National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Abhijit Sahasrabuddhe
Student ID: x20180799

School of Computing
National College of Ireland

Supervisor: Dr.Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Abhijit Sahasrabuddhe
Student ID:	x20180799
Programme:	Programme Name
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr.Christian Horn
Submission Due Date:	16/12/2021
Project Title:	Configuration Manual
Word Count:	1233
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th December 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Abhijit Sahasrabuddhe
x20180799

1 Introduction

The goal of this documentation is to compile a list of all the tasks that must be completed throughout the project implementation stage. Software and hardware requirements are outlined in order to recreate the project in the future. The coding and deployment processes are covered in this article, as well as the procedures that must be followed in order to run the code.

2 System Configuration

2.1 Hardware Configuration

Table below shows the hardware configuration of the system used to implement code

Table 1: System Configuration

	System Configuration
Operating System	Windows 10 Home Single Language 64-bit
Memory	16.0 GB RAM
CPU	AMD Ryzen 7 4800H
Cores	16
GPU	AMD Radeon RX 5600M Series

2.2 Software Configuration

This section includes details regarding software used during execution of this project.

2.2.1 Python

For implementation of this project python is used as coding language with python version as 3.9.5 using jupyter notebook to execute the code as shown in Fig[1]. It is one of the leading interface for python coding.

2.2.2 Other Software

Google chrome was used to access jupyter notebook and overleaf which is cloud-based collaborative LaTeX editor. For report writing latex has been used to format and align

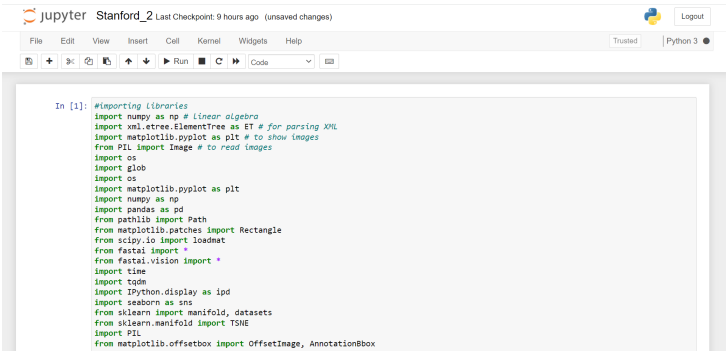


Figure 1: Jupyter notebook

the document. It supports supports document creation using Latex. It is very user friendly and user interface for same is as shown in below figure Fig[2].

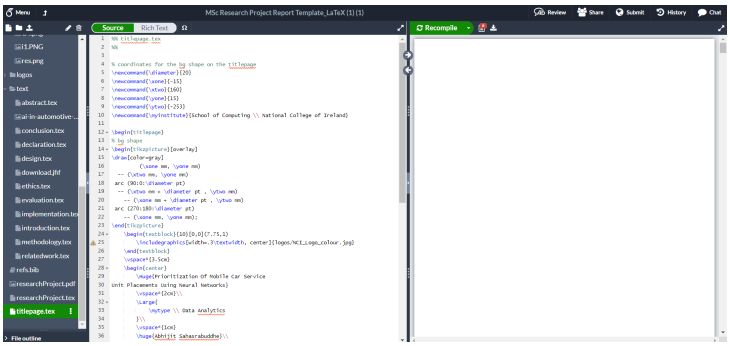


Figure 2: Overleaf LaTex Editor

3 Data Preparation

Data set used as part of this research is taken from Stanford Cars Data set Krause et al. (2013). It is an open source data set provided by Stanford University. Dr. Jonathan Krause and his Stanford University team created the Stanford Automotive Data Collection, which is a large, fine-grained car data set. The public Stanford vehicles data collection contains a total of 16,185 automotive images. There are 196 vehicle classes in this data collection. The authors utilized an unidentified automobile website to build a list of all cars from 1990 to 2012 in order to create a list of car labels. The data is separated into two categories: training and testing. The metadata for all photographs includes class names and bounding boxes. Year, production, and model categories are common classifications (for example, 2012 Tesla Model S or 2012 BMW M3). Each image has its own dimensions. Bounding boxes are used in the pre-processing step to create initial images that focus on the things of interest, which in this case are the vehicles.

After downloading data set and related MAT files, once MAT file is converted to data frame data can be accessed easily using the created data frame.

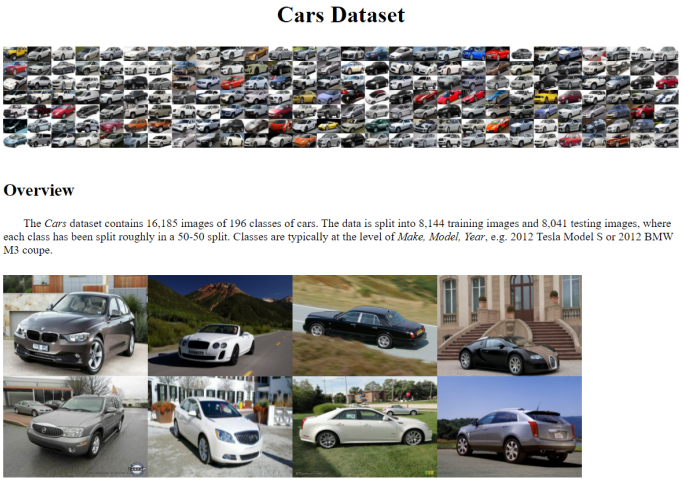


Figure 3: Stanford Cars Data Set

4 Implementation

As part of implementation multiple tasks needs to carried out in sequence as given below after which the data pre-processing, model build and implementation can be reproduced.

4.1 Converting MAT Files

MAT files are provided with data set and used to create data frame. using this data frame data can be accessed easily. For converting MAT file to data frame below code is used.

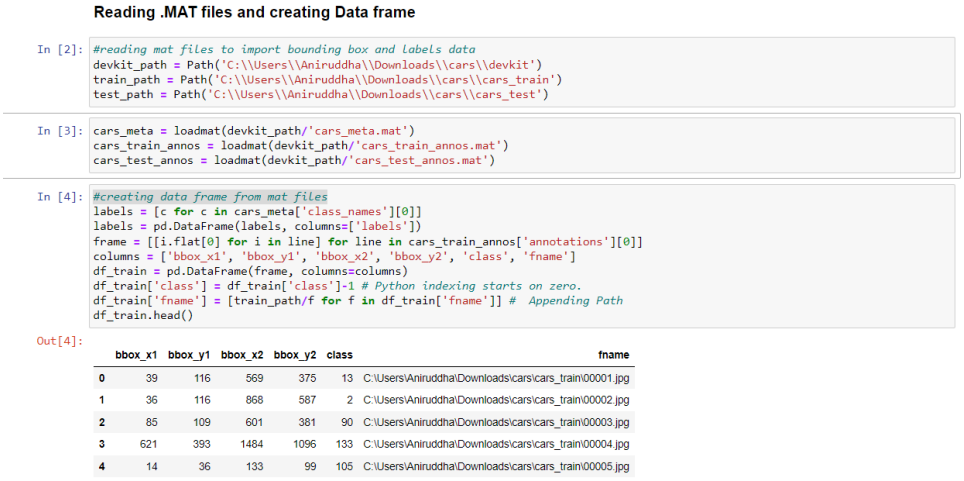


Figure 4: MAT File Conversion

File path need to be adjusted accordingly before running this code. once data frame is created labels and classes are added by merging along with null data check as shown in below figure.

```
In [5]: #Merging labels in dataframe
        df_train = df_train.merge(labels, left_on='class', right_index=True)
        df_train = df_train.sort_index()
        df_train.head()
```

Out[5]:

	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname	labels
0	39	116	569	375	13	C:\Users\Aniruddha\Downloads\cars\cars_train\00001.jpg	Audi TTS Coupe 2012
1	36	116	868	587	2	C:\Users\Aniruddha\Downloads\cars\cars_train\00002.jpg	Acura TL Sedan 2012
2	85	109	601	381	90	C:\Users\Aniruddha\Downloads\cars\cars_train\00003.jpg	Dodge Dakota Club Cab 2007
3	621	393	1484	1096	133	C:\Users\Aniruddha\Downloads\cars\cars_train\00004.jpg	Hyundai Sonata Hybrid Sedan 2012
4	14	36	133	99	105	C:\Users\Aniruddha\Downloads\cars\cars_train\00005.jpg	Ford F-450 Super Duty Crew Cab 2012

```
In [6]: #checking Null data
        df_train.isnull().sum()
```

```
Out[6]: bbox_x1    0
        bbox_y1    0
        bbox_x2    0
        bbox_y2    0
        class      0
        fname      0
        labels     0
        dtype: int64
```

Figure 5: Data Frame Merge To Add labels

After the new data frame created is exported to CSV file so that it can be used easily instead of creating data frame from MAT files each time.

Creating CSV from dataframe

```
In [8]: # exporting dataframe created from mat files to CSV
        df_train.to_csv('C:\\Users\\Aniruddha\\Downloads\\cars\\train.csv', index=False)
```

Figure 6: Exporting Data frame To CSV

4.2 Converting Source Images to Gray Scale

Once merged data frame is done and exported to CSV it can be used for handling data easily. Next step is to convert data to gray scale as instead of using only one color channel in model ,model is retaining all three color channels and gray scale images are given as input to the model as shown in figure below Fig[7]. So once this code is run images will be converted to gray scale and stored at given path as given in data frame under fname column.

Converting Data to Black and White

```
In [9]: ###Data is converted to black and white and stored at new location
        import cv2
        from os import listdir,makedirs
        from os.path import isfile,join

        path = r'C:\\Users\\Aniruddha\\Downloads\\cars\\cars_train' # Source
        dstpath = r'C:\\Users\\Aniruddha\\Downloads\\cars\\BAW' # Destination

        try:
            makedirs(dstpath)
        except:
            print ("Directory already exist, images will be written in BAW folder")

        # Folder won't used
        files = [f for f in listdir(path) if isfile(join(path,f))]

        for image in files:
            try:
                img = cv2.imread(os.path.join(path,image))
                gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
                dstPath = join(dstpath,image)
                cv2.imwrite(dstPath,gray)
            except:
                print ("{} is not converted".format(image))
```

Figure 7: Converting Data To Grey scale

4.3 Cropping Images As Per Given Bounding Boxes

Bounding boxes are given for all images along with data set highlighting the car. Using this data raw images are cropped and stored at new location using code shown in below figure Fig[7]. Destination path needs to be updated in code where cropped images will be stored.

Cropping images using bounding boxes

In [10]:
```
### Cropping images as per bounding box sizes provided in MAT files and saving at new location
import pandas as pd
import cv2
import numpy as np
from PIL import Image
# from PIL import open
dp= "C:\\Users\\Aniruddha\Downloads\\cars\\test"
df = df_train
for i in range(len(df)):
    name = df.loc[i]['fname']
    n1=str(name)
    n2=n1.replace('cars_train','test')
    im= Image.open(name)
#     print(i)
    image = im.crop((df.loc[i]['bbox_x1'], df.loc[i]['bbox_y1'], df.loc[i]['bbox_x2'], df.loc[i]['bbox_y2']) )
    pix = np.array(image)
    cv2.imwrite(n2,pix)
```

Figure 8: Image Crop

4.4 Class Merge

Original Data set contains 196 classes defining make model and year of the car. TO reduce the classes classes are manually merger by Brand ignoring year and model as goal of this research is to identify car brand.After manually merging classes in CSV document previously exported this new class distribution is used for further analysis. Code snippet shown in below figure is used to get class distribution in original data.

Class Distribution

In [7]:
```
#Checking class distribution in data using bar plots
import seaborn as sns
plt.figure(figsize = (5, 32))
sns.countplot(y = 'labels', data = df_train, order = df_train['labels'].value_counts().index,\
              palette = "husl")
plt.title('Barplot of Model in the Re-split Stanford Cars Training Set', y = 1.02)
plt.xlabel('Count'), plt.xticks(rotation = 90)
plt.show()
```

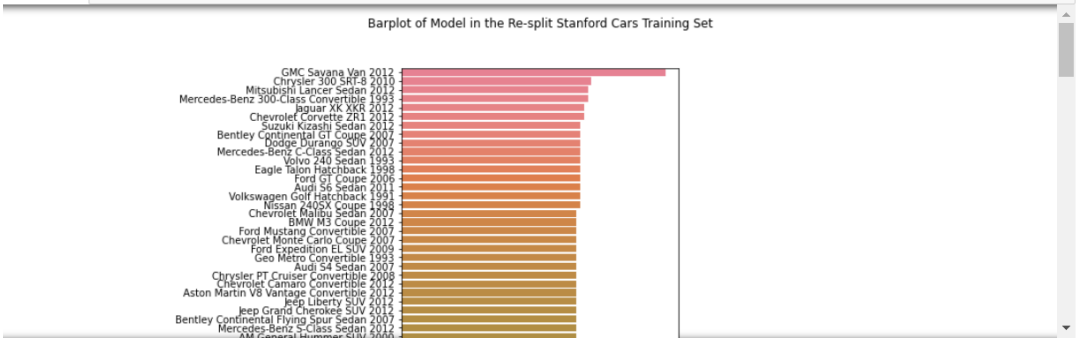


Figure 9: Class Distribution In Original Data

Once classes are merged to check new class distribution code shown in below figure Fig[10]. New class distribution is displayed using seaborn library horizontal bar plot. The

Matplotlib library was used to create the seaborn package. It's used to make statistics graphs that are more appealing and instructive.

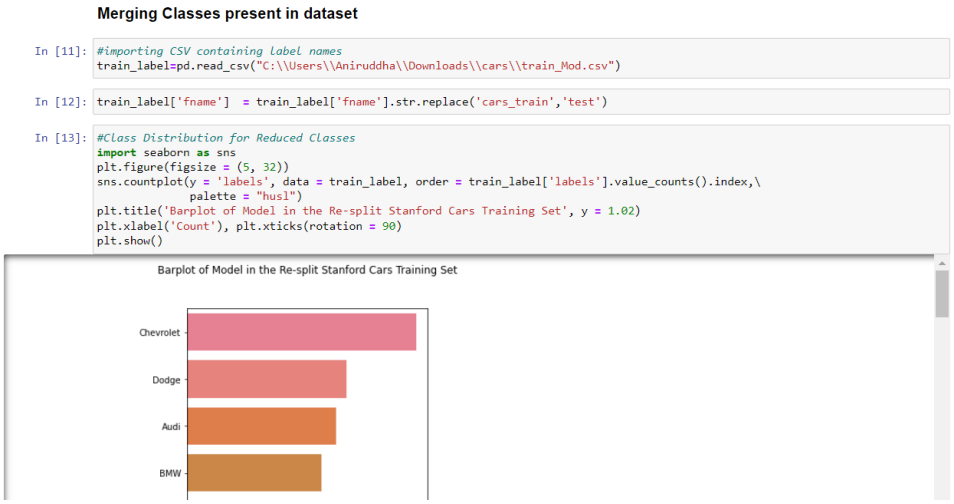


Figure 10: Class Distribution After Merge

4.5 Train Validation Test Split

Python code shown in below Fig[11] splits data in training , validation and testing set. This is done prior to building model to ensure model is correctly implemented on expected data.

Figure 11: Train Test Split

4.6 Model Build

As part of this research ResNet50 is used as base model along with added dense layers. Code shown in below figure imports the necessary libraries and and base ResNet50 model is imported and stored in variable "resnet".

Model-I Resnet50

In [15]:
```
import pickle
import numpy as np
from keras import backend as K

# from keras.applications import ResNet50,VGG19
from tensorflow.keras.applications.resnet50 import ResNet50

from keras.preprocessing.image import ImageDataGenerator
# from keras.optimizers import Adam,RMSprop
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array

from keras.layers import Flatten
from sklearn.linear_model import LogisticRegression
from keras.layers import Dense, Dropout, InputLayer
from keras.layers import Input,BatchNormalization
from keras.models import Sequential,Model
from keras.regularizers import l2
import matplotlib.pyplot as plt
import keras_metrics as km
```

In [16]:
```
#creating base model for ResNet50
IMAGE_SIZE = [224, 224]
resnet = ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
for layer in resnet.layers:
    layer.trainable = False
```

Figure 12: Model Build

Adding Full Connected Dense Layers

In [17]:
```
# ,kernel_regularizer=l2(0.01), bias_regularizer=l2(0.01)
flatten = Flatten()(resnet.output)
dense = Dense(256, activation = 'relu')(flatten)
dense = Dense(128, activation = 'relu')(dense)
prediction = Dense(32, activation = 'softmax')(dense)
```

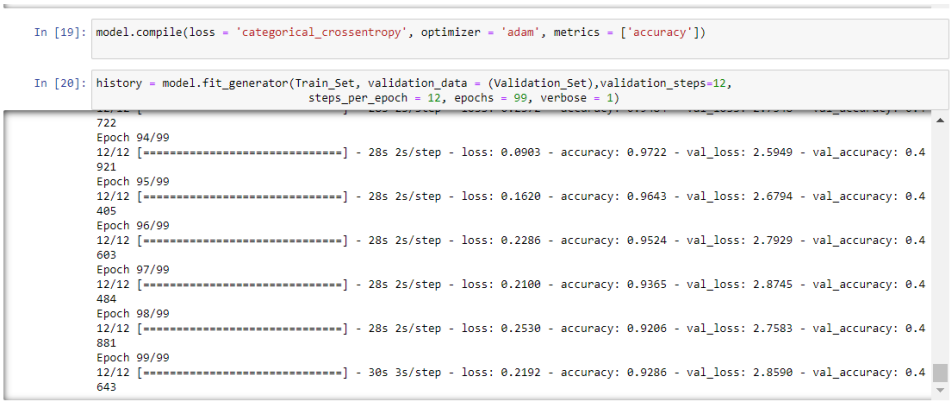
In [18]:
```
model = Model(inputs = resnet.input, outputs = prediction )
model.summary()
```

```
 conv5_block3_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block3_add[0][0]']

 flatten (Flatten)              (None, 100352)       0           ['conv5_block3_out[0][0]']

 dense (Dense)                  (None, 256)          25690368    ['flatten[0][0]']

 dense_1 (Dense)                (None, 128)          32896       ['dense[0][0]']

 dense_2 (Dense)                (None, 32)           4128        ['dense_1[0][0]']

==================================================================================================
Total params: 49,315,104
Trainable params: 25,727,392
Non-trainable params: 23,587,712
__________________________________________________________________________________________________
```

Figure 13: Model Build Adding Dense Layers

4.7 Model Compile and Run

Once base model code is run and model is built model need to be compiled and implemented on training and validation data to train the model and validate with small subset of the data. Number of epochs and validation steps need to defined while fitting the model.



```
In [19]: model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

In [20]: history = model.fit_generator(Train_Set, validation_data = (Validation_Set),validation_steps=12,
                                       steps_per_epoch = 12, epochs = 99, verbose = 1)

722
Epoch 94/99
12/12 [==============================] - 28s 2s/step - loss: 0.0903 - accuracy: 0.9722 - val_loss: 2.5949 - val_accuracy: 0.4
921
Epoch 95/99
12/12 [==============================] - 28s 2s/step - loss: 0.1620 - accuracy: 0.9643 - val_loss: 2.6794 - val_accuracy: 0.4
405
Epoch 96/99
12/12 [==============================] - 28s 2s/step - loss: 0.2286 - accuracy: 0.9524 - val_loss: 2.7929 - val_accuracy: 0.4
603
Epoch 97/99
12/12 [==============================] - 28s 2s/step - loss: 0.2100 - accuracy: 0.9365 - val_loss: 2.8745 - val_accuracy: 0.4
484
Epoch 98/99
12/12 [==============================] - 28s 2s/step - loss: 0.2530 - accuracy: 0.9206 - val_loss: 2.7583 - val_accuracy: 0.4
881
Epoch 99/99
12/12 [==============================] - 30s 3s/step - loss: 0.2192 - accuracy: 0.9286 - val_loss: 2.8590 - val_accuracy: 0.4
643
```

Figure 14: Model Compile And Run

Once model fitting is done accuracy and loss graphs are plotted using code shown in below Fig[15]

```
In [21]: ##Loss Graph plotting
plt.plot(history.history['loss'],label = 'train_loss')
plt.plot(history.history['val_loss'], label = 'testing_loss')
plt.title('loss')
plt.legend()
plt.show()
```



```
In [22]: ##Accuracy Graph plotting
plt.plot(history.history['accuracy'], label='training_accuracy')
plt.plot(history.history['val_accuracy'], label='validation accuracy')
plt.title('Accuracy')
plt.legend()
plt.show()
```


Figure 15: Accuracy- Loss Plots

4.8 Changing Input Data

4.8.1 Gray Scale Input Data

Once model created in implemented on normal colored training data next run input data is changed to gray scale. To do so file path under column fname need to be changes so that it is pointed to correct directory containing grey scale images converted previously Fig[16]. Once changes are done models is implemented on this data as shown in compile and run section of this document.

Model-II Resnet50 - Black & White Images

In [25]:
```
#Image classes are reduced to 32 from 196 by merging cars of same brand in same class irrespective of the model and make
df_train=pd.read_csv("C:\\Users\\Aniruddha\\Downloads\\cars\\train_Mod.csv")
```

In [26]:
```
df_train['fname'] = df_train['fname'].astype(str)
df_train['fname']  = df_train['fname'].str.replace('cars_train','BAW')
df_train
```

Out[26]:

	Unnamed: 0	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname	labels
0	0	39	116	569	375	13	C:\Users\Aniruddha\Downloads\cars\BAW\00001.jpg	Audi
1	1	36	116	868	587	2	C:\Users\Aniruddha\Downloads\cars\BAW\00002.jpg	Acura
2	2	85	109	601	381	90	C:\Users\Aniruddha\Downloads\cars\BAW\00003.jpg	Dodge
3	3	621	393	1484	1096	133	C:\Users\Aniruddha\Downloads\cars\BAW\00004.jpg	Hyundai
4	4	14	36	133	99	105	C:\Users\Aniruddha\Downloads\cars\BAW\00005.jpg	Ford
...	...	...	...	...	...	...	...	...
7031	8138	117	84	583	403	187	C:\Users\Aniruddha\Downloads\cars\BAW\08139.jpg	Toyota
7032	8139	3	44	423	336	77	C:\Users\Aniruddha\Downloads\cars\BAW\08140.jpg	Chrysler
7033	8141	26	246	660	449	162	C:\Users\Aniruddha\Downloads\cars\BAW\08142.jpg	Mercedes-Benz
7034	8142	78	526	1489	908	111	C:\Users\Aniruddha\Downloads\cars\BAW\08143.jpg	Ford
7035	8143	20	240	862	677	16	C:\Users\Aniruddha\Downloads\cars\BAW\08144.jpg	Audi

7036 rows × 8 columns

In [27]:
```
Train_df=df_train.copy()
Train_df['fname'] = Train_df['fname'].astype(str)
Train_df['fname']  = Train_df['fname'].str.replace('BAW','test_BW')
Train_df
```

Out[27]:

	Unnamed: 0	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname	labels
0	0	39	116	569	375	13	C:\Users\Aniruddha\Downloads\cars\test_BW\00001.jpg	Audi
1	1	36	116	868	587	2	C:\Users\Aniruddha\Downloads\cars\test_BW\00002.jpg	Acura
2	2	85	109	601	381	90	C:\Users\Aniruddha\Downloads\cars\test_BW\00003.jpg	Dodge
3	3	621	393	1484	1096	133	C:\Users\Aniruddha\Downloads\cars\test_BW\00004.jpg	Hyundai
4	4	14	36	133	99	105	C:\Users\Aniruddha\Downloads\cars\test_BW\00005.jpg	Ford
...	...	...	...	...	...	...	...	...
7031	8138	117	84	583	403	187	C:\Users\Aniruddha\Downloads\cars\test_BW\08139.jpg	Toyota
7032	8139	3	44	423	336	77	C:\Users\Aniruddha\Downloads\cars\test_BW\08140.jpg	Chrysler
7033	8141	26	246	660	449	162	C:\Users\Aniruddha\Downloads\cars\test_BW\08142.jpg	Mercedes-Benz
7034	8142	78	526	1489	908	111	C:\Users\Aniruddha\Downloads\cars\test_BW\08143.jpg	Ford
7035	8143	20	240	862	677	16	C:\Users\Aniruddha\Downloads\cars\test_BW\08144.jpg	Audi

Figure 16: Gray Scale Input Data

4.8.2 Front/Rear View Input Data

For next step same model is used with selected input data showing front or rear view of the car. From cropped images, selected images showing front and rear view of the car are placed in new folder and path is changed accordingly in fname column of the data frame. Using this code (Fig[17]) model in implemented on images showing only front view or rear view of the car. Once changes are done models is implemented on this data as shown in compile and run section of this document.

Changing fname to use selected subset of data

```
In [35]: import pandas as pd
         train_label=pd.read_csv("C:\\Users\\Aniruddha\\Downloads\\cars\\train_Mod.csv")

In [36]: train_label
```

Out[36]:

	Unnamed: 0	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname	labels
0	0	39	116	569	375	13	C:\Users\Aniruddha\Downloads\cars\cars_train\00001.jpg	Audi
1	1	36	116	868	587	2	C:\Users\Aniruddha\Downloads\cars\cars_train\00002.jpg	Acura
2	2	85	109	601	381	90	C:\Users\Aniruddha\Downloads\cars\cars_train\00003.jpg	Dodge
3	3	621	393	1484	1096	133	C:\Users\Aniruddha\Downloads\cars\cars_train\00004.jpg	Hyundai
4	4	14	36	133	99	105	C:\Users\Aniruddha\Downloads\cars\cars_train\00005.jpg	Ford
...	...	...	...	...	...	...	...	...
7031	8138	117	84	583	403	187	C:\Users\Aniruddha\Downloads\cars\cars_train\08139.jpg	Toyota
7032	8139	3	44	423	336	77	C:\Users\Aniruddha\Downloads\cars\cars_train\08140.jpg	Chrysler
7033	8141	26	246	660	449	162	C:\Users\Aniruddha\Downloads\cars\cars_train\08142.jpg	Mercedes-Benz
7034	8142	78	526	1489	908	111	C:\Users\Aniruddha\Downloads\cars\cars_train\08143.jpg	Ford
7035	8143	20	240	862	677	16	C:\Users\Aniruddha\Downloads\cars\cars_train\08144.jpg	Audi

7036 rows × 8 columns

```
In [37]: train_label['fname'] = train_label['fname'].astype(str)

In [38]: train_label['fname']  = train_label['fname'].str.replace('cars_train','FB')

In [39]: train_label
```

Out[39]:

	Unnamed: 0	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname	labels
0	0	39	116	569	375	13	C:\Users\Aniruddha\Downloads\cars\FB\00001.jpg	Audi
1	1	36	116	868	587	2	C:\Users\Aniruddha\Downloads\cars\FB\00002.jpg	Acura
2	2	85	109	601	381	90	C:\Users\Aniruddha\Downloads\cars\FB\00003.jpg	Dodge
3	3	621	393	1484	1096	133	C:\Users\Aniruddha\Downloads\cars\FB\00004.jpg	Hyundai
4	4	14	36	133	99	105	C:\Users\Aniruddha\Downloads\cars\FB\00005.jpg	Ford
...	...	...	...	...	...	...	...	...

Figure 17: Front/Rear View input Data

4.8.3 Selected Class Input Data

After class merge there is class imbalance in data so to reduce it classes are limited to AUDI, Hyundai, Ford, Dodge and BMW which is done manually in CSV file. In this step model with same build is implemented on above mentioned car classes alone. Once changes are done models is implemented on this data as shown in compile and run section of this document.

```
In [2]: import pandas as pd
        train_label=pd.read_csv("C:\\Users\\Aniruddha\\Downloads\\cars\\train_Mod_Class.csv")

In [3]: train_label['fname'] = train_label['fname'].astype(str)
        # train_label['fname']  = train_label['fname'].str.replace('cars_train','FB')

In [4]: train_label.count
```

Out[4]:

```
<bound method DataFrame.count of      Unnamed: 0  bbox_x1  bbox_y1  bbox_x2  bbox_y2  class  \
0             1       39      116      569      375     13
1             2       39       52      233      150     13
2             3       61       55      566      408     12
3             4       28       80      759      466     24
4             5      161      191      425      307     14
...         ...      ...      ...      ...      ...    ...
2704       3610       24       27      616      346    137
2705       3611       48      123      708      480    137
2706       3612       40       97      614      456    138
2707       3613       20       69      576      480    131
2708       3614      111       58      451      312    129

                                                  fname   labels
0     C:\Users\Aniruddha\Downloads\cars\cars_train\00001.jpg     Audi
1     C:\Users\Aniruddha\Downloads\cars\cars_train\00017.jpg     Audi
2     C:\Users\Aniruddha\Downloads\cars\cars_train\00041.jpg     Audi
3     C:\Users\Aniruddha\Downloads\cars\cars_train\00046.jpg     Audi
4     C:\Users\Aniruddha\Downloads\cars\cars_train\00053.jpg     Audi
...                                                     ...      ...
2704  C:\Users\Aniruddha\Downloads\cars\cars_train\08019.jpg  Hyundai
2705  C:\Users\Aniruddha\Downloads\cars\cars_train\08034.jpg  Hyundai
2706  C:\Users\Aniruddha\Downloads\cars\cars_train\08045.jpg  Hyundai
2707  C:\Users\Aniruddha\Downloads\cars\cars_train\08064.jpg  Hyundai
2708  C:\Users\Aniruddha\Downloads\cars\cars_train\08084.jpg  Hyundai

[2709 rows x 8 columns]>
```

Figure 18: Reduced Classes Data

References

Krause, J., Stark, M., Deng, J. and Fei-Fei, L. (2013). 3d object representations for fine-grained categorization, *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia.