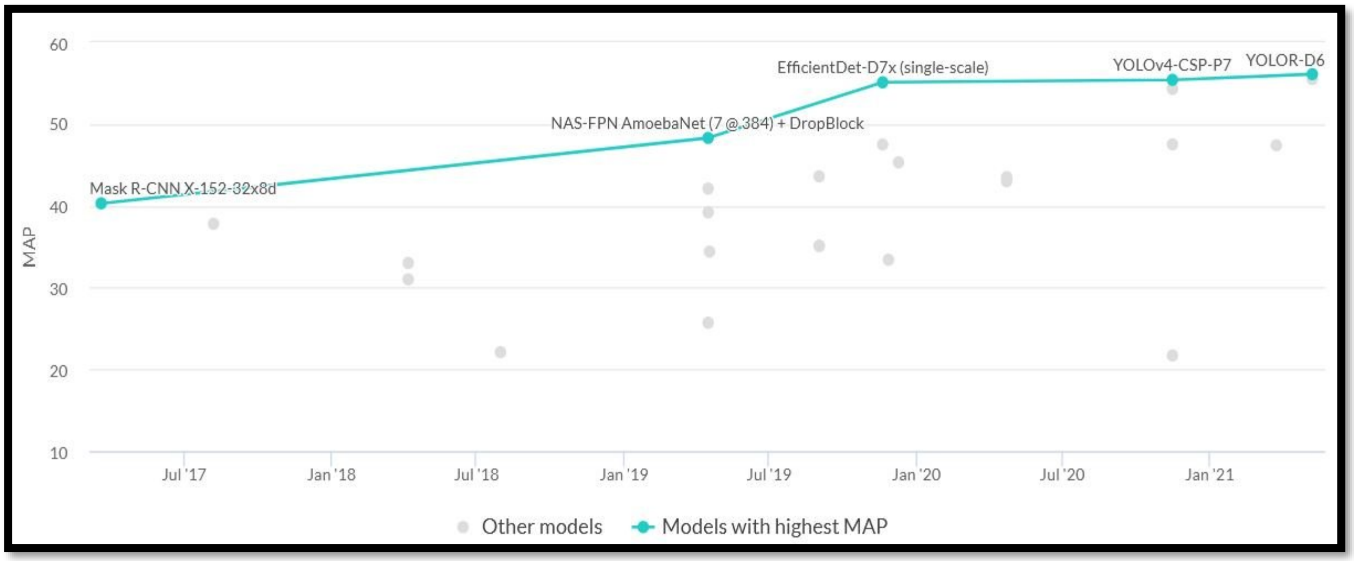
We find that accuracies of ResNet50V2 AND Mobile Net to be comparable but the MobileNet provided more overfitting than ResNet50V2 . This led ResNet50 to be the choice for final model hyperparameter tuning

5.Comparison to benchmark

We find a noticeable improvement in segmentation efficiency for the model where the IOU increased from close to 60% to 70 % which is over 16 % increase in the region mapping for validation set providing a clear evidence of better learning and more generalized model . However the classification accuracy marginally decreased from 40.1% to 38.3 % for validation set name accuracy there was no overall change in trend of overfitting for the model . The model achieved an average precision of 65 aligned with the existing benchmarks .We find that the final hyper tunned model works with better segmentation accuracy but similar classification accuracy

Benchmarking the Model accuracy

The industry benchmark for the Stanford dataset could not be obtained best real-time object detection algorithm for COCO dataset in terms of Mean Average Precision in 2021 is YOLOR (MAP 56.1). The algorithm is closely followed by YOLOv4 (MAP 55.4) and EfficientDet (MAP 55.1).



Source: https://viso.ai/deep-learning/object-detection/

The initial model runs provided a benchmark for selection of the best model and performing hyperparameter tunning. Our initial model runs indicated RESNET50V2 to be the best model for Hyperparameter tunning.

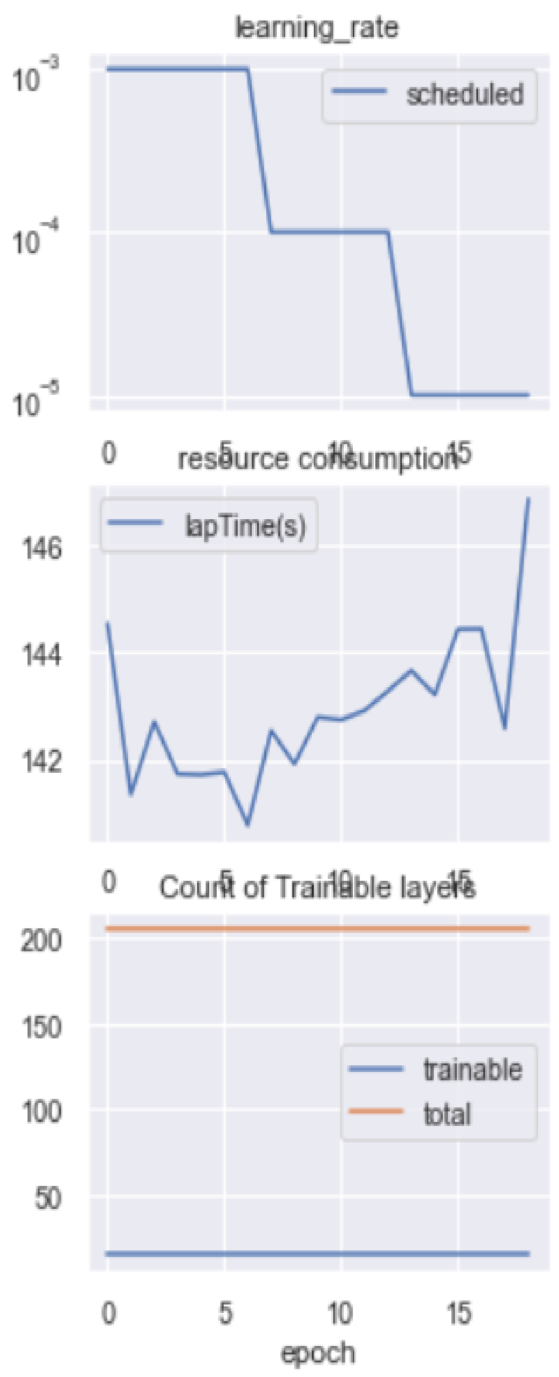
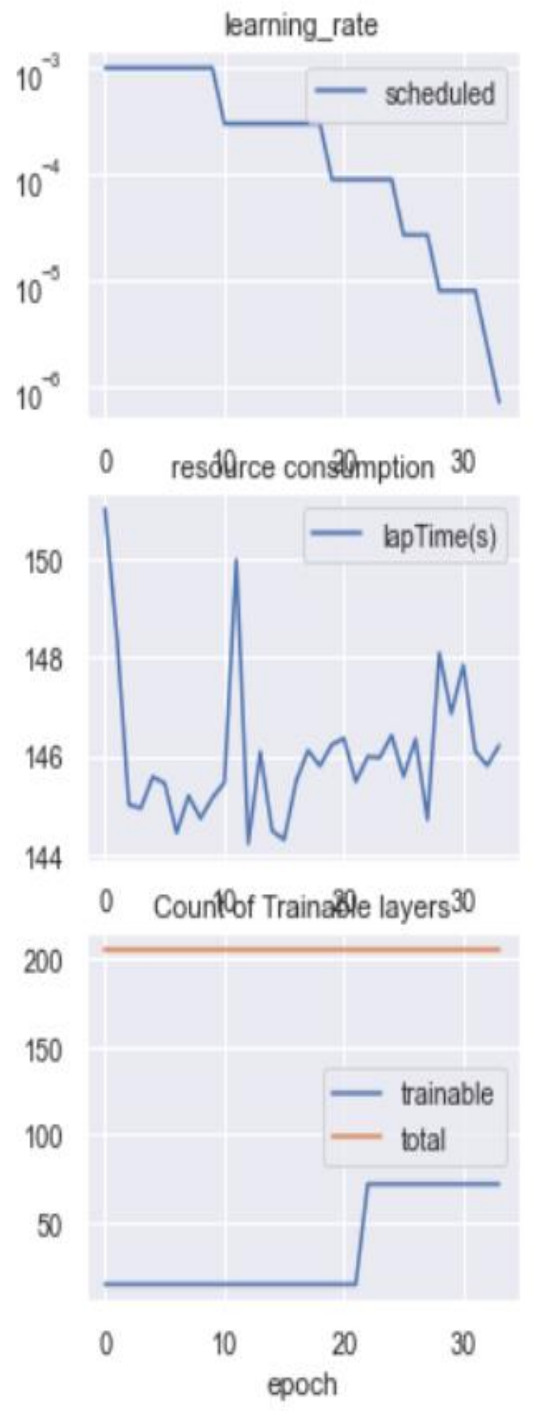
Hyper parameter tunning is highly computationally heavy and we had to optimize the hyperparameter tunning parameters to the computational bandwidth available for the team. The following parameters were hyper tuned for the model.

1. Optimizer options included Adam and SGD
2. Patience
3. Slope
4. Un Freeze rate set to 30% after 20th epoch over and above 15 layers already unfreeze for the model
5. Learning Rate
6. Output loss balance
7. Dropout rate
8. Regularization

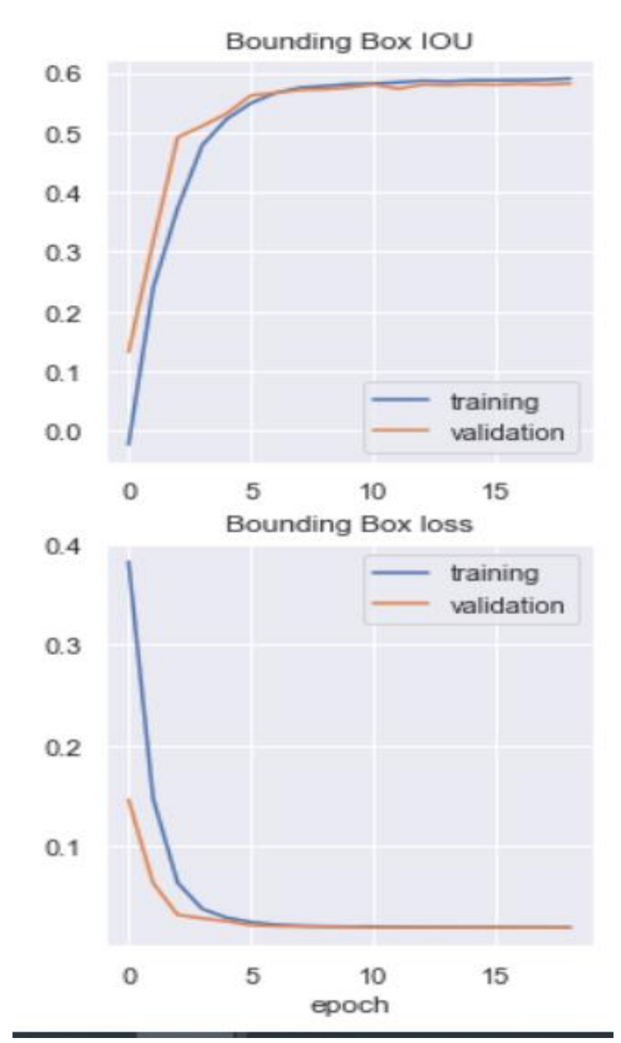
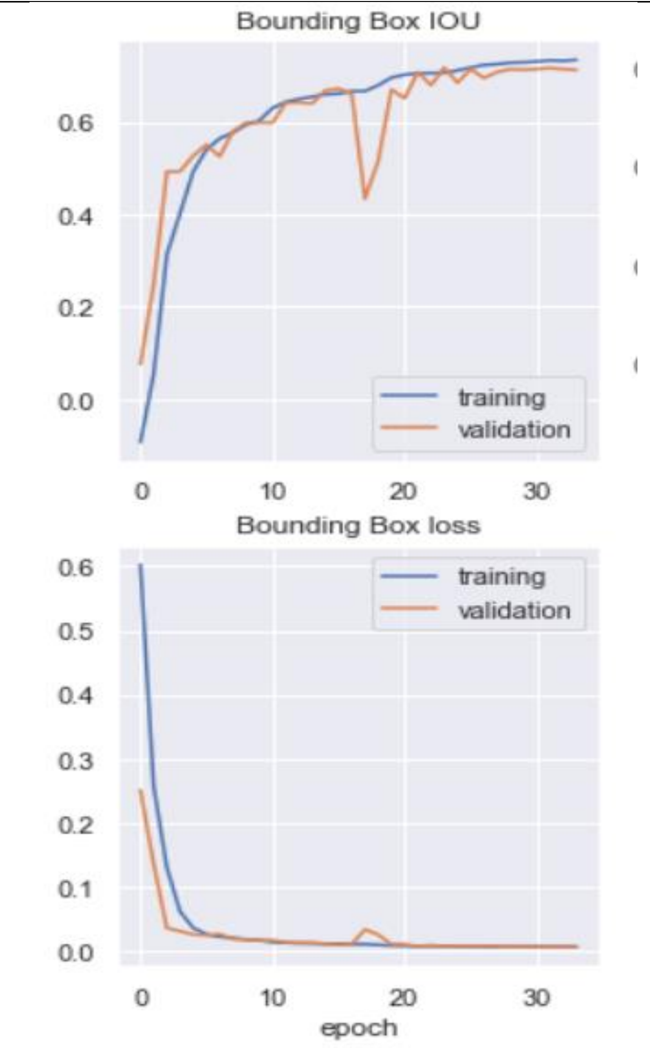
Parameter Tuning Range

```
optims = [Adam(), SGD(momentum=0.1)]
patiences = [4,8]
slopes = [0.3,0.5]
factors = [0.1,0.3]
uFreezePercents = [0,0.3,0.7]
learningRates = [1e-3,1e-7]
lossBalancing = [0.2,0.8]
dOuts = [0.5,0.7]
lmbdas = [0.1,0.01]
```

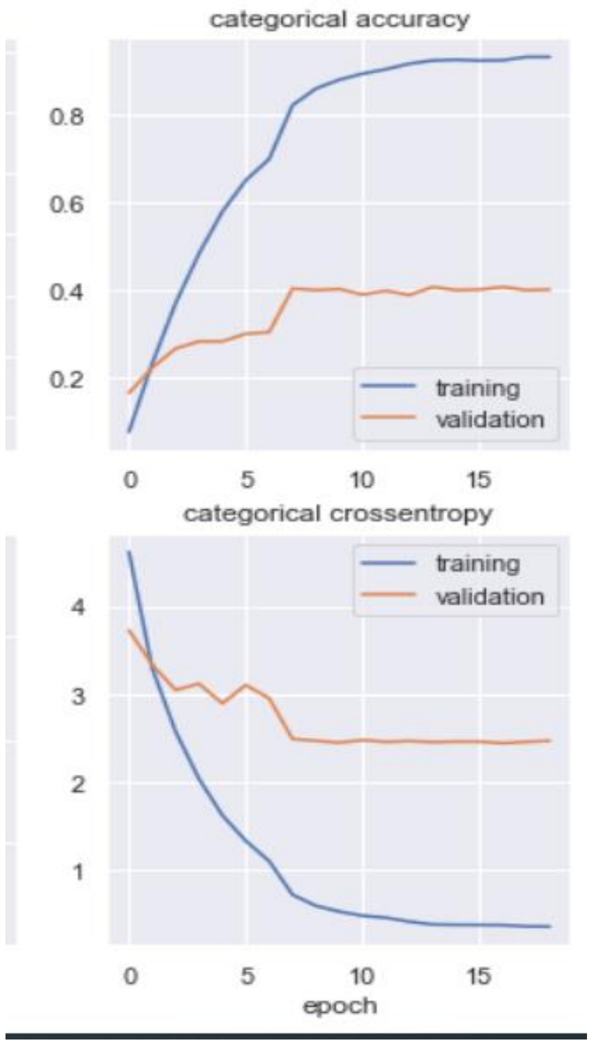
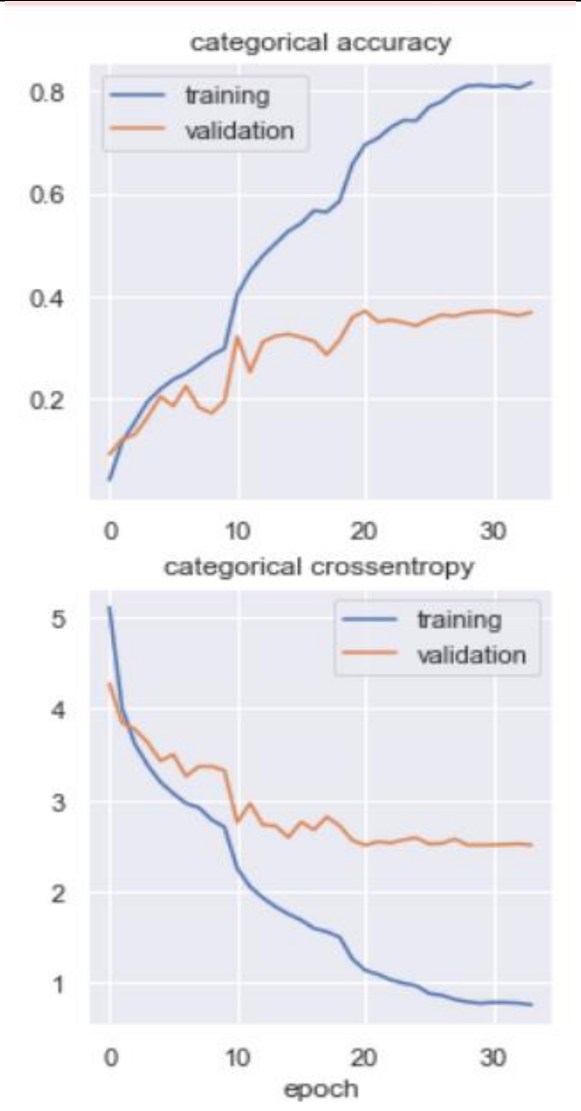
Figure 5.1 : Model hyper tuning comparative Learning Rate , Resource Use and unFreezed layers for hypertunning

PRE HYPERTUNING LR, RESOURCE USE, UNFRZEE	POST HYPERTUNING LR, RESOURCE USE, UNFRZEE
	

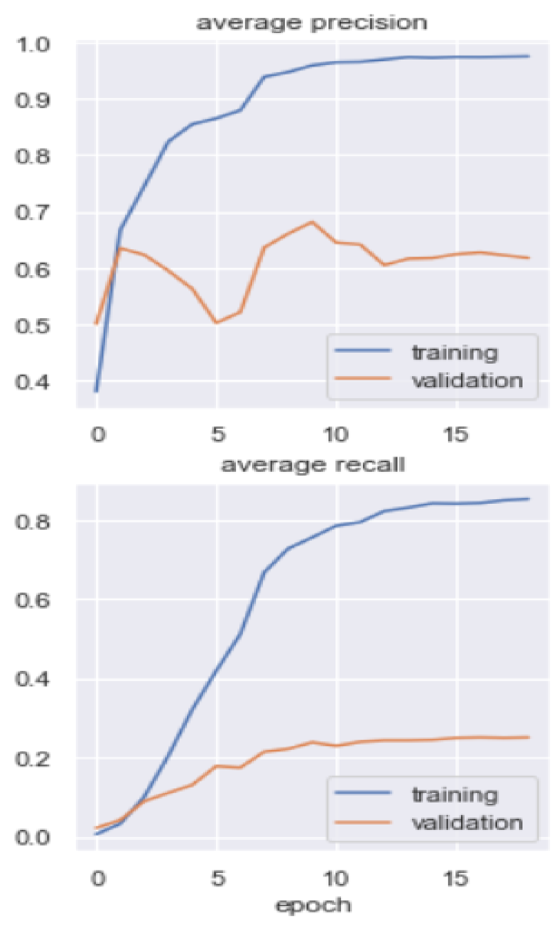
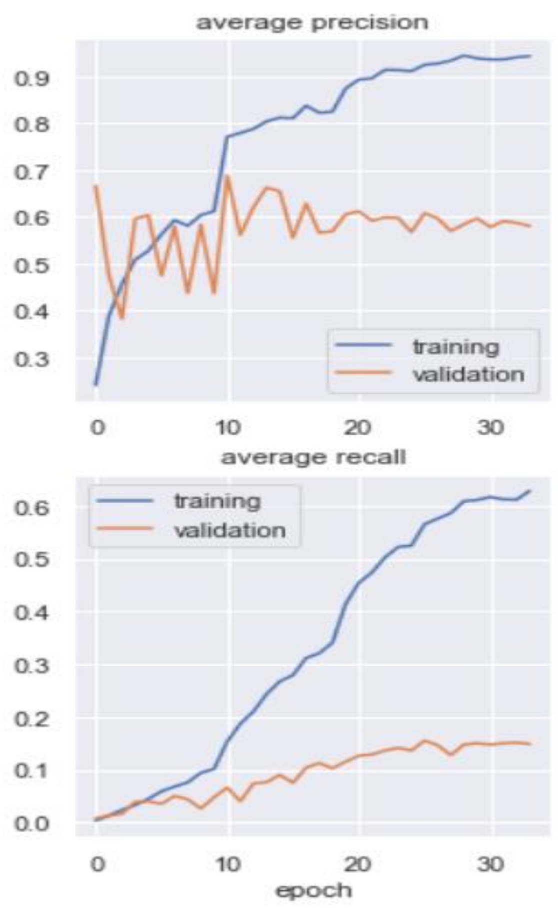
The model hyper tunning let to fine tunning in terms of learning rate and trainable layers . After epoch 20, 30% of the layers were unfreezed to provide better flexibility while training . This increased the trainable parameters for the model . This also led to slight increasing in training time and computational requirement . The overall hyperparameter tunning took over 10 hrs and the following change in parameters were obtained . **We find a noticeable improvement in segmentation efficiency for the model where the IOU increased from close to 60% to 70 % which is over 16 % increase in the region mapping .** Figure 5.2 provides the details

PRE HYPERTUNING IOU Bounding Box	POST HYPERTUNING IOU Bounding Box
	

However, we find that name classification efficiency marginally deteriorated for the validation set for the hyper tuned model . The overall validation accuracy for the model before hypertunning was 40.89% in contrast to training accuracy of 93.6% indicating a major overfitting for the training data set . However, for the hypertunned model the validation accuracy stands at 37.3% and training accuracy at 90.15% with no % change in overfitting .

PRE HYPERTUNING Classification Accuracy	POST HYPERTUNING Classification Accuracy
	

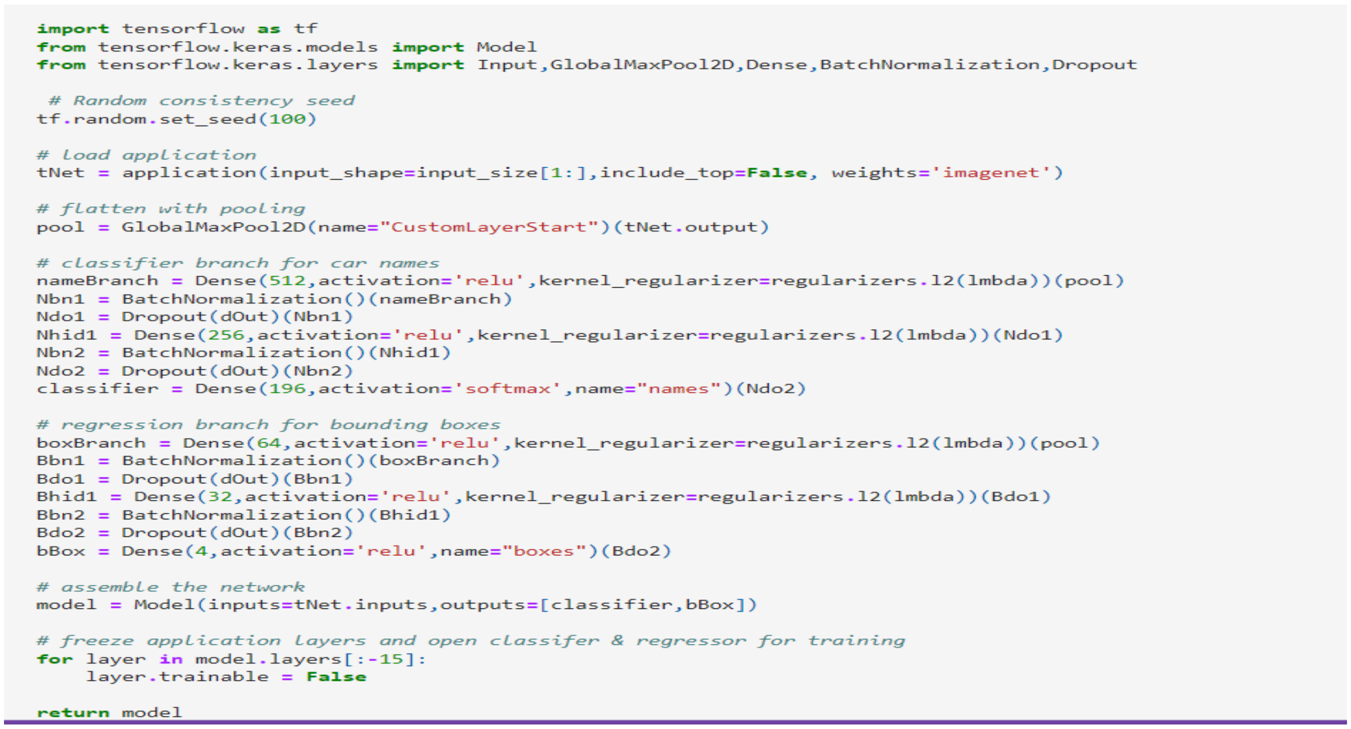
The overall computational constrains did not allow the team to experiment with other hyperparameters like batch size and increasing the unfreezing of the layers . We also saw a lot of oscillations in validation set due to changing and increasing learning rate and constrains posed by the existing model choices . The net section provides a detailed configuration of the final trained model.
Average precision of the models were compared and are detailed in figure below

PRE HYPERTUNING Classification Precision	POST HYPERTUNING Classification Precision
	

The average precision for the model was close to 65 % which is aligned with industry bench march for COCO data set.

Details of the Final Model

The final trained and hype tuned model for our work has been RESNET50V2. The topography of the final model is presented as appendix 1 . The following model configuration was defined. A pretrained transfer net with 15 unfreezed layer for custom training was initially used. The weights from the trained model were used to create the GUI based model



```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,GlobalMaxPool2D,Dense,BatchNormalization,Dropout

 # Random consistency seed
tf.random.set_seed(100)

# load application
tNet = application(input_shape=input_size[1:],include_top=False, weights='imagenet')

# flatten with pooling
pool = GlobalMaxPool2D(name="CustomLayerStart")(tNet.output)

# classifier branch for car names
nameBranch = Dense(512,activation='relu',kernel_regularizer=regularizers.l2(lmbda))(pool)
Nbn1 = BatchNormalization()(nameBranch)
Ndo1 = Dropout(dOut)(Nbn1)
Nhid1 = Dense(256,activation='relu',kernel_regularizer=regularizers.l2(lmbda))(Ndo1)
Nbn2 = BatchNormalization()(Nhid1)
Ndo2 = Dropout(dOut)(Nbn2)
classifier = Dense(196,activation='softmax',name="names")(Ndo2)

# regression branch for bounding boxes
boxBranch = Dense(64,activation='relu',kernel_regularizer=regularizers.l2(lmbda))(pool)
Bbn1 = BatchNormalization()(boxBranch)
Bdo1 = Dropout(dOut)(Bbn1)
Bhid1 = Dense(32,activation='relu',kernel_regularizer=regularizers.l2(lmbda))(Bdo1)
Bbn2 = BatchNormalization()(Bhid1)
Bdo2 = Dropout(dOut)(Bbn2)
bBox = Dense(4,activation='relu',name="boxes")(Bdo2)

# assemble the network
model = Model(inputs=tNet.inputs,outputs=[classifier,bBox])

# freeze application layers and open classifer & regressor for training
for layer in model.layers[:-15]:
    layer.trainable = False

return model
```

6.Visualizations

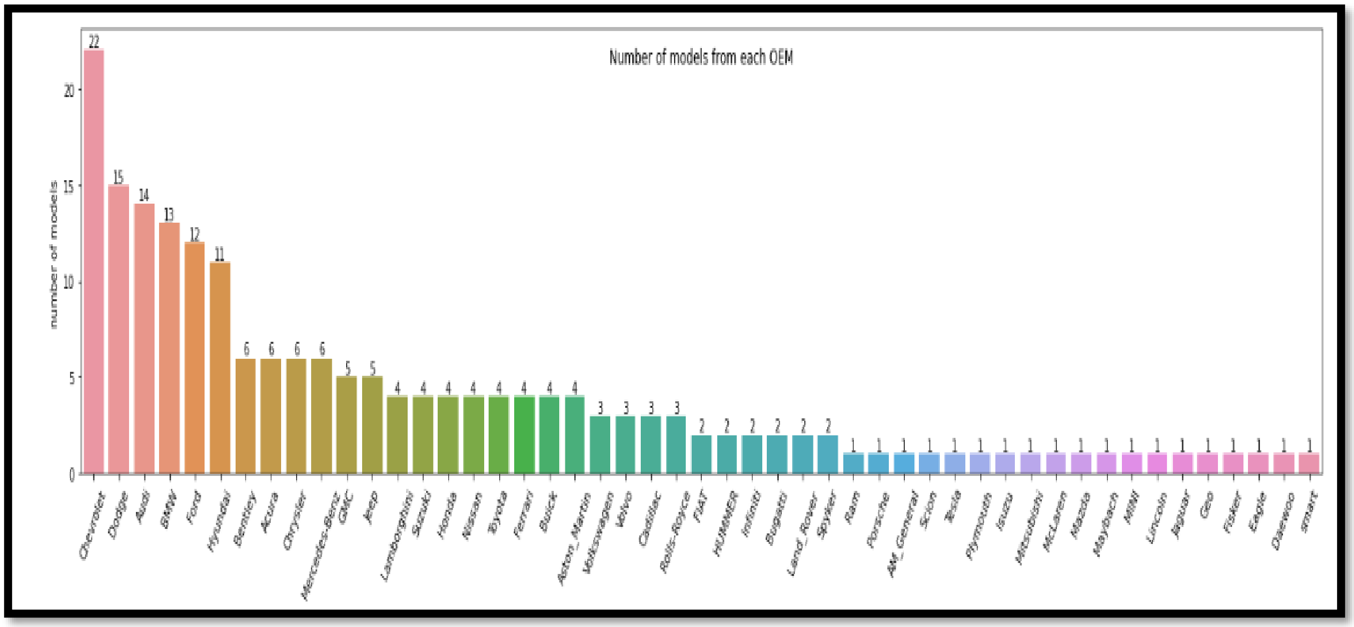
Visualizations are an important component of all the three milestone of the project . We here provide key visualizations for the three milestones that is EDA, model development and GUI driven model deployment

EDA Visualizations and inferences

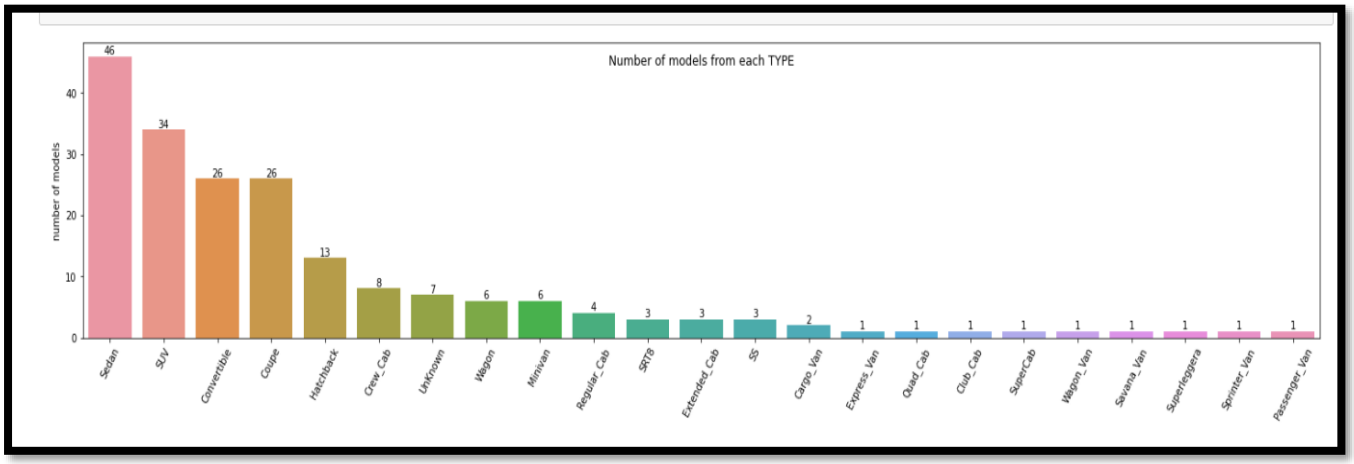
Visual review revealed there are no missing data but inconsistencies in terms of size of images and number of images for each car type. The frequency distribution was initially performed with respect to distribution of classes and number of images for OEM, car type and year of make. This was followed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

Following insights were revealed

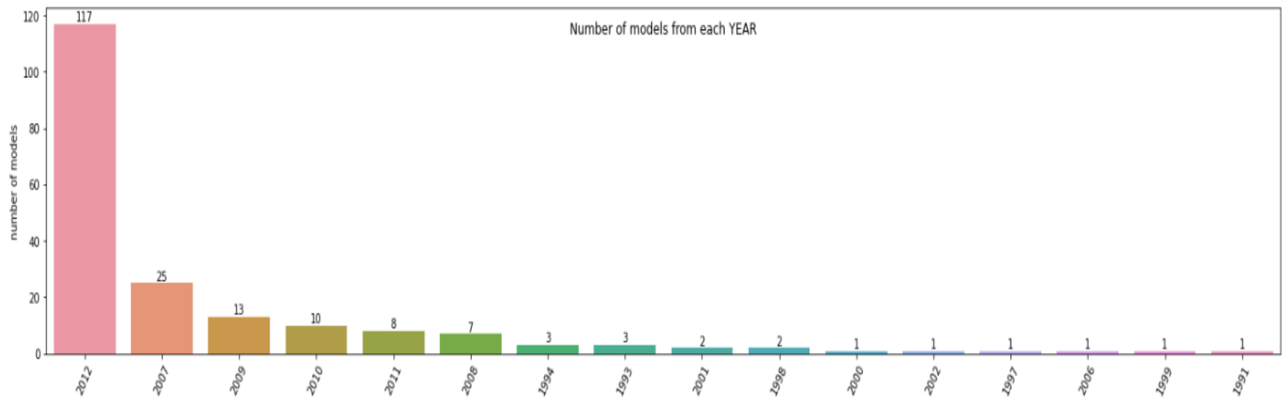
1. Chevrolet has highest number of classes in the data set



2. Sedan has highest number of classes in car type



3. 2012 is the predominant make year available in the data.



4. 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. Along with Sedans SUVs also have class instances with much higher number of training images as compared to median 2012 car make year dominates even in combined analysis emerging as the strongest bias in the data

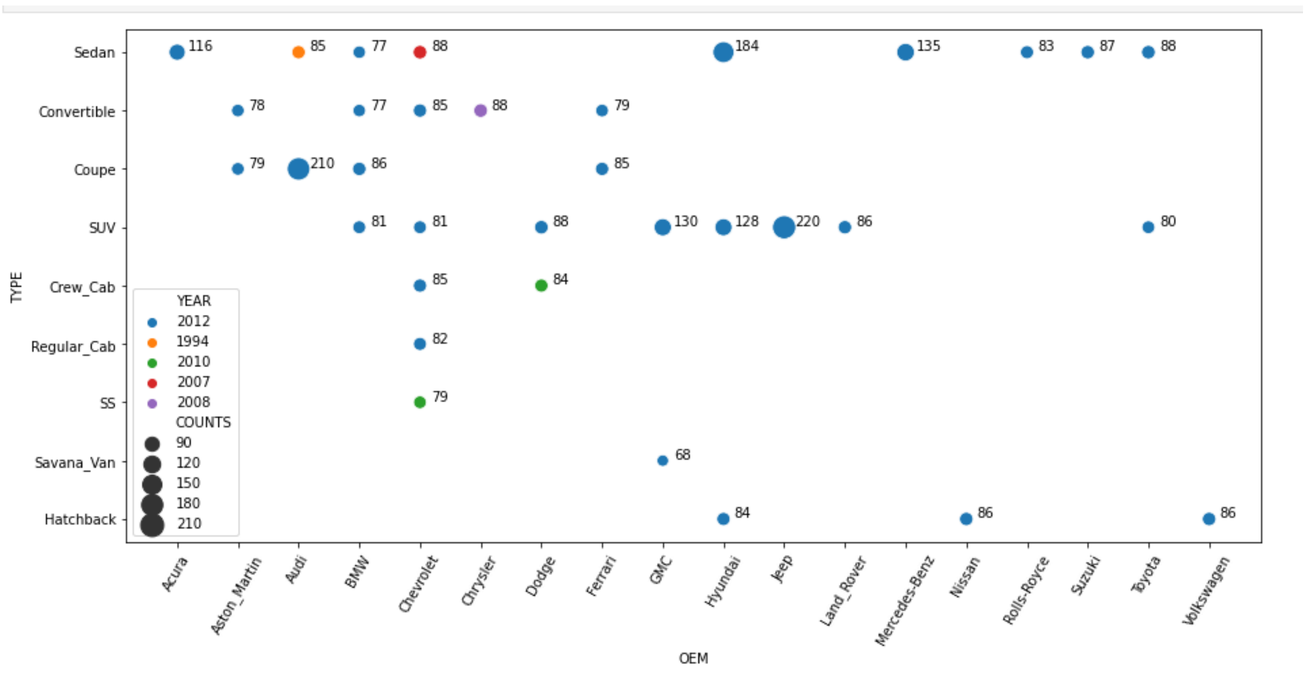


Image vary from as large as 210 Mega pixels to 4.5 kilo pixels requiring image resizing. Further, All the images also have different back ground and resolution making bounding box for training the model essential and was performed for the data

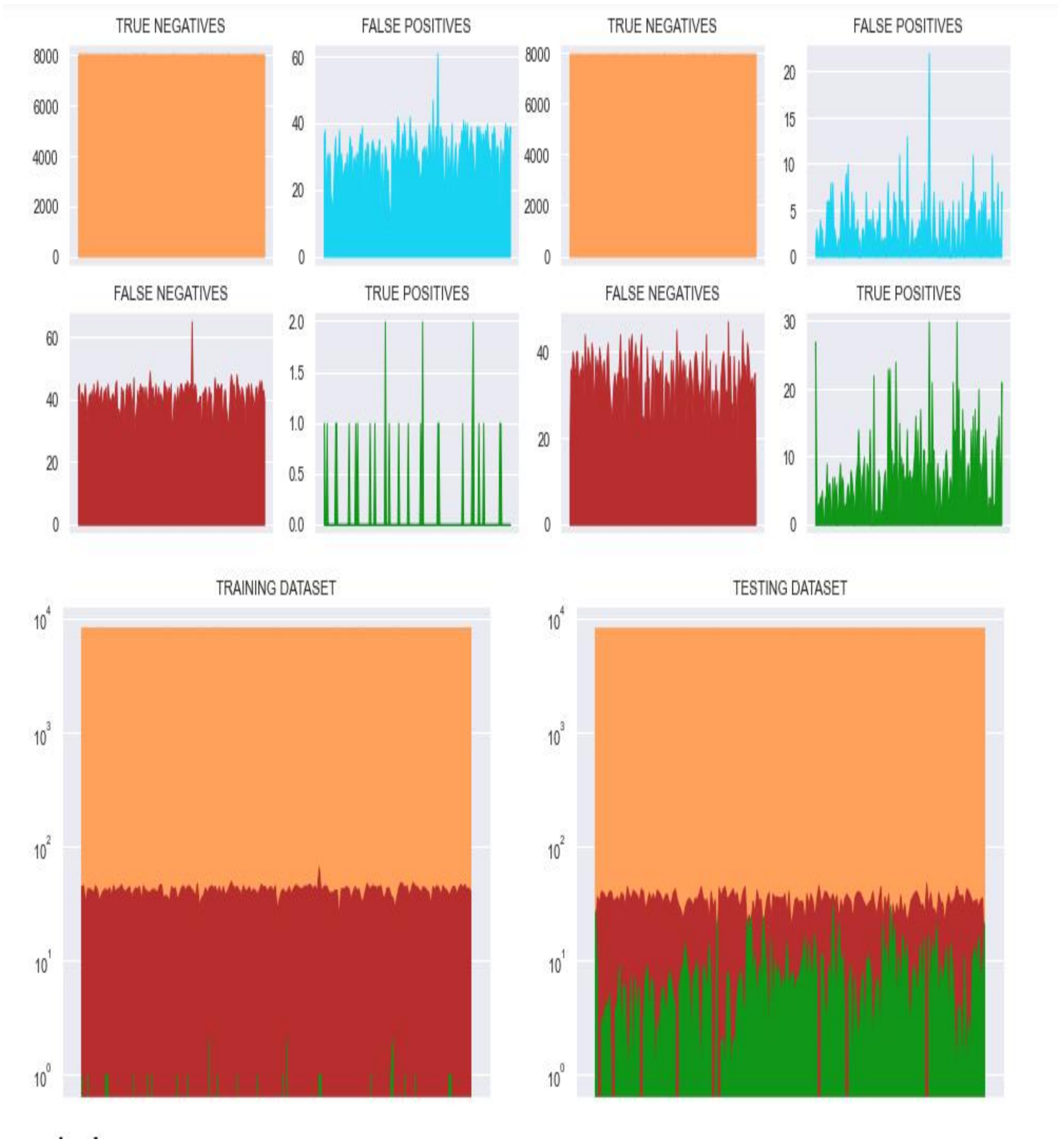
The overall inference from the dataset was that data is neither homogenous in car instances nor consistent with respect to image size therefore would be requiring significant preprocessing and improvements.

During model development and run it was realized that under the existing computation ability between the group though image resizing would be possible but data augmentation would critically increase the computational requirement for the model runs and only subset of the data could be used for training. A tradeoff was designed where the models were trained with the existing biases but full dataset to design the model. The imbalance has lead to biases in training specifically for classification but has resulted in better segmentation performance in the models .

The next section provides visual details for model training and the key insights from the confusion matrix

Model training and impacts of data quality

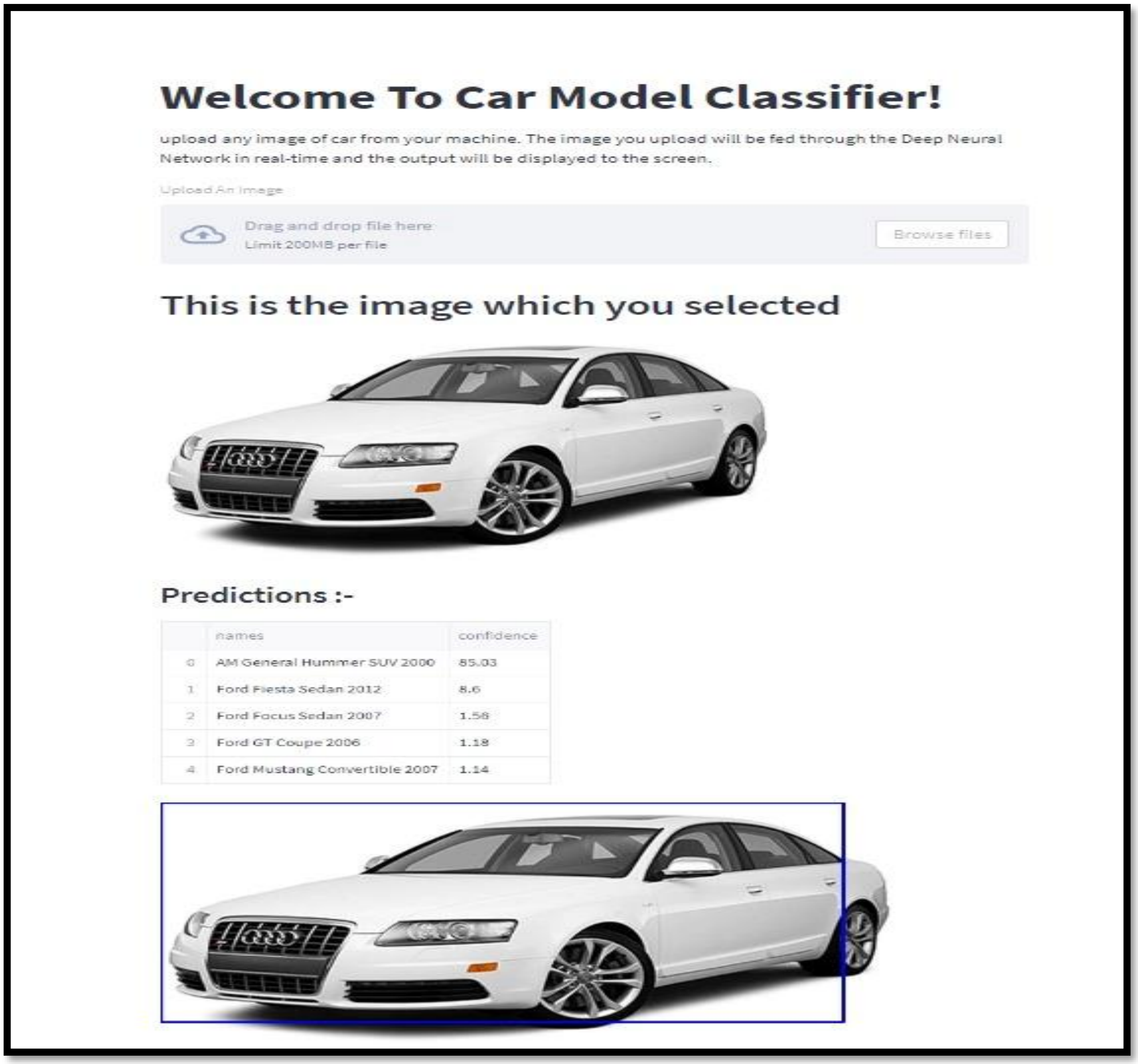
The visualization of the confusion matrix for the final hyper tunned ResNet50V2 is detailed below. We find that a highly biased data lead to compromised set of True positives with significant spikes for few categories of cars that show highest frequency of training data. The true negatives are much better represented in the dataset . As the model trains through the transfer learning models the ability of models to predict true positive improve on the validation set. The models trained are able to provide a good segmentation efficiency for the validation set but the classification efficiency during simultaneous prediction was limited to about 40% . However, an independent classification only model run provide an overall all classification efficiency of 70 % in validation set after data augmentation. Computational bandwidth raised a key constrain to model run and tunning .


TRUE NEGATIVES
FALSE POSITIVES
TRUE NEGATIVES
FALSE POSITIVES
FALSE NEGATIVES
TRUE POSITIVES
FALSE NEGATIVES
TRUE POSITIVES
TRAINING DATASET
TESTING DATASET

Model Deployment and GUI access

A simple interface illustrated as image below was created by the team which allows upload of image to be predicted along with bounding box

DEVELOPED GUI FOR THE MODEL



Welcome To Car Model Classifier!

upload any image of car from your machine. The image you upload will be fed through the Deep Neural Network in real-time and the output will be displayed to the screen.

This is the image which you selected

Predictions :-

	names	confidence
0	AM General Hummer SUV 2000	85.03
1	Ford Fiesta Sedan 2012	8.6
2	Ford Focus Sedan 2007	1.58
3	Ford GT Coupe 2006	1.18
4	Ford Mustang Convertible 2007	1.14

7.Implications

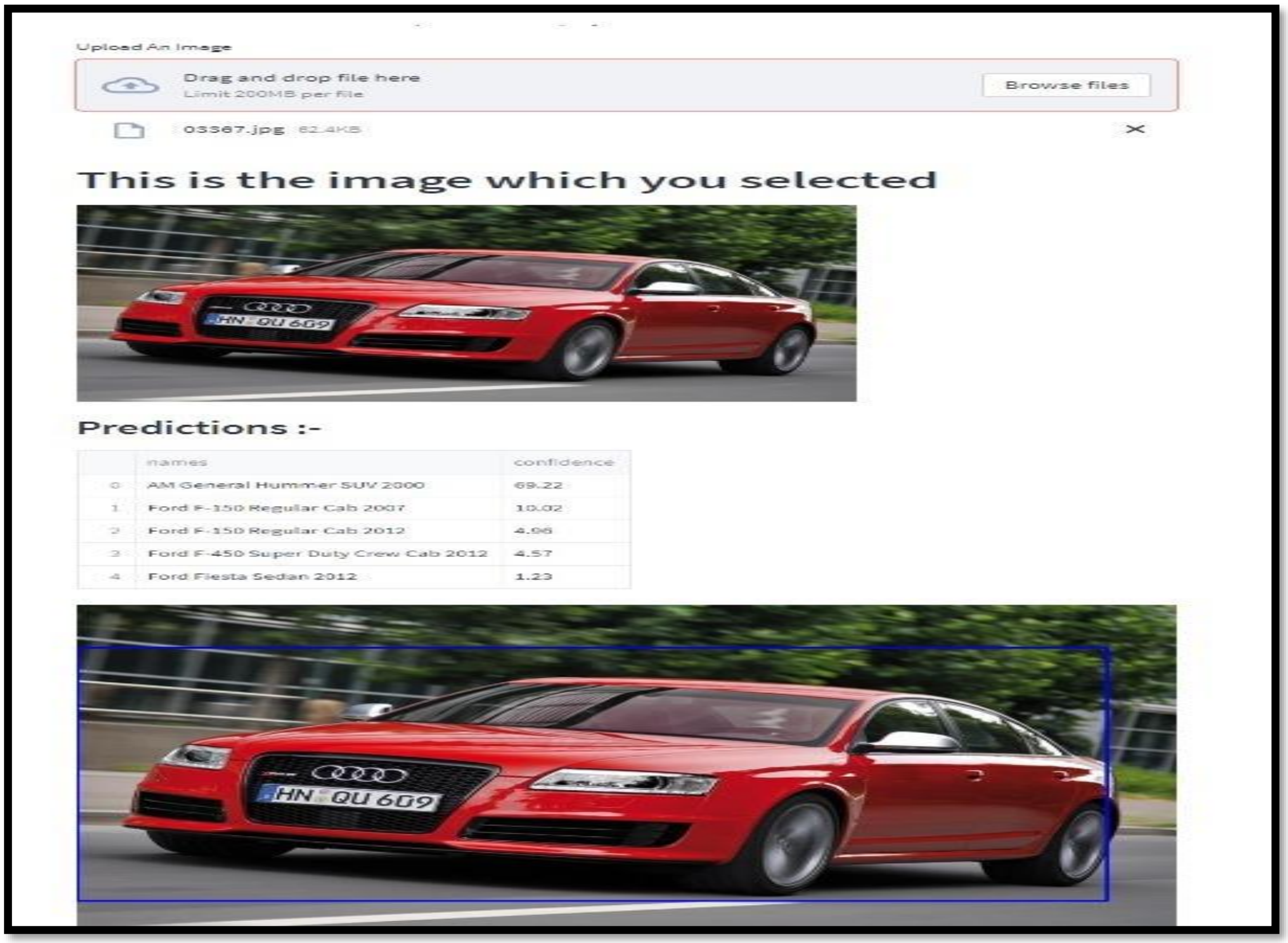
Effect of the solution on the problem in the domain or business?

We developed a multiclass object detection model for cars which enables identification of moving cars on the road by a camera. 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.

Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label together leading to object recognition.

The developed solution provides an ability for car detection on road conditions at a significant level of confidence of about 70% along with preliminary classification. The steps of classification and localization are simultaneously performed using a process of environment development which alleviated the computational constrains and provides a consistent model for prediction. Although there is ample scope of performance improvement the solution developed stands on its own in terms utility as the preliminary car detection model for surveillance purposes.

Further, the developed GUI provides an ability to detect a car from the picture and suitably classify it. As the training dataset had images of various dimensions and resolution the overall car detection ability of the model is good and reliable along with category prediction. The trained model is able to perform classification and localization simultaneously and provide an accessible output from the GUI developed.


Upload An Image

Drag and drop file here
Limit 200MB per file

03367.jpg 62.4KB

This is the image which you selected

Predictions :-

	names	confidence
0	AM General Hummer SUV 2000	69.22
1	Ford F-150 Regular Cab 2007	10.02
2	Ford F-150 Regular Cab 2012	4.98
3	Ford F-450 Super Duty Crew Cab 2012	4.57
4	Ford Fiesta Sedan 2012	1.23

Ability of the model to make recommendations

The model at the moment shows good car detection abilities of up to 70 % and is sufficiently generalized for real time applications for car detection.

The GUI developed provides an ability to access the train model and provide predictions for an image of interest. IOU of the validation set for the final model stands at 70%. The confidence level of classification stands at 38.3% and can be further improved by data augmentation. The model still provides a good starting point for surveillance purpose for vehicle detection and can be used as a preliminary model to scan cars on road as make, type, model and OEM.

8.Limitations

There is an array of limitations associated with the work.

1. One of the biggest limitations and the defining factor for the structure of this work has been the available computational capabilities for preprocessing, training and hype tunning the model
2. Image augmentation which was well recommended during EDA could not be performed for the hybrid model due lack of computational bandwidth for processing data and the risk of information loss while training with smaller dataset considering highly heterogenous image size, resolution and background
3. The model provides a segmentation efficiency of 70% thus detecting a car and its localization even after the hyperparameter tunning
4. Classification efficiency of the hybrid final model has been close to 40% for the hyper tuned model which needs to be further improved for better results.
5. The model has been only trained for car data so deployment for vehicle identification on roads would require further training with more generic vehicular data
6. Data augmentation was not performed so the models developed have inherent data biases which were evident while model predicted
7. During model development and run it was realized that under the existing computation ability between the group though image resizing would be possible but data augmentation would critically increase the computational requirement for the model runs and only subset of the data could be used for training. A tradeoff was designed where the models were trained with the existing biases but full dataset to design the model. The imbalance has led to biases in training specifically for classification but has resulted in better segmentation performance in the models.
8. As the model is only trained on the cars data deployment under real world condition for on road vehicle detection would not be possible and the model would need to be further trained for a range of vehicles on road
9. The use and exploration of other advanced model like YOLO and DETER which are expected to provide a much better results for simultaneous segmentation

and classification had to be aborted due to lack of time and varied input required which cannot be process in the existing pipeline.

10. The GUI developed at the moment do not provide ability to use videos for detection

Future Work

The project work has created a strong foundation for the team members to further enhance and develop expertise in deep learning-based object detection model which are progressively finding use in various sectors and walks of day-to-day life. Following few future extensions and enhancement of model development work have been delineated by the team

1. Working on improvement of classification efficiency of existing models
2. Exploring efficient ways of using state of art models like YOLO6, DETER for the problem statement
3. Exploring innovative ways for data augmentation to remove the existing biases from the dataset
4. Training more generic vehicle detection models for real time use on road
5. Exploring applications for night vision and driverless navigation
6. Creating associations with other research group for resource sharing and better access to high computing abilities for model training

9.Closing Reflections

This report delineates 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. The project was a unique opportunity to work as a team with various peers from different backgrounds and specialized skill and synthesize an industry standard deliverable within designated time frame. The exercise not only provided in-depth exposure into object detection through computer vision but also created space for innovation and exploration of various aspects of deep learning which is lot only restricted to complex model architecture but also input data analysis and preprocessing along with making the synthesized knowledge available in public domain through GUI development.

Learnings from the project

Learning from the project have been many

1. The project provided an in-depth exposure from end-to-end model design and deployment by the team
2. Hands on learning and exposure to project management applications while creating GitHub repository and click up account
3. An exposure to work on real life solution as a team which can be easily mimicked into a ML AI development team
4. The project provided enough scope for experimentation and designing novel methods and designs to work under resource constrained condition
5. Image classification -Bounding Bo combo generator was developed from scratch
6. Classification- localization merged in single model reducing overheads and prediction time
7. Multilevel confusion matrix visualized innovatively in a condensed plot
8. custom TensorFlow callbacks used for

   a) validation loss descends gradient-based learning rate scheduler

   b) transferred Native Model layers unfreeze for training after sufficient training of custom layers

c) epoch-epoch execution times monitored, creating scope for hyper tuning towards faster execution times

9. A modular programming mode was followed to leverage existing computing efficiency and executing parallel workflow
10. A GUI created for the model using streamlet and Heroku platform for the model deployment
11. All-encompassing pipeline implemented for the project

What could have been done differently

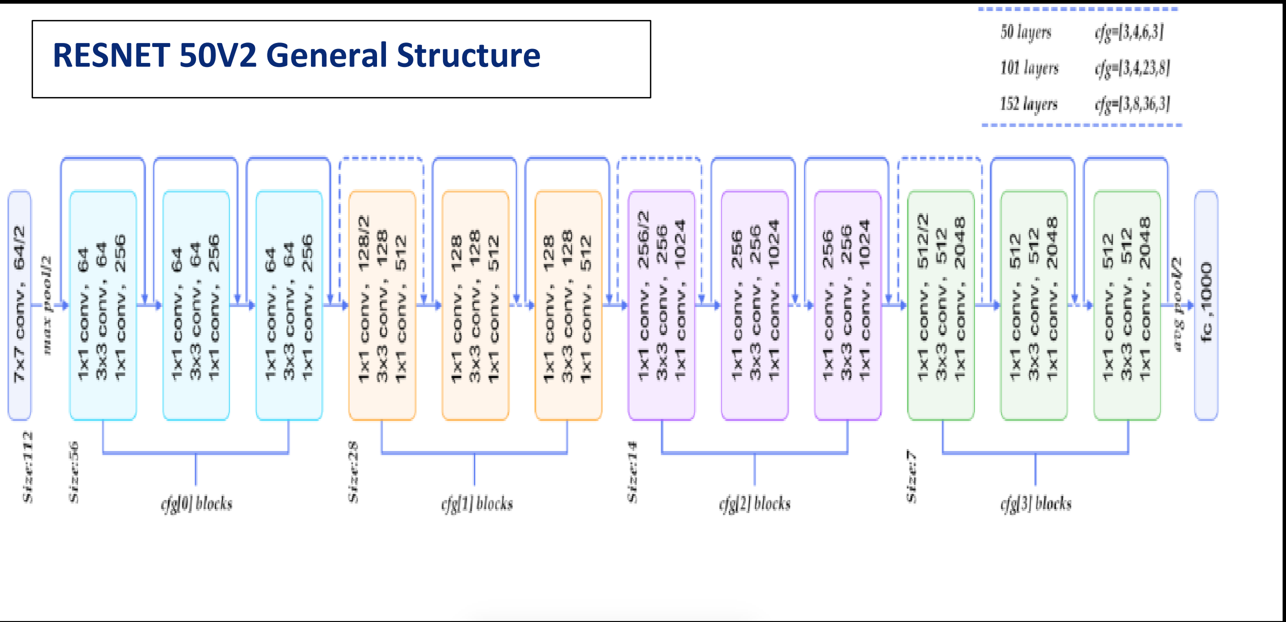
1. The decision of skipping data augmentation led to compromising of prediction accuracy of the model as is evident from the confusion matrix. Exploring other less computationally expensive methods for data augmentation would have helped in better model results

2. The analysis through other state of art models like newly released Yolo6 and DETER could not be done due to lack of time lesser combability with the created pipeline**.** A stand-alone analysis was attempted for YOLO V6 but could not be completed within the designated timeframe by the team. Greater focus of newer implementation in future could help getting better accuracy models with these new architecture

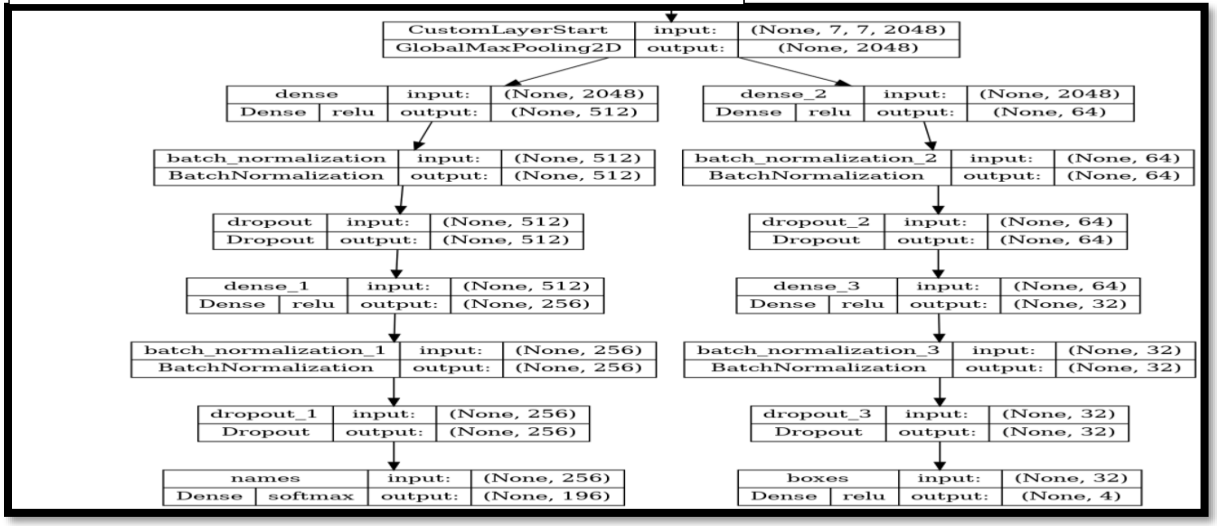
Appendix 1:

FINAL MODEL CONFIGURATION RESNET50V2

RESNET 50V2 General Structure



RESNET 50V2 CUSTOM TRAINED LAYER



CustomLayerStart | input: | (None, 7, 7, 2048)
GlobalMaxPooling2D | output: | (None, 2048)
dense | input: | (None, 2048)
Dense | relu | output: | (None, 512)
batch_normalization | input: | (None, 512)
BatchNormalization | output: | (None, 512)
dropout | input: | (None, 512)
Dropout | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | relu | output: | (None, 256)
batch_normalization_1 | input: | (None, 256)
BatchNormalization | output: | (None, 256)
dropout_1 | input: | (None, 256)
Dropout | output: | (None, 256)
names | input: | (None, 256)
Dense | softmax | output: | (None, 196)
dense_2 | input: | (None, 2048)
Dense | relu | output: | (None, 64)
batch_normalization_2 | input: | (None, 64)
BatchNormalization | output: | (None, 64)
dropout_2 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_3 | input: | (None, 64)
Dense | relu | output: | (None, 32)
batch_normalization_3 | input: | (None, 32)
BatchNormalization | output: | (None, 32)
dropout_3 | input: | (None, 32)
Dropout | output: | (None, 32)
boxes | input: | (None, 32)
Dense | relu | output: | (None, 4)