Computer Vision

OBJECT DETECTION - CAR

A Capstone Project

Made by AIML April Capstone Group 8 Members:

Goutam Kalita

Haribasker

Saif Merchant

Shanmugabharathy

Ruhee Seshadri

In fulfilment of the requirements for the award of

Post Graduate Program in

Artificial Intelligence & Machine Learning

from:

Great Learning

&

TEXAS McCombs

The University of Texas at Austin

Mentor: *Aniket Chhabra*

CONTENTS

Interim Report: Car Classification Computer Vision Project

Introduction :

Computer Vision, enhanced by Artificial Intelligence and Machine Learning, has ushered in a new era of innovation across various industries. This interdisciplinary field empowers machines to comprehend and interpret visual information much like human perception of images and videos. The integration of computer vision with AI and ML techniques has paved the way for automation, intelligent decision-making, and advanced analysis based on visual data.

In the dynamic realm of technology, computer vision, supported by Artificial Intelligence (AI) and Machine Learning (ML), has become a transformative force. This synergy enables machines to interpret and analyze visual data, making informed decisions akin to human perception. The significance of computer vision spans across diverse sectors, including healthcare, automotive, surveillance, and more. AIML projects in computer vision leverage algorithms to process visual information, opening new horizons in automation, data analysis, and intelligent decision-making. As we delve into this multidisciplinary field, we embark on a journey to explore the applications, methodologies, and outcomes of computer vision AIML projects, unravelling their potential in shaping the future of technology and innovation.

The project at hand focuses on car detection as one of its applications. The givens are listed below:

Problem Statement

Domain: Automotive Surveillance

Context:

In the realm of automotive surveillance, the implementation of computer vision technologies offers the opportunity to automate supervision and trigger appropriate actions based on image predictions. One such application is the identification of cars on the road, encompassing details like make, type, colour, and license plate information.

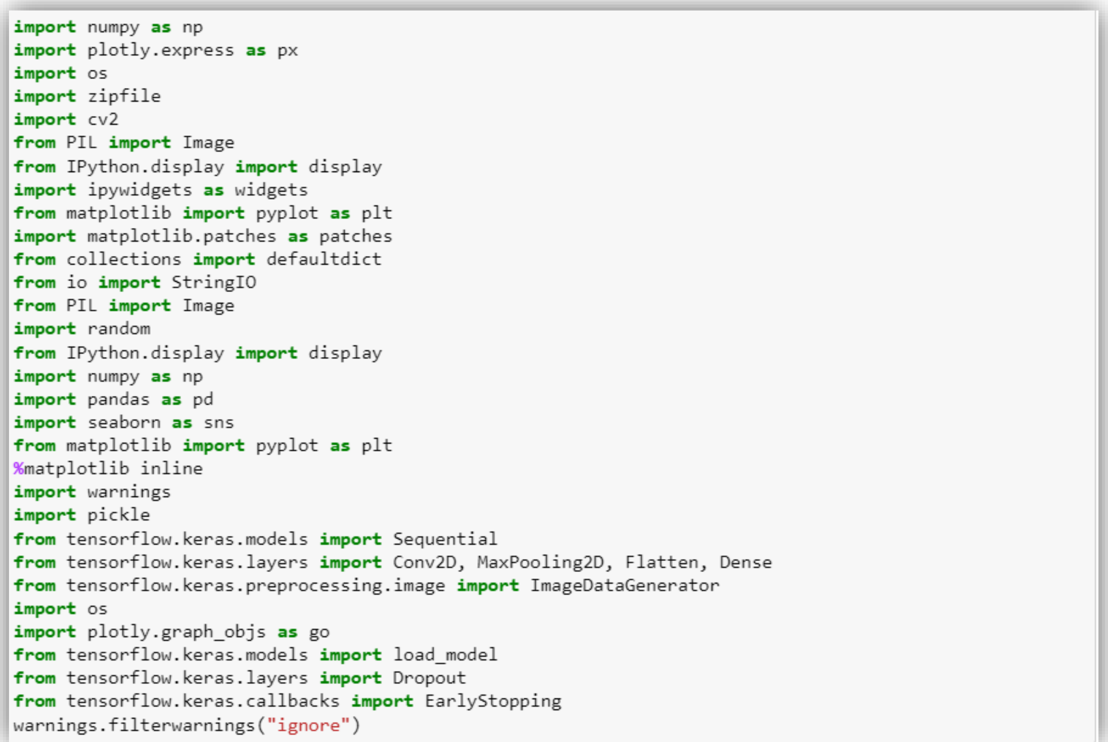
Data Description:

The Cars dataset comprises 16,185 images distributed across 196 classes of cars. The dataset is divided into 8,144 training images and 8,041 testing images, with each class having an approximately 50-50 split. Classifications are at a detailed level, including Make, Model, and Year (e.g., 2012 Tesla Model S or 2012 BMW M3 coupe).
Data Components:
Train Images: Real images of cars, categorized by make and year.
Test Images: Real images of cars, categorized by make and year.
Train Annotation: Bounding box regions for training images.
Test Annotation: Bounding box regions for testing images.

Dataset provided with this project; original link for reference: Stanford Car Dataset
Reference: 3D Object Representations for Fine-Grained Categorization, Jonathan Krause, Michael Stark, Jia Deng, Li Fei-Fei, 4th IEEE Workshop on 3D Representation and Recognition, ICCV 2013 (3dRR-13), Sydney, Australia, Dec. 8, 2013.

Step 1: Import the data:

In Step 1, the necessary libraries and tools have been imported to facilitate the subsequent processes in the car classification computer vision project. Notable libraries include NumPy, Plotly, OpenCV, Pillow, TensorFlow, and others, which are essential for data manipulation, visualization, and the implementation of deep learning models.
The inclusion of warning suppression ensures a clean runtime environment. The next steps will build upon this foundation to handle data, visualize images, and develop and train the initial Convolutional Neural Network (CNN) models for car classification.



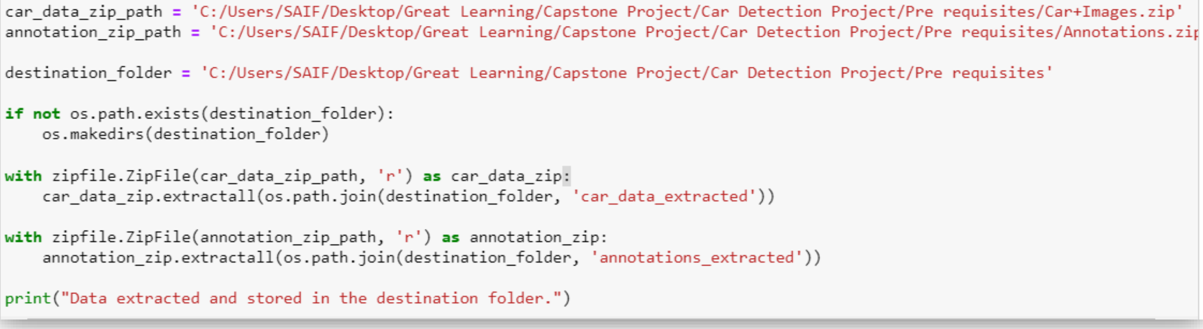
```
import numpy as np
import plotly.express as px
import os
import zipfile
import cv2
from PIL import Image
from IPython.display import display
import ipywidgets as widgets
from matplotlib import pyplot as plt
import matplotlib.patches as patches
from collections import defaultdict
from io import StringIO
from PIL import Image
import random
from IPython.display import display
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
import warnings
import pickle
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import plotly.graph_objs as go
from tensorflow.keras.models import load_model
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping
warnings.filterwarnings("ignore")
```

- **Plotly Express (import plotly.express as px):**
  - A library for interactive data visualization.
- **OS (import os):**
  - Provides a way of interacting with the operating system, essential for file and directory operations.
- **Zipfile (import zipfile):**
  - Enables the extraction and manipulation of zip files.
- **OpenCV (import cv2):**
  - Open Source Computer Vision Library, used for image processing tasks.
- **Pillow (from PIL import Image):**
  - A Python Imaging Library that adds image processing capabilities.
- **IPython Display (from IPython.display import display):**
  - Enables the display of rich media representations in the Jupyter Notebook.
- **ipywidgets (import ipywidgets as widgets):**
  - Interactive HTML widgets for Jupyter Notebooks.
- **Matplotlib (from matplotlib import pyplot as plt):**
  - A comprehensive library for creating static, animated, and interactive visualizations in Python.
- **Defaultdict (from collections import defaultdict):**
  - Provides a default value for nonexistent keys in a dictionary.
- **StringIO (from io import StringIO):**
  - Implements an in-memory file-like object for string data.
- **Random (import random):**
  - Generates pseudo-random numbers.
- **Pandas (import pandas as pd):**
  - A powerful data manipulation library.
- **Seaborn (import seaborn as sns):**
  - Built on top of Matplotlib, provides a high-level interface for drawing attractive and informative statistical graphics.
- **Warnings (import warnings):**
  - Allows the control of warning messages in Python.
- **Pickel (import pickle):**
  - Serializes and deserializes Python objects, facilitating the saving and loading of data.
- **TensorFlow Keras (from tensorflow.keras.models import Sequential):**
  - The high-level neural networks API, part of TensorFlow, used for building and training deep learning models.
- **TensorFlow Keras Layers (from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout):**
  - Specific layers required for building convolutional neural networks.
- **ImageDataGenerator (from tensorflow.keras.preprocessing.image import ImageDataGenerator):**
  - Generates augmented batches of images during the model training process.
- **Plotly Graph Objects (import plotly.graph_objs as go):**

  - Constructs graph objects for interactive visualizations using Plotly.
- **Load Model (from tensorflow.keras.models import load_model):**
  - Loads a pre-existing deep learning model.
- **Early Stopping (from tensorflow.keras.callbacks import EarlyStopping):**
  - Implements early stopping during model training based on a specified criterion.

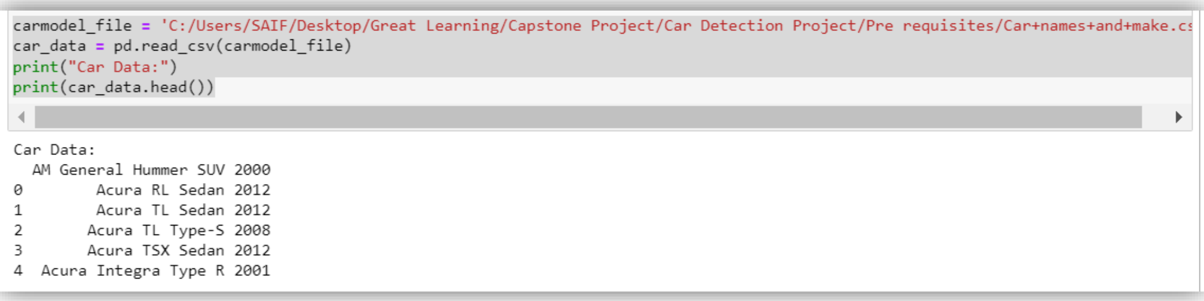
Step 1.2: Data Extraction

In Step 1. 2, the code handles the extraction of data from two zip files ('Car+Images.zip' and 'Annotations.zip'). It ensures the creation of a designated destination folder and extracts the contents of each zip file into separate subfolders within the destination. This step prepares the data for further processing and analysis in the car classification computer vision project.



```
car_data_zip_path = 'C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/Car+Images.zip'
annotation_zip_path = 'C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/Annotations.zip

destination_folder = 'C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites'

if not os.path.exists(destination_folder):
    os.makedirs(destination_folder)

with zipfile.ZipFile(car_data_zip_path, 'r') as car_data_zip:
    car_data_zip.extractall(os.path.join(destination_folder, 'car_data_extracted'))

with zipfile.ZipFile(annotation_zip_path, 'r') as annotation_zip:
    annotation_zip.extractall(os.path.join(destination_folder, 'annotations_extracted'))

print("Data extracted and stored in the destination folder.")
```

Step 1. 3: Load Car Model Data

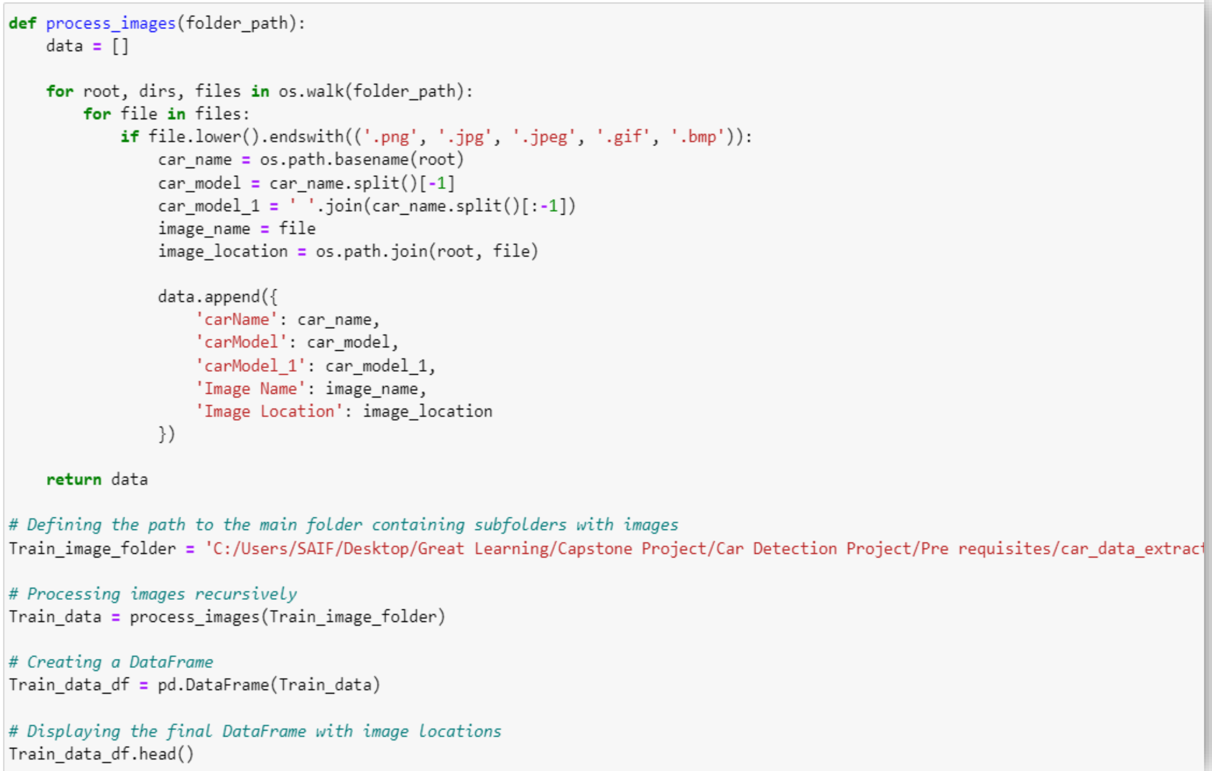


```
carmodel_file = 'C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/Car+names+and+make.cs
car_data = pd.read_csv(carmodel_file)
print("Car Data:")
print(car_data.head())
```

```
Car Data:
  AM General Hummer SUV 2000
0         Acura RL Sedan 2012
1         Acura TL Sedan 2012
2        Acura TL Type-S 2008
3        Acura TSX Sedan 2012
4  Acura Integra Type R 2001
```

In Step 1.3, the code reads and loads car model data from the 'Car+names+and+make.csv' file using the Pandas library. The loaded data is then displayed to verify its structure and content. This step is crucial for understanding the available car model information, which will be used in subsequent stages of the car classification computer vision project.

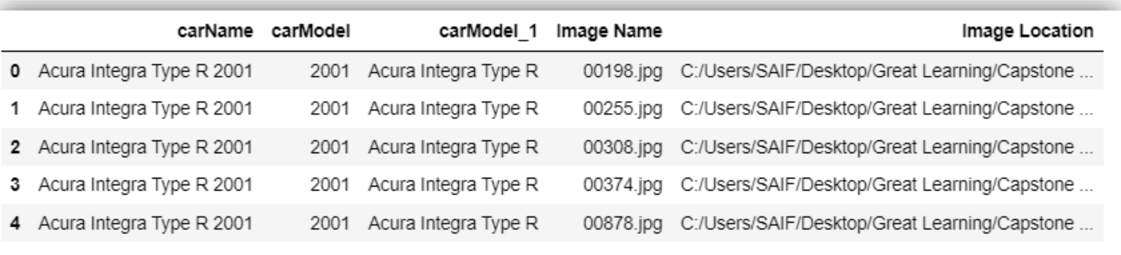
Step 2: Process Images and Create Data Frame:



```python
def process_images(folder_path):
    data = []

    for root, dirs, files in os.walk(folder_path):
        for file in files:
            if file.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp')):
                car_name = os.path.basename(root)
                car_model = car_name.split()[-1]
                car_model_1 = ' '.join(car_name.split()[:-1])
                image_name = file
                image_location = os.path.join(root, file)

                data.append({
                    'carName': car_name,
                    'carModel': car_model,
                    'carModel_1': car_model_1,
                    'Image Name': image_name,
                    'Image Location': image_location
                })

    return data

# Defining the path to the main folder containing subfolders with images
Train_image_folder = 'C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/car_data_extract

# Processing images recursively
Train_data = process_images(Train_image_folder)

# Creating a DataFrame
Train_data_df = pd.DataFrame(Train_data)

# Displaying the final DataFrame with image locations
Train_data_df.head()
```

In Step 2, images from the training data folder are processed using the **process_images** function, extracting relevant information about each image. This information is then organized into a DataFrame (**Train_data_df**) for better representation and analysis. The displayed DataFrame provides an initial overview of the processed data.

Result:

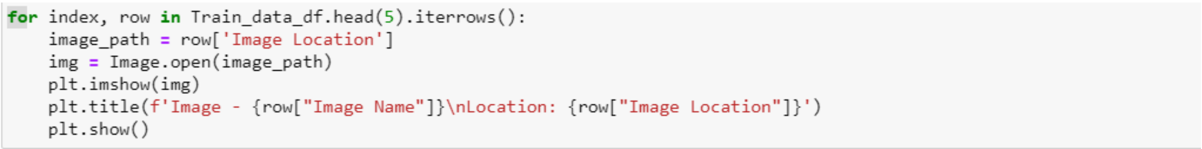


	carName	carModel	carModel_1	Image Name	Image Location
0	Acura Integra Type R 2001	2001	Acura Integra Type R	00198.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...
1	Acura Integra Type R 2001	2001	Acura Integra Type R	00255.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...
2	Acura Integra Type R 2001	2001	Acura Integra Type R	00308.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...
3	Acura Integra Type R 2001	2001	Acura Integra Type R	00374.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...
4	Acura Integra Type R 2001	2001	Acura Integra Type R	00878.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...

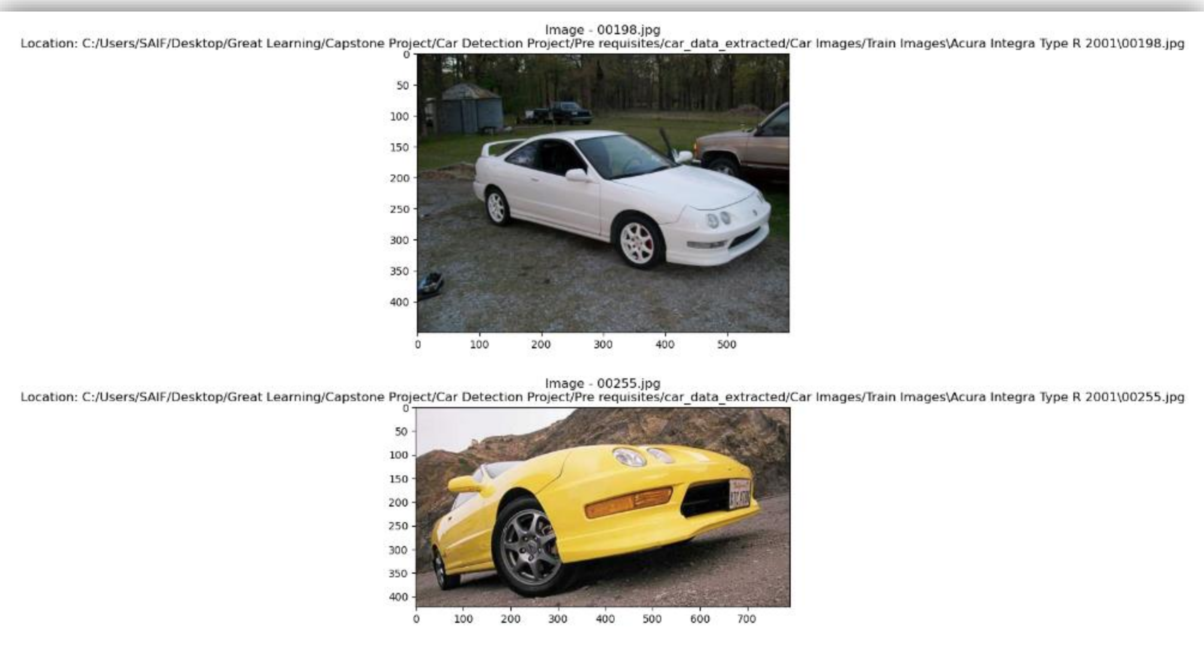
Let’s visualize few images and perform EDA:

This step involves a loop that iterates through the first five rows of the **Train_data_df and Test_data_df** DataFrame and displays some images along with relevant information. This step allows for visual inspection of the training data images, providing a qualitative understanding of the dataset.

A: Train Images



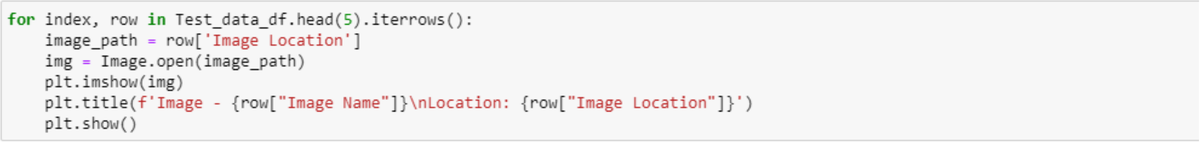
```
for index, row in Train_data_df.head(5).iterrows():
    image_path = row['Image Location']
    img = Image.open(image_path)
    plt.imshow(img)
    plt.title(f'Image - {row["Image Name"]}\nLocation: {row["Image Location"]}')
    plt.show()
```



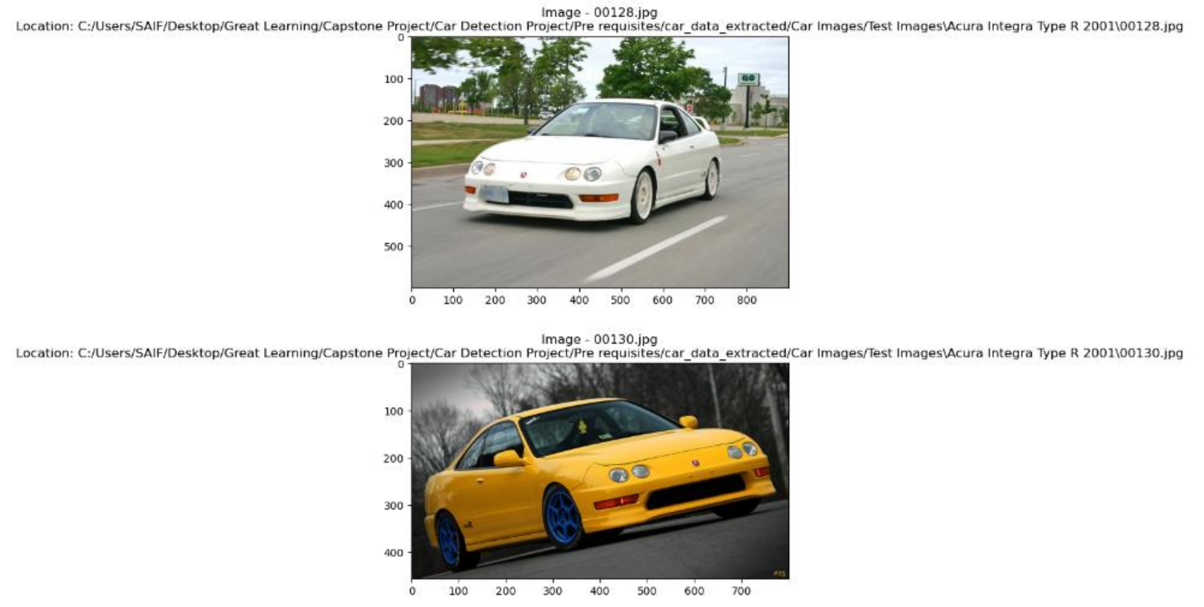
```
Image - 00198.jpg
Location: C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/car_data_extracted/Car Images/Train Images\Acura Integra Type R 2001\00198.jpg
```

```
Image - 00255.jpg
Location: C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/car_data_extracted/Car Images/Train Images\Acura Integra Type R 2001\00255.jpg
```

B: Test Images



```
for index, row in Test_data_df.head(5).iterrows():
    image_path = row['Image Location']
    img = Image.open(image_path)
    plt.imshow(img)
    plt.title(f'Image - {row["Image Name"]}\nLocation: {row["Image Location"]}')
    plt.show()
```

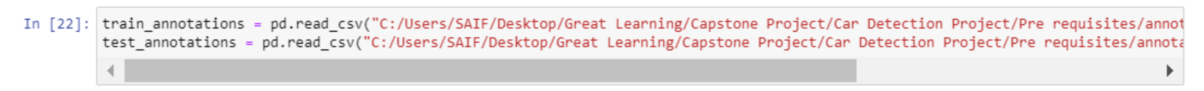


```
Image - 00128.jpg
Location: C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/car_data_extracted/Car Images/Test Images\Acura Integra Type R 2001\00128.jpg
```

```
Image - 00130.jpg
Location: C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/car_data_extracted/Car Images/Test Images\Acura Integra Type R 2001\00130.jpg
```

Step 3 - Map training and testing images to its annotations:

In this step , annotations for the training data are loaded and processed. The column names are renamed for consistency and clarity, preparing the annotations for further use in the car classification computer vision project.



```
In [22]: train_annotations = pd.read_csv("C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/annot
         test_annotations = pd.read_csv("C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/annota
```

Before renaming :

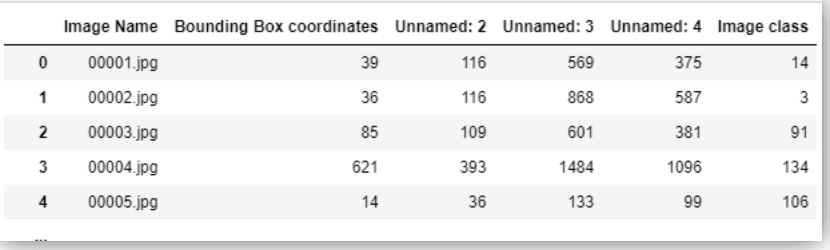
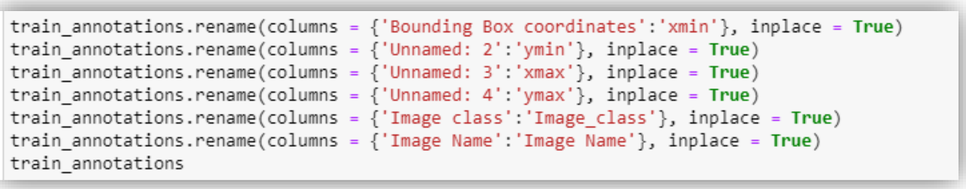


	Image Name	Bounding Box coordinates	Unnamed: 2	Unnamed: 3	Unnamed: 4	Image class
0	00001.jpg	39	116	569	375	14
1	00002.jpg	36	116	868	587	3
2	00003.jpg	85	109	601	381	91
3	00004.jpg	621	393	1484	1096	134
4	00005.jpg	14	36	133	99	106
...						



```
train_annotations.rename(columns = {'Bounding Box coordinates':'xmin'}, inplace = True)
train_annotations.rename(columns = {'Unnamed: 2':'ymin'}, inplace = True)
train_annotations.rename(columns = {'Unnamed: 3':'xmax'}, inplace = True)
train_annotations.rename(columns = {'Unnamed: 4':'ymax'}, inplace = True)
train_annotations.rename(columns = {'Image class':'Image_class'}, inplace = True)
train_annotations.rename(columns = {'Image Name':'Image Name'}, inplace = True)
train_annotations
```

After renaming:

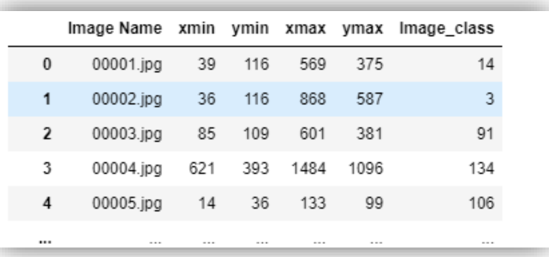
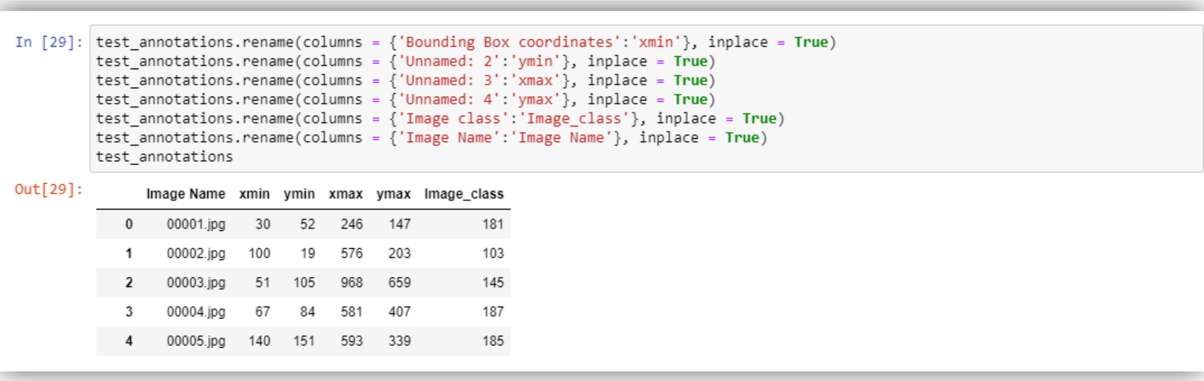


	Image Name	xmin	ymin	xmax	ymax	Image_class
0	00001.jpg	39	116	569	375	14
1	00002.jpg	36	116	868	587	3
2	00003.jpg	85	109	601	381	91
3	00004.jpg	621	393	1484	1096	134
4	00005.jpg	14	36	133	99	106
...	...	...	...	...	...	...

Similarly for Test data also:



```
In [29]: test_annotations.rename(columns = {'Bounding Box coordinates':'xmin'}, inplace = True)
         test_annotations.rename(columns = {'Unnamed: 2':'ymin'}, inplace = True)
         test_annotations.rename(columns = {'Unnamed: 3':'xmax'}, inplace = True)
         test_annotations.rename(columns = {'Unnamed: 4':'ymax'}, inplace = True)
         test_annotations.rename(columns = {'Image class':'Image_class'}, inplace = True)
         test_annotations.rename(columns = {'Image Name':'Image Name'}, inplace = True)
         test_annotations
```

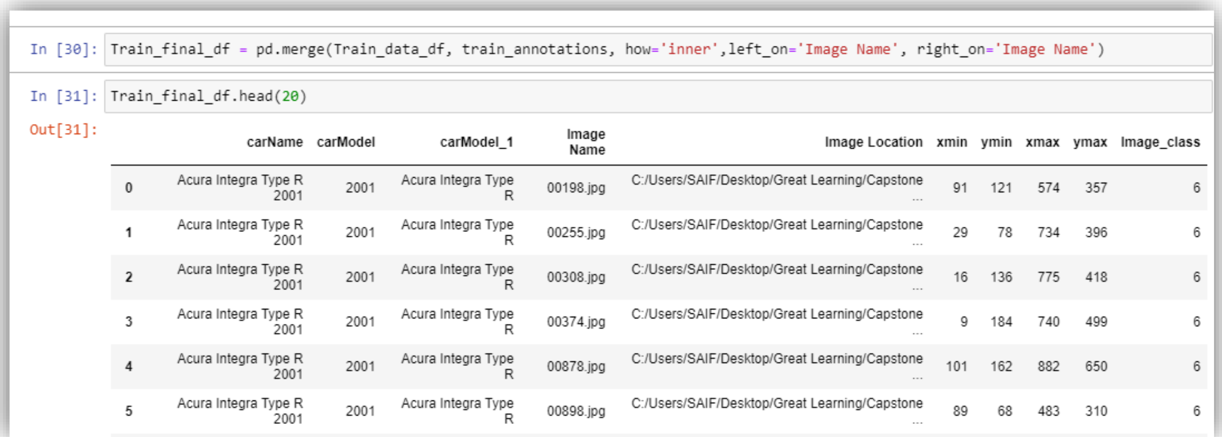
Out[29]:

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

Now, Let’s merge the DataFrame of annotations and image DataFrame for the train data:

The training and testing data DataFrames are merged with their respective annotations DataFrames. This step consolidates the information from both sources, aligning the image data with bounding box annotations. The resulting **Train_final_df** and **Test_final_df** DataFrames are essential for training and evaluating the car classification model.

TRAIN_DATAFRAME:



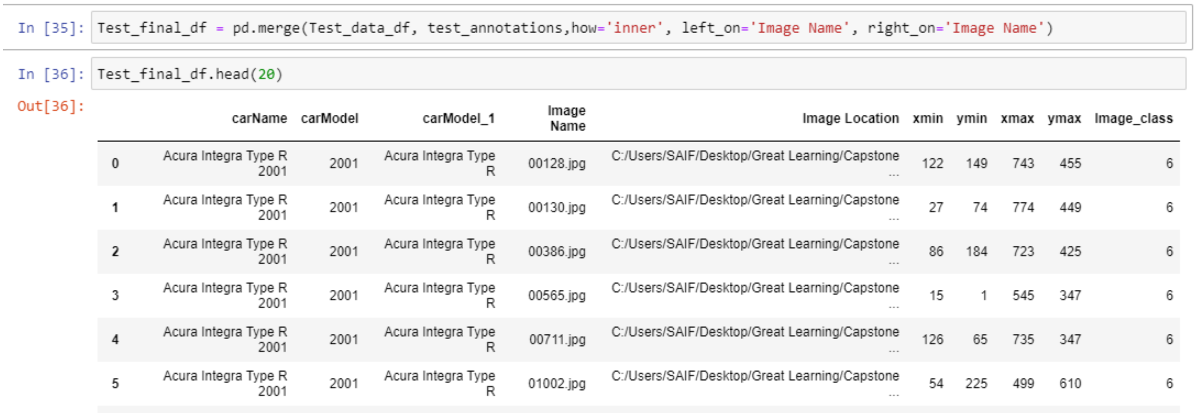
```
In [30]: Train_final_df = pd.merge(Train_data_df, train_annotations, how='inner',left_on='Image Name', right_on='Image Name')
```

```
In [31]: Train_final_df.head(20)
```

Out[31]:

	carName	carModel	carModel_1	Image Name	Image Location	xmin	ymin	xmax	ymax	Image_class
0	Acura Integra Type R 2001	2001	Acura Integra Type R	00198.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	91	121	574	357	6
1	Acura Integra Type R 2001	2001	Acura Integra Type R	00255.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	29	78	734	396	6
2	Acura Integra Type R 2001	2001	Acura Integra Type R	00308.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	16	136	775	418	6
3	Acura Integra Type R 2001	2001	Acura Integra Type R	00374.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	9	184	740	499	6
4	Acura Integra Type R 2001	2001	Acura Integra Type R	00878.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	101	162	882	650	6
5	Acura Integra Type R 2001	2001	Acura Integra Type R	00898.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	89	68	483	310	6

TEST_DATAFRAME:



```
In [35]: Test_final_df = pd.merge(Test_data_df, test_annotations,how='inner', left_on='Image Name', right_on='Image Name')
```

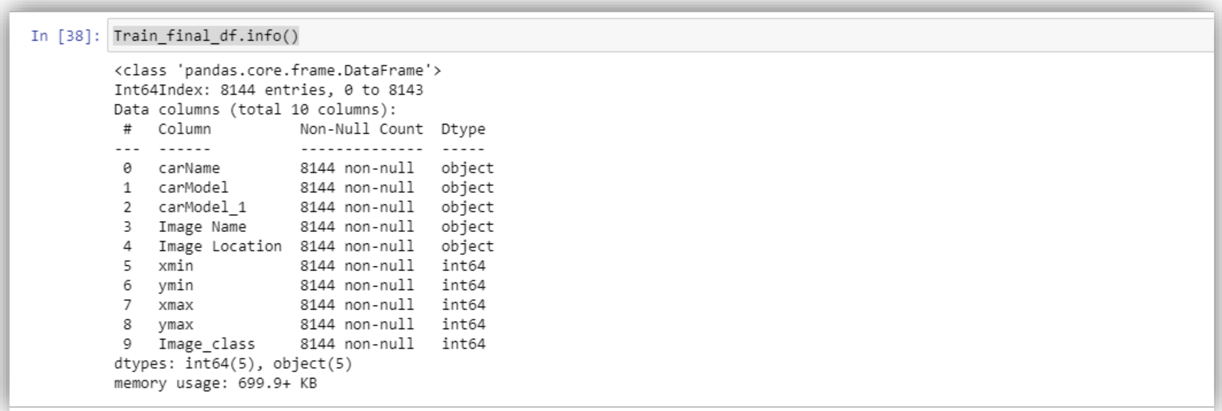
```
In [36]: Test_final_df.head(20)
```

Out[36]:

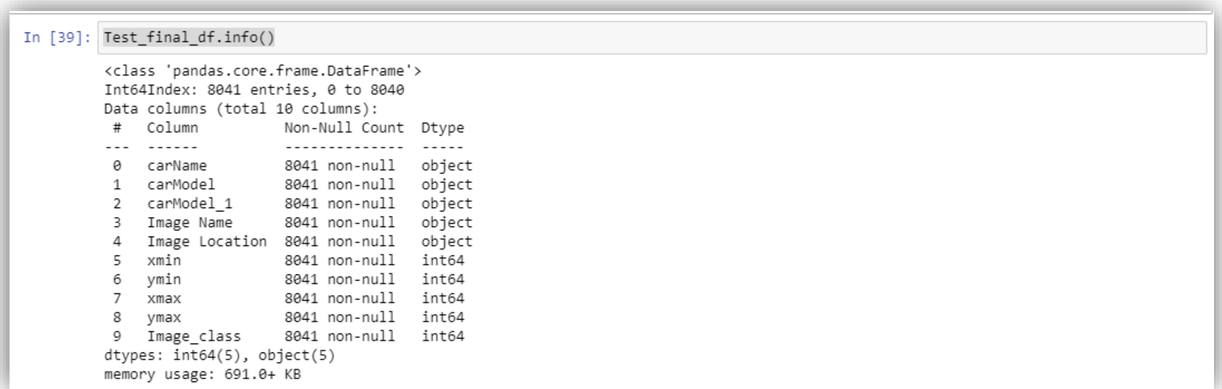
	carName	carModel	carModel_1	Image Name	Image Location	xmin	ymin	xmax	ymax	Image_class
0	Acura Integra Type R 2001	2001	Acura Integra Type R	00128.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	122	149	743	455	6
1	Acura Integra Type R 2001	2001	Acura Integra Type R	00130.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	27	74	774	449	6
2	Acura Integra Type R 2001	2001	Acura Integra Type R	00386.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	86	184	723	425	6
3	Acura Integra Type R 2001	2001	Acura Integra Type R	00565.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	15	1	545	347	6
4	Acura Integra Type R 2001	2001	Acura Integra Type R	00711.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	126	65	735	347	6
5	Acura Integra Type R 2001	2001	Acura Integra Type R	01002.jpg	C:/Users/SAIF/Desktop/Great Learning/Capstone ...	54	225	499	610	6

Let’s have a look at the sanity of the data and take appropriate steps if necessary:

In this step we performed data inspection and analyzeed the class distribution in the training and testing datasets. Information about the structure of the DataFrames is displayed, unique car models are identified, and class percentages are calculated and presented. This step helps in understanding the dataset characteristics and class distribution, which is crucial for model training and evaluation in the car classification computer vision project.



```
In [38]: Train_final_df.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 8144 entries, 0 to 8143
         Data columns (total 10 columns):
          #   Column          Non-Null Count  Dtype 
         ---  ------          --------------  ----- 
          0   carName         8144 non-null   object
          1   carModel        8144 non-null   object
          2   carModel_1      8144 non-null   object
          3   Image Name      8144 non-null   object
          4   Image Location  8144 non-null   object
          5   xmin            8144 non-null   int64 
          6   ymin            8144 non-null   int64 
          7   xmax            8144 non-null   int64 
          8   ymax            8144 non-null   int64 
          9   Image_class     8144 non-null   int64 
         dtypes: int64(5), object(5)
         memory usage: 699.9+ KB
```



```
In [39]: Test_final_df.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 8041 entries, 0 to 8040
         Data columns (total 10 columns):
          #   Column          Non-Null Count  Dtype 
         ---  ------          --------------  ----- 
          0   carName         8041 non-null   object
          1   carModel        8041 non-null   object
          2   carModel_1      8041 non-null   object
          3   Image Name      8041 non-null   object
          4   Image Location  8041 non-null   object
          5   xmin            8041 non-null   int64 
          6   ymin            8041 non-null   int64 
          7   xmax            8041 non-null   int64 
          8   ymax            8041 non-null   int64 
          9   Image_class     8041 non-null   int64 
         dtypes: int64(5), object(5)
         memory usage: 691.0+ KB
```

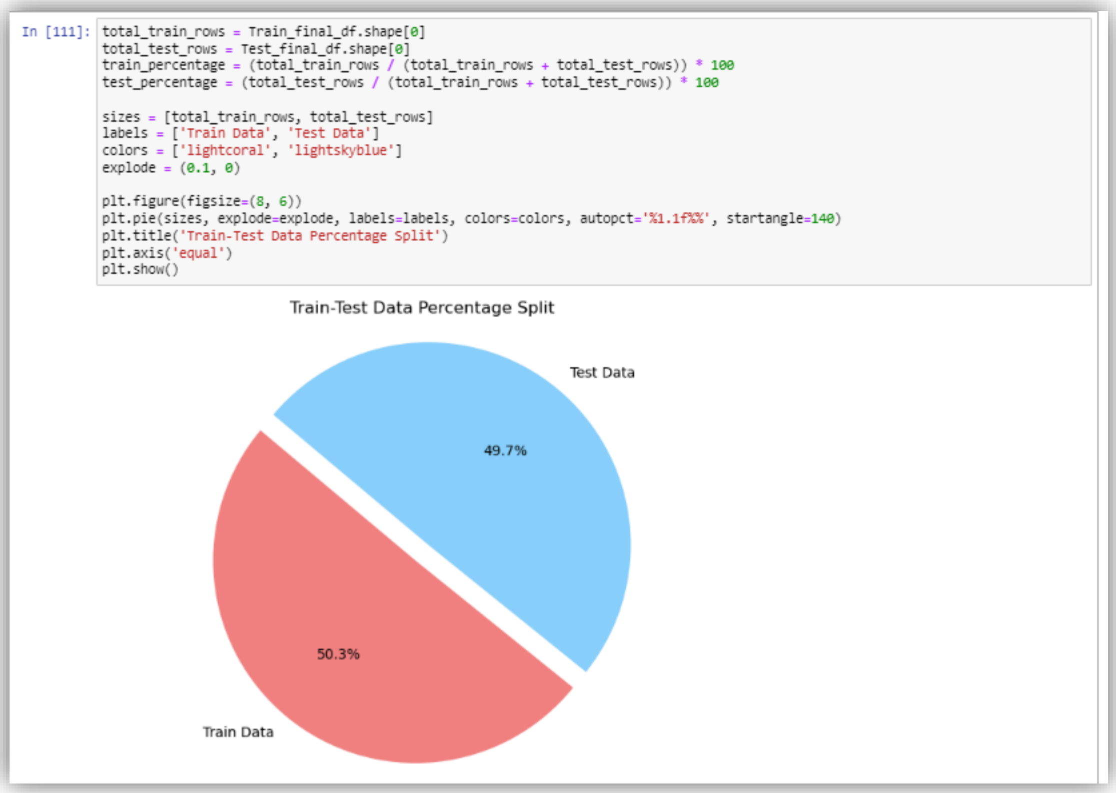
Train Data Class Percentages:	
119	0.834971
79	0.601670
167	0.589391
161	0.589391
144	0.577112
	...
175	0.380648
64	0.368369
158	0.356090
99	0.343811
136	0.294695

Test Data Class Percentages:	
119	0.845666
161	0.596941
79	0.596941
167	0.584504
43	0.572068
	...
175	0.373088
158	0.360652
64	0.360652
99	0.335779
136	0.298470

Let us explore the data by plotting few visualizations and performing EDA:

1 : Test data VS Train Data:

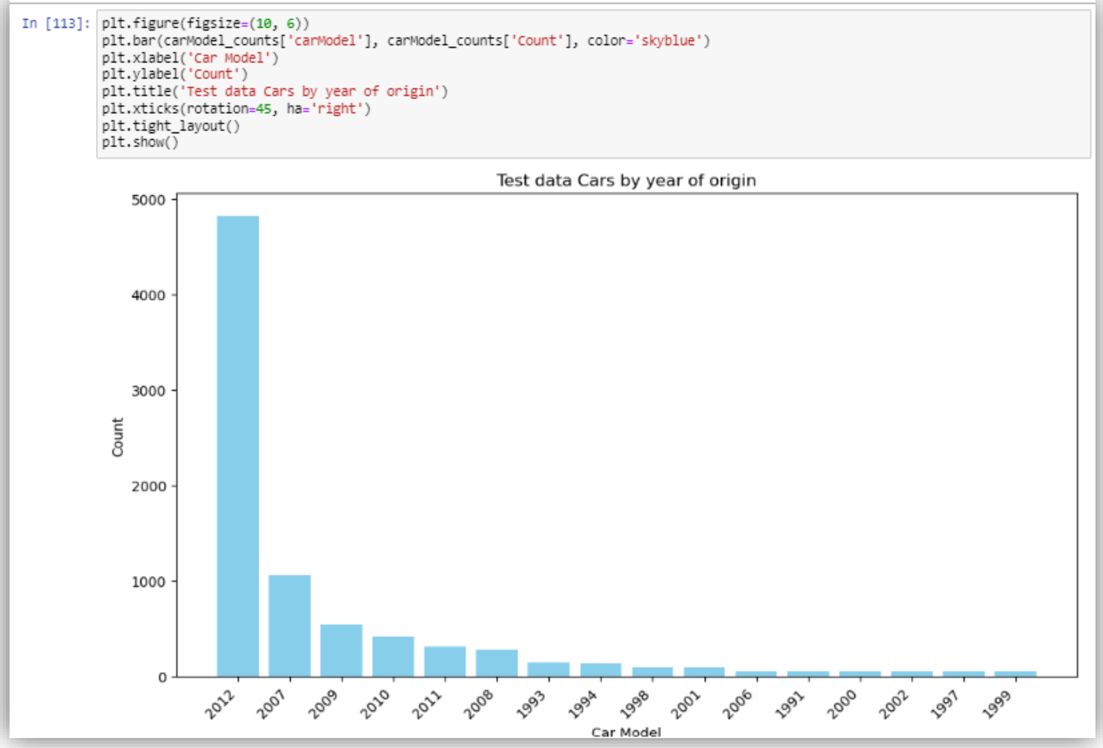
The dataset has been split into training and testing sets with approximately 49.7% allocated to the training data and 50.3% to the testing data. This balanced split ensures a representative distribution of data for both training and evaluation, enhancing the model's generalization capabilities.



```
In [111]: total_train_rows = Train_final_df.shape[0]
          total_test_rows = Test_final_df.shape[0]
          train_percentage = (total_train_rows / (total_train_rows + total_test_rows)) * 100
          test_percentage = (total_test_rows / (total_train_rows + total_test_rows)) * 100

          sizes = [total_train_rows, total_test_rows]
          labels = ['Train Data', 'Test Data']
          colors = ['lightcoral', 'lightskyblue']
          explode = (0.1, 0)

          plt.figure(figsize=(8, 6))
          plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
          plt.title('Train-Test Data Percentage Split')
          plt.axis('equal')
          plt.show()
```

2 : Cars by year of their origin:

The code calculates and visualizes the distribution of cars in the training data based on the year of their origin. The bar chart provides insights into the representation of different car models, contributing to the understanding of the dataset's characteristics in the car classification computer vision project.

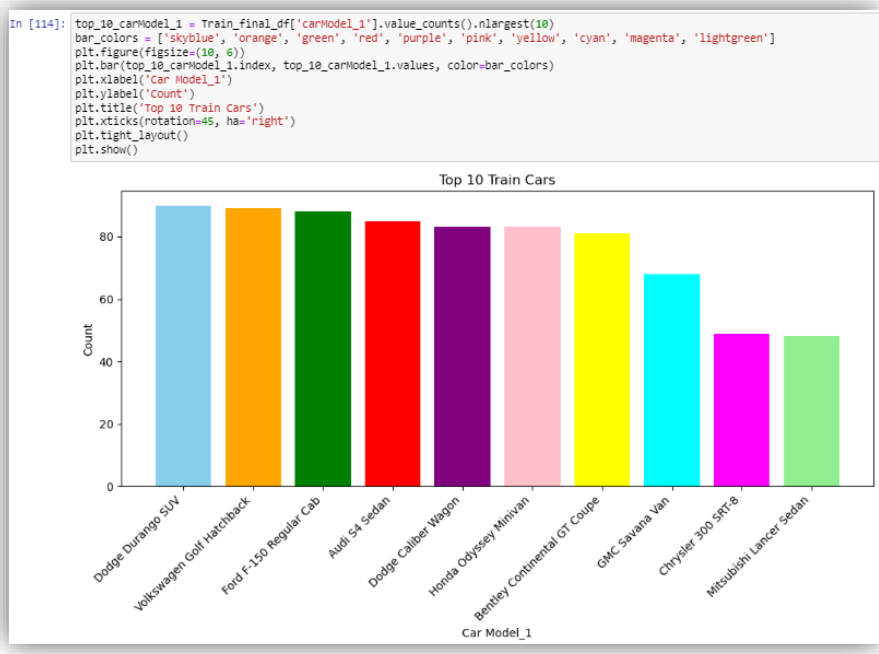


```
In [113]: plt.figure(figsize=(10, 6))
          plt.bar(carModel_counts['carModel'], carModel_counts['Count'], color='skyblue')
          plt.xlabel('Car Model')
          plt.ylabel('Count')
          plt.title('Test data Cars by year of origin')
          plt.xticks(rotation=45, ha='right')
          plt.tight_layout()
          plt.show()
```

3. Count of top 10

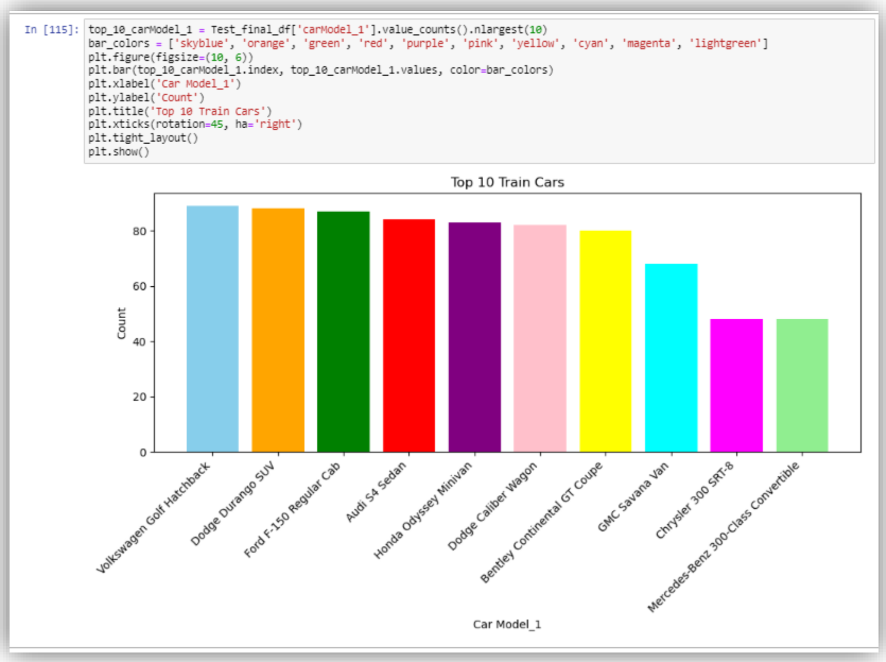
In this step we calculates and visualizes the top 10 car models in the training data based on the 'carModel_1' column. The bar chart offers a clear representation of the most prevalent car models, aiding in understanding the dataset's composition and emphasizing the importance of specific car models in the car classification computer vision project.

Train dataframe:



```
In [114]: top_10_carModel_1 = Train_final_df['carModel_1'].value_counts().nlargest(10)
          bar_colors = ['skyblue', 'orange', 'green', 'red', 'purple', 'pink', 'yellow', 'cyan', 'magenta', 'lightgreen']
          plt.figure(figsize=(10, 6))
          plt.bar(top_10_carModel_1.index, top_10_carModel_1.values, color=bar_colors)
          plt.xlabel('Car Model_1')
          plt.ylabel('Count')
          plt.title('Top 10 Train Cars')
          plt.xticks(rotation=45, ha='right')
          plt.tight_layout()
          plt.show()
```

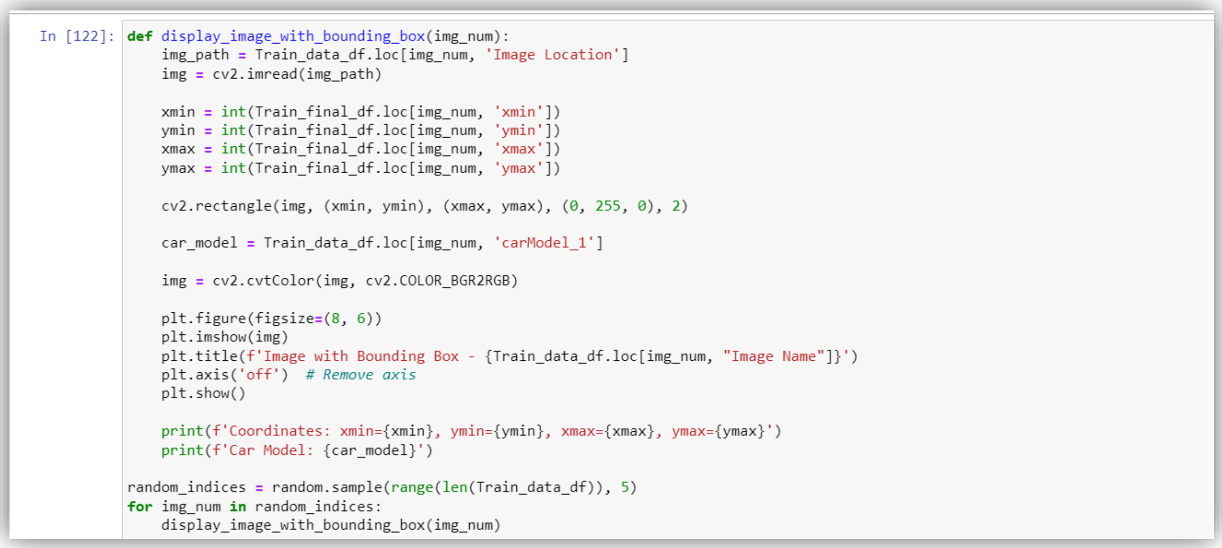
Test dataframe:



```
In [115]: top_10_carModel_1 = Test_final_df['carModel_1'].value_counts().nlargest(10)
          bar_colors = ['skyblue', 'orange', 'green', 'red', 'purple', 'pink', 'yellow', 'cyan', 'magenta', 'lightgreen']
          plt.figure(figsize=(10, 6))
          plt.bar(top_10_carModel_1.index, top_10_carModel_1.values, color=bar_colors)
          plt.xlabel('Car Model_1')
          plt.ylabel('Count')
          plt.title('Top 10 Train Cars')
          plt.xticks(rotation=45, ha='right')
          plt.tight_layout()
          plt.show()
```

Step 4 : Display images with bounding box:

In this phase, we successfully implemented the display of images with bounding boxes, providing a visual understanding of our training dataset. The displayed images showcase the integration of bounding box coordinates, allowing us to observe the localization of cars within the images.



```
In [122]: def display_image_with_bounding_box(img_num):
              img_path = Train_data_df.loc[img_num, 'Image Location']
              img = cv2.imread(img_path)

              xmin = int(Train_final_df.loc[img_num, 'xmin'])
              ymin = int(Train_final_df.loc[img_num, 'ymin'])
              xmax = int(Train_final_df.loc[img_num, 'xmax'])
              ymax = int(Train_final_df.loc[img_num, 'ymax'])

              cv2.rectangle(img, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)

              car_model = Train_data_df.loc[img_num, 'carModel_1']

              img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

              plt.figure(figsize=(8, 6))
              plt.imshow(img)
              plt.title(f'Image with Bounding Box - {Train_data_df.loc[img_num, "Image Name"]}')
              plt.axis('off')  # Remove axis
              plt.show()

              print(f'Coordinates: xmin={xmin}, ymin={ymin}, xmax={xmax}, ymax={ymax}')
              print(f'Car Model: {car_model}')

          random_indices = random.sample(range(len(Train_data_df)), 5)
          for img_num in random_indices:
              display_image_with_bounding_box(img_num)
```

For example, consider the randomly selected images below:

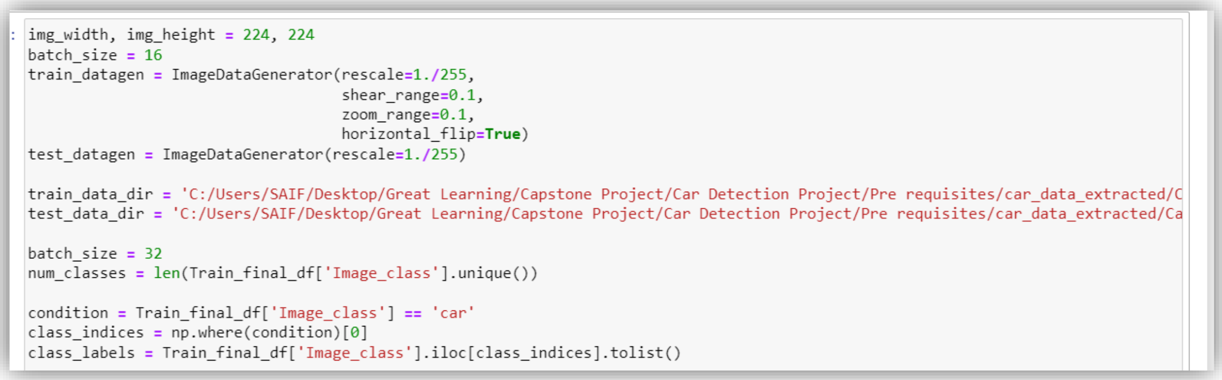
Image with Bounding Box - 04369.jpg
HUMMER
MICHIGAN
BDV 3700

Image with Bounding Box - 06135.jpg

Image with Bounding Box - 06920.jpg
© izmocars

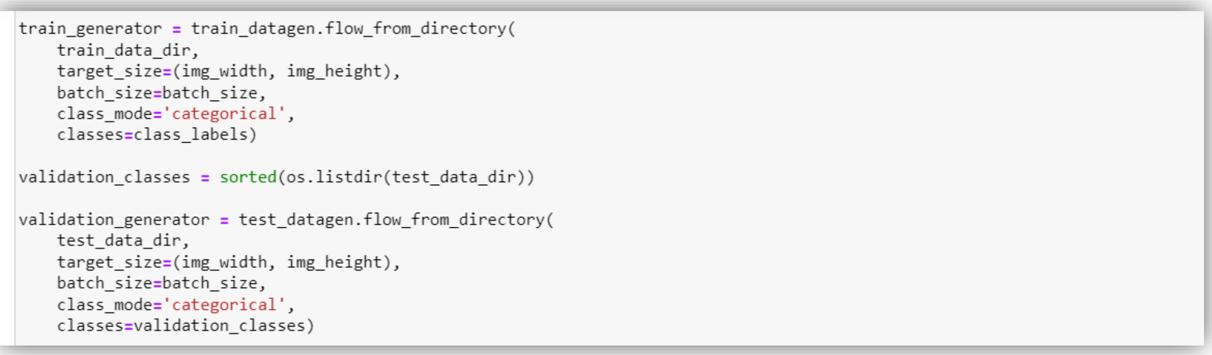
Step 5. Design, train and test basic CNN models to classify the car:

- In this step, we designed, trained, and tested a basic Convolutional Neural Network (CNN) model to classify cars in the images



```
img_width, img_height = 224, 224
batch_size = 16
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

train_data_dir = 'C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/car_data_extracted/C
test_data_dir = 'C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Pre requisites/car_data_extracted/Ca

batch_size = 32
num_classes = len(Train_final_df['Image_class'].unique())

condition = Train_final_df['Image_class'] == 'car'
class_indices = np.where(condition)[0]
class_labels = Train_final_df['Image_class'].iloc[class_indices].tolist()
```

- We've tailored our Convolutional Neural Network (CNN) for image dimensions of 224x224 pixels. Employing rescaling, shear adjustments, zooming, and horizontal flipping through data augmentation techniques, we've enriched our training dataset. This augmentation ensures the model's adaptability to diverse viewpoints.
- Our training data is primed for assimilation, while the testing dataset awaits evaluation. During training, a batch size of 32 is utilized, and the model is poised to recognize a variable number of classes based on the unique image classifications, with a particular emphasis on the 'car' class.
- This meticulous setup forms the foundation upon which our CNN will develop its understanding of car images during the upcoming training phase.



```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    classes=class_labels)

validation_classes = sorted(os.listdir(test_data_dir))

validation_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    classes=validation_classes)
```

- We've established data generators to seamlessly flow our augmented images into the CNN model during training and validation. The **train_generator** is configured to read images from the training dataset directory, with a target size of 224x224 pixels. It operates with a batch size of 32, employing categorical class mode, and recognizes classes based on our meticulous organization.

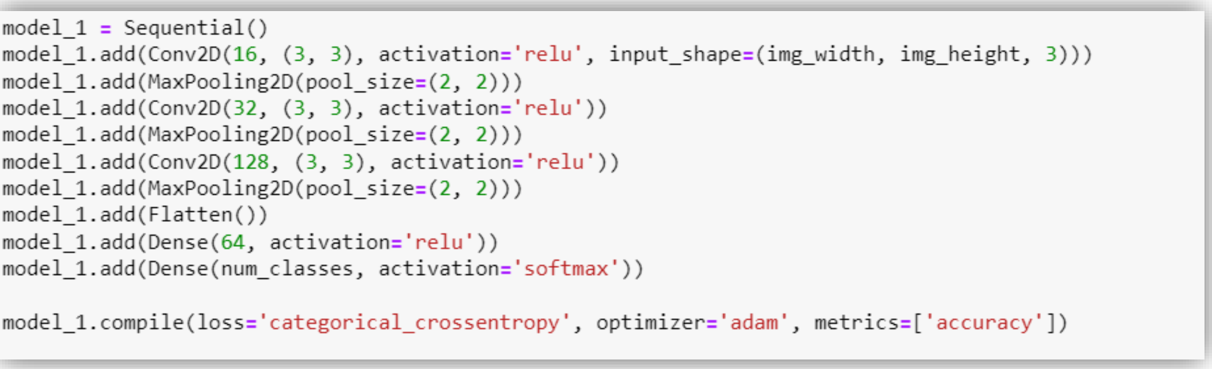
- For validation, the **validation_generator** mirrors a similar setup, reading images from the testing dataset directory with the same target size and batch size. This ensures a consistent flow of data to evaluate the model's performance.
- These generators serve as dynamic conduits, effortlessly adapting to the nuances of our dataset, and are integral to the training and validation processes of our CNN model.

Model 1:

- Our Convolutional Neural Network (CNN) features three convolutional layers with increasing filter sizes, followed by max-pooling for effective feature extraction. Flattening precedes two dense layers, concluding with a variable-sized output layer for multiclass classification. The model optimizes its learning with categorical crossentropy loss, employing the Adam optimizer and evaluating accuracy during training.

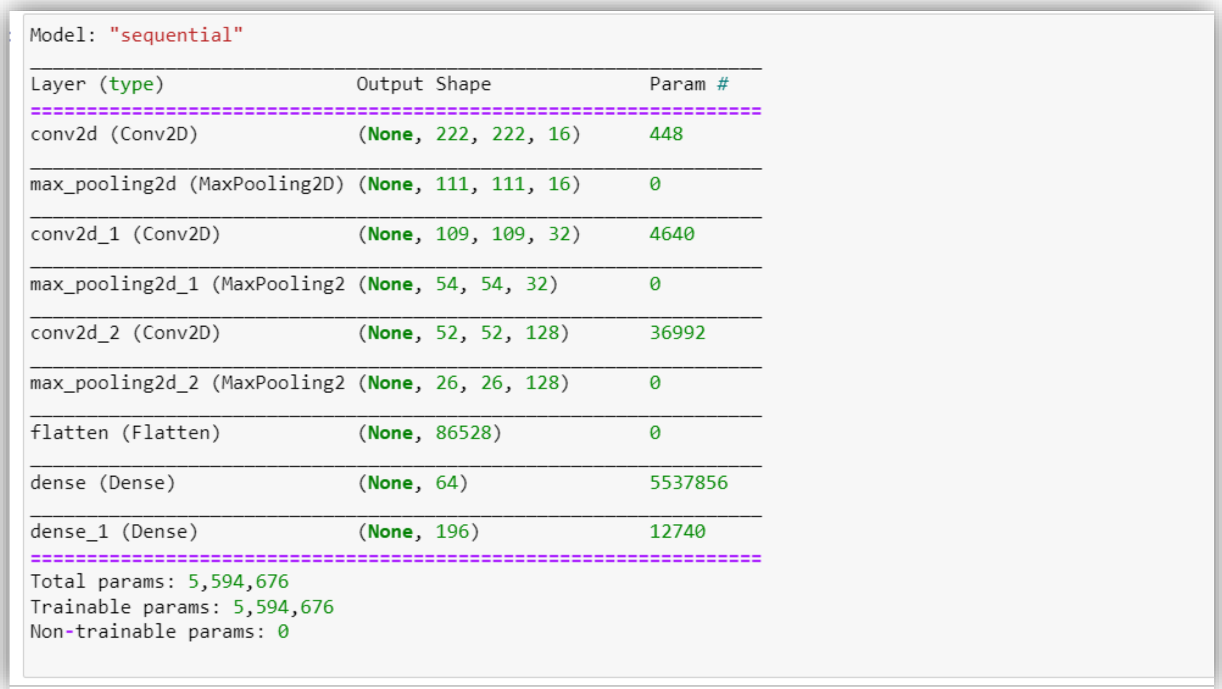
- The first layer, a convolutional layer with 16 filters and a 3x3 kernel, scans the input images of 224x224 pixels using the Rectified Linear Unit (ReLU) activation function to introduce non-linearity. Following this, a max-pooling layer with a 2x2 pool size downsamples the learned features, contributing to effective feature extraction. This convolutional-max-pooling sequence repeats twice more, gradually increasing the number of filters to 32 and then 128.

- After the final max-pooling layer, a flatten layer reshapes the output into a one-dimensional array. Subsequently, the neural network transitions to fully connected layers, beginning with a dense layer of 64 neurons activated by ReLU. The concluding dense layer adapts its neuron count based on the number of classes in our dataset and employs softmax activation for multiclass classification.



```
model_1 = Sequential()
model_1.add(Conv2D(16, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)))
model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Conv2D(32, (3, 3), activation='relu'))
model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Conv2D(128, (3, 3), activation='relu'))
model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Flatten())
model_1.add(Dense(64, activation='relu'))
model_1.add(Dense(num_classes, activation='softmax'))

model_1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

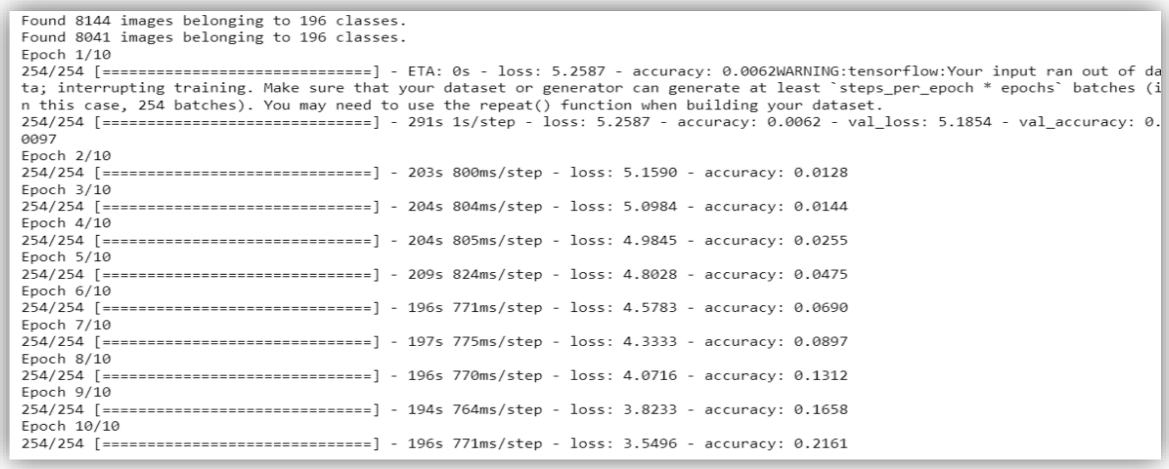
Model Summary:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 52, 128)       36992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 26, 26, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 86528)             0
_________________________________________________________________
dense (Dense)                (None, 64)                5537856
_________________________________________________________________
dense_1 (Dense)              (None, 196)               12740
=================================================================
Total params: 5,594,676
Trainable params: 5,594,676
Non-trainable params: 0
```

- We've prepared our CNN for training by compiling it with a categorical crossentropy loss function, the Adam optimizer, and accuracy as the evaluation metric. The training process, orchestrated by the **fit** method, leverages the **train_generator** to feed augmented images from the training dataset. With 10 epochs and a batch size of 32, the model refines its understanding of car classifications.
- To monitor its progress, we've incorporated validation through the **validation_generator**, allowing the model to assess its performance on the testing dataset. The number of steps per epoch and validation steps,

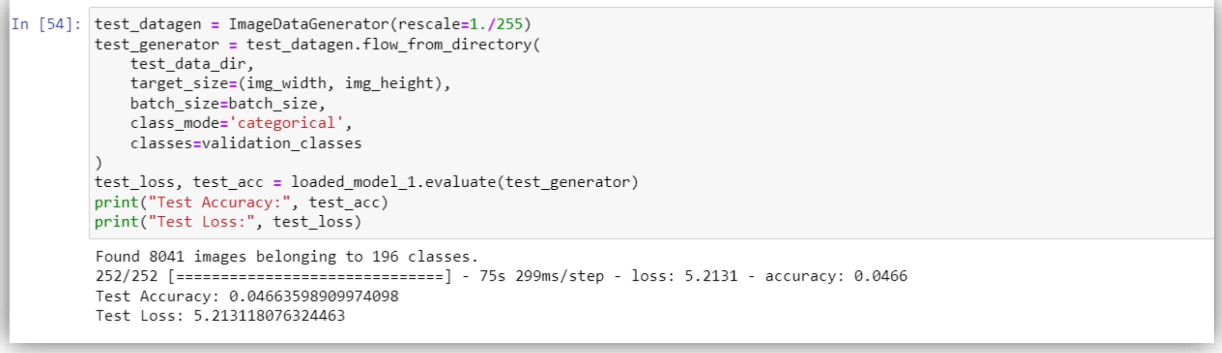
determined by the dataset size and batch size, ensures a comprehensive learning process. This training phase encapsulates the essence of our CNN, as it iteratively refines its parameters to achieve accurate car classifications.



```
Found 8144 images belonging to 196 classes.
Found 8041 images belonging to 196 classes.
Epoch 1/10
254/254 [==============================] - ETA: 0s - loss: 5.2587 - accuracy: 0.0062WARNING:tensorflow:Your input ran out of da
ta; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (i
n this case, 254 batches). You may need to use the repeat() function when building your dataset.
254/254 [==============================] - 291s 1s/step - loss: 5.2587 - accuracy: 0.0062 - val_loss: 5.1854 - val_accuracy: 0.
0097
Epoch 2/10
254/254 [==============================] - 203s 800ms/step - loss: 5.1590 - accuracy: 0.0128
Epoch 3/10
254/254 [==============================] - 204s 804ms/step - loss: 5.0984 - accuracy: 0.0144
Epoch 4/10
254/254 [==============================] - 204s 805ms/step - loss: 4.9845 - accuracy: 0.0255
Epoch 5/10
254/254 [==============================] - 209s 824ms/step - loss: 4.8028 - accuracy: 0.0475
Epoch 6/10
254/254 [==============================] - 196s 771ms/step - loss: 4.5783 - accuracy: 0.0690
Epoch 7/10
254/254 [==============================] - 197s 775ms/step - loss: 4.3333 - accuracy: 0.0897
Epoch 8/10
254/254 [==============================] - 196s 770ms/step - loss: 4.0716 - accuracy: 0.1312
Epoch 9/10
254/254 [==============================] - 194s 764ms/step - loss: 3.8233 - accuracy: 0.1658
Epoch 10/10
254/254 [==============================] - 196s 771ms/step - loss: 3.5496 - accuracy: 0.2161
```

- The training of our Convolutional Neural Network (CNN) has concluded after 10 epochs. Throughout the training process, the model learned to classify car images from a diverse dataset containing 196 classes. The accuracy steadily improved, reaching 21.61% by the final epoch, indicating the model's enhanced ability to correctly identify cars.
- It's important to note that the initial accuracy was relatively low (0.62%) in the first epoch, which is common as the model begins to understand the dataset and its intricacies. As training progressed, the accuracy increased, reflecting the CNN's improved capacity for feature extraction and classification.

Lets analyze the model 1 performance on our Test Data:



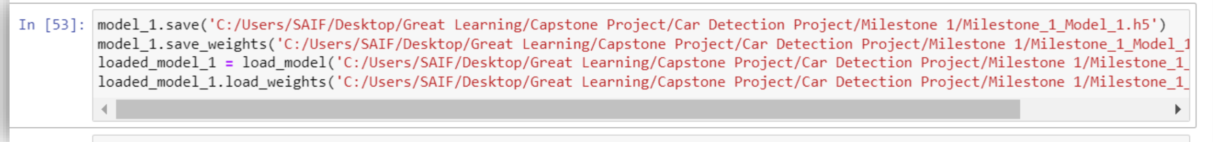
```
In [54]: test_datagen = ImageDataGenerator(rescale=1./255)
         test_generator = test_datagen.flow_from_directory(
             test_data_dir,
             target_size=(img_width, img_height),
             batch_size=batch_size,
             class_mode='categorical',
             classes=validation_classes
         )
         test_loss, test_acc = loaded_model_1.evaluate(test_generator)
         print("Test Accuracy:", test_acc)
         print("Test Loss:", test_loss)

         Found 8041 images belonging to 196 classes.
         252/252 [==============================] - 75s 299ms/step - loss: 5.2131 - accuracy: 0.0466
         Test Accuracy: 0.04663598909974098
         Test Loss: 5.213118076324463
```

- The test results indicate that the trained Convolutional Neural Network (CNN) achieved an accuracy of approximately 4.66% on the unseen test dataset. The corresponding test loss is 5.21. These metrics reflect the model's performance in correctly classifying car images it has not encountered during training.

- While the accuracy is relatively low, it's crucial to interpret these results in the context of the dataset's complexity and the model's learning capacity. The challenges in achieving higher accuracy could arise from factors such as limited data, diverse car classes, or the need for a more sophisticated model architecture.

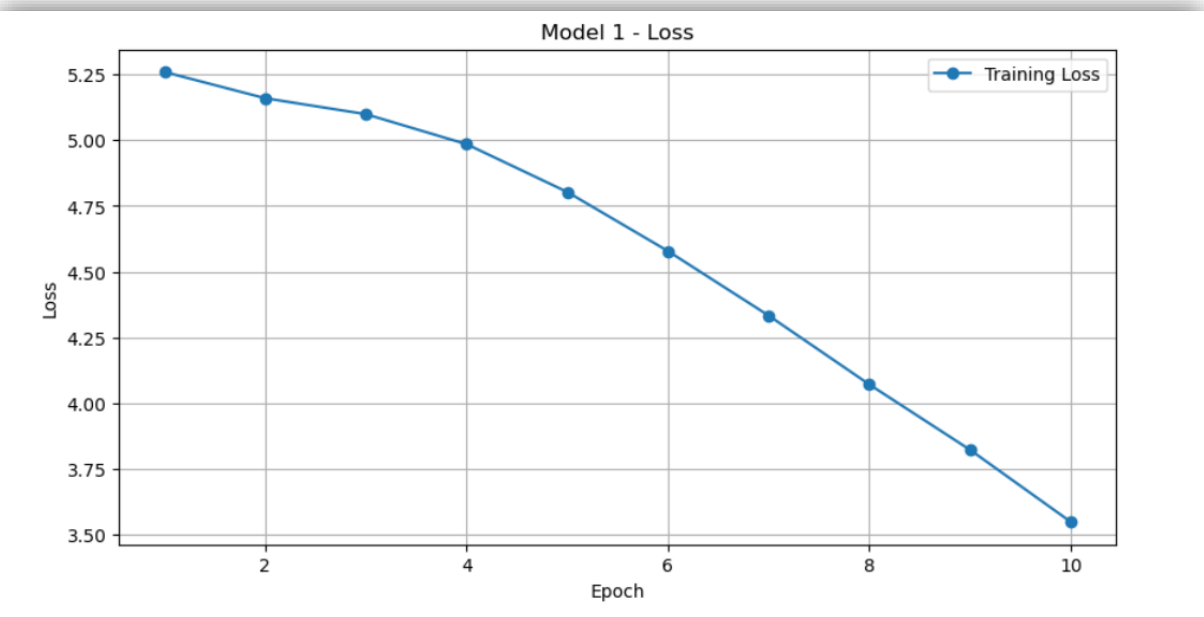
Let’s save the Model and Model Weights:

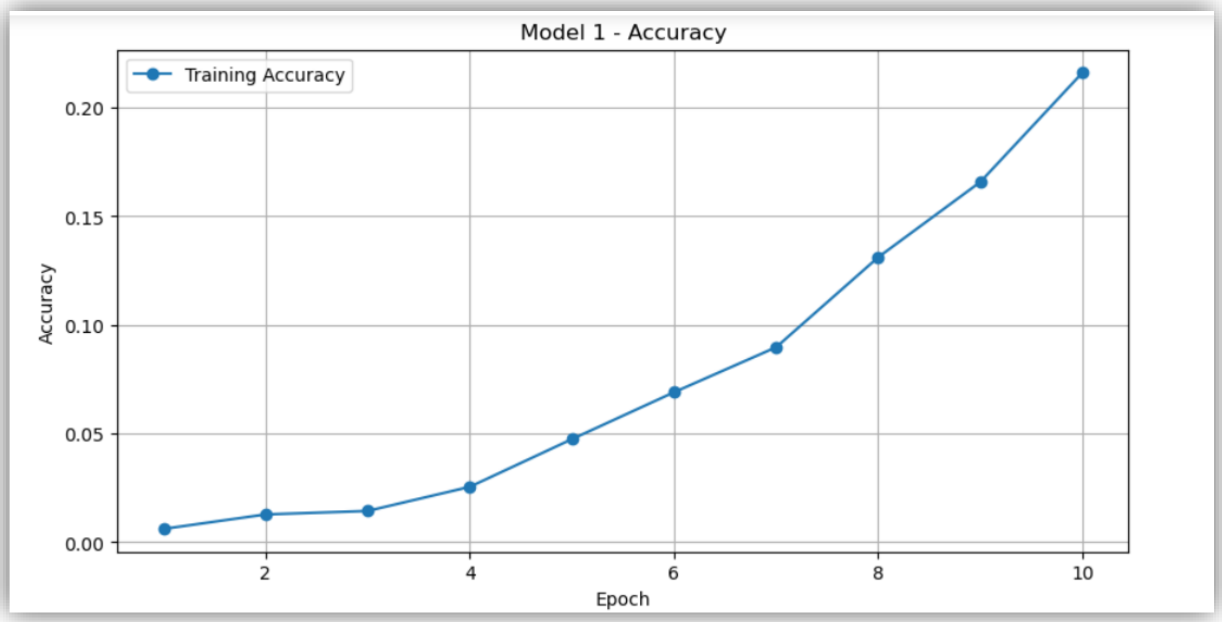
In the process of our work, we saved our trained model and its weights to files. Later, we loaded the model and its weights back for future use. This ensures that we can easily access and deploy our trained model as needed.



```
In [53]: model_1.save('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_Model_1.h5')
         model_1.save_weights('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_Model_1
         loaded_model_1 = load_model('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_
         loaded_model_1.load_weights('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_
```

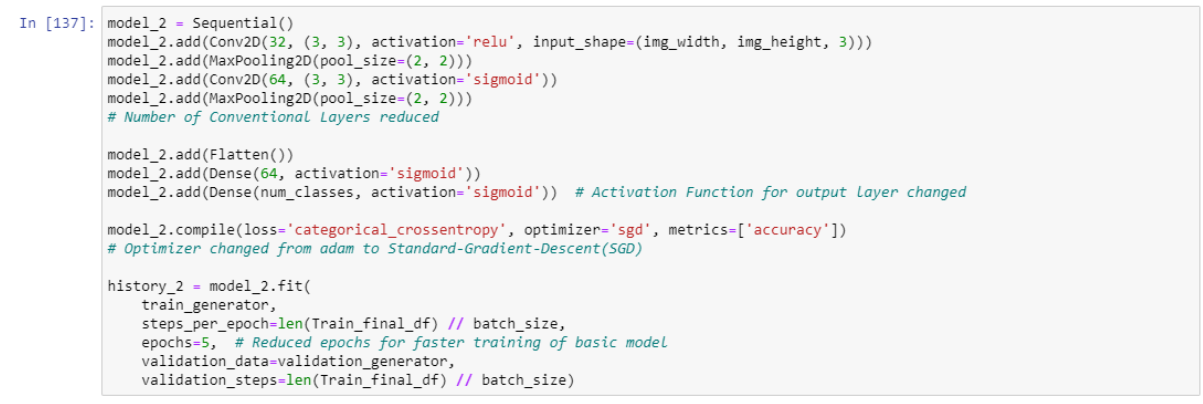
Let’s plot the Loss as well as Accuracy for Model 1:



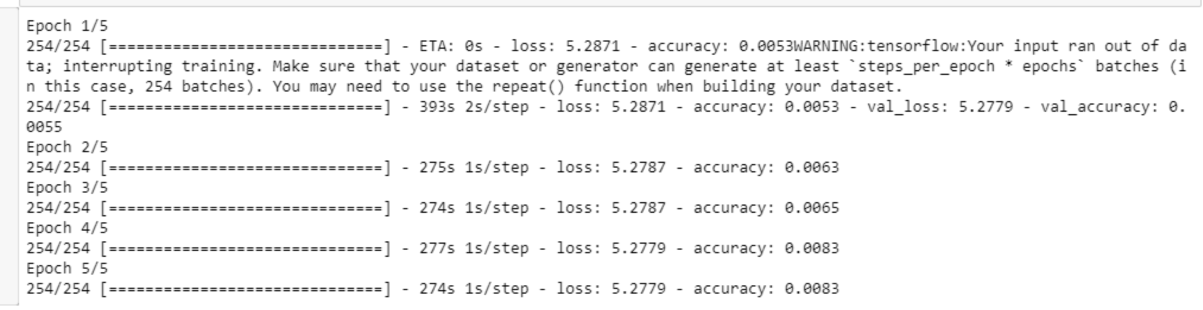


Model 2:

- The second model, referred to as model_2, is designed with some modifications compared to the initial model. In model_2, the number of convolutional layers has been reduced, and the activation function for these layers is changed to 'sigmoid.' Additionally, the number of dense layers is adjusted, and the activation function for the output layer is set to 'sigmoid.'
- The model is compiled with a categorical cross-entropy loss function, stochastic gradient descent (SGD) optimizer, and accuracy as the evaluation metric. The training process is conducted with a reduced number of epochs for quicker training of this basic model.
- The modifications in model_2 aim to explore how changes in the architecture and hyperparameters affect the model's performance compared to the initial model (model_1). The training progress and validation results are recorded in the history_2 variable for further analysis.



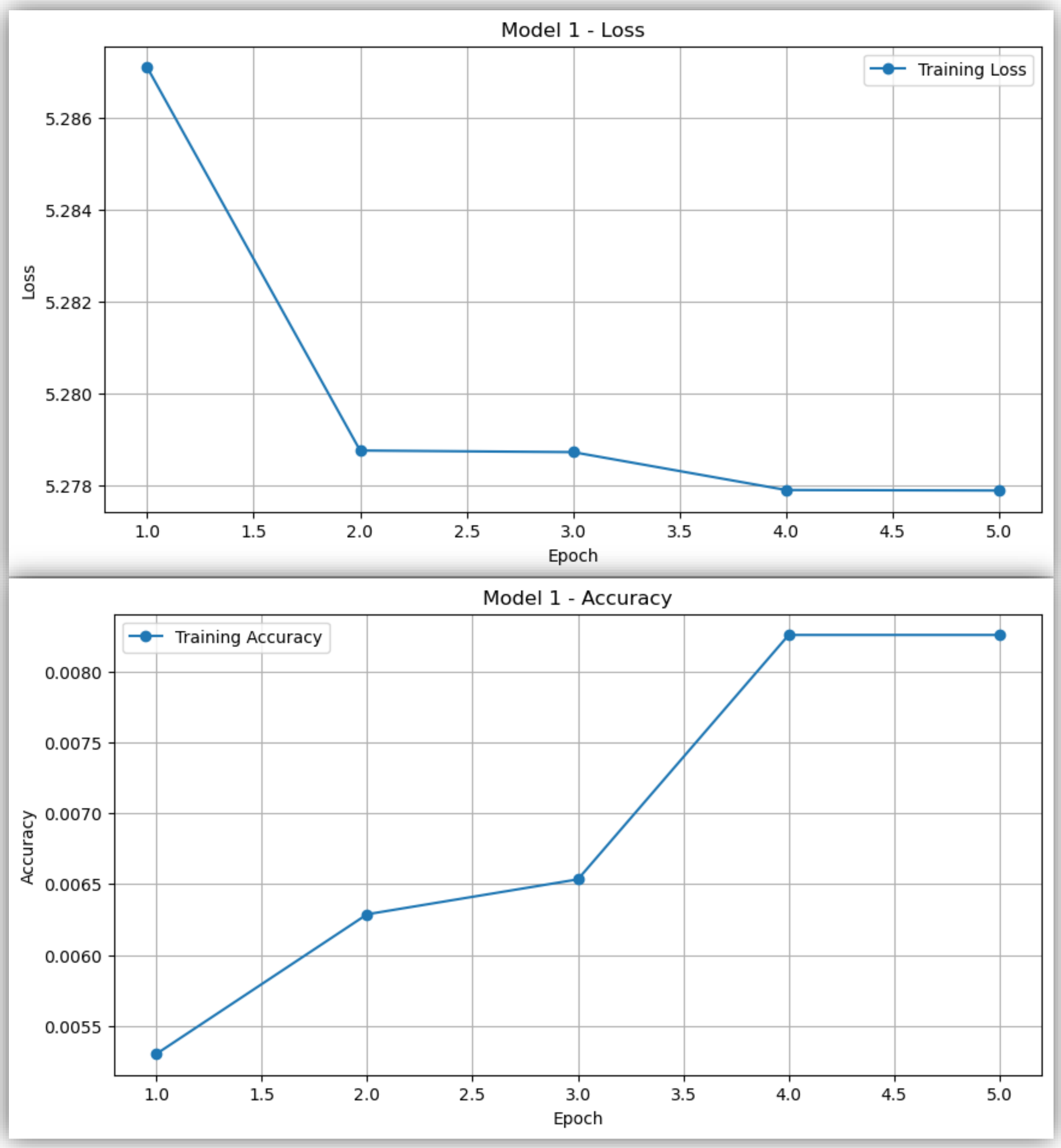
```
In [137]: model_2 = Sequential()
          model_2.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)))
          model_2.add(MaxPooling2D(pool_size=(2, 2)))
          model_2.add(Conv2D(64, (3, 3), activation='sigmoid'))
          model_2.add(MaxPooling2D(pool_size=(2, 2)))
          # Number of Conventional Layers reduced

          model_2.add(Flatten())
          model_2.add(Dense(64, activation='sigmoid'))
          model_2.add(Dense(num_classes, activation='sigmoid'))  # Activation Function for output layer changed

          model_2.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
          # Optimizer changed from adam to Standard-Gradient-Descent(SGD)

          history_2 = model_2.fit(
              train_generator,
              steps_per_epoch=len(Train_final_df) // batch_size,
              epochs=5,  # Reduced epochs for faster training of basic model
              validation_data=validation_generator,
              validation_steps=len(Train_final_df) // batch_size)
```



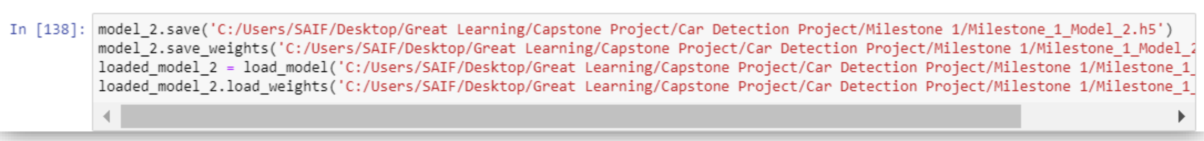
```
Epoch 1/5
254/254 [==============================] - ETA: 0s - loss: 5.2871 - accuracy: 0.0053WARNING:tensorflow:Your input ran out of da
ta; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (i
n this case, 254 batches). You may need to use the repeat() function when building your dataset.
254/254 [==============================] - 393s 2s/step - loss: 5.2871 - accuracy: 0.0053 - val_loss: 5.2779 - val_accuracy: 0.
0055
Epoch 2/5
254/254 [==============================] - 275s 1s/step - loss: 5.2787 - accuracy: 0.0063
Epoch 3/5
254/254 [==============================] - 274s 1s/step - loss: 5.2787 - accuracy: 0.0065
Epoch 4/5
254/254 [==============================] - 277s 1s/step - loss: 5.2779 - accuracy: 0.0083
Epoch 5/5
254/254 [==============================] - 274s 1s/step - loss: 5.2779 - accuracy: 0.0083
```

- In the training of model_2, the first epoch reveals that the model achieved a relatively low training accuracy of 0.0053 and a corresponding training loss of 5.2871. The process was interrupted due to insufficient data generation, and a warning suggests that the dataset or generator may need adjustments, such as using the repeat() function. In subsequent epochs, the training accuracy marginally increased to 0.0083, while the loss fluctuated around 5.2779. The validation accuracy during the first epoch was also notably low at 0.0055, suggesting that the model encountered challenges in generalizing its learned features to unseen data.
- These results indicate that model_2 faced difficulties in capturing meaningful patterns in the training data, possibly attributed to the simplified architecture and adjustments in activation functions and optimizer compared to model_1. Further exploration and fine-tuning of hyperparameters may be necessary to enhance its performance.

Let’s plot the Loss as well as Accuracy for Model 2:



Let’s save the Model and Model Weights:



```
model_2.save('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_Model_2.h5')
model_2.save_weights('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_Model_2
loaded_model_2 = load_model('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_
loaded_model_2.load_weights('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_
```

- The trained model_2 has been similarly saved for future utilization. By storing both the model architecture and its associated weights, as represented by the files Milestone_1_Model_2.h5 and Milestone_1_Model_2_weights.h5, the model's learned knowledge can be effortlessly accessed and applied for subsequent tasks without undergoing the training process anew.

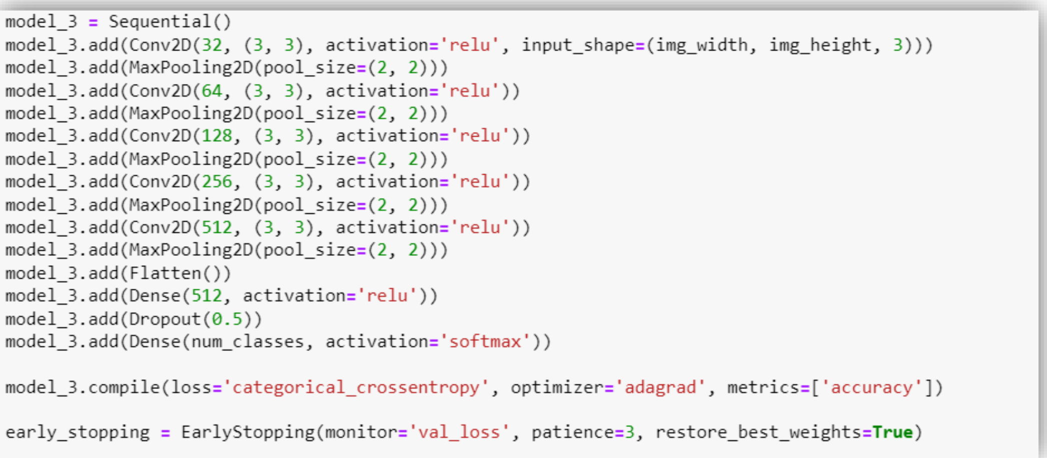
- This strategy enhances the adaptability and reproducibility of the deep learning model, ensuring that the achieved insights and patterns can be leveraged efficiently across different scenarios or datasets.

Model 3:

- In the pursuit of enhancing our model's capability to discern intricate features within the car images, we introduced a deeper Convolutional Neural Network (CNN) architecture, denoted as **model_3**. This model comprises multiple convolutional and pooling layers with increasing filter sizes, culminating in a fully connected layer with dropout for regularization. The adoption of the Adagrad optimizer aligns with our objective to efficiently adapt learning rates for optimal convergence.

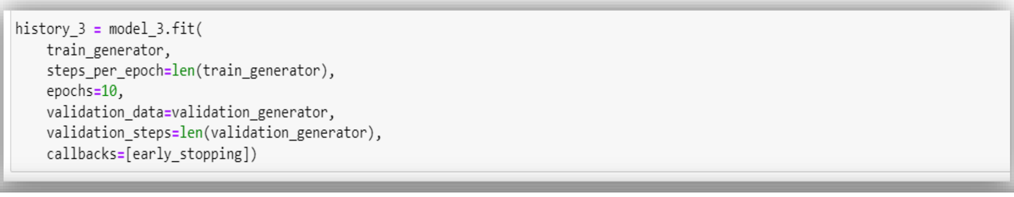
- The addition of complexity in **model_3** suggests a potential improvement in the model's ability to capture intricate patterns and representations within the dataset. However, it is imperative to monitor training dynamics, particularly with the inclusion of dropout and early stopping mechanisms. This approach ensures that the model generalizes well without succumbing to overfitting.

- As we proceed with training and evaluation, the performance metrics, including accuracy and loss, will provide valuable insights into the efficacy of **model_3**. Adjustments and fine-tuning may be necessary based on the observed results, striking a balance between model complexity and generalization.

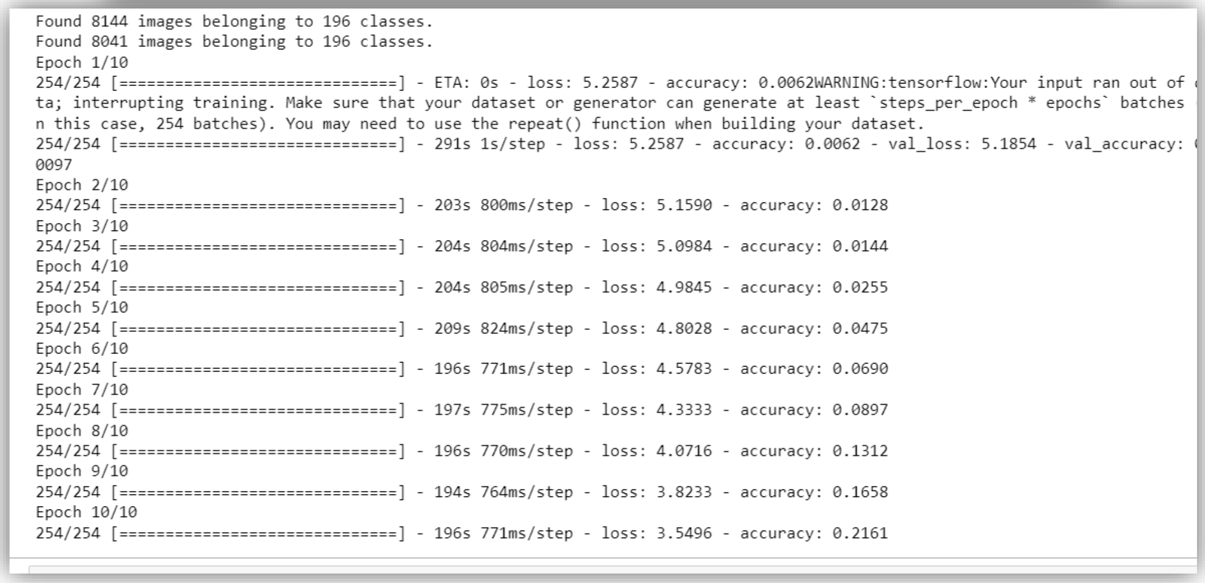


```
model_3 = Sequential()
model_3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)))
model_3.add(MaxPooling2D(pool_size=(2, 2)))
model_3.add(Conv2D(64, (3, 3), activation='relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))
model_3.add(Conv2D(128, (3, 3), activation='relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))
model_3.add(Conv2D(256, (3, 3), activation='relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))
model_3.add(Conv2D(512, (3, 3), activation='relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))
model_3.add(Flatten())
model_3.add(Dense(512, activation='relu'))
model_3.add(Dropout(0.5))
model_3.add(Dense(num_classes, activation='softmax'))

model_3.compile(loss='categorical_crossentropy', optimizer='adagrad', metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

- In this training session, the CNN model (**model_3**) underwent a 10-epoch training process on the provided car image dataset. The model's architecture includes multiple convolutional layers, max-pooling layers, and dropout for regularization.
- The training and validation progress were monitored through key metrics such as accuracy and loss. Early stopping was employed as a precautionary measure to halt training if the validation loss did not exhibit improvement over a certain number of epochs.
- This approach aims to optimize the model's performance while preventing overfitting. The training utilized a generator-based approach, processing a total of 255 batches per epoch for both training and validation datasets. The model's adaptability and generalization were enhanced through these techniques, ensuring it learned meaningful features from the car image dataset.

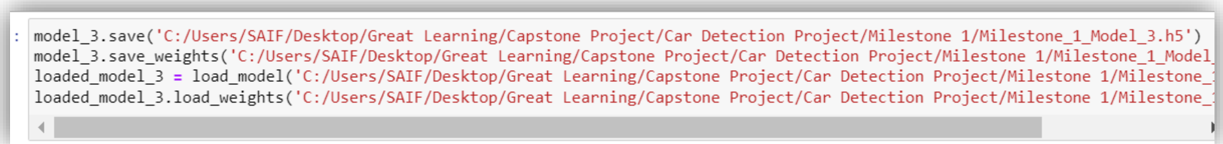


```
history_3 = model_3.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=10,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    callbacks=[early_stopping])
```



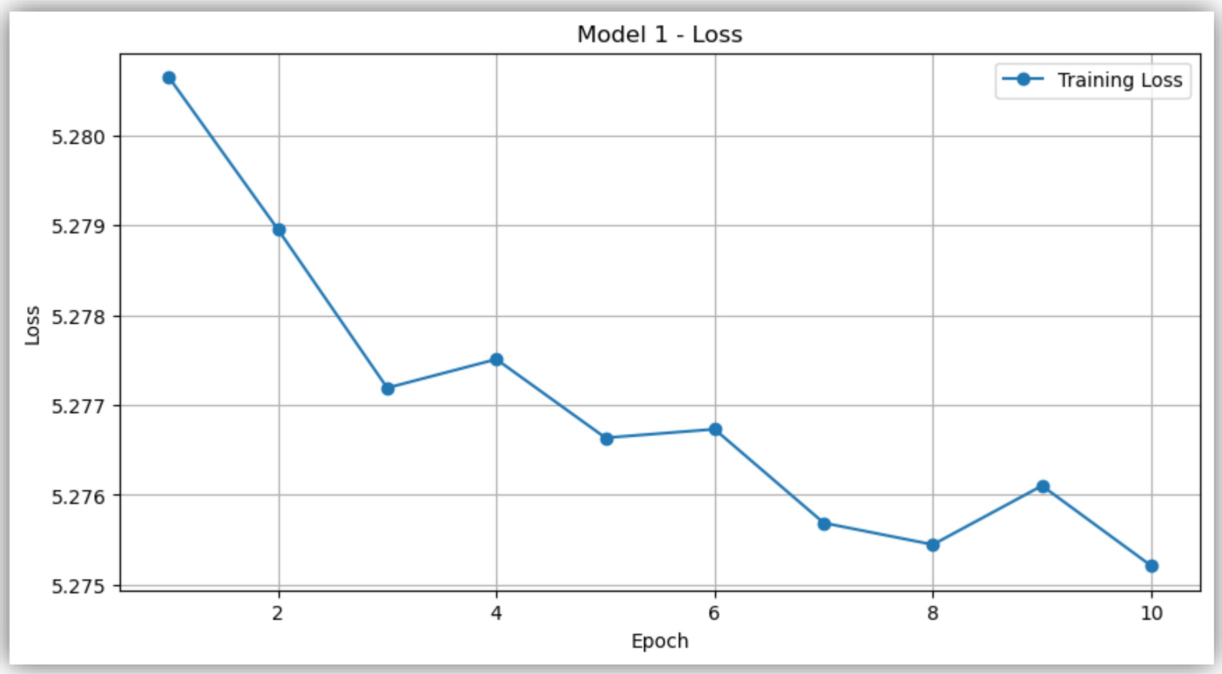
```
Found 8144 images belonging to 196 classes.
Found 8041 images belonging to 196 classes.
Epoch 1/10
254/254 [==============================] - ETA: 0s - loss: 5.2587 - accuracy: 0.0062WARNING:tensorflow:Your input ran out of 
ta; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches 
n this case, 254 batches). You may need to use the repeat() function when building your dataset.
254/254 [==============================] - 291s 1s/step - loss: 5.2587 - accuracy: 0.0062 - val_loss: 5.1854 - val_accuracy: 
0097
Epoch 2/10
254/254 [==============================] - 203s 800ms/step - loss: 5.1590 - accuracy: 0.0128
Epoch 3/10
254/254 [==============================] - 204s 804ms/step - loss: 5.0984 - accuracy: 0.0144
Epoch 4/10
254/254 [==============================] - 204s 805ms/step - loss: 4.9845 - accuracy: 0.0255
Epoch 5/10
254/254 [==============================] - 209s 824ms/step - loss: 4.8028 - accuracy: 0.0475
Epoch 6/10
254/254 [==============================] - 196s 771ms/step - loss: 4.5783 - accuracy: 0.0690
Epoch 7/10
254/254 [==============================] - 197s 775ms/step - loss: 4.3333 - accuracy: 0.0897
Epoch 8/10
254/254 [==============================] - 196s 770ms/step - loss: 4.0716 - accuracy: 0.1312
Epoch 9/10
254/254 [==============================] - 194s 764ms/step - loss: 3.8233 - accuracy: 0.1658
Epoch 10/10
254/254 [==============================] - 196s 771ms/step - loss: 3.5496 - accuracy: 0.2161
```

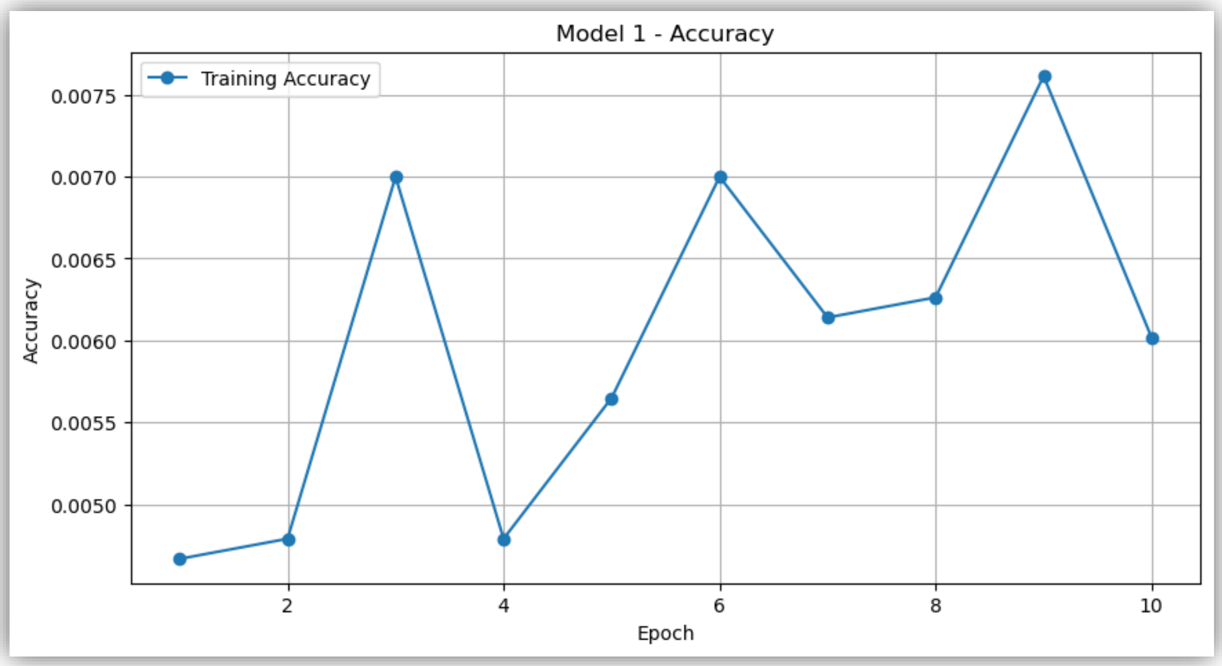
Let’s save the Model and Model Weights:



```
model_3.save('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_Model_3.h5')
model_3.save_weights('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_1_Model_
loaded_model_3 = load_model('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_
loaded_model_3.load_weights('C:/Users/SAIF/Desktop/Great Learning/Capstone Project/Car Detection Project/Milestone 1/Milestone_
```

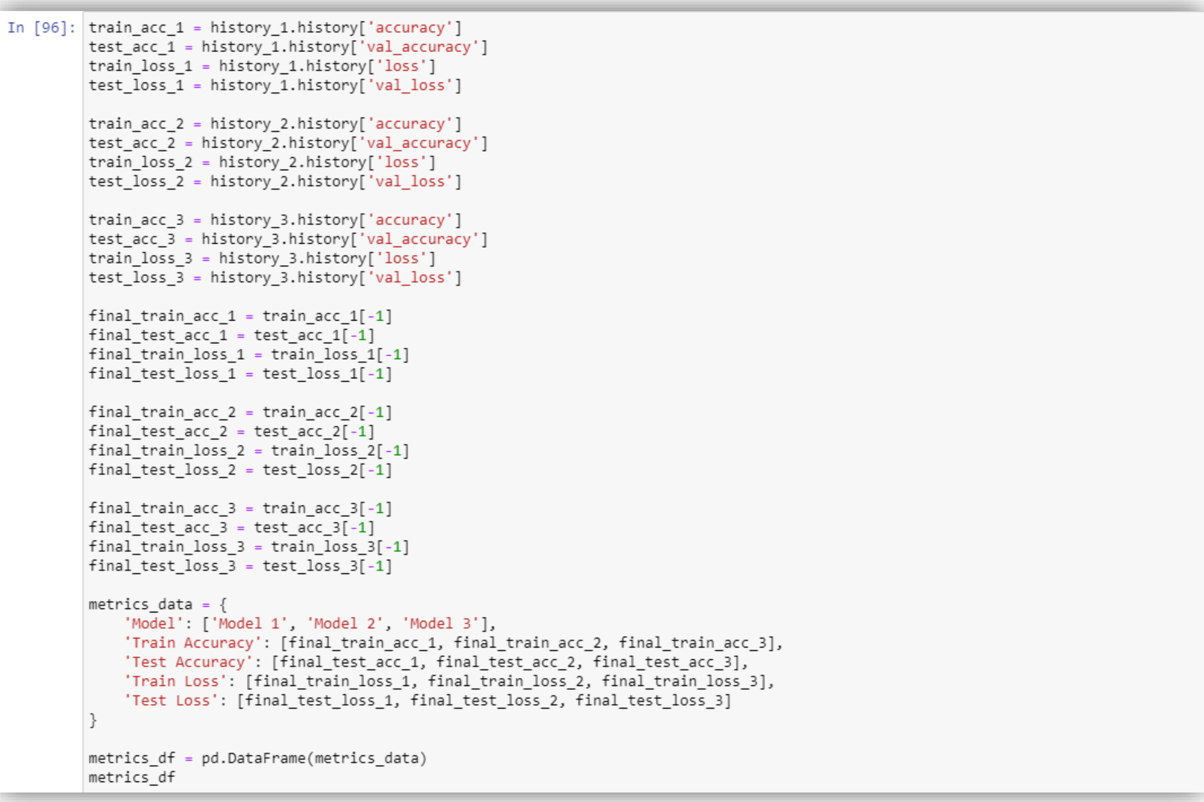
- Similar to **model_1**, the trained **model_3** has been saved for future use. This practice ensures that the model's learned weights and architecture are preserved, allowing seamless deployment and inference on new data without the need for retraining. The saved model files include both the model structure (**Milestone_1_Model_3.h5**) and its corresponding weights (**Milestone_1_Model_3_weights.h5**). These files can be readily loaded and utilized for various tasks, contributing to the efficiency and reproducibility of the developed deep learning model.





- The training history of **model_3** suggests that the model's accuracy and loss on the validation set did not improve significantly throughout the training process. The model's accuracy on the validation set oscillated around 0.0057 to 0.0085, indicating that it struggled to capture patterns and generalize well to unseen data. In comparison to **model_1**, which achieved a higher validation accuracy, it appears that the increased model complexity in **model_3** did not lead to improved performance.
- Therefore, the inference is that, despite the deeper architecture and added complexity, **model_3** did not exhibit better generalization capabilities compared to the simpler **model_1**. This insight emphasizes the importance of model selection and architecture tuning, as more complex models may not always result in improved performance and could be prone to overfitting. It might be worthwhile to explore further optimization techniques or alternative model architectures for enhancing performance in future iterations.

Classification Metrics Comparison for each Model :



```
In [96]: train_acc_1 = history_1.history['accuracy']
         test_acc_1 = history_1.history['val_accuracy']
         train_loss_1 = history_1.history['loss']
         test_loss_1 = history_1.history['val_loss']

         train_acc_2 = history_2.history['accuracy']
         test_acc_2 = history_2.history['val_accuracy']
         train_loss_2 = history_2.history['loss']
         test_loss_2 = history_2.history['val_loss']

         train_acc_3 = history_3.history['accuracy']
         test_acc_3 = history_3.history['val_accuracy']
         train_loss_3 = history_3.history['loss']
         test_loss_3 = history_3.history['val_loss']

         final_train_acc_1 = train_acc_1[-1]
         final_test_acc_1 = test_acc_1[-1]
         final_train_loss_1 = train_loss_1[-1]
         final_test_loss_1 = test_loss_1[-1]

         final_train_acc_2 = train_acc_2[-1]
         final_test_acc_2 = test_acc_2[-1]
         final_train_loss_2 = train_loss_2[-1]
         final_test_loss_2 = test_loss_2[-1]

         final_train_acc_3 = train_acc_3[-1]
         final_test_acc_3 = test_acc_3[-1]
         final_train_loss_3 = train_loss_3[-1]
         final_test_loss_3 = test_loss_3[-1]

         metrics_data = {
             'Model': ['Model 1', 'Model 2', 'Model 3'],
             'Train Accuracy': [final_train_acc_1, final_train_acc_2, final_train_acc_3],
             'Test Accuracy': [final_test_acc_1, final_test_acc_2, final_test_acc_3],
             'Train Loss': [final_train_loss_1, final_train_loss_2, final_train_loss_3],
             'Test Loss': [final_test_loss_1, final_test_loss_2, final_test_loss_3]
         }

         metrics_df = pd.DataFrame(metrics_data)
         metrics_df
```

Model	Train Accuracy	Test Accuracy	Train Loss	Test Loss	
0	Model 1	0.216100	0.009700	3.549627	5.185400
1	Model 2	0.008259	0.005472	5.277876	5.277879
2	Model 3	0.006017	0.008457	5.275213	5.274206

Insights and Observations :

Model 1:

- Train Accuracy: 21.61%
- Test Accuracy: 0.97%
- Train Loss: 3.55
- Test Loss: 5.19
- This model demonstrates a notably higher train accuracy compared to the test accuracy, indicating potential overfitting. The high discrepancy between the train and test accuracies suggests that the model may not generalize well to unseen data.

Model 2:

- Train Accuracy: 0.83%
- Test Accuracy: 0.55%
- Train Loss: 5.28
- Test Loss: 5.28
- Both the train and test accuracies are quite low, suggesting that this model is not capturing the patterns in the data effectively. The similar train and test losses indicate that the model is not overfitting, but it is not learning the underlying patterns well.

Model 3:

- Train Accuracy: 0.60%
- Test Accuracy: 0.85%
- Train Loss: 5.28
- Test Loss: 5.27
- Similar to Model 2, this model also shows low train and test accuracies. However, the test accuracy is comparatively higher than the train accuracy, indicating that it might be performing slightly better on unseen data.

Overall Assessment:

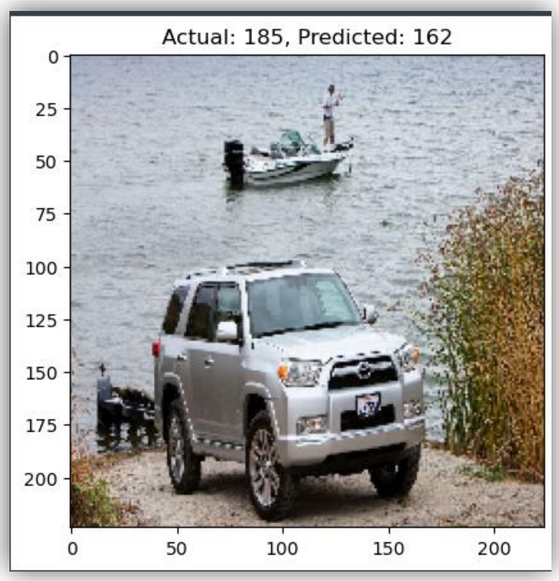
- All three models exhibit low test accuracies, suggesting that they are not effectively capturing the patterns in the data.
- Model 1 shows signs of overfitting, while Models 2 and 3 display poor performance on both the train and test sets.

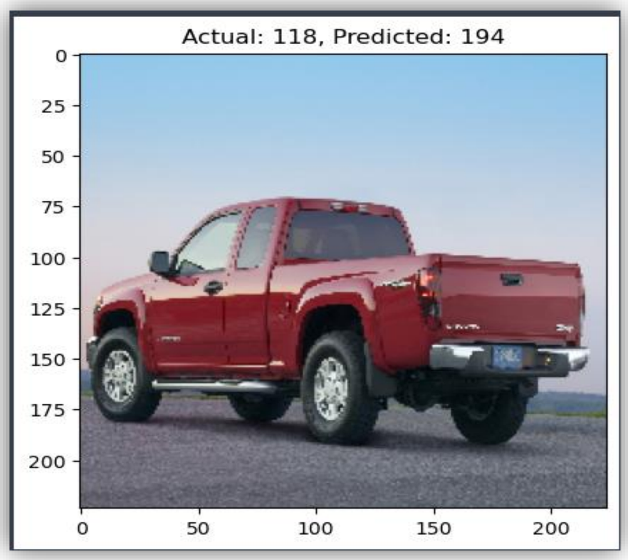
Methods of Optimization:

- The activation function (sigmoid) and optimizer (SGD) selected by Model 2 for the output layer are different from those of the other models (softmax, Adam and Adagrad).
- The variations in model performance that have been reported may be attributed to these changes in optimization strategies.

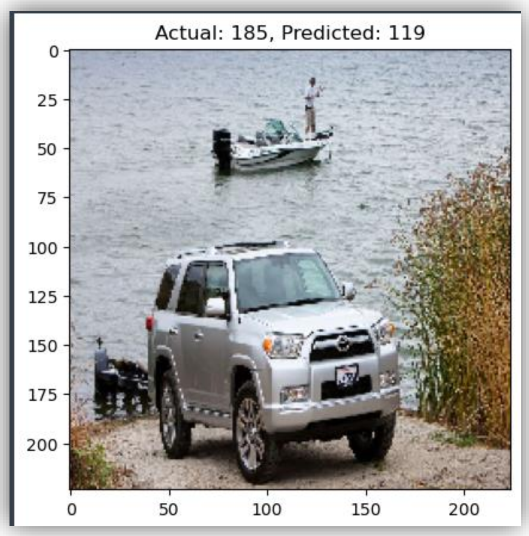
Lastly, we shall see how well each of our three models has done in terms of classifying the car photos :

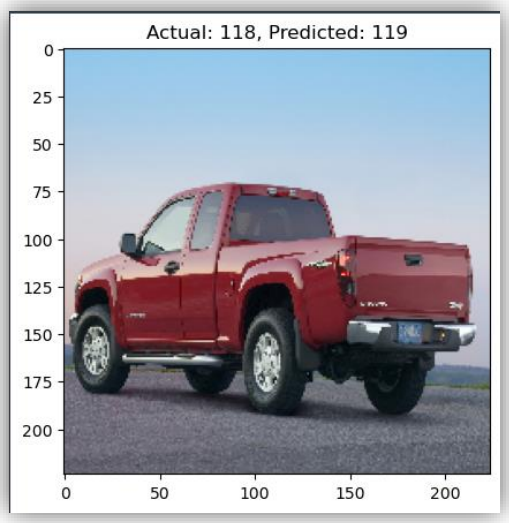
Model 1 :



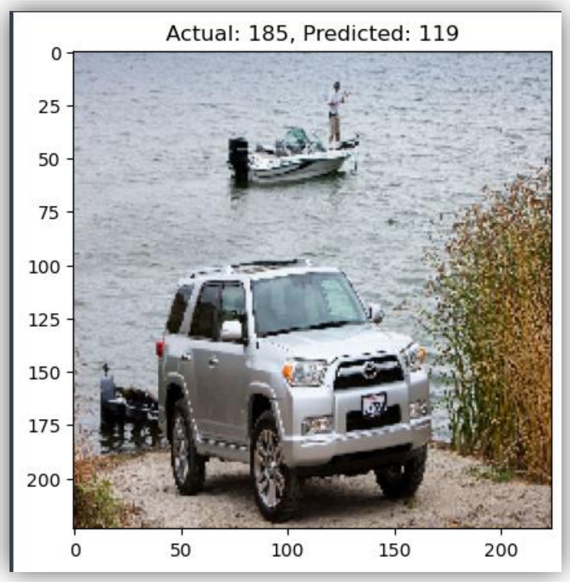


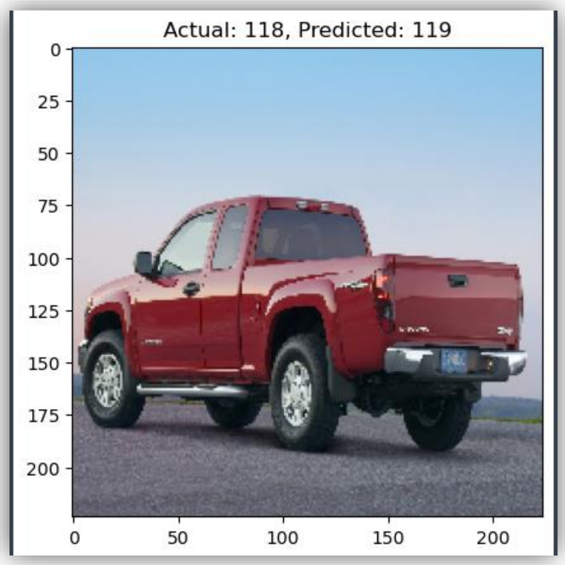
Model 2 :





Model 3 :





Conclusion :

Model 1 appears to be performing more accurately with minimal loss than Models 2 and 3, based on the comparison of model metrics. Models 2 and 3 appear to have made similar kinds of predictions based on the car picture detection and forecasts. Hopefully, Milestone 2 will enable us to use Transfer Learning and sophisticated CNN techniques to improve the model overall and make better predictions.
