Computer Vision

OBJECT DETECTION - CAR

## A Capstone Project

*Submitted by****:***

Sindhu Lathasenthil

R. Sethuraman

Gokulkrishnan M

Santanu Mukherjee

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

March 2025

Table of Contents

|  |  |  |
| --- | --- | --- |
| Sl. No. | Topic | Page No. |
| 1 | Introduction | 3 |
| 1.1 | Problem Statement | 3 |
| 1.2 | Exploring the Input data | 4 |
| 1.3 | Mapping Images to their Classes | 6 |
| 2 | Data Preparation for the Model Building | 7 |
| 2.1 | Image Data Augmentation | 8 |
| 2.2 | Combining the Training Images | 11 |
| 2.3 | Image Data Preprocessing | 12 |
| 3 | Model Building | 14 |
| 3.1 | Model Training | 21 |
| 3.2 | Model Saving | 25 |
| 4 | Model Evaluation |  |
| 5 | Comparison to Benchmark |  |
| 6 | Visualization of Model Predictions |  |
| 6.1 | GUI Creation and Model Deployment |  |
| 6.2 | GUI using HTML Webpage |  |
| 7 | Implications |  |
| 8 | Limitations |  |
| 9 | Closing Reflections |  |
| 10 | References |  |

# Introduction

Computer vision is a field of artificial intelligence (AI) and computer science that focuses on enabling machines to interpret and understand visual information from the world, similar to how humans use their eyes and brains to perceive and make sense of visual data. It involves the development of algorithms and models that allow computers to process, analyze, and understand images and videos.

It has numerous applications across various industries, including autonomous vehicles, healthcare (e.g., medical imaging), agriculture (e.g., crop monitoring), security (e.g., facial recognition), and entertainment (e.g., augmented reality and virtual reality). The field has seen significant advancements in recent years, largely due to the development of deep learning techniques and the availability of large datasets for training models.

## Problem Statement

Computer vision can be used to automate supervision and generate action appropriate action trigger if the event is predicted from the image of interest. For example, a car moving on the road can be easily identified by a camera as make of the car, type, color, number plates etc.

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

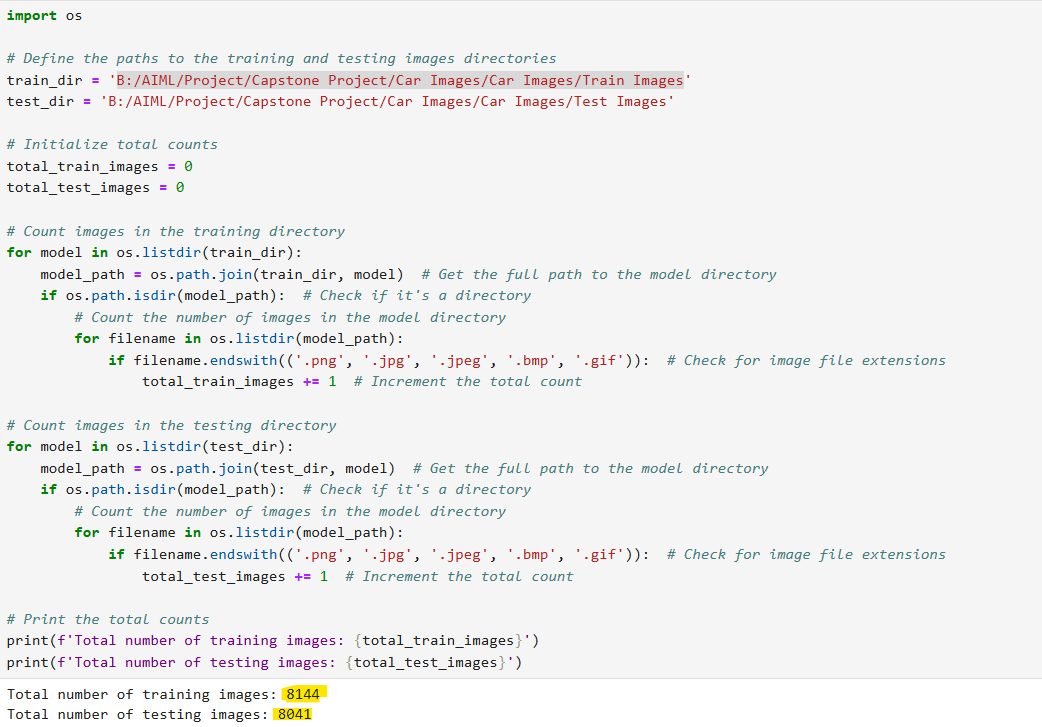
Data description:

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

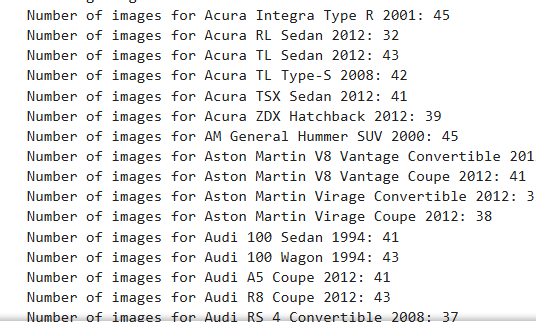
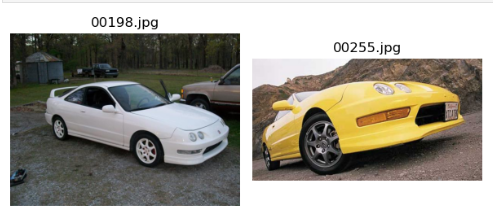
## Exploring the Input data

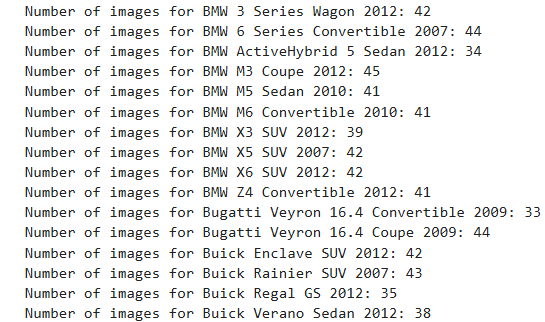
Input data has 8,144 training images and 8,041 testing images. Few images were randomly chosen and plotted using Matplotlib imread. The images are having different dimensions in height and width, the cars are in different orientations, colors and backgrounds. A few of the images are displayed below and a few class names along with make model year & count are also listed.

**Sample code to list the count of both training & testing data.**



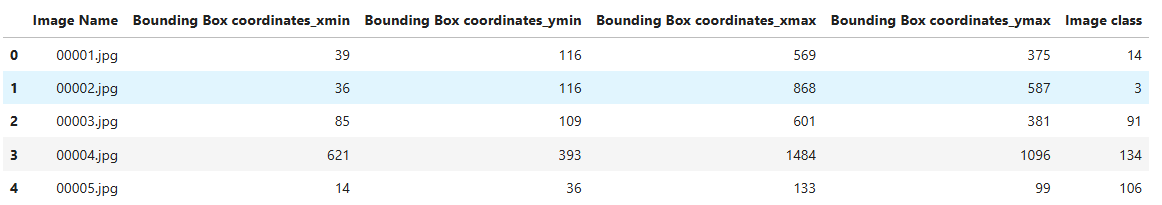
### Sample Images Class names: Make Model Year & Count

****

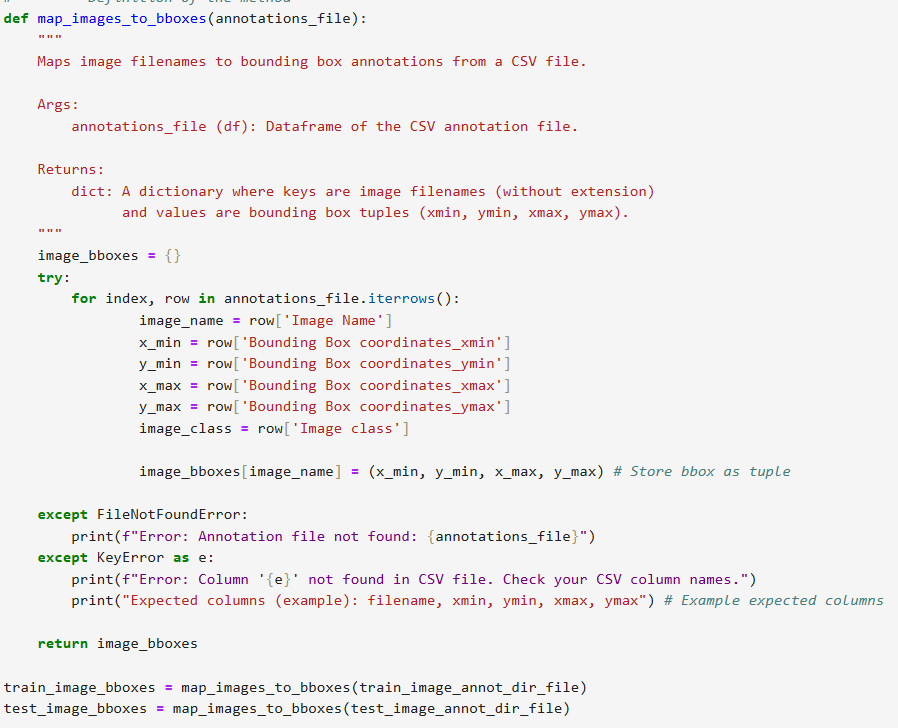
****

The annotations for both train and test data are given in the csv format. Each csv file contains the following information; Image name, corresponding bounding box coordinates and the class label of the car in the image.

Annotations for the first 5 images from Train Annotations.csv are shown below.



## 1.3 Mapping Images to their Classes

In this step, all the images are mapped to their respective classes. Created function for mapping and bounding box.

After mapping is done, few images with their bounding boxes are plotted as shown below.





# 2. Data Preparation for the Model Building

In the data preparation, all the images with bounding boxes need to be resized. We need to convert the labels to one hot encoding. With that, the data will be ready to be used for the deep learning model.

Here is the snap attached for reference to split both original & resized images. The original train images are resized to 128X128.





* Converted the images to cv2 arrays and preprocessed the images for standardization
* Resized the bounding boxes according to the resized images
* Two different CNN models were used
* ResNet & Google net model

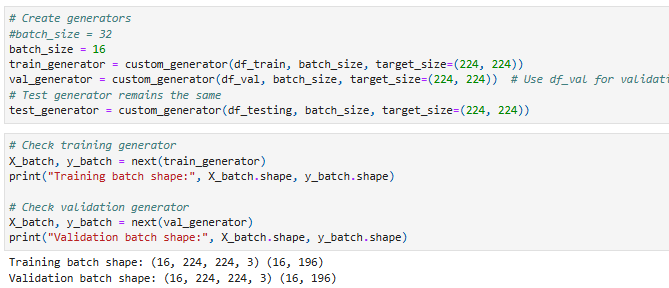
2.1 Image Data Augmentation

It helps to increases the complexity for model training, avoid overfitting of the model and improves the validation class accuracy.

ImageDataGenerator is used for real-time data augmentation and preprocessing of image data.

It is a powerful tool for enhancing the training images through augmentation, which can lead to better model performance and generalization. It is widely used in image classification tasks and other computer vision applications.

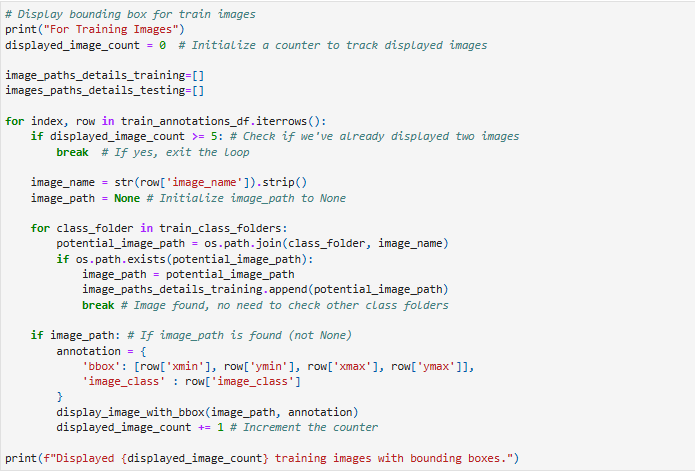
It applies a transformation to an image according to given parameters. “flow\_from\_directory” function takes the path to a directory & generates batches of augmented data. Here is the code for the ImageDataGenerator.

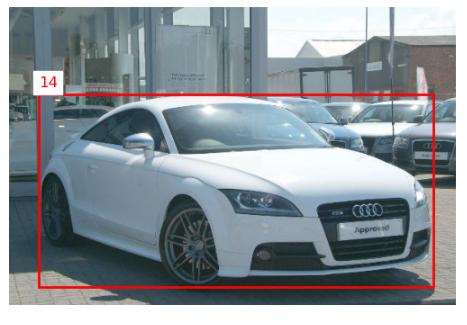


ImageDataGenarator does the augmentation only for the images not for the bounding boxes.

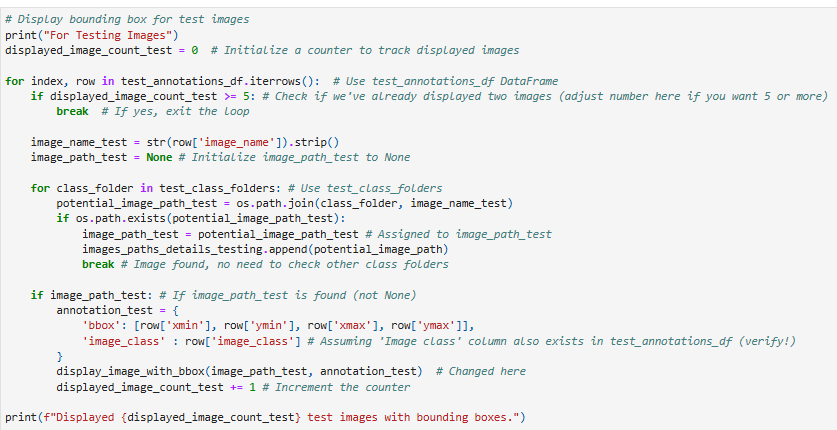
Since it is used only for the categorical prediction, it can be used only to improve the class label accuracy and a second model is still required to predict the bounding box predictions.

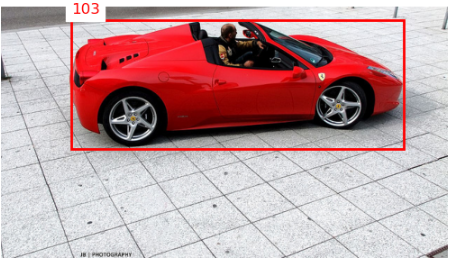
Code to display bounding box for training images.



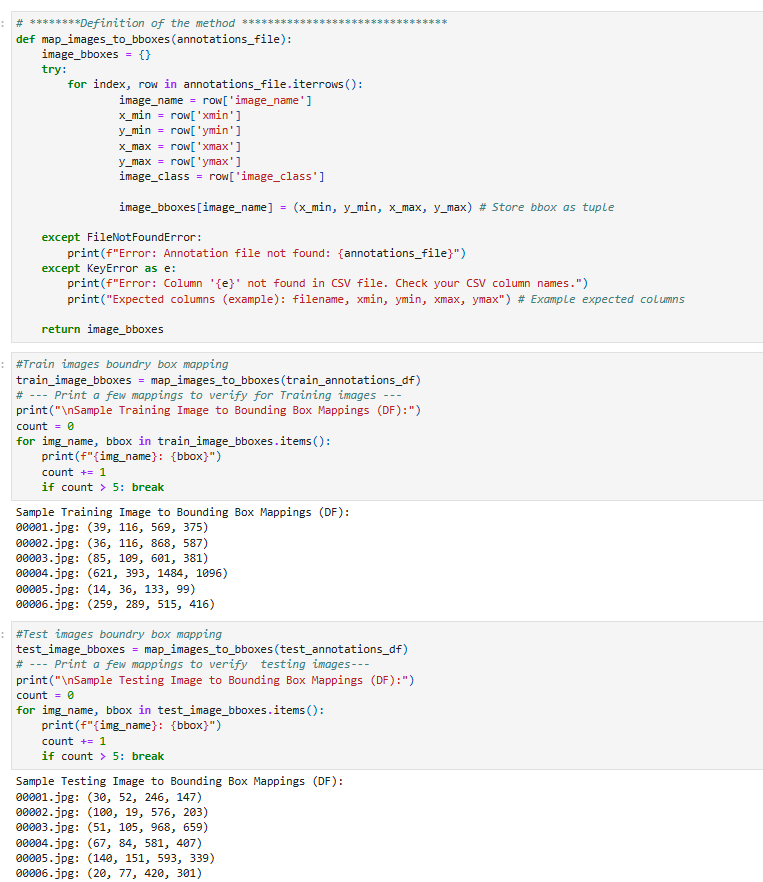
Code to display bounding box for testing images.



## . Combining the Training Images

In this step, the following code is used to create the data frame with filenames, bounding boxes and classes for the original train images.

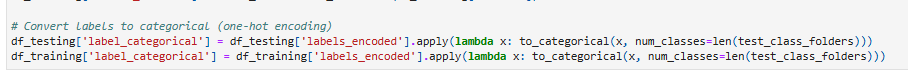


## Image Data Preprocessing

In this step, data was loaded into the data frame & its images were converted to the format of (224,224,3) and the labels were encoded accordingly. The image Classes were determined. The Bounding Boxes were determined and tested with images for both training and testing data sets. The images are converted to cv2 arrays and further standardized by dividing with 255.



Label Encoding is an important pre-processing step, it refers to converting the labels into the numeric form to convert it into the machine-readable form.



The same above procedure has been followed for the test images and saved into NumPy arrays. With that, both train and test arrays are ready for the model input.

Note:

There are four models used for training the datasets as part of milestone 1 the same is listed

1. MobileNetV2

2. GoogleNet

3. AlexNet

4. ResNet

The models were trained on the training data sets and evaluated on the validation/testing data sets

However, we are proceeding with final report by keeping Googlenet and Resnet models.

As it will undergo hyper parameter tunning as commonly data imbalance and accuracy is less

compared to other models such as Mobilenet and Alexnet which are light weight models/Shallow

models.

Hence, those two models were dropped from further fine tuning and comparing them with other

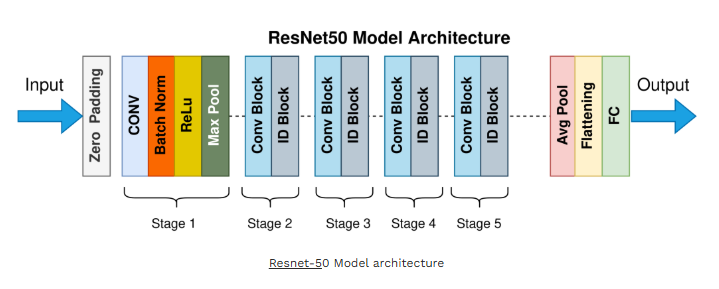
models. Design and Test an RCNN Model with a combination of ResNet/GoogleNet (depending

on the performance with respect to the testing data set) as feature extractors

# Model Building

# ResNet (Without Fine Tuned)

ResNet50 is a CNN architecture that is part of the ResNet (Residual Network) family, which was introduced by Kaiming and later it titled "Deep Residual Learning for Image Recognition" in 2015. ResNet50 is specifically a 50-layer deep neural network that utilizes residual learning to facilitate the training of very deep networks.



It is a powerful and efficient deep learning model that leverages residual connections to enable the training of very deep networks. Its architecture and performance make it a popular choice for many computer vision tasks.

## Create & Compile ResNet50 CNN Model:

## 

## 

## Set Epochs & evaluate the model predictions

## 

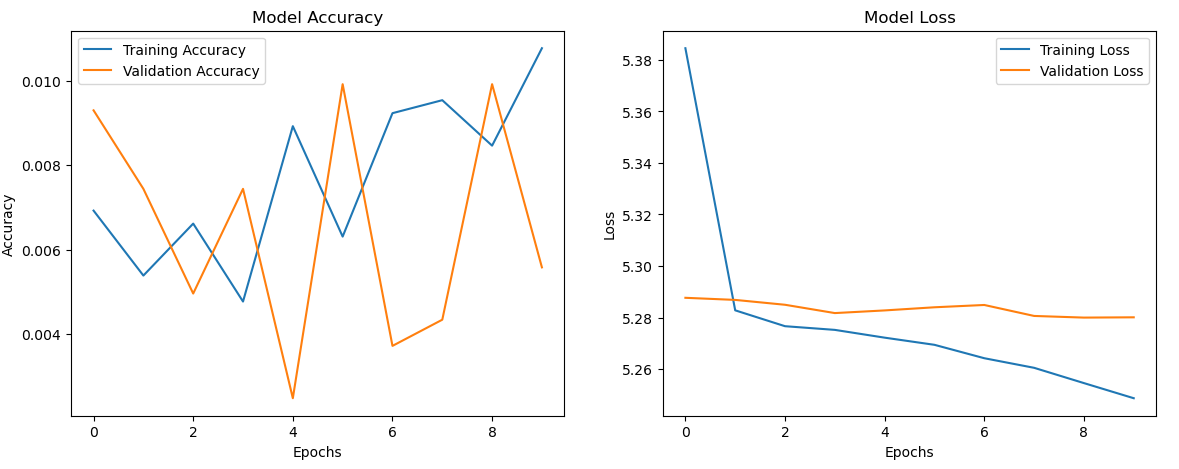
## 

## Code to plot model accuracy & loss graph

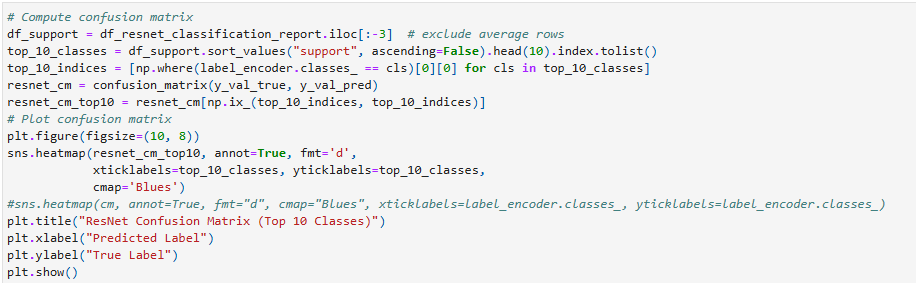
## 

## 

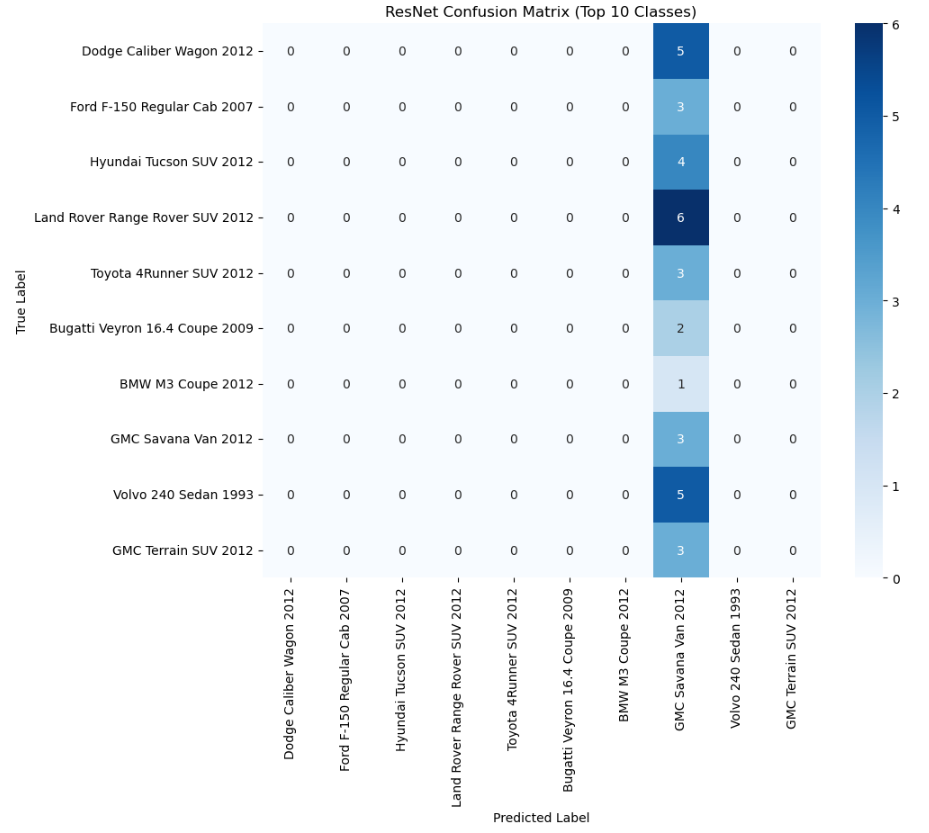
1. Graph for Model Accuracy Vs Model Loss



1. Code to create confusion matrix



Plot confusion matrix



**Summary of ResNet Model (Without Fine Tuned):**

**The model is not doing well due to**

1. Extremely low accuracy which indicates that the model is struggling to learn.
2. High Precision and Very low Recall indicating class imbalance issues

**Further actions could be**

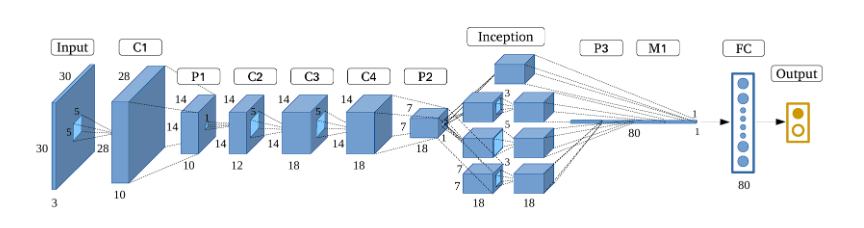
1. Check for class Imbalance
2. Fine tune parameters and retrain the model

## 

# GoogleNet (Without Fine Tuned)

GoogleNet is a CNN architecture that was introduced by researchers at Google in 2014 as part of the Inception project. It was designed to improve the efficiency and accuracy of deep learning models for image classification tasks. The architecture won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014, achieving state-of-the-art performance.

**Architecture Overview**



1. **Input Layer:** The input is typically a 224x224 RGB image.
2. **Convolutional Layers (C1-C4):** The initial layers consist of standard convolutional layers followed by max pooling.
3. **Inception:** Multiple inception modules are stacked together. Each module has branches with different convolutional filters and pooling operations.
4. **Global Average Pooling Layer (P1-P3):** After the last Inception module, a global average pooling layer is applied.
5. **Output Layer:** The final output is produced using a SoftMax layer for classification.

GoogleNet represents a significant advancement in deep learning architectures, particularly for image classification tasks. Its innovative use of Inception modules, 1x1 convolutions, and global average pooling has influenced many subsequent architectures and remains a foundational model in the field of computer vision.

## Create & Compile GoogleNet CNN Model:

## 

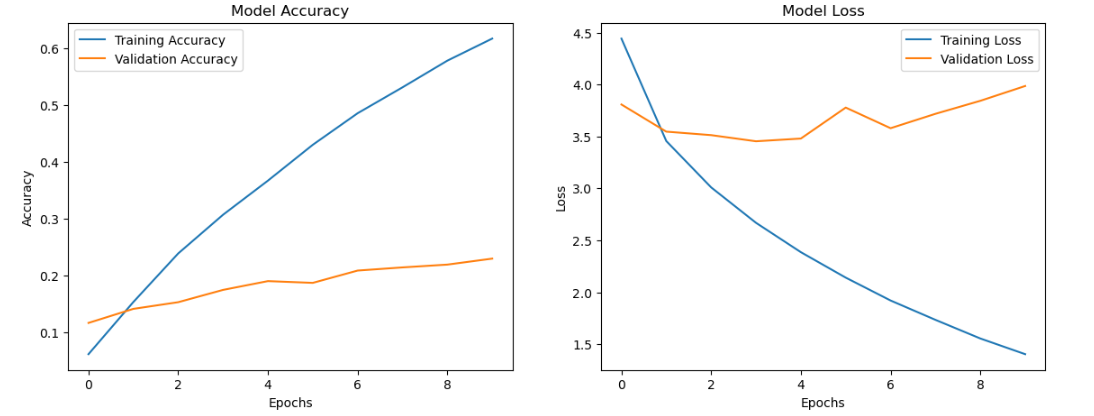
## Set Epochs & evaluate the model predictions

## 

## Code to plot model accuracy & loss graph

## 

1. Graph for Model Accuracy Vs Model Loss



## Code to create confusion matrix.

## 

Plot confusion matrix

## 

## **Summary of GoogleNet (Without Fine Tuned)**

## The inception modules allow the model to learn features at different scales, which can be beneficial for detecting cars of various sizes and orientations.

## Training Accuracy: Steadily increases, reaching ~75%. Validation Accuracy: Stagnates around 20-25%, indicating poor generalization. Training Loss: Decreases smoothly, showing effective learning on training data.

## Validation Loss: Plateaus and increases after a few epochs, a sign of overfitting.

## Classification Report Analysis: Overall Accuracy: 25%, indicating poor performance on validation data. Precision: ~41% Recall: ~26% (Very Low) F1-Score: ~24%

## Key Issues Identified: Indicates skewed performance, likely due to class imbalance. Overfitting. High Bias (Poor Performance on Validation Data) Potential Class Imbalance

## 3.1 Model Training

## ResNet (With Fine Tuned)

## Fine-tune the ResNet model for end-to-end object detection.

This is a modified CNN trained/fine-tuned on the car dataset for object detection and classification.

Code snippet of fine-tuning the ResNet Model

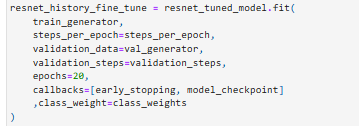


For model training, we initially used the original train images without augmentation and test

images with the batch size of 32, epochs 20. To improve the model performance, we will be

using augmented images, increase the number of epochs and use the callbacks with reduced

learning rates and early stopping.



## **ResNet Summary (With Fine Tuned)**

## Despite tuning, the ResNet model's performance declined, with accuracy dropping from 0.55% (untuned) to 0.43% (tuned).

## Precision remains misleadingly high due to sparse predictions, while recall and F1-scores are nearly zero in both cases — indicating that the model fails to generalize across classes.

## Which suggests issues such as label mismatch, improper preprocessing, or training data imbalance,

## leading to severe underfitting or poor class prediction confidence even after tuning.

## 

## GoogleNet (With Fine Tuned)

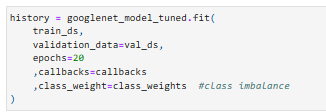
Fine-tune the GoogleNet model for end-to-end object detection.

This is a modified CNN trained/fine-tuned on the car dataset for object detection and classification.

Code snippet of fine-tuning the GoogleNet Model

## 

For model training, we initially used the original train images without augmentation and test images with the batch size of 16, epochs 20. To improve the model performance, we will be using augmented images, increase the number of epochs and use the callbacks with reduced learning rates and early stopping.



**GoogleNet Summary (With Fine Tuned)**

* The untuned GoogleNet model achieved an accuracy of 0.49%, slightly outperforming the tuned model, which dropped to 0.43%.
* Although macro and weighted precision appear high, the recall and F1-scores are nearly zero, confirming that the model rarely makes correct predictions.
* This indicates that tuning did not improve performance and may have disrupted learning, likely due to preprocessing inconsistencies or label mapping mismatches or fine-tuning strategies

## Model Saving

ResNet

Finally, both model weights are saved in h5 format



GoogleNet



## Model Evaluation

For the model evaluation, we looked at the loss and accuracy. Here are the comparisons of the two models:

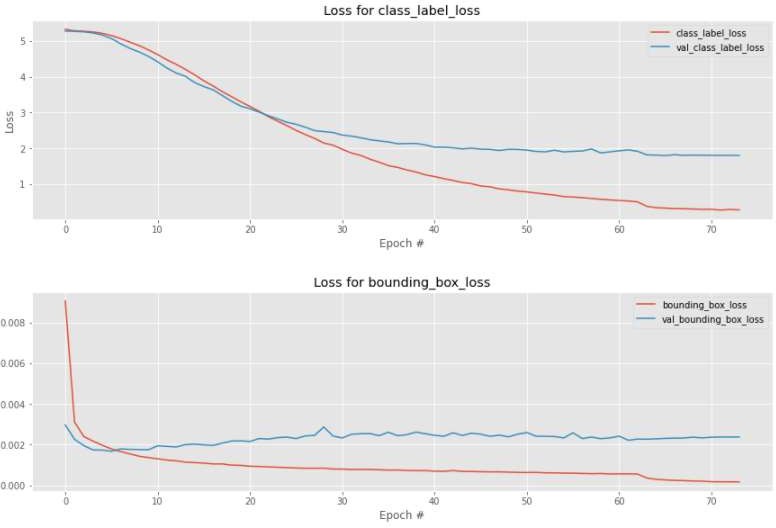
### VGG16 model: Mobile Net model:

|  |  |  |
| --- | --- | --- |
| Model Accuracy (%) | Bounding Box (Train/Test) | Class Label (Train/Test) |
| VGG (74 epochs with early stop) | 90 / 84.3 | 91.6 / **57.5** |
| Mobile Net (100 epochs) | 91.4 / 86.6 | 94.9 / 50.8 |

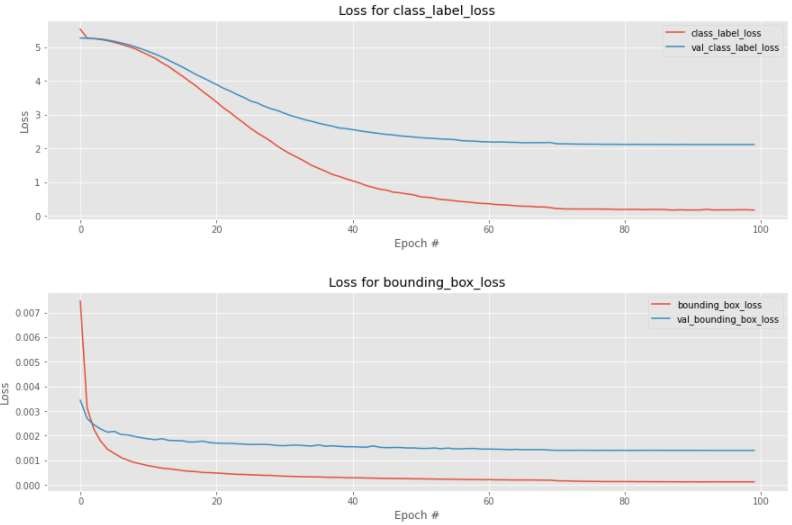
Both the models are showing over 80% accuracy for the Regression and over 50% accuracy for the Classification. VGG16 model is giving better class validation accuracy compared to the Mobile Net as shown in the figure. Bounding box accuracies are not too far between train and test for both the models but the class label accuracies are under predicted on validation set with 30% less accuracy.

Loss Comparison:

VGG16:



Mobile Net:



The losses are almost stabilized after 70 epochs in the both models.

VGG16 model was stopped earlier after 70 epochs but Mobile Net model could run completely up to 100 epochs.

# Comparison to Benchmark

Our benchmark for this object detection model is to predict the bounding box around the car and predict the correct label which is make, model and year of the car, for the input test image.

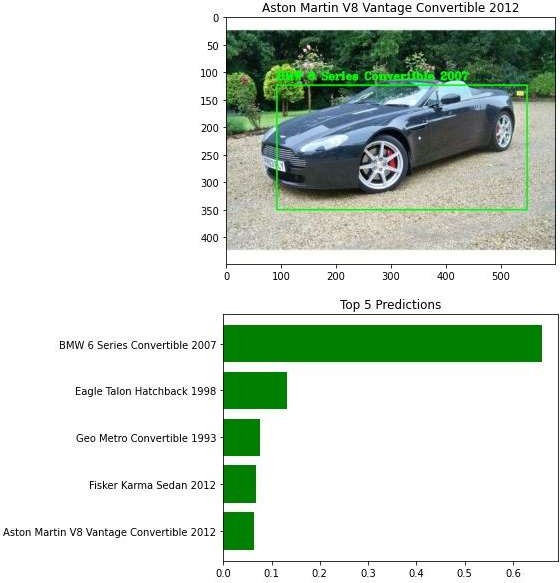
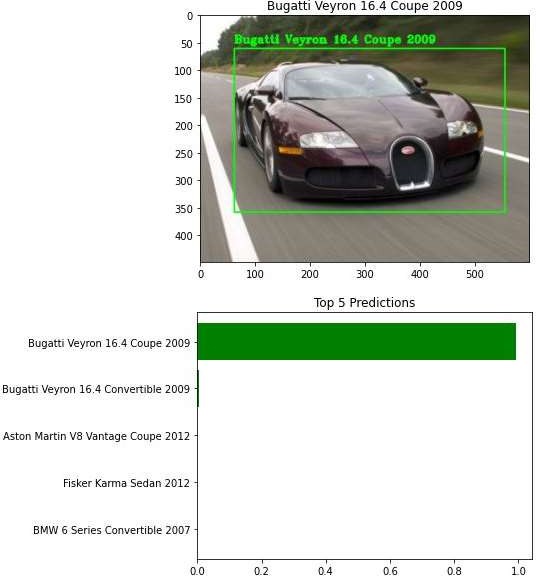
There are several competitions on Kaggle for the object detection where the objects are detected, localized and labeled for very generic categories. For the Stanford car dataset, there are models which showed 86% class label accuracy in just 4 epochs but those were not trained for predicting both bounding boxes and labels. The complexity in our task was to predict both in a single model without referring to any available models in the competitions or online. Though our model could produce satisfactory results, there is always an opportunity for the improvement.

With the given time and availability of resources we could train our model to achieve above 80% validation accuracy for the bounding boxes but only less than 60% accuracy on the class label prediction. That means, we are able to detect the car with the correct bounding boxes and fulfill our first requirement. And, we are able to predict the correct class label for almost 60% of the test images, which is not really bad after looking at the complexity of the various car constructions. Car styles vary from a sleek sports car to a large pickup truck under so many brands.

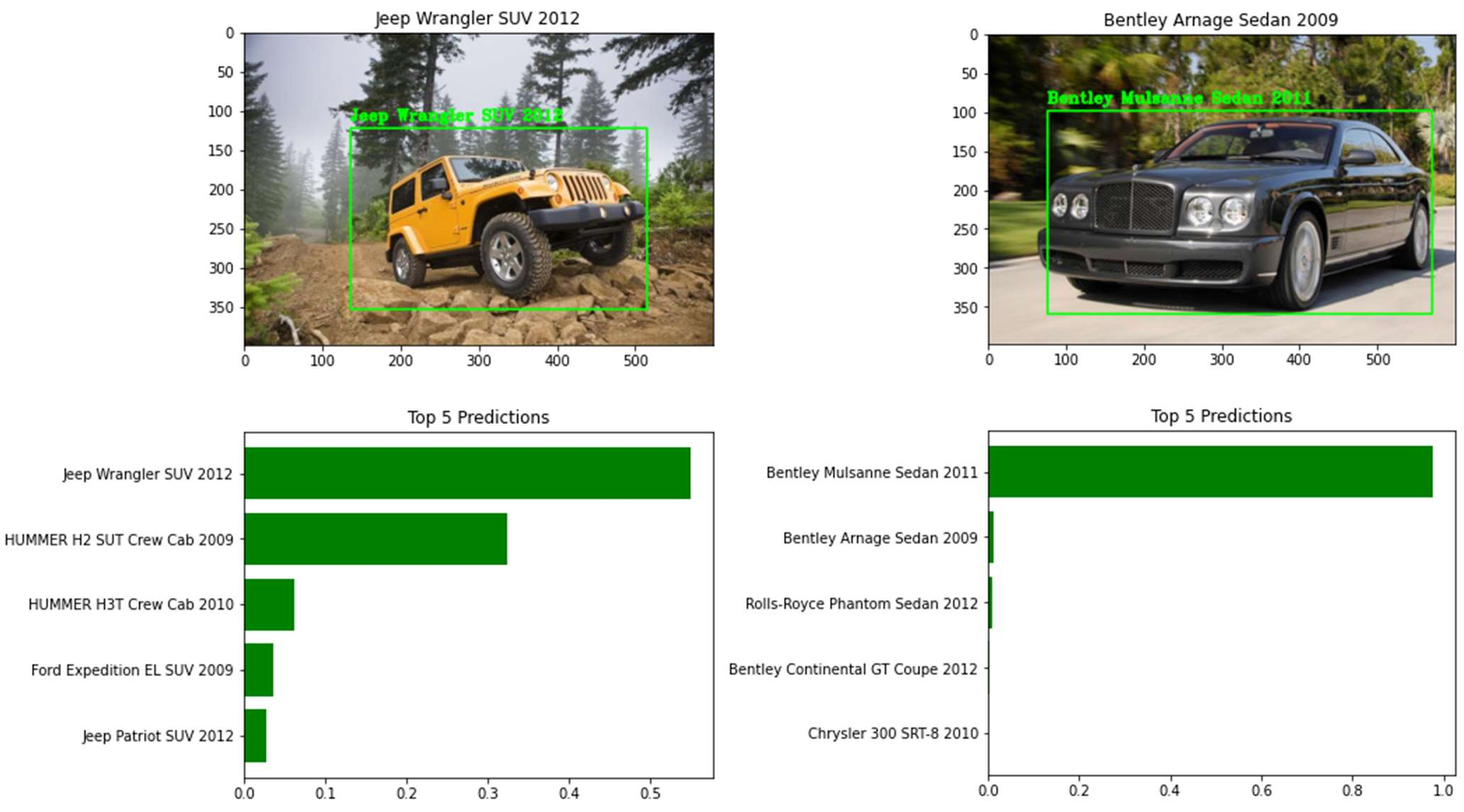
However, the model could predict the correct label for more than 90% of the test car images at least in top 5 predictions. Overall, the prediction for the bounding boxes and the class labels are satisfactory compared to the benchmark. The comparison of the predictions between VGG16 and Mobile net models are shown in the next step.

# Visualization of Model Predictions

**VGG** model could predict almost 60% of car labels correctly and it could predict the correct label at least in top5 predictions for more than 30% in the remaining cars.



**Mobile Net** model also giving similar results compared to VGG16 but with less accuracy.



## GUI Creation and Model Deployment

### Model Deployment:

In a typical machine learning and deep learning project, we usually start by defining the problem statement followed by data collection and preparation, understanding of the data, and model building.

But, in the end, we want our model to be available for the end-users so that they can make use of it. Model Deployment is one of the last stages of any machine learning project and can be a little tricky. How do you get your machine learning model to your client/stakeholder? What are the different things you need to take care of when putting your model into production? And how can you even begin to deploy a model?

Here comes the role of Flask. And, the following text will walk through all the steps in brief as our agenda is to deploy the model and use it on our localhost and it can be further extended to deploy on the web as well using Heroku platform.

### About Flask:

Flask is a web application framework written in Python. It has multiple modules that make it easier for a web developer to write applications without having to worry about the details like protocol management, thread management, etc.

Flask gives us a variety of choices for developing web applications and it gives us the necessary tools and libraries that allow us to build a web application.



### Installing Libraries:

If you have an anaconda installed, then you need to follow this to install all the required libraries in an environment. This will create a new environment. The environment name can be changed in the .yml file.



The new environment with respective dependencies will get created. Now, to use the newly created environment you need to activate it.

*Activating Environment*

**Command (CPU):** *conda object-detection-cpu*

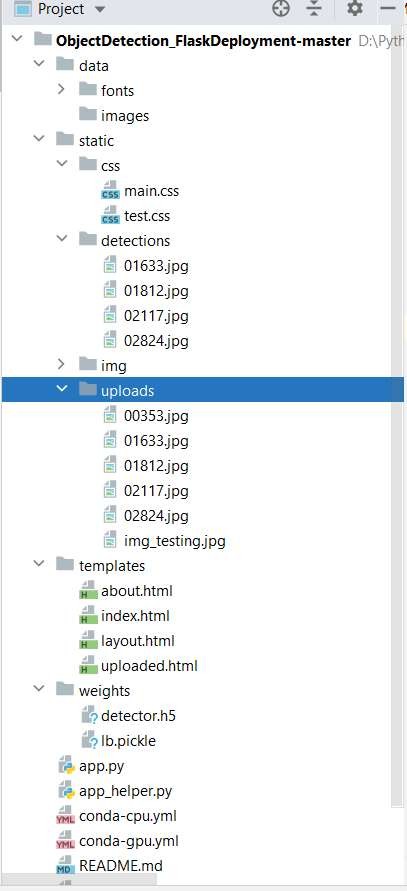
**Command (GPU):** *conda object-detection-gpu*

### Workflow:

Webpage template: - Here, we will design a user interface where the user can submit the query in the form of an image.

GUI: - Backend code will use the saved model weights to predict class & bounding boxes and send the results back to the webpage.

### GUI Folder structure:

****

**Explanation of input files:**

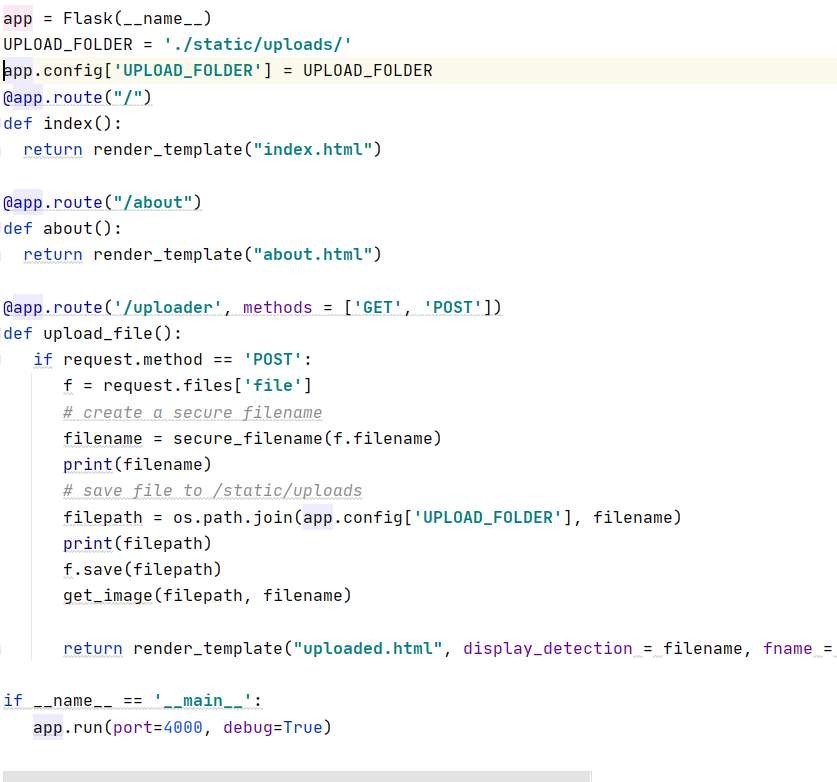
“ObjectDetectionFlaskDeployment” is the master directory and following are the sub directories. “/data” contains the frontend files such as fonts, pictures that are been used on the webpage. “/static/css” contains CSS related files for the graphics required on the webpage.

“/static/upload” contains images that the user to upload during the prediction through the web. “/static/detection” contains images with predicted class labels and bounding

boxes and these images are displayed on the web page.

“/templates” folder contains HTML files that are used in designing the user web interface. “/weights” contains model weight file -detector.h5 and labels binary pickle file- lb.pickle.

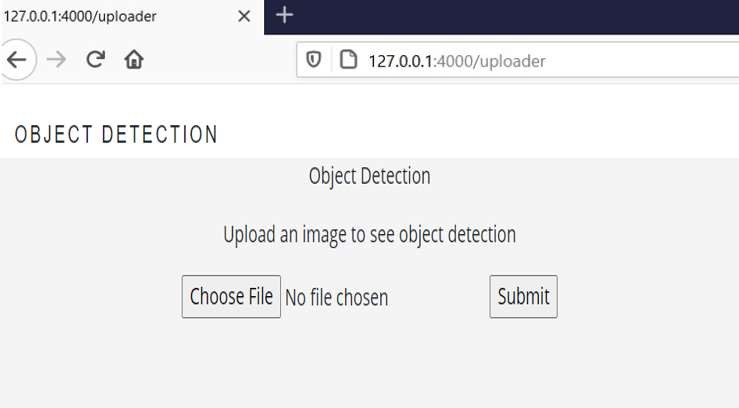
“/app.py” is the starting point where our flask starts servers running. This file contains a run method where it starts the whole application. Once this app starts, it invokes methods: GET & POST, from there it calls the function: upload\_files.



Once the above application up and running, it will provide one localhost URL as shown in the picture below.



Using the above URL, one can browse and upload the car image for the predictions. URL will redirect to the below page.



Once the car image is browsed using *choose File* option and hit *submit* then internally from app.py will call app\_helper.py file.

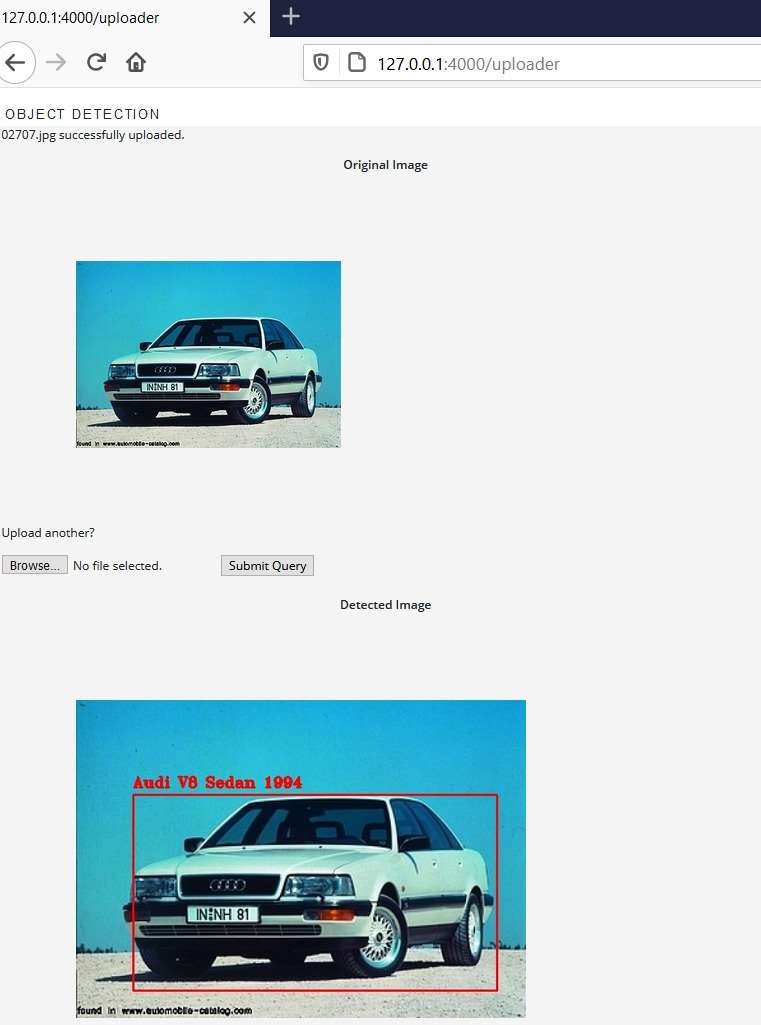
“app\_helper.py” file contains a helper function that helps in predicting the image using model weights (detector.h5) and label file(lb.pickle)

Once the file is uploaded into the above GUI and hit submit, internally it uses both weight and pickle files to find a class label (car name), bounding box coordinates for object detection.

Using the above output: class labels & bonding box coordinates, using cv2 functions, we are drawing a bounding box and putting the label on top of the image.

## GUI using HTML web page

After submitting the image, the GUI plots the input image and with the bounding box and car class label.



# Implications

This Car detection model could help in the automotive and surveillance domains to trace the exact car when spotted on the public or private places.

In the automotive domain, autonomous vehicles are expected to dramatically redefine the future of transportation. However, there are still significant engineering challenges to be solved before one can fully realize the benefits of self-driving cars. The challenge for self-driving car system is to identify 3D objects, which an important step before detecting their movement. One such challenge is building models that reliably predict the movement of traffic agents, such as cars, cyclists, and pedestrians. This car detection model should be able to detect the dimensions of the cars around and maintain the distance to avoid crash.

In the surveillance domain, the most common and well-known application from the category of traffic surveillance and law enforcement is license plate recognition. Due to growing demand, other categories of vehicle classification have also been added recently: Make, model and color of the car. Other application is to recognize the cars parked on roadside, which cause unnecessary blockage and create safety issues for other road users. The surveillance system requires computer vision-based technology to facilitate and enhance the accuracy and effectiveness of detection and recognition of vehicles. This is another application where our car detection model could help in solving the problem by detecting and predicting the correct car make, model and year. And it can be further developed to extract the text on the number plates as well.

An increasing number of vehicles and insufficient data transparency make the situation worse, and this issue is becoming critical in cities with high transportation density. Hence, there are increasing numbers of applications for computer vision car detection, However, current systems have limitations, such as being affected by vehicle speeds. To solve this problem, we need to record the videos, process the clippings and generate images, process them, train the models and make predictions. To perform all these tasks in no time, it requires very highspeed computation resources.

# Limitations

In this project, there were only few images available for each class of the cars but there are 196 such classes which makes it complex to train and predict the right class. With the limited data availability and resources, we could achieve only less than 60% accuracy in predicting the right class of the car. In the real world, there are several thousand models available and it’s really a challenge for our model to train. For a few thousand images, our model was taking lot of time on GPU and it will take huge amount of time to train for the cars in the real world.

Major challenges to implement our model in the real world as follow.

#1. Dual target: object classification and localization: The first major complication of object detection is its added goal: not only do we want to classify image objects but also to determine the objects’ positions, generally referred to as the object localization task.

#2. Speed for real-time detection: Object detection algorithms need to not only accurately classify and localize important objects, but they also need to be incredibly fast at prediction time to meet the real-time demands of video processing.

#3. Multiple spatial scales and aspect ratios: For many applications of object detection, items of interest may appear in a wide range of sizes and aspect ratios.

#4. Limited data: The limited amount of annotated data currently available for object detection proves to be another substantial hurdle. Object detection datasets typically contain ground truth examples for about a dozen to a hundred classes of objects, while image classification datasets can include upwards of 100,000 classes. Gathering ground truth labels along with accurate bounding boxes for object detection, however, remains incredibly tedious work.

#5. Class imbalance: It proves to be an issue for most classification problems, and object detection is no exception. Consider a typical photograph. More likely than not, the photograph contains a few main objects and the remainder of the image is filled with background.

To address these issues, we must use a multi-task loss function to penalize both misclassifications and localization errors. We need to boost the speed by using the faster algorithms like YOLO with test time of 155 frames per second (fps) which is faster than of R- CNN with 0.02 fps. We need to use several techniques to ensure detection algorithms capture objects at multiple scales and views.

# Closing Reflections

Computer vision is the ability of artificially intelligent systems to “see” like humans, it has been a subject of increasing interest and rigorous research for decades now. As a way of emulating the human visual system, the research in the field of computer vision purports to develop machines that can automate tasks that require visual cognition.

However, the process of deciphering images, due to the significantly greater amount of multi-dimensional data that needs analysis, is much more complex than understanding other forms of binary information. This makes developing AI systems that can recognize visual data more complicated.

Deep learning which is also based on machine learning require lots of mathematical and deep learning frameworks understanding. By using dependencies such as TensorFlow, Keras, OpenCV etc., we can detect each and every object in the image by the area object in a highlighted rectangular box and identify each and every object and assign its tag to the object.

This also includes the training of different deep learning models, the evaluation of the performance by the measure of accuracy of each model, selection of the best model suited for the application and the problem that we want to solve.

This project helped us as a team to strengthen our basics and understanding the significant aspects of the implementation of a computer vision project. Understanding how things work internally is crucial in computer vision because it helps us figure out how exactly the computer analyzes and processes the data as well as appreciate the beauty behind its methodologies.

Through this Computer vision Car detection project, we acquired a lot of knowledge by referring to various blogs and competitions. This is a very interesting subject for us to learn further and gain confidence to implement our knowledge in the real-world applications like automotive surveillance and especially in the autonomous vehicle development.

# References

Data Source: https://[www.kaggle.com/jutrera/stanford-car-dataset-by-classes-folder](http://www.kaggle.com/jutrera/stanford-car-dataset-by-classes-folder)

Image Augmentation with Bounding boxes: [https://medium.com/@a.karazhay/guide-augment-](https://medium.com/%40a.karazhay/guide-augment-) images-and-multiple-bounding-boxes-for-deep-learning-in-4-steps-with-the-notebook - 9b263e414dac

Training for both Regression & Classification: https://[www.pyimagesearch.com/2020/10/12/](http://www.pyimagesearch.com/2020/10/12/) multi-class-object-detection-and-bounding-box-regression-with-keras-tensorflow-and-deep- learning/

Plotting top 5 predictions: https://github.com/wengsengh/Car-Models-Classifier/blob/master/ car\_models\_classifier.ipynb

Real world challenges: https://towardsdatascience.com/5-significant-object-detection-challenges- and-solutions-924cb09de9dd