Models:

SSD:

Model Implementation & Results:

TensorFlow Object Detection API was used to Train, Evaluate, and Build the Model. MobileNet V2 (320x320) was used as the backbone Model. Due to system restriction, The model was trained for random 6 classes from the dataset. The batch size, and the number of steps were also reduced in the pipeline config file. The optimal batch size was 24 and the optimal number of steps was 2000 to train the model within the system restictions.

The best model had a MAP value of 0.605 for IoU between 0.50 and 0.95. The classification and object detection was decent in this iteration of the model. There is scope for improvement by increasing the number of steps and other hyperparameters based on system configuration.

Faster-RCNN:

Algorithms like R-CNN & Fast R-CNN use selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. But Faster-RCNN eliminates the selective search algorithm and lets the network learn the region proposals.

SSD models are faster on average but cannot beat the Faster R-CNN in accuracy

Model Implementation & Results:

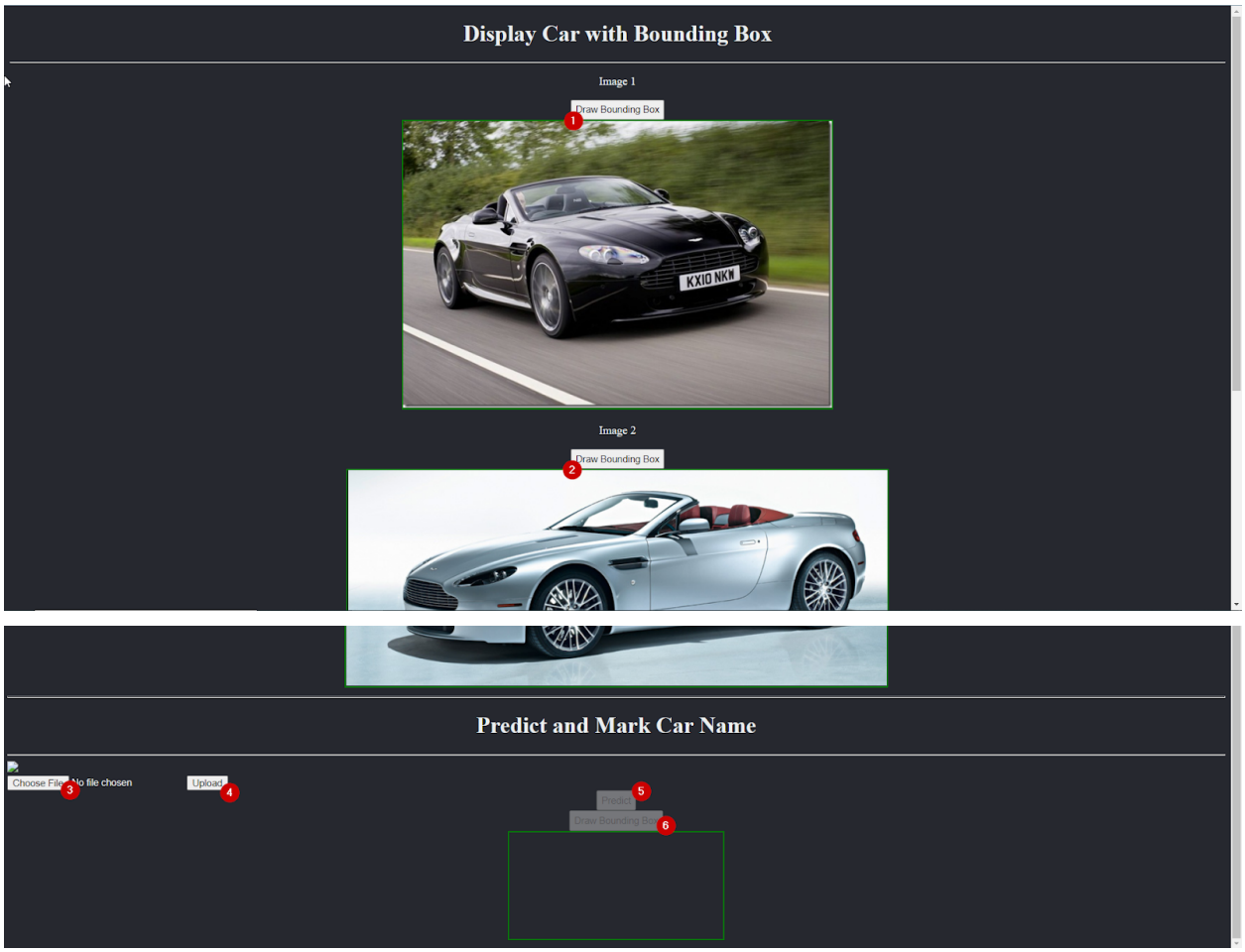
Similar to SSD, TensorFlow Object Detection API was used to Train, Evaluate, and Build the Model. Resnet101 v1 (640x640) was used as the backbone Model.

Hyperparameter based on system restriction -
No of classes trained: 6
Batch size: 12
No of steps: 500

Score:
MAP value of 0.379 for IoU between 0.50 and 0.95.

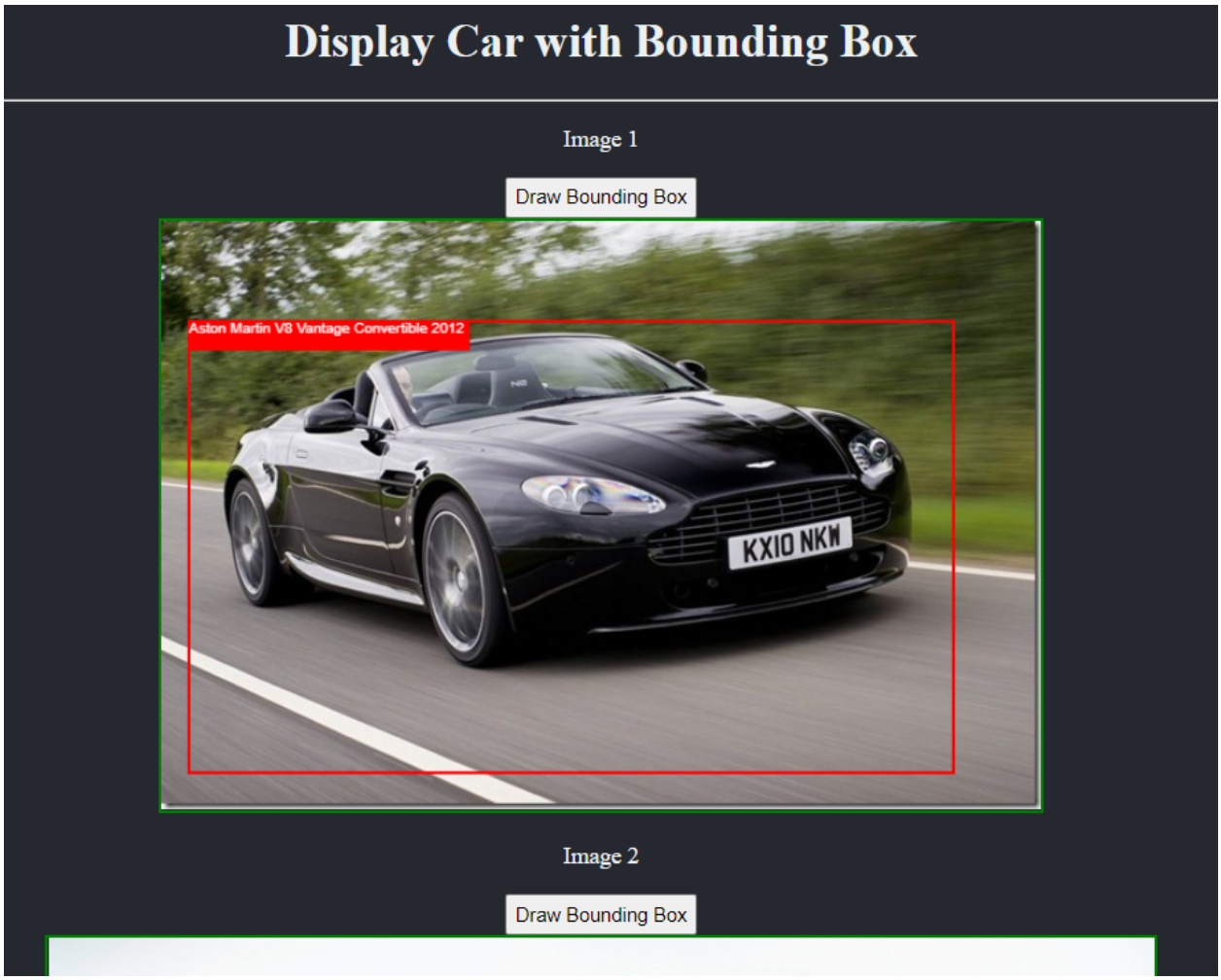
Clickable UI:

Created clickable Web UI to display images with bounding Box and detect the Car Name and Make on a New Input Image Using Angular Framework.



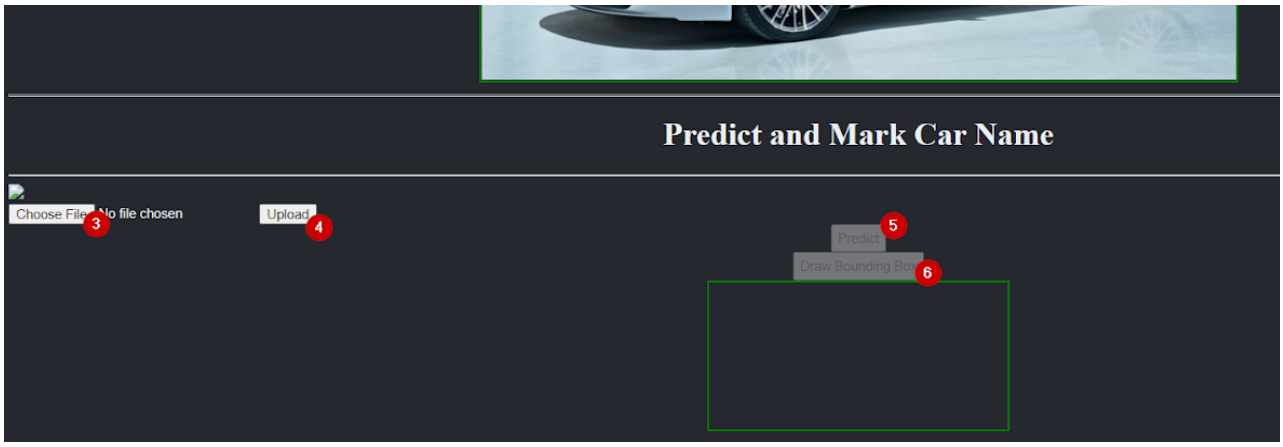
Part 1: Display Car with Bounding Box.

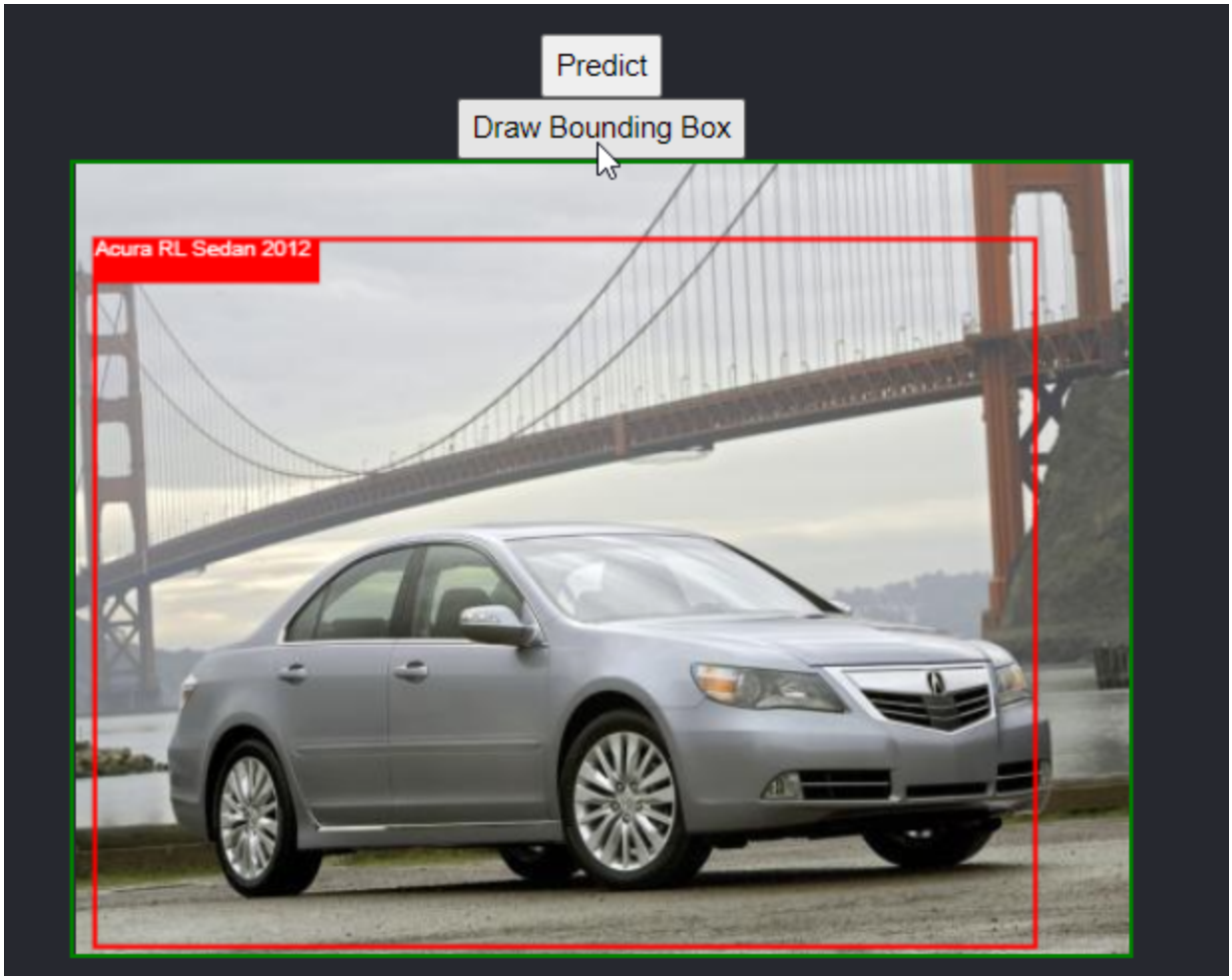
There are two images placed in the interface. On the click of “Draw Bounding Box” Button, a bounding box with car name and make is drawn on the image. The coordinates of box and class are preloaded in the code.



Part 2: Predict and Mark Car Name.

There is a “Choose File” button to browse through the local system files and “Upload” it to the Web UI. Clicking on the “Predict” button, the application loads “Model.json” and predicts the bounding box coordinates and the car name and make. Then Clicking on the “Draw Bounding Box” button shows the result on the uploaded image.





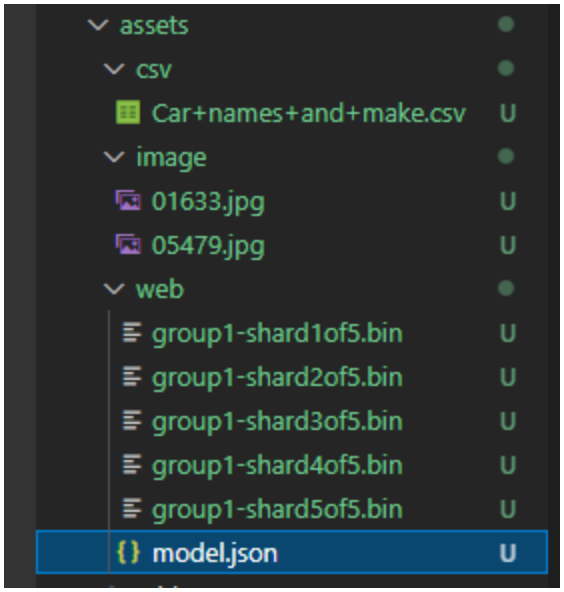
Model Building and Integration:

After training the model in a Python notebook, it was converted into .json format using tensorflowjs_converter script from **Tensorflow.js library**.

```
!tensorflowjs_converter \
    --input_format=tf_saved_model \
    --output_node_names='detection_boxes,detection_classes,detection_features,detection_multiclass_scores,detection_scores,num_detections
    --saved_model_tags=serve \
    --output_format=tfjs_graph_model \
    /content/drive/MyDrive/GL_AIML_Course/Projects/AIML_Learning/Capstone_Project/Notebooks/SavedModels_SSD/saved_model \
    /content/drive/MyDrive/GL_AIML_Course/Projects/AIML_Learning/Capstone_Project/Notebooks/SavedModels_SSD/detection_model/web_model
```

```
2022-03-20 06:25:56.586261: W tensorflow/core/common_runtime/gpu/gpu_bfc_allocator.cc:39] Overriding allow_growth setting because the TF_
Writing weight file /content/drive/MyDrive/GL_AIML_Course/Projects/AIML_Learning/Capstone_Project/Notebooks/SavedModels_SSD/detection_mod
```

The Class Labels, Preloaded Images and Model.json are loaded into the web application as shown below.



Tensorflow.js library is imported into Angular application.

```
import * as tf from '@tensorflow/tfjs';
```

On click of “Predict” button,

- the model is loaded,
- The image is loaded and preprocessed,
- The Output/result is predicted.

```
async Predict() {

  //Load Model
  const modelURL = '../assets/web/model.json';
  this.model = await tf.loadGraphModel(modelURL);
  let i = new Image();

  //Load Image
  const batched = tf.tidy(() => {
    i.src = this.UploadedImage;
    let img:any;
    if (!(i instanceof tf.Tensor)) {
      img = tf.browser.fromPixels(i);
      img = tf.cast(img, 'int32');
    }
    return img.expandDims(0);
  });
  const height = batched.shape[1];
  const width = batched.shape[2];

  //Predict Result
  const result = await this.model.executeAsync(batched);
  console.log(result);

  const scores = result[2].dataSync() as Float32Array;
  const boxes = result[4].dataSync() as Float32Array;
  const clas = result[1].dataSync() as Float32Array;

  this.out3 = { Image: i,ImageName: '', minX: boxes[0]*width, minY: boxes[1]*height,
   maxX: boxes[2]*width, maxY: boxes[3]*height,
    class: this.ClassName[clas[0]].toString(), score: scores[0] }

}
```

Challenges/Obstacles:

- The latest Angular version was not compatible with the tensorflow.js library. So we had to downgrade Angular CLI version to `8.2.8`
- The below error was encountered when tried to load the highest map score model i.e. SSD.

  Error: TensorList shape mismatch: Shapes -1 and 3 must match

Reference:

[Get Started | TensorFlow.js]

[Integration of TensorFlow JS model with Angular Application | by Kush Hingol | Analytics Vidhya | Medium]