Project Report

Deep learning based car detection model

Capstone project — interim report

Author(s)

Santosh KAKARLA, Prudhvi KALAVAKUNTA, Shraddha NANDAGAVE S, Sivateja RAMBHOTLA, Shyamdeep T

Deep learning based car detection model

DATE
26th June 2022

NUMBER OF PAGES AND ATTACHMENTS
26

ABSTRACT
The objective of this project is to develop a deep learning based classifier and object detector to detect cars of different makes and year. In this work, we use Resnet for classification purpose and Faster R-CNN for object detection purpose. We first discuss the problem statement and then present the data analysis extensively. Finally, we discuss the classifier and the object detection models used in this work along with their performance during prediction. The python code developed for solving this problem is submitted separately as '.ipynb' and 'html' files.

KEYWORDS:
Image classification, Object detection, CNN, R-CNN, Fast R-CNN, Faster R-CNN

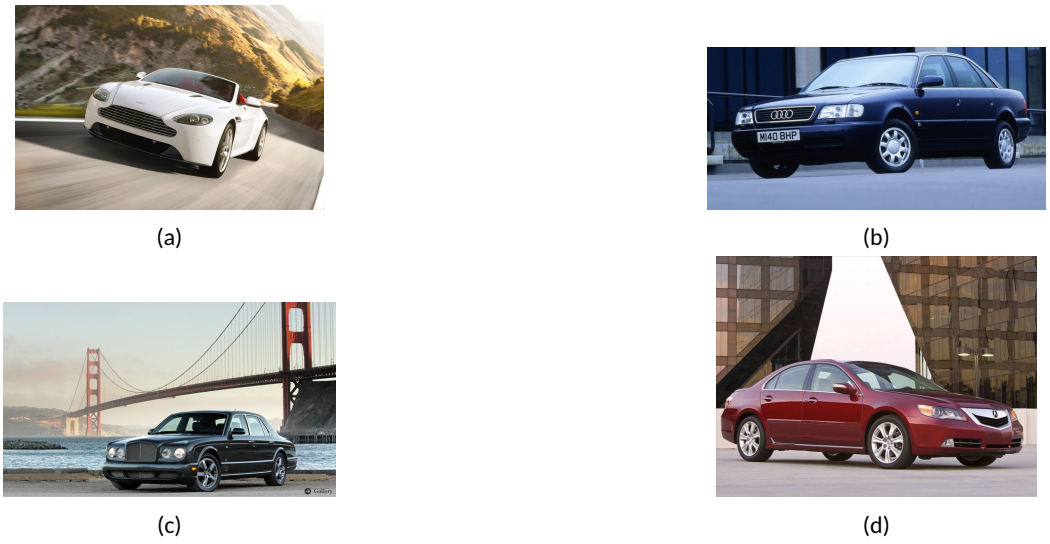
Contents

Chapter 1

Deep learning based car detection model

1.1 Data description

The given problem is related to the automotive field and surveillance. Moreover, the solution of the given problem is directly applicable to the self-deriving cars. The dataset consists of the 16,185 images of cars belonging to 196 different classes that are labeled according to their make and year. In addition to this, we are given the coordinates of the bounding box that localize the position of a car of particular class in the given image. The training and testing images consists of 8,144 and 8,041 samples respectively. In the Figure 1.1, we show some samples of the car images from the given dataset.



(a) (b) (c) (d)

Figure 1.1: Images of cars in the given dataset: a) Aston Martin V8 Vantage Convertible 2012, b) Audi 100 Sedan 1994, c) Bentley Arnage Sedan 2009, d) Acura RL Sedan 2012

The primary objective of this project is to develop a deep learning based object detection model that classifies cars according to their make and year.

1.2 Steps involved in the first milestone

As per the project document, the following steps are performed in the first milestone:

1. pre-processing of the data such as importing the data, mapping the given images to its labels and annotations,
2. Exploratory Data Analysis (EDA) in which we study the data extensively.

We explain these steps in detail in the following sections.

1.2.1 Pre-processing of the data

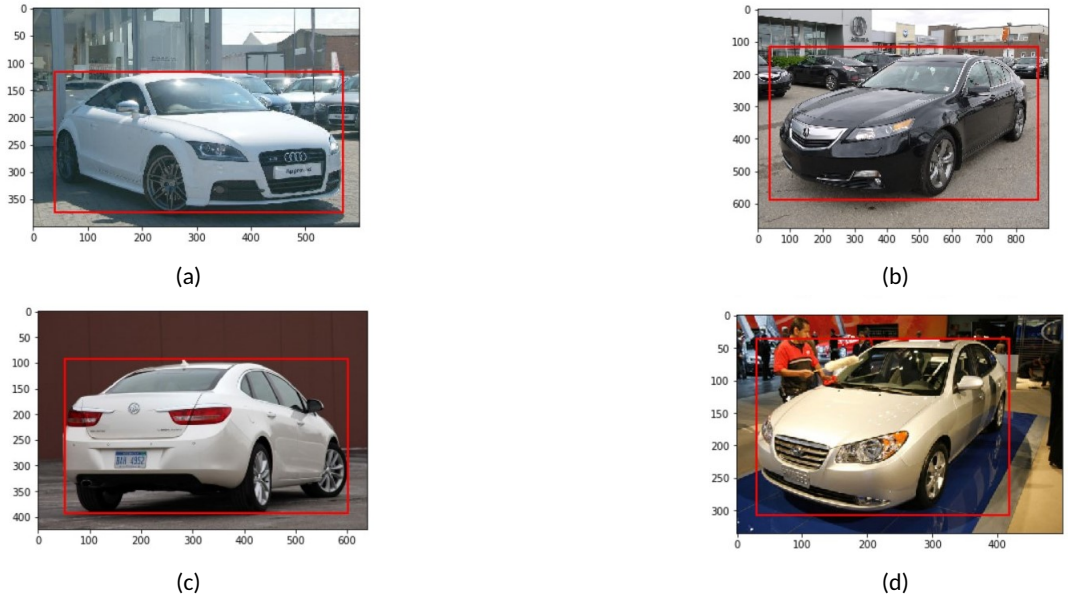
In this step, we perform the following activities:

1. **Import the data**: we used the tensorflow in-built method, $tf.keras.utils.image\_dataset\_from\_directory$ for this purpose. This method gives a tensorflow dataset that can be used during the training and testing process.
2. **Map training and testing images to its classes**: this step is automatically completed during the previous step as we get a tensorflow dataset with class names assigned automatically using the names of the folders in which the images are kept.
3. **Map training and testing images to its annotation**: in order to do this, we first import the annotation files as a dataframe using *pandas* library. Later, we use the class names obtained in the first step to map the correct annotations to its corresponding images.
4. **Display images with bounding box**: Here, we use the *imread* method of *cv2* library to read the files and then *imshow* method of *matplotlib* to display them. In the Figure 1.2, we show some images of the cars with the corresponding bounding box displayed on them.

1.2.2 Exploratory Data Analysis

In this step, we analyse the data in order to get some insights on various aspects of it. Here, we summarize the activities performed and their outcomes. More detailed results and figures can be found in the '.ipynb' and 'html' files submitted seperately.

1. Check for null values - we observed that there are no null values in the data,
2. Summary statistics - we present in Figure 1.2.2 the summary statistics of the data. Here, (x_0, y_0) and (x_1, y_1) are the bottom left and the top right coordinates of the bounding boxes respectively and the 'Year' is the year of making of the cars in the data. Rest of the statistical quantities are self explanatory. We observe that the pixel values of the bounding boxes goes from 65 to 5205 pixels suggesting that the range of the bounding boxes sizes is very large.



(a) (b) (c) (d)

Figure 1.2: Images of cars in the given dataset: a) Audi TTS Coupe 2012, b) Acura TL Sedan 2012, c) Buick Verano Sedan 2012, d) Hyundai Elantra Sedan 2007

	x0	y0	x1	y1	Year
count	8144.000000	8144.000000	8144.000000	8144.000000	8144.000000
mean	64.906803	108.661223	638.208620	416.431606	2009.559921
std	82.198684	104.551635	410.776734	273.786000	4.435031
min	1.000000	1.000000	76.000000	47.000000	1991.000000
25%	19.000000	42.000000	392.000000	248.000000	2008.000000
50%	39.000000	80.000000	569.000000	360.000000	2012.000000
75%	79.000000	138.250000	746.250000	477.000000	2012.000000
max	1648.000000	1508.000000	5205.000000	3389.000000	2012.000000

Figure 1.3: Summary statistics of the given data

3. Counting the unique no of items - we observed that there are repetitions in the bounding box coordinates, suggesting that the position of several cars in their respective images is uniform across several images.
4. Pairplot - while observing the pairplot, we can say that there are no striking patterns found out between any of the attributes of the given data.
5. Histogram - the histogram plots say that there is a very good balance between the classes in the given data. Also, it is extremely rare to find very large bounding boxes.
6. Barplot - The distribution of 'number of cars per year' attribute says that most of the cars whose images are collected are made after the year 2010.

1.3 Steps involved in the second milestone

There are two different tasks to be performed on the dataset during this project. They are:

1. classification of the cars according to the given labels,
2. object detection task that consists of both classifying a car according to the given labels and localizing it in the given image.

Image classification The purpose here is to build a machine learning model that can be trained on the given labeled images and be able to predict the class of the unseen images that contain similar objects in it. For the present dataset, we have labeled training and testing images that belong to 196 classes of cars. The machine learning model should be able to correctly predict the make along with the year of the car that embedded in the labels of the training and the testing images.

Object detection The difference between the image classification and the object detection is that the later deals with the localization of a particular object in the given image along with the classification of the object according to the training labels. The localization of a particular object is achieved using the 'bounding box' data. This data contains the coordinates of a rectangular (in general) box inside which the pixels belonging to the observed object in the given image are present.

1.3.1 Image classification models

In order to classify the given images using deep learning, Convolutional Neural Networks (CNNS) are employed. The basic architecture of CNN consists of the convolutional layers, the pooling layers, and the fully connected layers arranged in this particular order. The convolutional layer extracts the features from the image using kernels which have learnable parameters. These features that are extracted by the convolutional layer are more deeper (ex.512 layers) than the given image which may have 4 layers (3 channels related to colors and 1 channel related opaqueness) in general. The pooling layer then performs the dimension reduction on the features extracted by the convolutional layer. Finally, the fully connected neural network layers that also have learnable parameters will do the classification task. It has to be noted that the fully connected layer can also be used to perform regression. LeNet [1] is one of the earlier models that use the CNN architecture for image classification. Improved versions of this series include AlexNet [2], ZFNet [3] and VGGNet [4]. However, later models like GoogleNet [5] and ResNet [6] have more accuracy than the earlier models.

1.3.2 Object detection models

The Object detection models based on deep learning can be divided into the following categories:

1. models that use region proposals - R-CNN [7], Fast R-CNN [8], Faster R-CNN [9], Mask R-CNN [10], etc...

2. models that do not use region proposals - You Only Look Once (YOLO) [11], Single Shot Detector (SSD) [12], etc...

The general architecture of the object detection models that uses region proposals consists of a region proposal generator followed by an algorithm to extract the features in the given image and finally a machine learning models/methods to classify and regress the image and the bounding box coordinates. In the case of deep learning based object detection models, the neural networks are employed to perform these tasks.

R-CNN Here, the object detection is performed in 3 steps. In the first step, the selective search algorithm is used to propose 2000 Regions Of Interest (ROI). Then, CNNs are used to extract features from each of the region proposals independently. Finally, these features are sent to a pre-trained model to predict classes of different objects in the image. However, since it is a multi-stage model, R-CNN cannot be trained end-to-end. Also, the selective search algorithm is very expensive which makes it difficult for real-time deployment.

Fast R-CNN In this case, a new layer called ROI pooling layer is proposed. This allows to extract a fixed length feature vector from all ROI proposals. In this way, the CNN layers shares computations with all the ROIs only once unlike in the case R-CNN. This gives a faster performance than the R-CNN. However, it still uses the selective search algorithm for ROI proposals which is expensive.

Faster R-CNN This problem is alleviated by proposing the so called 'region proposal network (RPN)'. This is a CNN that proposes bounding boxes of different sizes and aspect ratios which are also termed as 'anchor boxes'. Finally, the proposed RPNs are then passed to Fast R-CNN to perform ROI pooling and classification purpose. In this way, we have an end-to-end model that can deployed for the real-time object detection.

YOLO In the case of the object detection models that do not use region proposals, for example YOLO, the bounding box coordinates and the class probabilities are obtained directly from the image pixels. Here, we first divide the image into different segments using a grid with predefined size. Next, we assume that a particular object is in the grid if the midpoint of that object is in the grid. We finally feed this to a CNN architecture to predict the bounding box coordinates and the classes of the objects.

1.3.3 Selection of models and their implementation

In this section, we describe the models used for the classification and object detection. We select the following baseline models:

1. ResNet for classification: this network implements the 'residual networks' which skip connections if the vanishing/exploding gradient is detected in the layers. In this manner, the accuracy of the model is increased.

2. Faster R-CNN for object detection: we select this architecture because this is an end-to-end architecture that does not need external algorithms to propose the ROIs.

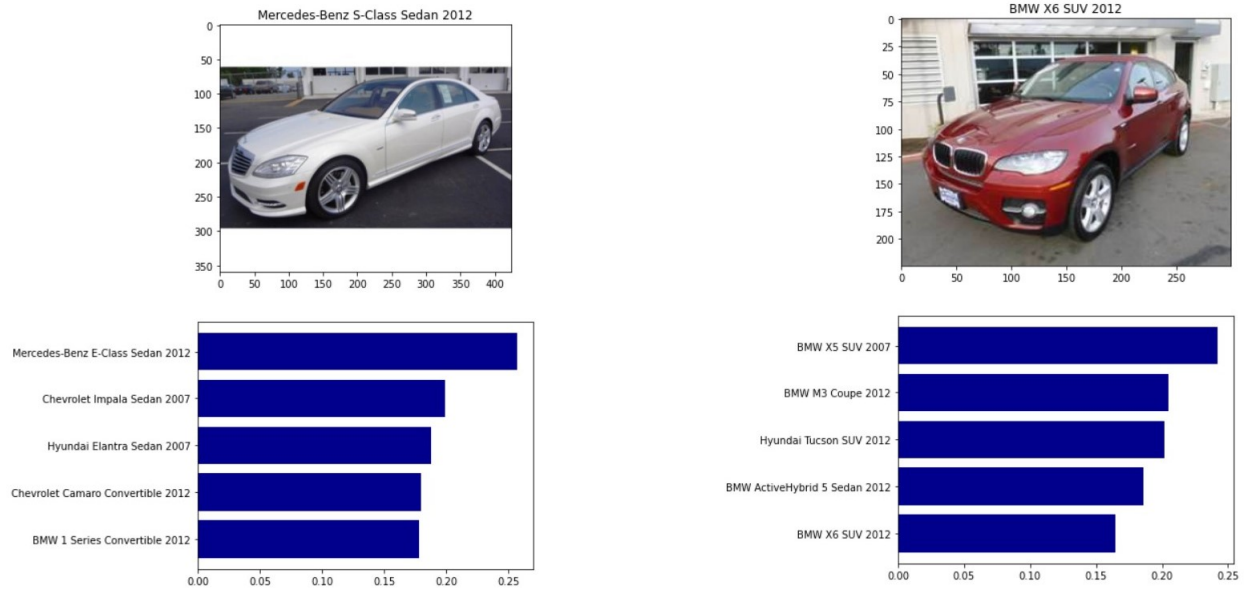
1.3.3.1 Image classification using ResNet

We describe here the steps involved in training and testing the ResNet classification model.

Pre-processing For this model, the image data that was imported during the first milestone is used as the input. For the sake of validation during training, we consider 20% of the training data. We transform the data by considering random rotations of 30^{deg} and horizontal flips. This transformations helps the model to generalize well on the data. Also, we resize all the images to 244×244 size.

Training We import a pretrained Resnet model from *torchvision* library. We set the number of classes to 196. Here, we use 'categorical cross-entropy' as the loss function for the classification. Finally, we obtained an accuracy of 96% on the validation dataset.

Testing During testing, we obtained 77% accuracy over all the classes. Some of the images along with the predicted probabilities are shown in the Figure 1.4.



(a) (b)

Figure 1.4: The correctly predicted class labels of the car images in the testing data, a) Mercedes-Benz S-Class Sedan 2012, b) BMW X6 SUV 2012

Evaluation metrics For the sake of evaluating the developed model, we consider precision, recall and f1-score. In the Table 1.1, we present the values obtained for the considered metrics per each class.

We observe that the class 61 is predicted with 100% precision with 97% recall and 0.99 as the f1-score. We also observe that the class 23 is predicted with the least precision of 43%.

Table 1.1: Values of the evaluation metrics of the image classification model with ResNet backbone.

Class	Precision	Recall	f1-score	Support
0	0.78	0.8863636363636364	0.8297872340425532	44.0
1	0.7755102040816326	0.8636363636363636	0.8172043010752688	44.0
2	0.6	0.65625	0.626865671641791	32.0
3	0.7872340425531915	0.8604651162790697	0.8222222222222222	43.0
4	0.6739130434782609	0.7380952380952381	0.7045454545454546	42.0
5	0.75	0.675	0.7105263157894738	40.0
6	0.8	0.717948717948718	0.7567567567567569	39.0
7	0.6274509803921569	0.7111111111111111	0.6666666666666666	45.0
8	0.5882352941176471	0.4878048780487805	0.5333333333333332	41.0
9	0.6666666666666666	0.6060606060606061	0.6349206349206349	33.0
10	0.7631578947368421	0.7631578947368421	0.7631578947368421	38.0
11	0.6	0.6	0.6	40.0
12	0.7631578947368421	0.6904761904761905	0.725	42.0
13	0.5892857142857143	0.8048780487804879	0.6804123711340206	41.0
14	0.7333333333333333	0.7674418604651163	0.7499999999999999	43.0
15	0.9310344827586207	0.75	0.8307692307692308	36.0
16	0.7111111111111111	0.7111111111111111	0.7111111111111111	45.0
17	0.42857142857142855	0.3076923076923077	0.3582089552238806	39.0
18	0.7	0.6666666666666666	0.6829268292682926	42.0
19	0.375	0.2857142857142857	0.3243243243243243	42.0
20	0.6981132075471698	0.8043478260869565	0.7474747474747474	46.0
21	0.631578947368421	0.6	0.6153846153846154	40.0
22	0.7666666666666667	0.5897435897435898	0.6666666666666667	39.0
23	0.43137254901960786	0.5238095238095238	0.4731182795698925	42.0
24	0.5833333333333334	0.6511627906976745	0.6153846153846155	43.0
25	0.875	0.8	0.8358208955223881	35.0
26	0.8157894736842105	0.7560975609756098	0.7848101265822786	41.0
27	0.75	0.6428571428571429	0.6923076923076924	42.0
28	0.8055555555555556	0.7073170731707317	0.7532467532467532	41.0
29	0.6857142857142857	0.5454545454545454	0.6075949367088607	44.0
30	0.96	0.7058823529411765	0.8135593220338982	34.0
31	0.7446808510638298	0.7954545454545454	0.7692307692307692	44.0
32	0.7555555555555555	0.8292682926829268	0.7906976744186047	41.0
33	0.46875	0.7317073170731707	0.5714285714285714	41.0
34	0.8648648648648649	0.8421052631578947	0.8533333333333334	38.0
35	0.6808510638297872	0.7804878048780488	0.7272727272727273	41.0
36	0.825	0.7857142857142857	0.8048780487804876	42.0
37	0.7647058823529411	0.65	0.7027027027027027	40.0
38	0.8095238095238095	0.8717948717948718	0.8395061728395062	39.0
39	0.6888888888888889	0.7045454545454546	0.6966292134831461	44.0
40	0.7291666666666666	0.7608695652173914	0.7446808510638298	46.0
41	0.8	0.5882352941176471	0.6779661016949153	34.0

Table 1.1: Values of the evaluation metrics of the image classification model with ResNet backbone.

Class	Precision	Recall	f1-score	Support
42	0.7560975609756098	0.8611111111111112	0.8051948051948052	36.0
43	0.7857142857142857	0.6285714285714286	0.6984126984126985	35.0
44	0.8387096774193549	0.8125	0.8253968253968254	32.0
45	0.75	0.8372093023255814	0.7912087912087912	43.0
46	0.8947368421052632	0.8095238095238095	0.8500000000000001	42.0
47	0.813953488372093	0.8333333333333334	0.8235294117647058	42.0
48	0.8	0.8	0.8000000000000002	35.0
49	0.7555555555555555	0.918918918918919	0.8292682926829269	37.0
50	0.8863636363636364	0.9069767441860465	0.896551724137931	43.0
51	0.7872340425531915	0.8409090909090909	0.8131868131868133	44.0
52	0.8333333333333334	0.8536585365853658	0.8433734939759037	41.0
53	0.5769230769230769	0.6666666666666666	0.6185567010309277	45.0
54	0.7272727272727273	0.7272727272727273	0.7272727272727273	44.0
55	0.8787878787878788	0.7073170731707317	0.7837837837837839	41.0
56	0.6666666666666666	0.6666666666666666	0.6666666666666666	39.0
57	0.9117647058823529	0.8378378378378378	0.8732394366197184	37.0
58	0.7755102040816326	0.8260869565217391	0.8	46.0
59	0.5	0.2413793103448276	0.32558139534883723	29.0
60	0.4411764705882353	0.42857142857142855	0.43478260869565216	35.0
61	1.0	0.9722222222222222	0.9859154929577464	36.0
62	0.6382978723404256	0.6976744186046512	0.6666666666666666	43.0
63	0.7435897435897436	0.7631578947368421	0.7532467532467534	38.0
64	0.8333333333333334	0.6818181818181818	0.7499999999999999	44.0
65	0.825	0.7333333333333333	0.776470588235294	45.0
66	0.6851851851851852	0.8809523809523809	0.7708333333333335	42.0
67	0.5813953488372093	0.5813953488372093	0.5813953488372093	43.0
68	0.6176470588235294	0.525	0.5675675675675677	40.0
69	0.6206896551724138	0.8181818181818182	0.7058823529411765	44.0
70	0.6333333333333333	0.5	0.5588235294117647	38.0
71	0.6875	0.75	0.717391304347826	44.0
72	0.7083333333333334	0.4594594594594595	0.5573770491803279	37.0
73	0.813953488372093	0.875	0.8433734939759036	40.0
74	0.7906976744186046	0.7727272727272727	0.7816091954022988	44.0
75	0.7209302325581395	0.6458333333333334	0.6813186813186812	48.0
76	0.782608695652174	0.8372093023255814	0.8089887640449438	43.0
77	0.875	0.813953488372093	0.8433734939759036	43.0
78	0.9333333333333333	0.9333333333333333	0.9333333333333333	45.0
79	0.8	0.8	0.8000000000000002	40.0
80	0.9393939393939394	0.8378378378378378	0.8857142857142858	37.0
81	0.8863636363636364	0.8666666666666667	0.8764044943820225	45.0
82	0.6808510638297872	0.7619047619047619	0.7191011235955055	42.0
83	0.5434782608695652	0.625	0.5813953488372093	40.0
84	0.9090909090909091	0.9302325581395349	0.9195402298850575	43.0
85	0.8048780487804879	0.8461538461538461	0.8250000000000001	39.0

Table 1.1: Values of the evaluation metrics of the image classification model with ResNet backbone.

Class	Precision	Recall	f1-score	Support
86	0.7567567567567568	0.6666666666666666	0.7088607594936708	42.0
87	0.6829268292682927	0.6829268292682927	0.6829268292682927	41.0
88	0.6486486486486487	0.631578947368421	0.64	38.0
89	0.8484848484848485	0.6829268292682927	0.7567567567567567	41.0
90	0.8478260869565217	0.8666666666666667	0.8571428571428571	45.0
91	0.9047619047619048	0.8837209302325582	0.8941176470588236	43.0
92	0.972972972972973	0.8181818181818182	0.8888888888888891	44.0
93	0.8205128205128205	0.8	0.8101265822784810	40.0
94	0.9142857142857143	0.7619047619047619	0.8311688311688312	42.0
95	0.6538461538461539	0.7727272727272727	0.7083333333333333	44.0
96	0.7027027027027027	0.6666666666666666	0.6842105263157895	39.0
97	0.7272727272727273	0.6956521739130435	0.7111111111111111	46.0
98	0.9310344827586207	1.0	0.9642857142857143	27.0
99	0.8888888888888888	0.7272727272727273	0.7999999999999999	33.0
100	0.8	0.717948717948718	0.7567567567567569	39.0
101	0.7352941176470589	0.5952380952380952	0.6578947368421053	42.0
102	0.7441860465116279	0.8205128205128205	0.7804878048780488	39.0
103	0.7619047619047619	0.7619047619047619	0.7619047619047619	42.0
104	0.8085106382978723	0.8837209302325582	0.8444444444444444	43.0
105	0.9473684210526315	0.972972972972973	0.9599999999999999	37.0
106	0.7254901960784313	0.8604651162790697	0.7872340425531916	43.0
107	0.8409090909090909	0.8409090909090909	0.8409090909090909	44.0
108	0.7959183673469388	0.8666666666666667	0.8297872340425533	45.0
109	0.8478260869565217	0.9285714285714286	0.8863636363636365	42.0
110	0.9487179487179487	0.9024390243902439	0.9249999999999999	41.0
111	0.7560975609756098	0.7380952380952381	0.746987951807229	42.0
112	0.8181818181818182	0.8	0.8089887640449439	45.0
113	0.8888888888888888	0.9090909090909091	0.8988764044943819	44.0
114	0.6981132075471698	0.8222222222222222	0.7551020408163266	45.0
115	0.775	0.7045454545454546	0.7380952380952381	44.0
116	0.7906976744186046	0.8095238095238095	0.8	42.0
117	0.8222222222222222	0.8409090909090909	0.8314606741573033	44.0
118	0.75	0.75	0.75	40.0
119	0.6744186046511628	0.8529411764705882	0.7532467532467532	68.0
120	0.813953488372093	0.8536585365853658	0.8333333333333333	41.0
121	0.7837837837837838	0.6904761904761905	0.7341772151898734	42.0
122	0.8780487804878049	0.8181818181818182	0.8470588235294119	44.0
123	0.8787878787878788	0.6744186046511628	0.7631578947368421	43.0
124	0.875	0.8974358974358975	0.8860759493670887	39.0
125	0.7837837837837838	0.7435897435897436	0.7631578947368421	39.0
126	0.7073170731707317	0.7631578947368421	0.7341772151898733	38.0
127	0.7857142857142857	0.8048780487804879	0.7951807228915663	41.0
128	0.8478260869565217	0.9285714285714286	0.8863636363636365	42.0
129	0.8	0.6666666666666666	0.7272727272727272	24.0

Table 1.1: Values of the evaluation metrics of the image classification model with ResNet backbone.

Class	Precision	Recall	f1-score	Support
130	0.7674418604651163	0.7857142857142857	0.7764705882352941	42.0
131	0.8048780487804879	0.7857142857142857	0.7951807228915663	42.0
132	0.7894736842105263	0.7142857142857143	0.7500000000000001	42.0
133	0.7659574468085106	0.8372093023255814	0.8	43.0
134	0.9428571428571428	0.7857142857142857	0.8571428571428571	42.0
135	0.8	0.8484848484848485	0.823529411764706	33.0
136	0.8055555555555556	0.7435897435897436	0.7733333333333334	39.0
137	0.8222222222222222	0.8604651162790697	0.8409090909090908	43.0
138	0.8125	0.6341463414634146	0.7123287671232876	41.0
139	0.6046511627906976	0.6190476190476191	0.611764705882353	42.0
140	0.75	0.7058823529411765	0.7272727272727272	34.0
141	0.9	0.84375	0.870967741935484	32.0
142	0.813953488372093	0.875	0.8433734939759036	40.0
143	0.6779661016949152	0.8695652173913043	0.7619047619047619	46.0
144	0.868421052631579	0.7857142857142857	0.825	42.0
145	0.7619047619047619	0.7111111111111111	0.735632183908046	45.0
146	0.8333333333333334	0.7954545454545454	0.8139534883720929	44.0
147	0.8372093023255814	0.8181818181818182	0.8275862068965518	44.0
148	0.8723404255319149	0.9534883720930233	0.9111111111111112	43.0
149	0.8648648648648649	0.7441860465116279	0.8	43.0
150	0.8	0.9090909090909091	0.8510638297872342	44.0
151	0.9032258064516129	0.8	0.8484848484848486	35.0
152	0.8108108108108109	0.8333333333333334	0.8219178082191781	36.0
153	0.8974358974358975	0.8333333333333334	0.8641975308641975	42.0
154	0.8695652173913043	0.9523809523809523	0.909090909090909	42.0
155	0.8823529411764706	0.7692307692307693	0.8219178082191781	39.0
156	0.875	0.9722222222222222	0.9210526315789473	36.0
157	0.7586206896551724	0.7586206896551724	0.7586206896551724	29.0
158	0.7837837837837838	0.8055555555555556	0.7945205479452055	36.0
159	0.7924528301886793	0.9545454545454546	0.865979381443299	44.0
160	0.7288135593220338	0.8958333333333334	0.8037383177570093	48.0
161	0.8260869565217391	0.8444444444444444	0.8351648351648352	45.0
162	0.5208333333333334	0.5813953488372093	0.5494505494505495	43.0
163	0.66	0.75	0.702127659574468	44.0
164	0.7894736842105263	0.8333333333333334	0.8108108108108109	36.0
165	0.6739130434782609	0.7560975609756098	0.7126436781609194	41.0
166	0.6956521739130435	0.6808510638297872	0.6881720430107526	47.0
167	0.7241379310344828	0.9130434782608695	0.8076923076923076	46.0
168	0.782608695652174	0.8181818181818182	0.8	44.0
169	0.8837209302325582	0.9047619047619048	0.8941176470588236	42.0
170	0.9459459459459459	0.9210526315789473	0.9333333333333332	38.0
171	0.8125	0.8863636363636364	0.8478260869565218	44.0
172	0.7948717948717948	0.7209302325581395	0.7560975609756098	43.0
173	0.8205128205128205	0.7804878048780488	0.8	41.0

Table 1.1: Values of the evaluation metrics of the image classification model with ResNet backbone.

Class	Precision	Recall	f1-score	Support
174	0.6666666666666666	0.7894736842105263	0.7228915662650601	38.0
175	0.7619047619047619	0.5333333333333333	0.6274509803921569	30.0
176	0.6956521739130435	0.7272727272727273	0.711111111111111	44.0
177	0.8	0.7804878048780488	0.7901234567901235	41.0
178	0.8292682926829268	0.7555555555555555	0.7906976744186047	45.0
179	0.8055555555555556	0.6904761904761905	0.7435897435897436	42.0
180	0.7647058823529411	0.6842105263157895	0.7222222222222222	38.0
181	0.7446808510638298	0.7608695652173914	0.7526881720430109	46.0
182	0.7115384615384616	0.8809523809523809	0.7872340425531914	42.0
183	0.6111111111111112	0.55	0.5789473684210527	40.0
184	0.6829268292682927	0.7368421052631579	0.7088607594936709	38.0
185	0.825	0.825	0.825	40.0
186	0.8333333333333334	0.813953488372093	0.8235294117647058	43.0
187	0.6041666666666666	0.6744186046511628	0.6373626373626373	43.0
188	0.825	0.868421052631579	0.8461538461538461	38.0
189	1.0	0.8809523809523809	0.9367088607594937	42.0
190	0.8367346938775511	0.8913043478260869	0.8631578947368421	46.0
191	0.96875	0.7209302325581395	0.8266666666666667	43.0
192	0.9318181818181818	0.9111111111111111	0.9213483146067416	45.0
193	0.717391304347826	0.8048780487804879	0.7586206896551724	41.0
194	0.8235294117647058	0.6511627906976745	0.7272727272727273	43.0
195	0.9736842105263158	0.925	0.9487179487179489	40.0

1.3.3.2 Object detection using Faster R-CNN

Here[1], we describe in detail the baseline architecture considered for the Faster R-CNN and also the parameters chosen for training the model.

Pre-processing In the case of Faster R-CNN, we have to give the annotation data along with the image data as the input to the model. As a baseline models, we use a pretrained 'VGG16' and 'ResNet50' CNN architecture for feature extraction part.

Training During the training process, we consider the anchor boxes of size 64, 128 and 256 and their variants with three ratios of $1:1$, $1:\frac{1}{\sqrt{2}}$ and $\frac{1}{\sqrt{2}}:1$ for the RPN layer. We also set the threshold of Intersection Over Union (IOU) for selecting positive and negative bounding boxes among those proposed by RPN to the value of 0.7. Later, we pass these region proposals to the ROI pooling layer that extract 512 features in the case of 'VGG16' and 1024 features in the case of 'ResNet50'. Finally, we choose 'binary cross-entropy' and 'categorical cross-entropy' as the loss functions for the

[1]Considering the timeline of the project and absence of availability of an open source API (according to the knowledge of the Authors), we use a publicly available libraries for Faster R-CNN in this GitHub account.

regression and the classification layers respectively. After the training is completed, we save the weights of both the RPN layer and the classifier layer to be used during the testing phase.

Testing During this phase, we use the given $8,041$ testing images. Unfortunately, we found out that the models with backbones as both 'VGG16' and 'ResNet50' predict only the background in the testing images but not the car in the image. The predicted results for models with backbones as both 'VGG16' and 'ResNet50' are shown in the Figure 1.5 and Figure 1.6 respectively. One possible reason for this is that the model is over-fitting in the training phase. This prevents the model from generalizing on the unseen images. We will propose the possible remedies for this problem in the section 1.5.

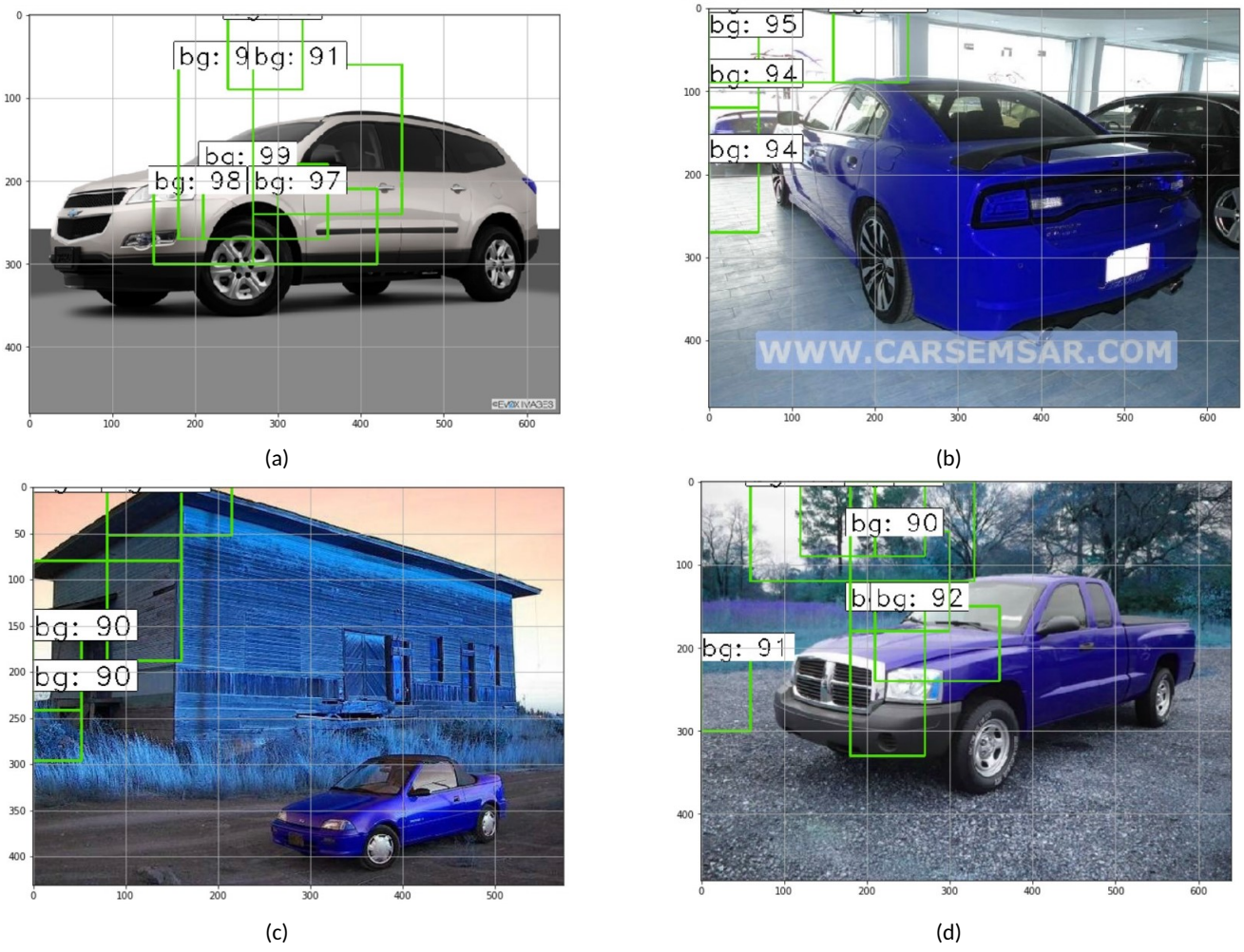


Figure 1.5: The predicted bounding boxes by the Faster R-CNN model with 'VGG16' as backbone on the testing data

1.3.3.3 Object detection using YOLOv5

Pre-processing The data preparation[2] involves reformatting the annotation data so that the centroid of the bounding box along with its height and width are provided to the algorithm. Along with this,

[2]We consider the YOLOv5 methodology as implemented in this GitHub account

(a) (b) (c) (d)

Figure 1.6: The predicted bounding boxes by the Faster R-CNN model with 'ResNet50' as backbone on the testing data

a file named '$cars\_data.yaml$' which consists of the information related to the location of the training and testing data along with the class names is to be provided.

Training By considering the run time of the model, here, we choose 'YOLOv5s' model which is the smallest of all the versions of this model with 270 layers and 7548241 parameters. For the purpose of training the model, we consider 640×640 as the image size. In order to make the model as general as possible, we consider rotations of 30^{deg} and also $0.1, 0.5$ as the factors for image translation and image scaling respectively. An IoU value of 0.2 is considered for distinguishing various bounding boxes. YOLOv5 algorithm also uses mosaics of the given images as shown in the Figure 1.7. The evolution of different losses during the training and validation is shown in Figure 1.8.

Testing Similar to the previous case of Faster R-CNN, the model is unable to predict any bounding boxes on the testing images. We explore some reasons for this in the section 1.5.

Evaluation metrics In Figure 1.9, we present evaluation of R-curve, P-curve, PR-curve and f1-curve. However, it is difficult to conclude on this evolution.

Figure 1.7: Mosaics of the given images used by the YOLOv5 algorithm

1.3.4 Benchmark comparison

We compare the accuracy of our classification model with the benchmark model proposed in [13]. We observe that the accuracy obtained using our classification model is 77% which is significantly higher than the one reported in the benchmark paper which is 67%. The reason for this improvement is due to the fact that the deep learning model built here generalize well on the unseen datasets when compared to the statistical models used in the benchmark paper [13].

1.4 Steps involved in the third milestone

In this section, we present the User Interfaces (UIs) developed to automate the steps performed in the two milestones. That is, for loading, mapping and printing data as well as for training and testing process. Here, we used *Tkinter* library to develop desktop-based UIs that run locally. In the below sections, we present the essential information about these UIs and their outputs.

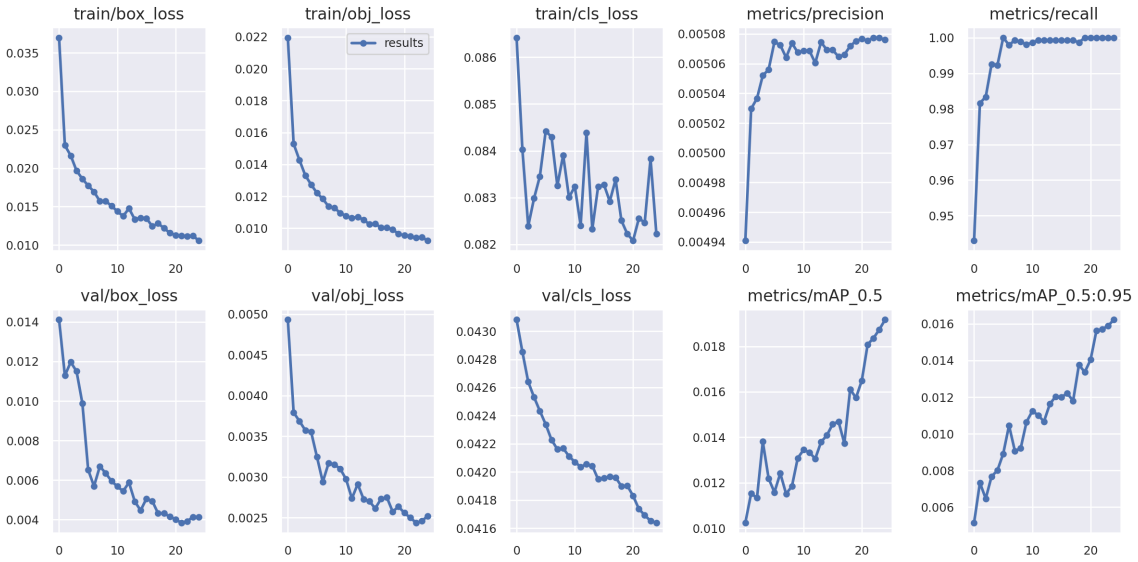


Figure 1.8: Evolution of different losses during the training and validation using the YOLOv5 algorithm

1.4.1 UI for loading, mapping and printing data

The code for automatizing the steps in the first milestone is contained in the $'UI\_milestone\_1.py'$ file. We assume that the training images and testing images are stored in the relative locations of this file as shown in the following variables in the code:

$$directory =' ./CarImages/TrainImages'$$
$$training_annotations = pd.read\_csv('./Annotations/TrainAnnotations.csv')$$

We point out that the above lines in the code has to be changed in case of any other storage paths other than those. The UI has three buttons, namely, *Load data*, *Map classes and annotations* and *Print image*. The outputs of the buttons are shown in Figure 1.14 and Figure 1.15.

1.4.2 UI for training the models

The UI for automating the training process takes the number of epochs as the input and trains the image classifier model with 'Rsenet30' architecture and object detection models (Faster R-CNN) with 'VGG16' and 'ResNet50' backbones. The corresponding buttons are displayed on the UI so that the model of interest can be selected. The corresponding code can be found in the file named $UI\_milestone\_2\_training.py$. The information about different losses during the epochs are shown in the output window. For the sake of illustration, screenshots of the outputs of the buttons related to classification and object detection models are presented in Figure 1.12 and Figure 1.13. Please note that the outputs (epoch information) for the *Train Faster R-CNN with VGG16* and *Train Faster R-CNN with ResNet50* buttons are the same.

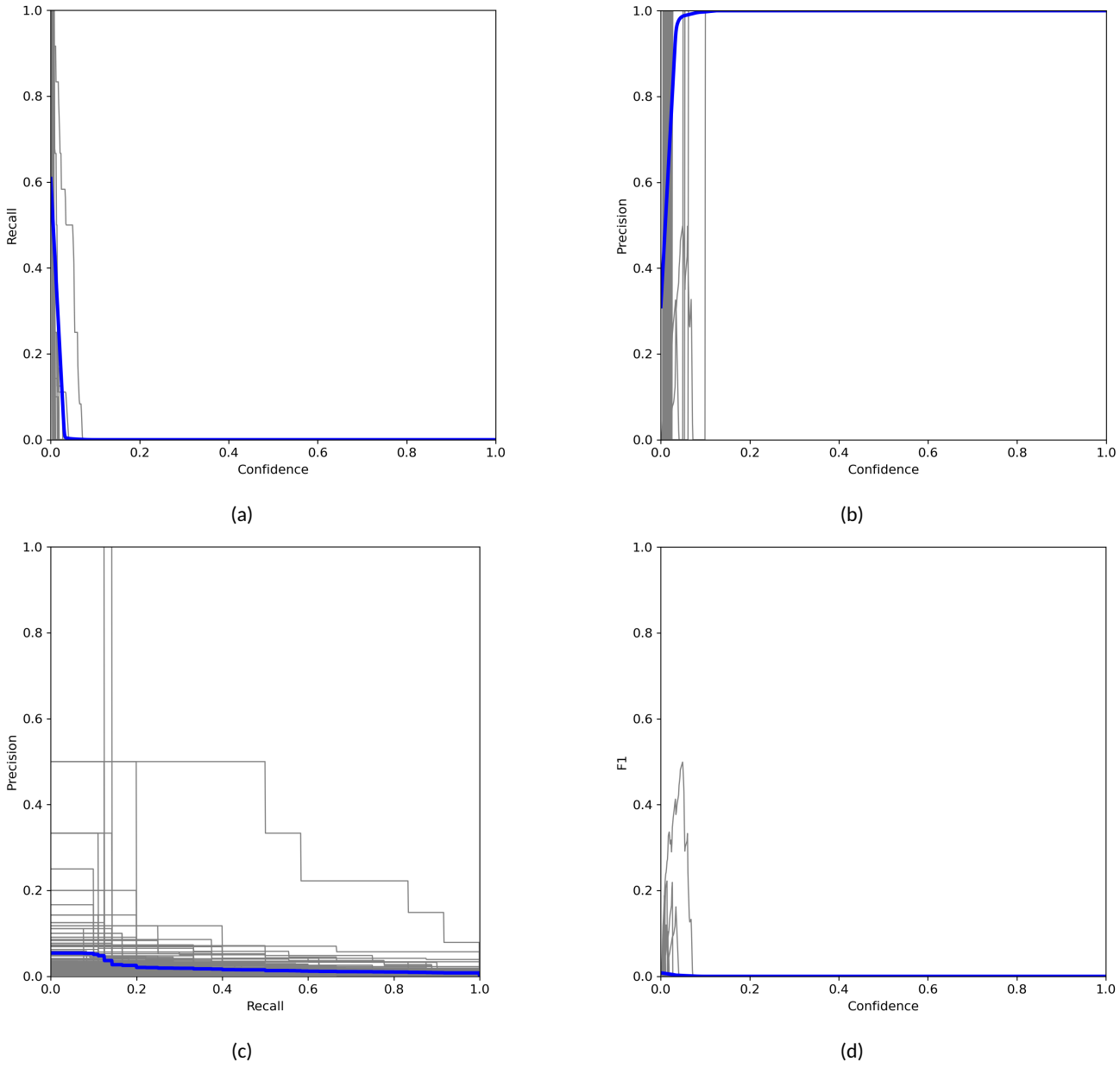


Figure 1.9: Evolution of different metrics used for the evaluation of the YOLOv5 algorithm

1.4.3 UI for testing the models

The input for this UI is the path to the *'.jpg'* file of the image of consideration. There are three buttons named *Predict class of the car (ResNet34)*, *Identify the object (ResNet50)* and *Identify the object (VGG16)*. Upon clicking the *Predict class of the car (ResNet34)* button the probabilities of predicted classes of the car in the image are displayed in the UI window. The other two buttons give the bounding box coordinates. The respective outputs are shown in the Figure 1.14 and Figure 1.15 while the corresponding code is in the $UI\_milestone\_2\_testing.py$ file.

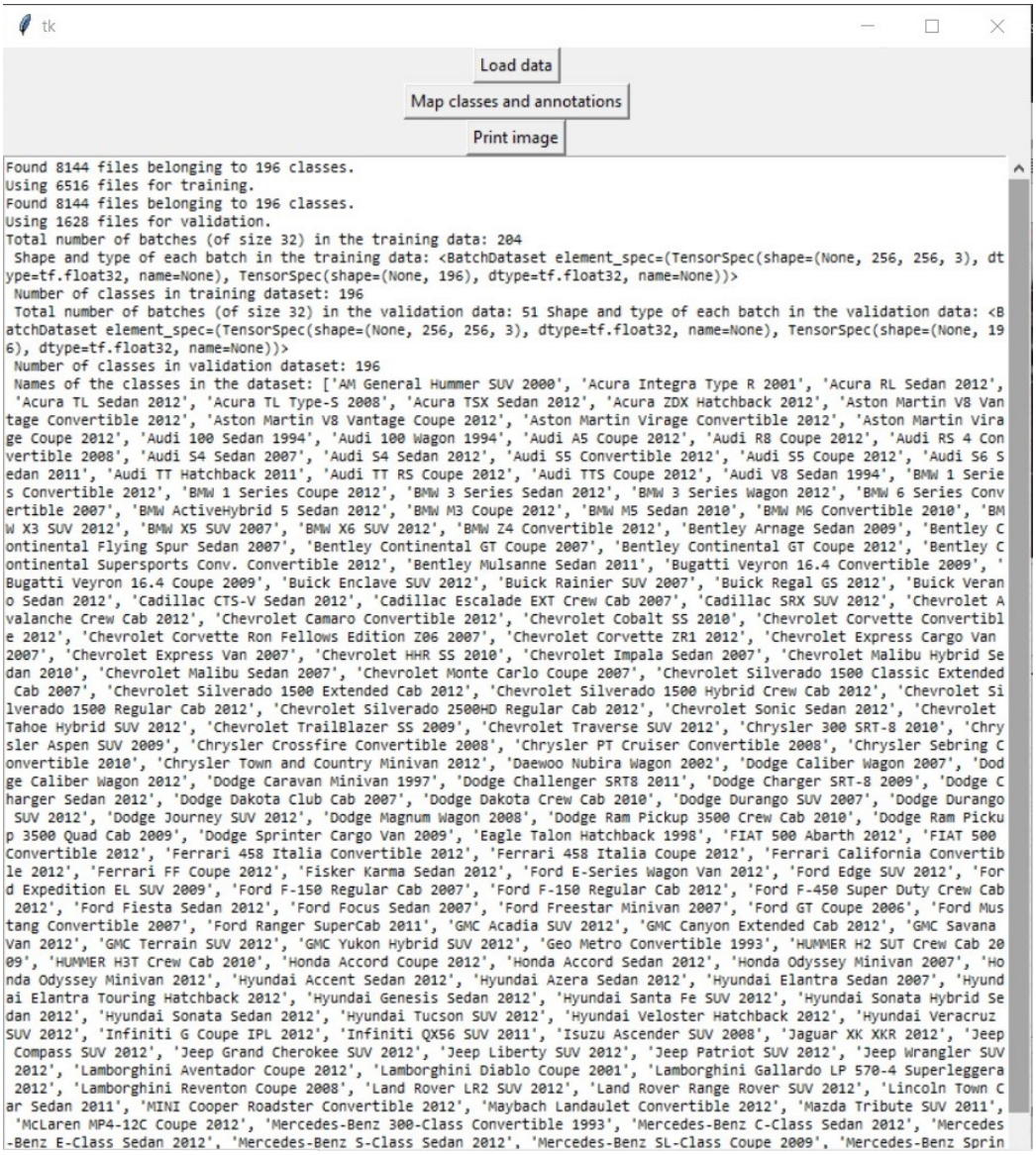


Figure 1.10: The screenshots of the UI for loading, mapping and printing data - *Load data* and *Map classes and annotations* buttons

1.5 Conclusions and perspectives

In this work, we have presented deep learning models for image classification and object detection applied to the given cars dataset. From EDA, we conclude that there are no anomalies in the given data and also the data for all classes is well balanced. However, we observed that the data is skewed towards the cars that are made recently but this does not effect the models that are built here. We observed that the pretrained 'ResNet34' model gives satisfying results on the classification of validation as well as testing data. We have built three models for the object detection. They are the Faster R-CNN architecture with the 'VGG16' and 'ResNet50' as the backbones and 'YOLOv5'. However, none of them perform well on the testing data. They reason is that the average number of images per class is less than 50 while generally, 1500 images per class is recommended by the authors who proposed these models.

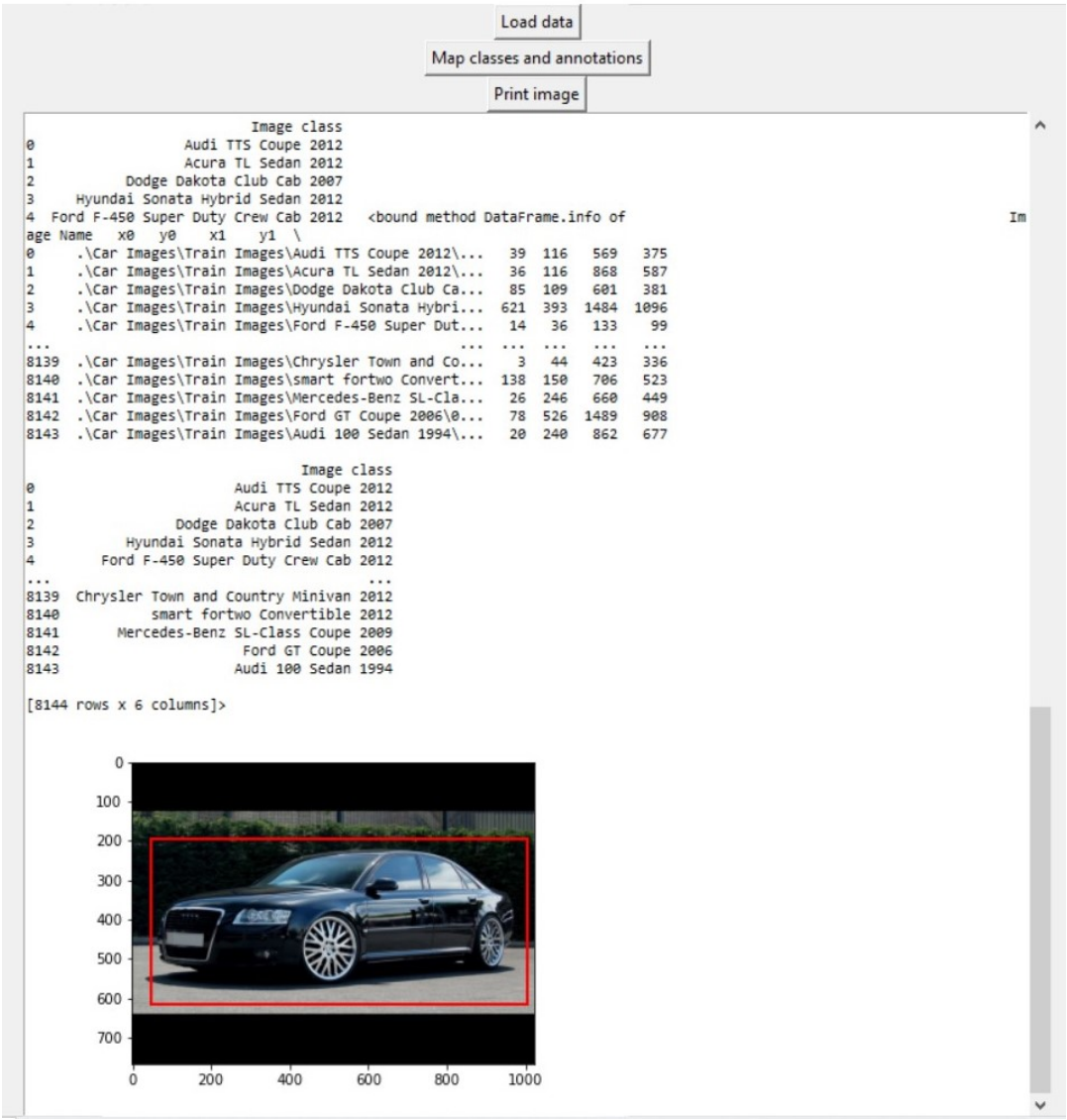


Figure 1.11: The screenshots of the UI for loading, mapping and printing data - *Print image* button

Implications The performance of the deep learning based car image classification model is significantly better than the statistical approach used in the benchmark model available in the literature [13]. Based on these results, we strongly recommend a deep learning based model for classification of the car images. From business point of view, this work reaffirms the usage of deep learning for real-time classification problem of the self-driving cars.

Limitations The main limitation of this work is the unavailability of the data. In order to built an object detection model based on deep learning, we need at least 1500 images per class. It is highly recommended that we rerun the models after we collect the required data.

Final remarks Finally, we conclude that deep learning based car detection model is very promising. However, the further steps include collection of higher volumes of the data. Also, another point to

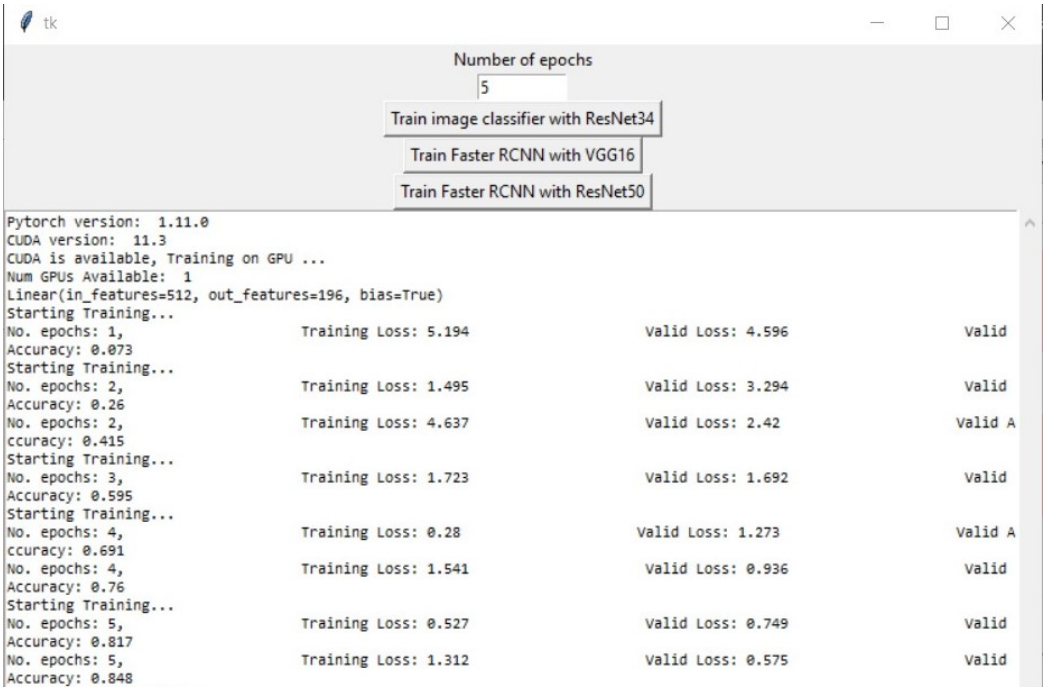


Figure 1.12: The screenshots of the UI for training the models - *Train image classifier with ResNet34* button

be improved is the computational time required for training the models. Although we have decreased the required computational time by running the models on the GPU instead of the CPU, it can be further decreased by using parallel computing techniques on multiple GPUs.

Figure 1.13: The screenshots of the UI for training the models - *Train Faster R-CNN with VGG16* and *Train Faster R-CNN with ResNet50* buttons

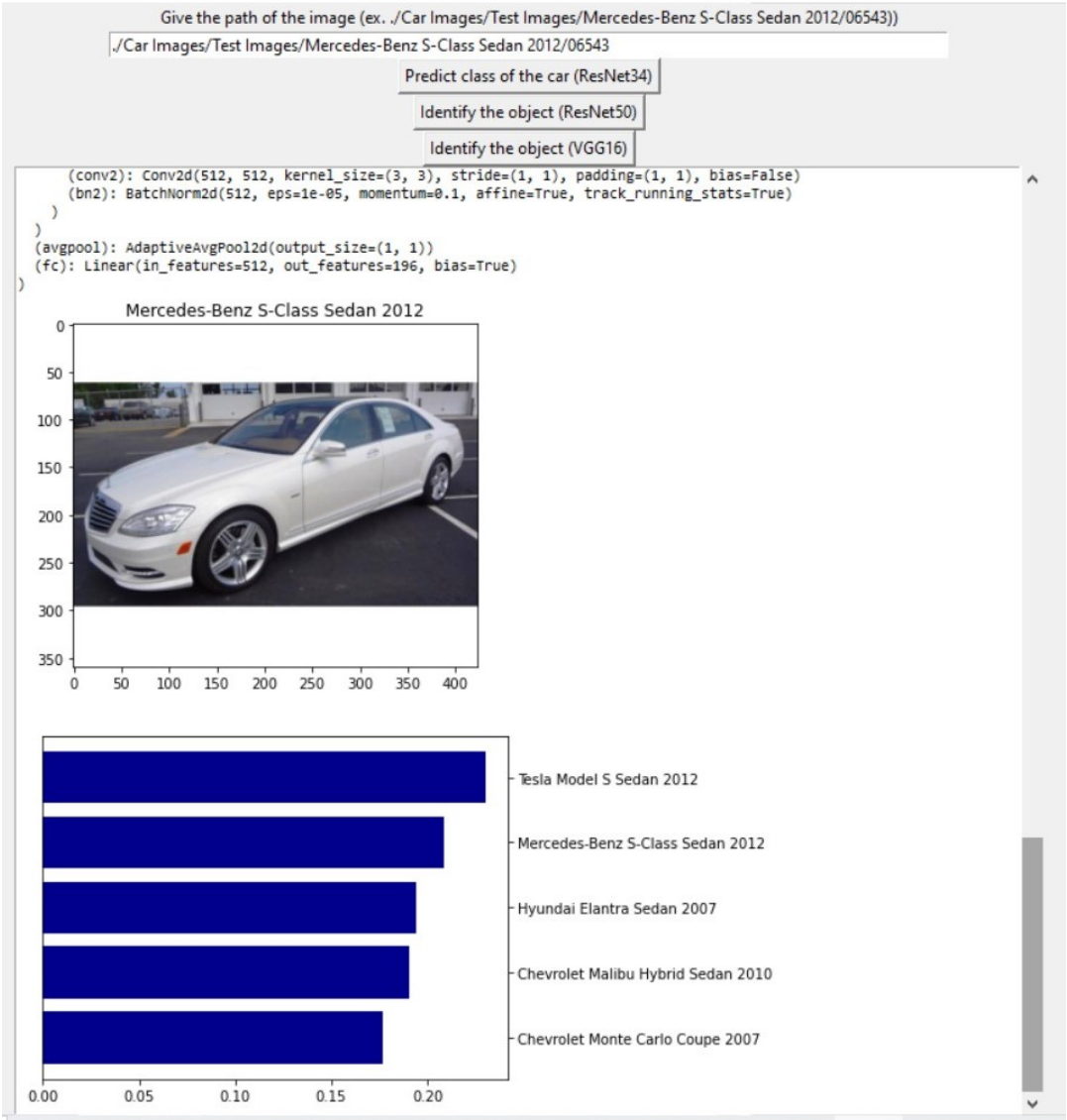


Figure 1.14: The screenshots of the UI for training the models - *Predict class of the car (ResNet34)* button

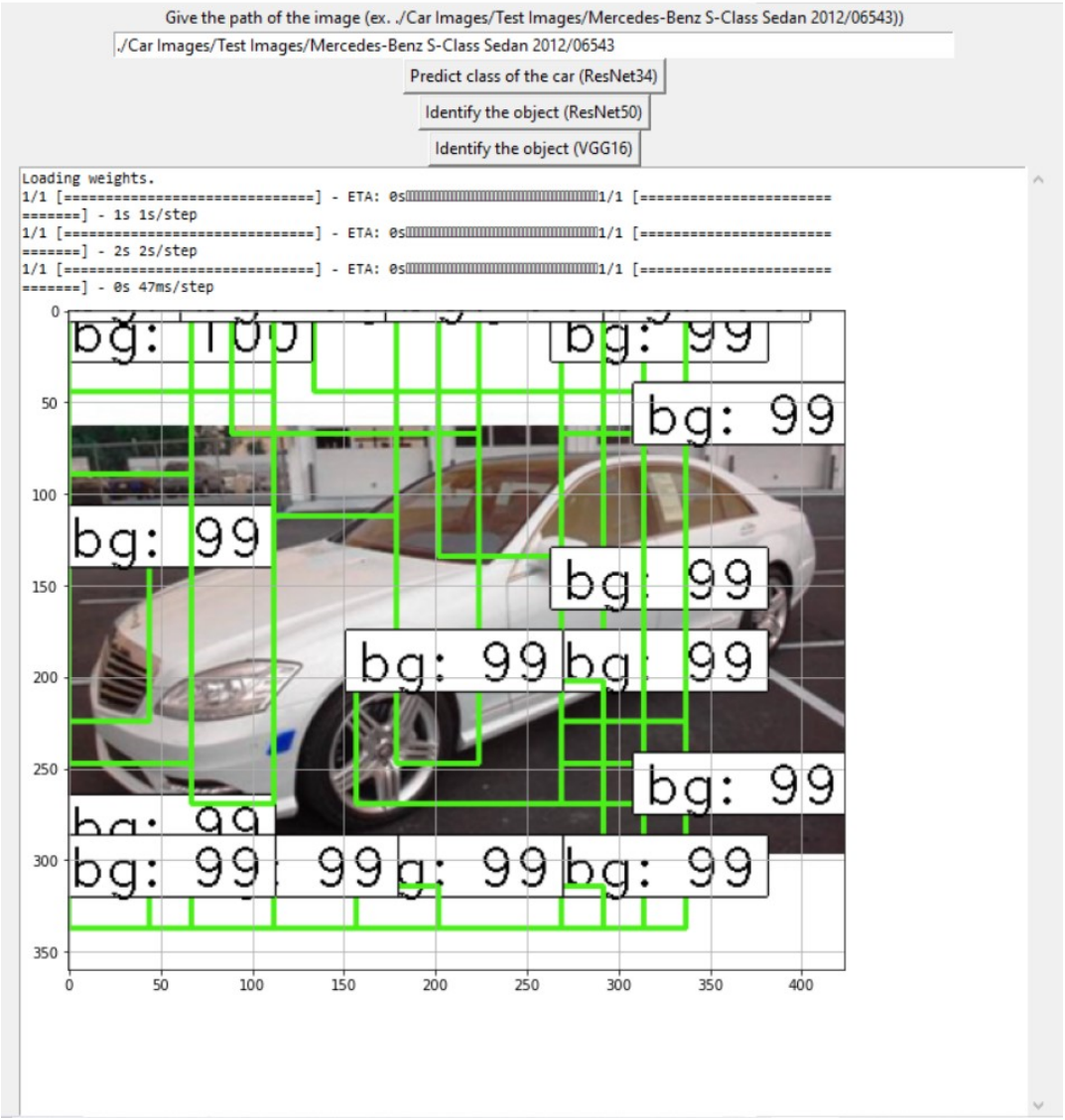


Figure 1.15: The screenshots of the UI for training the models - *Test Faster R-CNN with VGG16* and *Identify the object (ResNet50)* button

Bibliography

[1] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[3] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[8] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[10] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[13] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013.

Capstone Project