MACHINE LEARNING

(Predicting Amount of Purchase)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science and Engineering
BY

KARRI SRI SAI SOWMYA

221710310031

Under the Guidance of

Mr. M. Venkateswarlu

Assistant Professor

Department Of Computer Science and Engineering

GITAM School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
June 2020

DECLARATION

I submit this industrial training work entitled **"DISTRACTED DRIVER DETECTION**" to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology**" in "**Computer Science and Engineering**". I declare that it was carried out independently by me under the guidance of **Mr. M. Venkateswarlu**, Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD
Date:

Karri Sri Sai Sowmya
221710310031

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **"DISTRACTED DRIVER DETECTION"** is being submitted by Karri Sri Sai Sowmya (221710310031) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science and Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2019-20

It is faithful record work carried out by her at the **Computer Science and Engineering Department,** GITAM University Hyderabad Campus under my guidance and supervision.

Mr. M. Venkateswarlu
Assistant Professor
Department of ECE

Dr.K.Manjunathachari
Professor and HOD
Department of ECE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad,** Pro Vice Chancellor, GITAM Hyderabad and **Dr. CH. Sanjay,** Principal, GITAM Hyderabad

I would like to thank respected **Dr. K. Manjunathachari,** Head of the Department of Computer science and Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr. M. Venkateswarlu** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

KARRI SRI SAI SOWMYA

221710310031

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on Cereals data set My perception of understanding the given data set has been in the view of undertaking a client's requirement of overcoming the stagnant point of sales of the products being manufactured by client.

To get a better understanding and work on a strategical approach for solution of the client, I have adapted the view point of looking at ratings of the products and for further deep understanding of the problem, I have taken the stance of a consumer and reasoned out the various factors of choice of the products and they purchase , and my primary objective of this case study was to look up the factors which were dampening the sale of products and corelate them to ratings of products and draft out an outcome report to client regarding the various accepts of a product manufacturing , marketing and sale point determination

Table of Contents:

LIST OF FIGURES

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

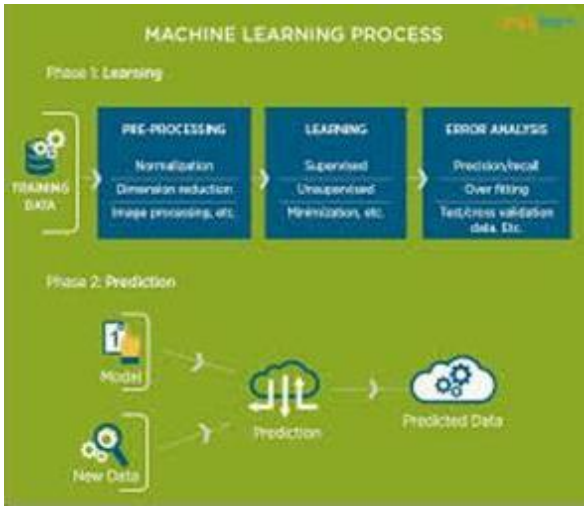

Figure 1 : The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

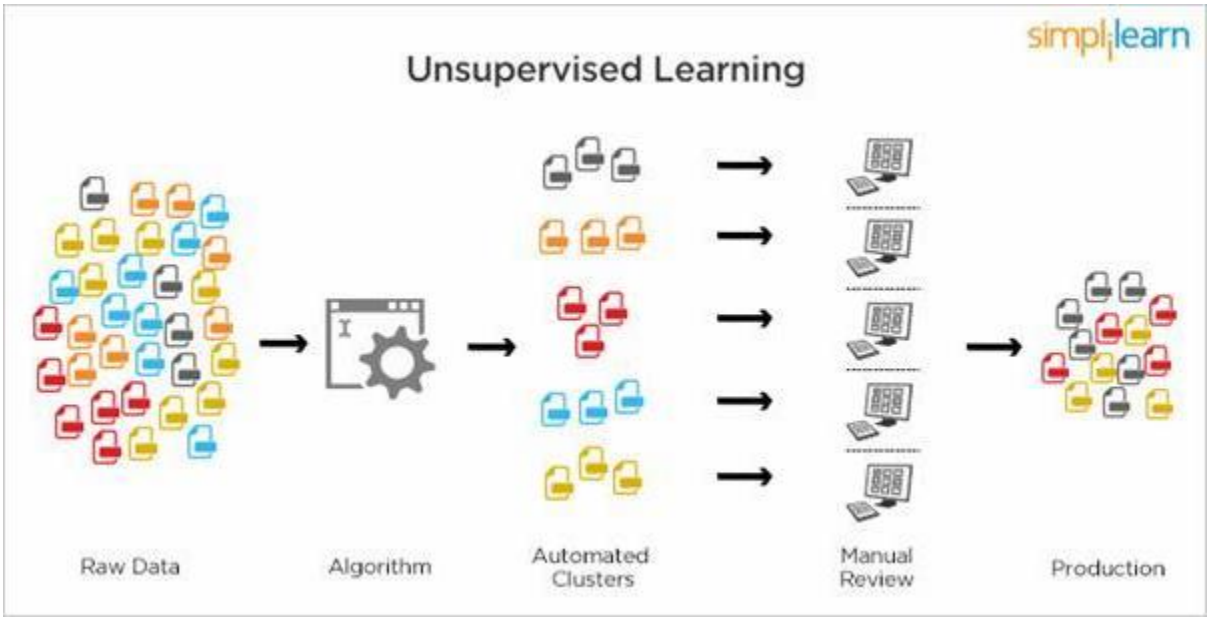

Figure 2 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

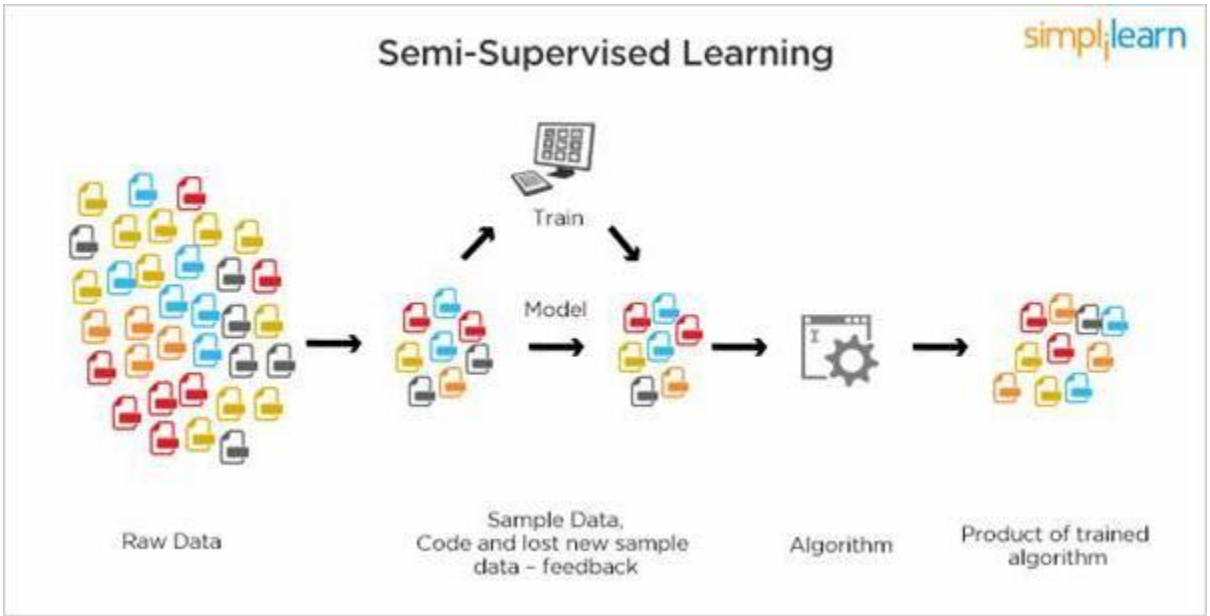

Figure 3 : Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special

types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

2.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

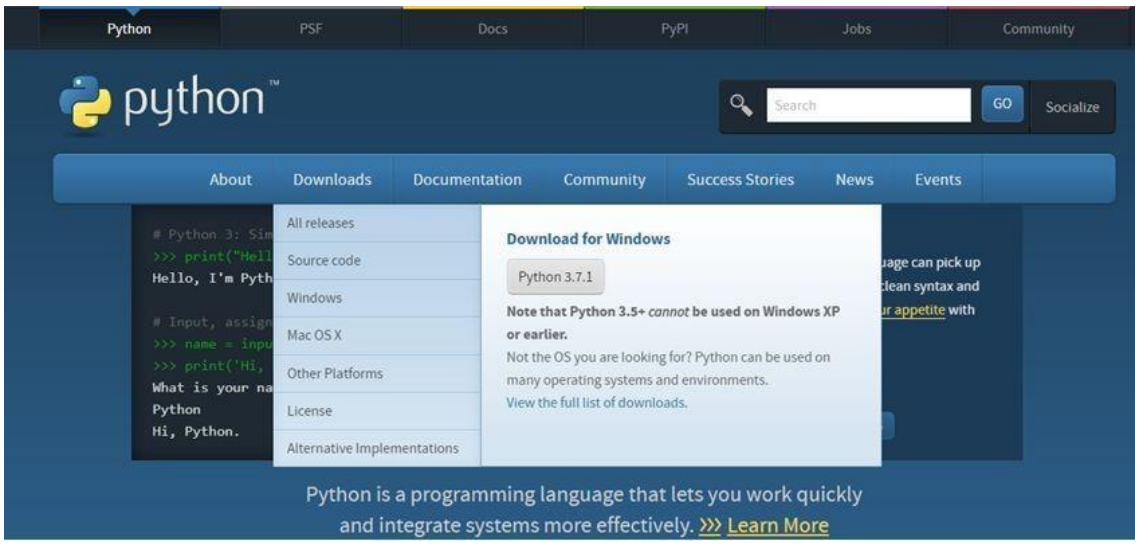

Figure 4 : Python download

2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.

- In WINDOWS:
- In windows
    - Step 1: Open Anaconda.com/downloads in web browser.
    - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
    - Step 3: select installation type(all users)
    - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
    - Step 5: Open jupyter notebook (it opens in default browser)

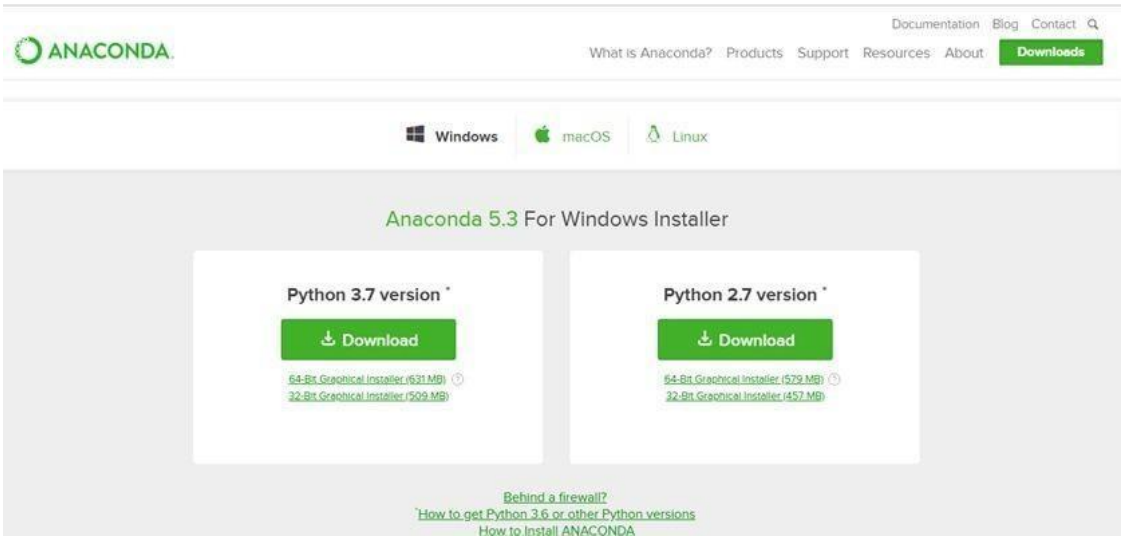

Figure 5 : Anaconda download

Figure 6 : Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists

- Tuples
- Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets

([]).

- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays

or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

- Defining a Class:
    - We define a class in a very similar way how we define a function.
    - Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is

indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Figure 7 : Defining a Class

2.7.2 __init__method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: init ().

CHAPTER 3

CASE STUDY

3.1 PROBLEM STATEMENT:

Our goal is to train a custom deep learning model to detect whether a person is driving safely or distracted.
In this data we are given driver images, each taken in a car with a driver doing something in the car (texting, eating, talking on the phone, makeup, reaching behind, etc). The objective of this work is to successfully predict the likelihood of what a driver is doing in each of the pictures in the dataset[1].

3.2 DATASET:

Data Set Link : https://www.kaggle.com/c/5048/download-all

The data consists on a set of images, each taken in a car where the driver is doing some action (e.g. texting, talking on the phone, doing their makeup). These are some examples:

The 10 classes to predict are:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

The given data set has the following Parameters

1. imgs.zip

    1.1. imgs

        1.1.1. train

            - c0
            - c1
            - c2

- c3
- c4
- c5
- c6
- c7
- c8
- c9

1.1.2. test

2. sample_submission.csv

3. driver_imgs_list.csv

3.3 OBJECTIVE OF CASE STUDY

To get a better understanding and chalking out a plan of solution of the client, we have adapted the view point of looking at product categories and for further deep understanding of the problem, we have considered all the factors of training data, testing data and validation data, which has images of distracted driver and safely driving driver.

This project focuses on driver distraction activities detection via images using different kinds of machine learning techniques. Our goal is to build a high-accuracy model to distinguish whether drivers is driving safely or conducting a particular kind of distraction activity. The input ofour model is images of driver taken in the car. We first preprocess these images to get input vectors, then use different classifiers to output a predicted type of distraction activity that drivers are conducting.

CHAPTER 4

MODEL BUILDING

4.1 VISUALIZATION OF THE DATA:

4.1.1 IMPORTING THE LIBRARIES:

The libraries have to be imported as per the requirement of the algorithm.We are importing os for reading the directories and matplotlib for plotting the graphs and images . The methods Sequential, Convo2D, Dense,Flatten,MaxPooling2D are imported from tensorflow and keras models.

```
import os
import matplotlib.pyplot as plt
import tensorflow as tf
```

Figure 8 : importing libraries

4.1.2 LOADING THE DATA SET

- Install Kaggle library

```
[ ]  ! pip install -q kaggle
```

Figure 9 : installing kaggle library

- Upload kaggle.json

```
[2]  from google.colab import files
     files.upload()
```

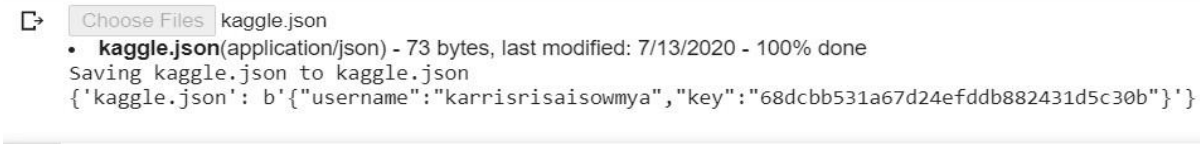

Figure 10: Uploading kaggle.json file

- Create a directory named kaggle and Move the `kaggle.json` file into `~/.kaggle`

```
[3]  ! mkdir ~/.kaggle
     ! cp kaggle.json ~/.kaggle/

[4]  ! chmod 600 ~/.kaggle/kaggle.json
```

Figure 11:Making a folder and moving the kaggle.json

- Download the data from kaggle

  For large datasets or competitions, you will get a "429 - Too Many Requests. A simple way to download data, in that case, is using wget command line, emulating "download all" button from kaggle:

  wget {download-all-button-URL}

  - Upload cookies.txt to your Colab

    As Kaggle needs user authentication, you must add your Kaggle cookies to wget. To do that, the simplest way is using this Chrome plugin https://chrome.google.com/webstore/detail/cookiestxt/njabckikapfpffapmjgojcnbfjonfjfg to obtain the cookies.txt file with your logged Kaggle information required

Figure 12:Uploading cookies.txt

- Download and unzip the data :

```
[ ] !wget -x --load-cookies cookies.txt "https://www.kaggle.com/c/5048/download-all" -O data.zip
    !unzip data.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: imgs/train/c7/img_56661.jpg
  inflating: imgs/train/c7/img_56699.jpg
  inflating: imgs/train/c7/img_56717.jpg
  inflating: imgs/train/c7/img_56724.jpg
  inflating: imgs/train/c7/img_56768.jpg
  inflating: imgs/train/c7/img_56779.jpg
  inflating: imgs/train/c7/img_56849.jpg
  inflating: imgs/train/c7/img_56899.jpg
  inflating: imgs/train/c7/img_56938.jpg
  inflating: imgs/train/c7/img_5706.jpg
  inflating: imgs/train/c7/img_57076.jpg
  inflating: imgs/train/c7/img_57122.jpg
  inflating: imgs/train/c7/img_57137.jpg
  inflating: imgs/train/c7/img_5715.jpg
  inflating: imgs/train/c7/img_57278.jpg
  inflating: imgs/train/c7/img_57321.jpg
  inflating: imgs/train/c7/img_57345.jpg
  inflating: imgs/train/c7/img_57355.jpg
  inflating: imgs/train/c7/img_5741.jpg
  inflating: imgs/train/c7/img_57433.jpg
  inflating: imgs/train/c7/img_5753.jpg
  inflating: imgs/train/c7/img_57559.jpg
  inflating: imgs/train/c7/img_57594.jpg
  inflating: imgs/train/c7/img_5771.jpg
  inflating: imgs/train/c7/img_57712.jpg
  inflating: imgs/train/c7/img_57733.jpg
  inflating: imgs/train/c7/img_57745.jpg
  inflating: imgs/train/c7/img_57775.jpg
  inflating: imgs/train/c7/img_57954.jpg
  inflating: imgs/train/c7/img_57989.jpg
  inflating: imgs/train/c7/img_58083.jpg
  inflating: imgs/train/c7/img_58096.jpg
```

Figure 13:Downloading and unzip the data

4.1.3 READING DIRECTORIES :

Using the command *os.listdir* we are reading the directories, at first it in content directory , and there are 8 parts ['.config', 'data.zip','cookies.txt','sample_submission.csv','driver_imgs_list.csv','kaggle.json','imgs','sample_data']

```
[9] os.listdir("/content")
```

```
['.config',
 'driver_imgs_list.csv',
 'sample_submission.csv',
 'imgs',
 'kaggle.json',
 'data.zip',
 'cookies.txt',
 'sample_data']
```

Figure 14: Reading directories

Our data set is present in ‘imgs’ directory , so now we are reading the ‘imgs’ directory to know the directories present in it.
On reading ‘imgs’ directory we got ‘test’ and ‘train’ directories .
Now in the ‘train’ directory there 9 folders namely ‘c0’,’c1’,’c2’,’c3’,’c4’,’c5’,’c6’,’c7’,’c8’,’c9’.which has the images of labels c0: safe driving , c1: texting – right , c2: talking on the phone – right , c3: texting – left , c4: talking on the phone – left , c5: operating the radio , c6: drinking , c7: reaching behind ,c8: hair and makeup , c9: talking to passenger. And ‘test’ directory has the images of all the labels in it .

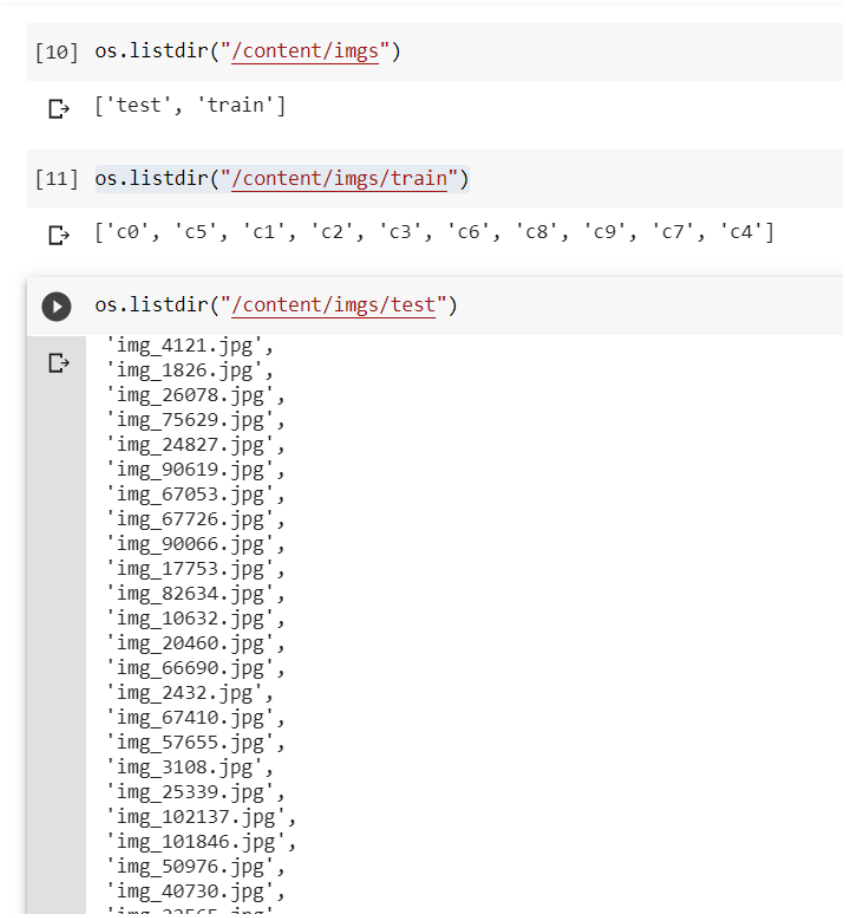

Figure 15:Reading sub directories

We are removing the data.zip as the data from it is extracted and also removing files 'driver_imgs_list.csv' and ‘sample_submission.csv’ as we are not using those data files in this project.

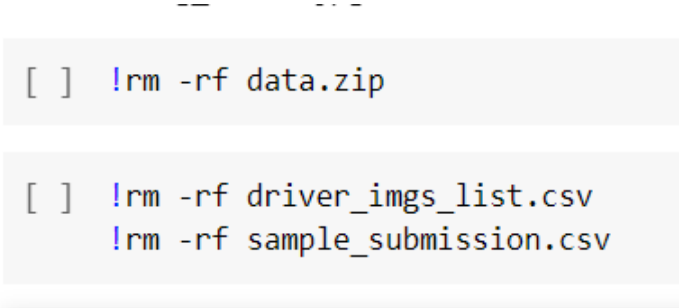

Figure 16: removing unnecessary files

By using *len* function the length of each label folder in train directory is checked . there are 2267 images in c1 folder , 2317 images in c2 folder , 2346 images in c3 folder , 2326 images in c4 folder , 2312 images in c5 folder , 2325 images in c6 folder , 2002 images in c7 folder , 1911 images in c8 folder ,2129 in c9 folder . Similarly, in the 'test' directory there are 79726 images.

```
[ ]  print(len(os.listdir("/content/imgs/train/c1")))
     print(len(os.listdir("/content/imgs/train/c2")))
     print(len(os.listdir("/content/imgs/train/c3")))
     print(len(os.listdir("/content/imgs/train/c4")))
     print(len(os.listdir("/content/imgs/train/c5")))
     print(len(os.listdir("/content/imgs/train/c6")))
     print(len(os.listdir("/content/imgs/train/c7")))
     print(len(os.listdir("/content/imgs/train/c8")))
     print(len(os.listdir("/content/imgs/train/c9")))
```

```
2267
2317
2346
2326
2312
2325
2002
1911
2129
```

```
[ ]  print(len(os.listdir("/content/imgs/test")))
```

```
79726
```

Figure 17: Length of the directories

4.1.3.1 LOADING THE DATA FROM DIRECTORIES:

The 'content' directory has all the data of 'imgs' . So first we assign *base_dir* to content folder . Using *os.path.join* we are assigning *img_dir* to 'imgs' directory which is in *base_dir*. Similarly *train_dir* and *test_dir* are assigned to 'train' directory and 'test' directory which are present in *imgs_dir*. All the sub folders in the 'train' directory are also assigned in the same manner as shown in the below figure.

```
[14] base_dir = "/content"
     imgs_dir = os.path.join(base_dir,'imgs')
     train_dir = os.path.join(imgs_dir,'train')
     test_dir = os.path.join(imgs_dir,'test')
     train_c1 = os.path.join(train_dir,'c1')
     train_c2 = os.path.join(train_dir,'c2')
     train_c3 = os.path.join(train_dir,'c3')
     train_c4 = os.path.join(train_dir,'c4')
     train_c5 = os.path.join(train_dir,'c5')
     train_c6 = os.path.join(train_dir,'c6')
     train_c7 = os.path.join(train_dir,'c7')
     train_c8 = os.path.join(train_dir,'c8')
     train_c9 = os.path.join(train_dir,'c9')
     train_c0 = os.path.join(train_dir,'c0')
```

Figure 18: Assigning data to directories

4.1.3.2 GIVING FILE NAMES :

Here the file names are assigned to list of files in the particular directories as *test_filenames* in for image files in test directory and *c0_filename* for image files in the *train_c0* . similarly all image files in other folders of train directory are assigned in the same manner as shown in the figure below.

```
[ ]  #filenames
     test_filename = os.listdir(test_dir)
```

```
c0_filename = os.listdir(train_c0)
c1_filename = os.listdir(train_c1)
c2_filename = os.listdir(train_c2)
c3_filename = os.listdir(train_c3)
c4_filename = os.listdir(train_c4)
c5_filename = os.listdir(train_c5)
c6_filename = os.listdir(train_c6)
c7_filename = os.listdir(train_c7)
c8_filename = os.listdir(train_c8)
c9_filename = os.listdir(train_c9)
```

Figure 19: Giving filenames to image files

File names of first 4 image files from the assigned folders are printed here using *print(file_name[:4])* as shown below.

```
[ ]  print(test_filename[:4])
     print(c0_filename[:4])
     print(c1_filename[:4])
     print(c2_filename[:4])
     print(c3_filename[:4])
     print(c4_filename[:4])
     print(c5_filename[:4])
     print(c6_filename[:4])
     print(c7_filename[:4])
     print(c8_filename[:4])
     print(c9_filename[:4])
```

```
['img_10827.jpg', 'img_11399.jpg', 'img_13554.jpg', 'img_73985.jpg']
['img_30957.jpg', 'img_98143.jpg', 'img_93208.jpg', 'img_86851.jpg']
['img_35833.jpg', 'img_78001.jpg', 'img_2314.jpg', 'img_30552.jpg']
['img_25781.jpg', 'img_27518.jpg', 'img_86648.jpg', 'img_57904.jpg']
['img_28481.jpg', 'img_48169.jpg', 'img_55817.jpg', 'img_49082.jpg']
['img_43975.jpg', 'img_29986.jpg', 'img_73464.jpg', 'img_64071.jpg']
['img_99978.jpg', 'img_36200.jpg', 'img_80483.jpg', 'img_40370.jpg']
['img_97404.jpg', 'img_72425.jpg', 'img_80694.jpg', 'img_1722.jpg']
['img_15262.jpg', 'img_95855.jpg', 'img_16477.jpg', 'img_36425.jpg']
['img_31951.jpg', 'img_89953.jpg', 'img_5355.jpg', 'img_23932.jpg']
['img_43060.jpg', 'img_19160.jpg', 'img_83436.jpg', 'img_31830.jpg']
```

Figure 20: Printing first four filenames

4.1.4 DISPLAYING THE IMAGE :

The whole dataset contains 22,424 images categorized in 10 classes. (9 classes for different distraction activities and 1 class for safe driving), and figures below shows illustrating images for each category. The size of each image is 640 × 480 pixels. We display these images using *matplotlob.pyplot.imshow(image_read)* . Firstly we read the image using *matplotlob.pyplot.imread(path_of_the_image_file).*

4.1.4.1 DISPLAYING AN SINGLE IMAGE :

- Displaying image from c0 folder in which the driver is driving safely.

```
### Display image in c0 (safe driving)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c0+'/img_83502.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d48bc2f28>
```

Figure 21: Image from c0 (driving safely)

- Displaying image from c1 folder in which the driver is testing-right.

```
### Display image in c1 (texting - right)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c1+'/img_7317.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d48718b70>
```

Figure 22: Image from c1 (texting – right)

- Displaying image from c2 folder in which the driver is talking on the phone – right .

```
### Display image in c2 (talking on the phone - right)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c2+'/img_42941.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d46e85208>
```

Figure 23: Image from c2 (talking on the phone – right)

- Displaying image from c3 folder in which the driver is texting-left.

```
### Display image in c3 (texting - left)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c3+'/img_6132.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d4a045048>
```

Figure 24: Image from c3 (texting-left)

- Displaying image from c4 folder in which the driver is talking on the phone – left .

```
### Display image in c4 (talking on the phone - left)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c4+'/img_57786.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d46e396d8>
```

Figure 25 : Image from c4 (talking on the phone - left)

- Displaying image from c5 folder in which the driver is operating the radio .

```
### Display image in c5 (operating the radio)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c5+'/img_80668.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d46d80b70>
```

Figure 26 : Image from c5 (operating the radio)

- Displaying image from c6 folder in which the driver is drinking.

```
### Display image in c6 (drinking)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c6+'/img_22932.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d46d5fc88>
```

Figure 27: Image from c6 (drinking)

- Displaying image from c7 folder in which the driver is reaching behind.

```
### Display image in c7 (reaching behind)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c7+'/img_80085.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d46cbfda0>
```

Figure 28: Image from c7 (reaching behind)

- Displaying image from c8 folder in which the driver is setting hair and makeup .

```
### Display image in c8 ( hair and makeup )
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c8+'/img_44416.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d46ca1eb8>
```

Figure 29: Image from c8 (hair and makeup)

- Displaying image from c9 folder in which the driver is talking to passenger .

```
### Display image in c9 (talking to passenger)
import matplotlib.pyplot as plt
plt.imshow(plt.imread(train_c9+'/img_64982.jpg'))
```

```
<matplotlib.image.AxesImage at 0x7f4d46c03fd0>
```

Figure 30: Image from c9 (talking to passenger)

4.1.4.2 DISPLAYING SET OF IMAGES

- Displaying set of images from c0 folder in which the driver is driving safely.

```
### Display of set of images in c0 ( safe driving )
plt.figure(figsize=(16,16))
j = 1
for i in range(8):
  img = plt.imread(os.path.join(train_c0,c0_filename[i]))
  plt.subplot(4,4,j)
  plt.imshow(img)
  plt.title(img.shape)
  plt.axis('off')
  j += 1
```

Figure 31: Set of images from c0(driving safely)

- Displaying set of images from c1 folder in which the driver is texting-right.

```
### Display of set of images in c1 ( texting - right)
plt.figure(figsize=(16,16))
j = 1
for i in range(8):
  img = plt.imread(os.path.join(train_c1,c1_filename[i]))
  plt.subplot(4,4,j)
  plt.imshow(img)
  plt.title(img.shape)
  plt.axis('off')
  j += 1
```

Figure 32: Set of images from c1 (texting - right)

- Displaying set of images from c2 folder in which the driver is talking on phone - right.

```
### Display of set of images in c2 (talking on the phone - right)
plt.figure(figsize=(16,16))
j = 1
for i in range(8):
  img = plt.imread(os.path.join(train_c2,c2_filename[i]))
  plt.subplot(4,4,j)
  plt.imshow(img)
  plt.title(img.shape)
  plt.axis('off')
  j += 1
```

Figure 33: Set of images from c2 (talking on the phone - right)

- Displaying set of images from c3 folder in which the driver is texting-left.

```
### Display of set of images in c3 (texting - left)
plt.figure(figsize=(16,16))
j = 1
for i in range(8):
  img = plt.imread(os.path.join(train_c3,c3_filename[i]))
  plt.subplot(4,4,j)
  plt.imshow(img)
  plt.title(img.shape)
  plt.axis('off')
  j += 1
```

Figure 34: Set of images from c3 (texting left)

- Displaying set of images from c4 folder in which the driver is talking on the phone - left.

```
### Display of set of images in c4 (talking on the phone - left)
plt.figure(figsize=(16,16))
j = 1
for i in range(8):
  img = plt.imread(os.path.join(train_c4,c4_filename[i]))
  plt.subplot(4,4,j)
  plt.imshow(img)
  plt.title(img.shape)
  plt.axis('off')
  j += 1
```

Figure 35: Set of images from c4 (talking on the phone-left)

- Displaying set of images from c5 folder in which the driver is operating the radio.

```
### Display of set of images in c5 (operating the radio)
plt.figure(figsize=(16,16))
j = 1
for i in range(8):
  img = plt.imread(os.path.join(train_c5,c5_filename[i]))
  plt.subplot(4,4,j)
  plt.imshow(img)
  plt.title(img.shape)
  plt.axis('off')
  j += 1
```

Figure 36: Set of images from c5 (hair and makeup)

- Displaying set of images from c6 folder in which the driver is drinking.

```
[ ]  ### Display of set of images in c6 (drinking)
     plt.figure(figsize=(16,16))
     j = 1
     for i in range(8):
       img = plt.imread(os.path.join(train_c6,c6_filename[i]))
       plt.subplot(4,4,j)
       plt.imshow(img)
       plt.title(img.shape)
       plt.axis('off')
       j += 1
```

Figure 37: Set of images from c6 (Drinking)

- Displaying set of images from c7 folder in which the driver is reaching behind.

```
### Display of set of images in c7 ( reaching behind )
plt.figure(figsize=(16,16))
j = 1
for i in range(8):
  img = plt.imread(os.path.join(train_c7,c7_filename[i]))
  plt.subplot(4,4,j)
  plt.imshow(img)
  plt.title(img.shape)
  plt.axis('off')
  j += 1
```

Figure 38: Set of images from c7 (Reaching Behind)

- Displaying set of images from c8 folder in which the driver is setting hair and makeup.

```
] ### Display of set of images in c8 ( hair and makeup )
  plt.figure(figsize=(16,16))
  j = 1
  for i in range(8):
    img = plt.imread(os.path.join(train_c8,c8_filename[i]))
    plt.subplot(4,4,j)
    plt.imshow(img)
    plt.title(img.shape)
    plt.axis('off')
    j += 1
```

Figure 39: Set of images from c8 (hair and makeup)

- Displaying set of images from c9 folder in which the driver is talking to passenger.

```
[ ] ### Display of set of images in c9 (talking to passenger)
    plt.figure(figsize=(16,16))
    j = 1
    for i in range(8):
      img = plt.imread(os.path.join(train_c9,c9_filename[i]))
      plt.subplot(4,4,j)
      plt.imshow(img)
      plt.title(img.shape)
      plt.axis('off')
      j += 1
```

Figure 40: Set of images from c9 (talking to passenger)

4.2 PREPROCESSING OF DATA :

- Read the image data
- We have different shapes here.
- We need same shape for each image
- reshaping the images
- c0,c1,c2,c3,c4,c5,c6,c7,c8,c9 labelling

4.2.1 CREATING TRAIN AND VALIDATION DATA

We are importing the libraries required which are tenserflow, keras, and from them we are importing image data generator.
Using Imagedatagenerator we are splitting the data from *train_dir* to training data and validation data using key word *validation_split.* Since we are giving validation_split = 0.3 , 70% of data is split into training data and 30% of the data is split into validation data. And we are scaling the image using the *rescale* key word .Since 255 is the maximin pixel value. Rescale 1./255 is to transform every pixel value from range [0,255] -> [0,1]. We are rescaling all the images of train and validation and storing them in ***train*** and ***val***.
Now we are dividing the images into 20 batches and each batch size is 244x244 using the class mode as categorical , the class mode is categorical because we have 10 classes .
These divided batches of training data images will be stored in ***train*** and validation data images will be stored in ***val***.

```
[ ] from tensorflow.keras.preprocessing.image import ImageDataGenerator

    # create a new generator
    imagegen = ImageDataGenerator(rescale=1./255, validation_split=0.3)
    # load train data
    train = imagegen.flow_from_directory(train_dir, class_mode="categorical",subset="training", batch_size=128, target_size=(224, 224))
    # load val data
    val = imagegen.flow_from_directory(train_dir, class_mode="categorical",subset="validation", batch_size=128, target_size=(224, 224))
```

```
Found 15702 images belonging to 10 classes.
Found 6722 images belonging to 10 classes.
```

Figure 41: Creating train and validation data from train_dir

We split the training data into two sets: training set containing 15,702 images, and validation set containing 6722 belonging to 10 classes each.

```
[ ] train
```

```
<keras_preprocessing.image.directory_iterator.DirectoryIterator at 0x7f4d46c2e550>
```

Figure 42: checking whether image is assigned

4.2.1.1 GENERATING RANDOM IMAGES USING IMAGE GENERATOR

- **SINGLE IMAGE**

Using *train.next()* we are printing random image from training data. The random image generated is one from the 15702 images.

```
imgs,labels = train.next()
print(imgs.shape)
print(labels.shape)
plt.imshow(imgs[0,:,:,:])
plt.title(labels[i])
```

```
(128, 224, 224, 3)
(128, 10)
/usr/local/lib/python3.6/dist-packages/matplotlib/t
  if s != self._text:
Text(0.5, 1.0, '[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]')
```

Figure 43: display image using image generator

- **SET OF IMAGES**

  Here , we are printing random images from training data. The random images generated are from the 15702 images.

```
# set of random images
plt.figure(figsize=(16,16))
pos=1 ##plot position
for i in range(20):
  plt.subplot(4,5,pos)
  plt.imshow(imgs[i,:,:,:])# To display the image
  plt.title(labels[i])
  plt.axis('off')
  pos+=1
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/text.py:1165: FutureWarning: elementwise comparison failed; returning
  if s != self._text:
```

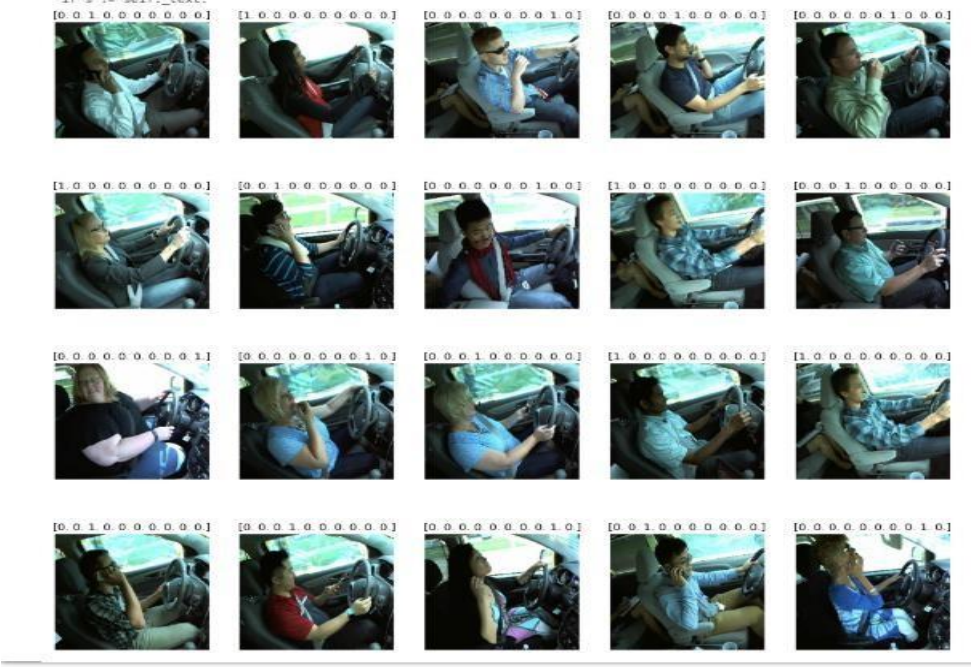

Figure 44: Random images using image generator

- **HISTOGRAM (PIXEL DISTRIBUTION)**

```
[ ]  plt.figure(figsize=(16,16))
     pos = 1  ## plot position
     for i in range(20):
       plt.subplot(4,5,pos)
       plt.hist(imgs[i,:,:,:].flat) # To display the histogram
       plt.title(labels[i])
       pos += 1
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/text.py:1165: FutureWarning: elementwise comparison failed; returning scalar in
  if s != self._text:
```

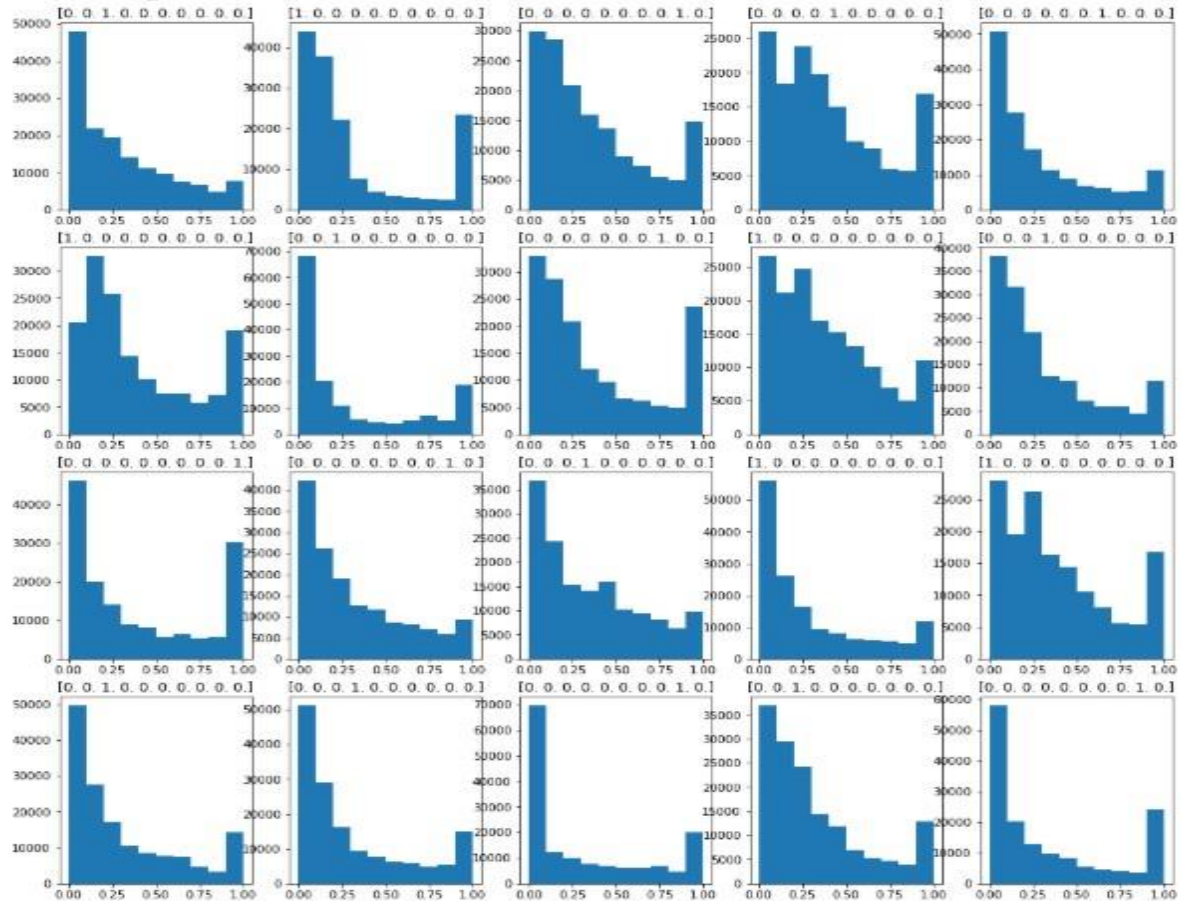

Figure 45: histogram plot

4.3 MODEL

4.3.1 BUILDING THE MODEL

The sequential function into model is invoked and using the function we are adding the convo layers by maxpooling .
We are giving different sizes for different layers of convo using convo2D and activation as relu
The final output is given with dense 10 as we have only 2 classes , activation as softmax .
Finally calling summary for knowing the total parameters of model. The output has total 4,19,460 parameters in which 4,19,220 are trainable and 240 are non-trainable.

```
[ ] ### Build the model
    from keras.models import Sequential
    from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, InputLayer, BatchNormalization, Dropout

    # build a sequential model
    model = Sequential()
    model.add(InputLayer(input_shape=(224, 224, 3)))

    # 1st conv block
    model.add(Conv2D(25, (5, 5), activation='relu', strides=(1, 1), padding='same'))
    model.add(MaxPool2D(pool_size=(2, 2), padding='same'))
    # 2nd conv block
    model.add(Conv2D(50, (5, 5), activation='relu', strides=(2, 2), padding='same'))
    model.add(MaxPool2D(pool_size=(2, 2), padding='same'))
    model.add(BatchNormalization())
    # 3rd conv block
    model.add(Conv2D(70, (3, 3), activation='relu', strides=(2, 2), padding='same'))
    model.add(MaxPool2D(pool_size=(2, 2), padding='valid'))
    model.add(BatchNormalization())
    # ANN block
    model.add(Flatten())
    model.add(Dense(units=100, activation='relu'))
    model.add(Dense(units=100, activation='relu'))
    model.add(Dropout(0.25))
    # output layer
    model.add(Dense(units=10, activation='softmax'))

    model.summary()
```

```
Using TensorFlow backend.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 25)      1900
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 25)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 56, 50)        31300
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 50)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 50)        200
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 70)        31570
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 70)          0
_________________________________________________________________
batch_normalization_2 (Batch (None, 7, 7, 70)          280
_________________________________________________________________
flatten_1 (Flatten)          (None, 3430)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               343100
_________________________________________________________________
dense_2 (Dense)              (None, 100)               10100
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
=================================================================
Total params: 419,460
Trainable params: 419,220
Non-trainable params: 240
_________________________________________________________________
```

Figure 46 : Creating the model

4.3.2 COMPILING THE MODEL

The model is compiled using compile function and loss which uses categorical cross entropy function and the metrics as accuracy.

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

Figure 47: Compiling the model.

4.3.3 TRAIN THE MODEL

```
[ ]  # fit on data for 30 epochs
     history = model.fit_generator(train, epochs=30, validation_data=val)
```

```
[ ] Epoch 1/30
    123/123 [==============================] - 133s 1s/step - loss: 1.0089 - accuracy: 0.6593 - val_loss: 2.1835 - val_accuracy: 0.2269
    Epoch 2/30
    123/123 [==============================] - 123s 1s/step - loss: 0.1232 - accuracy: 0.9674 - val_loss: 1.9122 - val_accuracy: 0.3816
    Epoch 3/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0542 - accuracy: 0.9852 - val_loss: 0.2124 - val_accuracy: 0.9328
    Epoch 4/30
    123/123 [==============================] - 126s 1s/step - loss: 0.0325 - accuracy: 0.9903 - val_loss: 0.0796 - val_accuracy: 0.9689
    Epoch 5/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0197 - accuracy: 0.9946 - val_loss: 0.0365 - val_accuracy: 0.9857
    Epoch 6/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0120 - accuracy: 0.9969 - val_loss: 0.0223 - val_accuracy: 0.9908
    Epoch 7/30
    123/123 [==============================] - 124s 1s/step - loss: 0.0273 - accuracy: 0.9915 - val_loss: 0.0301 - val_accuracy: 0.9851
    Epoch 8/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0250 - accuracy: 0.9926 - val_loss: 0.0721 - val_accuracy: 0.9558
    Epoch 9/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0191 - accuracy: 0.9936 - val_loss: 0.0455 - val_accuracy: 0.9701
    Epoch 10/30
    123/123 [==============================] - 124s 1s/step - loss: 0.0148 - accuracy: 0.9954 - val_loss: 0.1132 - val_accuracy: 0.9573
    Epoch 11/30
    123/123 [==============================] - 124s 1s/step - loss: 0.0151 - accuracy: 0.9959 - val_loss: 0.1090 - val_accuracy: 0.9691
    Epoch 12/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0141 - accuracy: 0.9952 - val_loss: 0.0292 - val_accuracy: 0.9918
    Epoch 13/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0047 - accuracy: 0.9987 - val_loss: 0.0724 - val_accuracy: 0.9917
    Epoch 14/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0048 - accuracy: 0.9987 - val_loss: 0.0028 - val_accuracy: 0.9894
    Epoch 15/30
    123/123 [==============================] - 125s 1s/step - loss: 0.0066 - accuracy: 0.9977 - val_loss: 0.0751 - val_accuracy: 0.9857
Epoch 16/30
123/123 [==============================] - 125s 1s/step - loss: 0.0090 - accuracy: 0.9969 - val_loss: 0.0085 - val_accuracy: 0.9912
Epoch 17/30
123/123 [==============================] - 123s 1s/step - loss: 0.0082 - accuracy: 0.9972 - val_loss: 0.0588 - val_accuracy: 0.9749
Epoch 18/30
123/123 [==============================] - 123s 1s/step - loss: 0.0297 - accuracy: 0.9905 - val_loss: 0.0989 - val_accuracy: 0.9540
Epoch 19/30
123/123 [==============================] - 123s 1s/step - loss: 0.0186 - accuracy: 0.9943 - val_loss: 0.0075 - val_accuracy: 0.9912
Epoch 20/30
123/123 [==============================] - 124s 1s/step - loss: 0.0084 - accuracy: 0.9977 - val_loss: 0.0060 - val_accuracy: 0.9949
Epoch 21/30
123/123 [==============================] - 124s 1s/step - loss: 0.0058 - accuracy: 0.9982 - val_loss: 0.0267 - val_accuracy: 0.9866
Epoch 22/30
123/123 [==============================] - 124s 1s/step - loss: 0.0034 - accuracy: 0.9992 - val_loss: 0.0643 - val_accuracy: 0.9952
Epoch 23/30
123/123 [==============================] - 125s 1s/step - loss: 0.0061 - accuracy: 0.9981 - val_loss: 0.2866 - val_accuracy: 0.9878
Epoch 24/30
123/123 [==============================] - 123s 998ms/step - loss: 0.0042 - accuracy: 0.9987 - val_loss: 0.0016 - val_accuracy: 0.9908
Epoch 25/30
123/123 [==============================] - 123s 1s/step - loss: 0.0130 - accuracy: 0.9966 - val_loss: 0.1332 - val_accuracy: 0.9914
Epoch 26/30
123/123 [==============================] - 122s 993ms/step - loss: 0.0100 - accuracy: 0.9975 - val_loss: 0.0487 - val_accuracy: 0.9790
Epoch 27/30
123/123 [==============================] - 122s 993ms/step - loss: 0.0171 - accuracy: 0.9953 - val_loss: 0.3090 - val_accuracy: 0.9863
Epoch 28/30
123/123 [==============================] - 123s 1s/step - loss: 0.0070 - accuracy: 0.9979 - val_loss: 0.0243 - val_accuracy: 0.9926
Epoch 29/30
123/123 [==============================] - 123s 997ms/step - loss: 0.0046 - accuracy: 0.9983 - val_loss: 3.8501e-04 - val_accuracy: 0.9946
Epoch 30/30
123/123 [==============================] - 123s 998ms/step - loss: 0.0023 - accuracy: 0.9994 - val_loss: 4.6670e-05 - val_accuracy: 0.9940
```

Figure 48: Training the model

In the first epoch the loss value is 1.0089 , accuracy value is 0.6953, val loss is 2.1835 and val accuracy is 0.2269 .
By the end of 30th epoch the loss value has decreased to 0.023 and and the accuracy increased to 0.9994 and val accuracy to 0.9940
The overall accuracy of the model is 99% which is a best fit model.

4.3.3.1 PLOT FOR TRAINING ACCURACY – VALIDATION ACCURACY AND TRAIN LOSS – VALIDATION LOSS :

Reading the total history of epochs for all the training and validation data and plotting the graph for accuracy and loss of training and validation data.

```
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = list(range(1,31))
plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
plt.plot(epochs,train_acc,label='train_acc')
plt.plot(epochs,val_acc,label='val_acc')
plt.title('accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(epochs,train_loss,label='train_loss')
plt.plot(epochs,val_loss,label='val_loss')
plt.title('loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f4d404bfeb8>
```

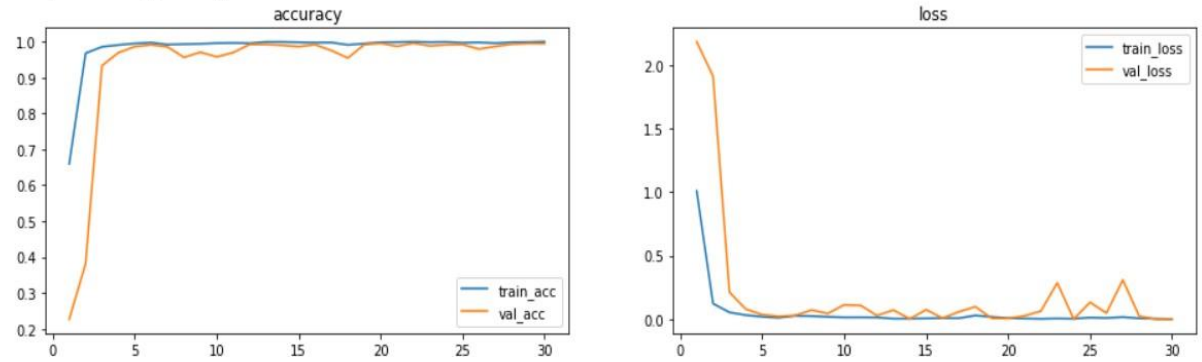

Figure 49: Graph of training model

The graph has accuracy increasing and loss decreasing for training data Accuracy and loss are fluctuating for validation data. Overall its increasing so it’s a best fit model.

4.4 TO PREDICT AN IMAGE

- Read the image
- check the shape
- Resize into required shape(1,244 X 244 X 3)
- Apply scaling

4.4.1 EVALUATING ON TRAIN DATA

4.4.1.1 Evaluation on train data – c0 label (driving safely)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c0' folder path is given , the randome_filename is chosen from the c0 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ]  import random, os
     path = '/content/imgs/train/c0'
     random_filename_train_c0 = random.choice([
         x for x in os.listdir(path)
         if os.path.isfile(os.path.join(path, x))
     ])
     img = plt.imread(os.path.join(train_dir,train_c0,random_filename_train_c0))
     print(type(img))
     img = tf.keras.preprocessing.image.img_to_array(img)
     print(img.shape)
     print(type(img))
     img = tf.image.resize(img,(224,224))
     ## Scaling
     img = img/255
     print(img.shape)
     img = np.expand_dims(img,axis=0)
     print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

Figure 50:Reading image from c0

```
[ ]  model.predict(img)
```

```
array([[9.9999976e-01, 2.0026547e-10, 4.4037674e-10, 1.7705308e-11,
        2.3044292e-11, 6.1268850e-08, 1.5863412e-13, 6.8490455e-13,
        1.2537706e-12, 9.9305673e-08]], dtype=float32)
```

Figure 51 : predicting the image from c0

```
[ ]  classes = model.predict_classes(img)
     print (classes)
     img = plt.imread(os.path.join(train_dir,train_c0,random_filename_train_c0))
     plt.imshow(img)
```

```
[0]
<matplotlib.image.AxesImage at 0x7f4d40201d30>
```

Figure 52: predicting class and displaying image from c0

It is predicting correctly that the image belongs to class 0 i.e; folder c0(driving safely)

4.4.1.2 Evaluation on train data – c1 label (texting-right)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c1' folder path is given , the randome_filename is chosen from the c1 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ] import random, os
    path = '/content/imgs/train/c1'
    random_filename_train_c1 = random.choice([
        x for x in os.listdir(path)
        if os.path.isfile(os.path.join(path, x))
    ])
    img = plt.imread(os.path.join(train_dir,train_c1,random_filename_train_c1))
    print(type(img))
    img = tf.keras.preprocessing.image.img_to_array(img)
    print(img.shape)
    print(type(img))
    img = tf.image.resize(img,(224,224))
    ## Scaling
    img = img/255
    print(img.shape)
    img = np.expand_dims(img,axis=0)
    print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
[ ] model.predict(img)
```

```
array([[5.1119917e-09, 1.0000000e+00, 3.4008107e-12, 6.1899471e-13,
        1.6164382e-14, 1.2823806e-15, 2.0951440e-11, 8.7369660e-14,
        1.0984864e-11, 1.8616346e-10]], dtype=float32)
```

```
[ ] classes = model.predict_classes(img)
    print (classes)
    img = plt.imread(os.path.join(train_dir,train_c1,random_filename_train_c1))
    plt.imshow(img)
```

```
[1]
<matplotlib.image.AxesImage at 0x7f4d40160f28>
```

Figure 53: predicting image in c1

It is predicting correctly that the image belongs to class 1 i.e; folder c1 (texting-right)

4.4.1.3 Evaluation on train data – c2 label (Talking on the phone – right)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c2' folder path is given , the randome_filename is chosen from the c2 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ] import random, os
    path = '/content/imgs/train/c2'
    random_filename_train_c2 = random.choice([
        x for x in os.listdir(path)
        if os.path.isfile(os.path.join(path, x))
    ])
    img = plt.imread(os.path.join(train_dir,train_c2,random_filename_train_c2))
    print(type(img))
    img = tf.keras.preprocessing.image.img_to_array(img)
    print(img.shape)
    print(type(img))
    img = tf.image.resize(img,(224,224))
    ## Scaling
    img = img/255
    print(img.shape)
    img = np.expand_dims(img,axis=0)
    print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
[ ] model.predict(img)
```

```
array([[6.8645011e-07, 8.1706082e-09, 9.9999821e-01, 4.7283766e-10,
        3.5840003e-08, 1.8087880e-08, 3.0804628e-08, 2.8262457e-09,
        2.3342585e-08, 9.2469611e-07]], dtype=float32)
```

```
[ ] classes = model.predict_classes(img)
    print (classes)
    img = plt.imread(os.path.join(train_dir,train_c2,random_filename_train_c2))
    plt.imshow(img)
```

```
[2]
<matplotlib.image.AxesImage at 0x7f4d400c6160>
```

Figure 54: predicting image in c2

It is predicting correctly that the image belongs to class 2 i.e; folder c2 (talking on phone - right)

4.4.1.4 Evaluation on train data – c3 label (texting-left)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c3' folder path is given , the randome_filename is chosen from the c3 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ] import random, os
    path = '/content/imgs/train/c3'
    random_filename_train_c3 = random.choice([
        x for x in os.listdir(path)
        if os.path.isfile(os.path.join(path, x))
    ])
    img = plt.imread(os.path.join(train_dir,train_c3,random_filename_train_c3))
    print(type(img))
    img = tf.keras.preprocessing.image.img_to_array(img)
    print(img.shape)
    print(type(img))
    img = tf.image.resize(img,(224,224))
    ## Scaling
    img = img/255
    print(img.shape)
    img = np.expand_dims(img,axis=0)
    print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
[ ] model.predict(img)
```

```
array([[2.0995181e-13, 1.9784053e-11, 5.5822135e-15, 1.0000000e+00,
        3.2226588e-10, 4.2082304e-15, 4.3444958e-16, 5.8841934e-16,
        1.0595154e-15, 1.0693311e-12]], dtype=float32)
```

```
] classes = model.predict_classes(img)
  print (classes)
  img = plt.imread(os.path.join(train_dir,train_c3,random_filename_train_c3))
  plt.imshow(img)
```

```
[3]
<matplotlib.image.AxesImage at 0x7f4d400a3358>
```

Figure 55: predicting image in c3

It is predicting correctly that the image belongs to class 3 i.e; folder c3 (texting left)

4.4.1.5 Evaluation on train data – c4 label (talking on the phone - left)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c4' folder path is given , the randome_filename is chosen from the c4 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ] import random, os
    path = '/content/imgs/train/c4'
    random_filename_train_c4 = random.choice([
        x for x in os.listdir(path)
        if os.path.isfile(os.path.join(path, x))
    ])
    img = plt.imread(os.path.join(train_dir,train_c4,random_filename_train_c4))
    print(type(img))
    img = tf.keras.preprocessing.image.img_to_array(img)
    print(img.shape)
    print(type(img))
    img = tf.image.resize(img,(224,224))
    ## Scaling
    img = img/255
    print(img.shape)
    img = np.expand_dims(img,axis=0)
    print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
] model.predict(img)
```

```
array([[4.9643771e-14, 2.6963228e-17, 2.4167439e-12, 5.5139232e-13,
        1.0000000e+00, 7.2880374e-12, 4.8854920e-10, 2.3813162e-14,
        4.9605343e-14, 3.5687548e-15]], dtype=float32)
```

```
] classes = model.predict_classes(img)
  print (classes)
  img = plt.imread(os.path.join(train_dir,train_c4,random_filename_train_c4))
  plt.imshow(img)
```

```
[4]
<matplotlib.image.AxesImage at 0x7f4d40080550>
```

Figure 56: predicting image in c4

It is predicting correctly that the image belongs to class 4 i.e; folder c4(talking on the phone -left)

4.4.1.6 Evaluation on train data – c5 label (operating radio)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c5' folder path is given , the randome_filename is chosen from the c5 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
import random, os
path = '/content/imgs/train/c5'
random_filename_train_c5 = random.choice([
    x for x in os.listdir(path)
    if os.path.isfile(os.path.join(path, x))
])
img = plt.imread(os.path.join(train_dir,train_c5,random_filename_train_c5))
print(type(img))
img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(224,224))
## Scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
[ ]  model.predict(img)
```

```
array([[3.2606264e-22, 4.1419827e-27, 9.2054927e-20, 5.5767603e-24,
        1.0298411e-21, 1.0000000e+00, 3.3256662e-18, 1.9299505e-21,
        7.1177147e-20, 2.0911966e-21]], dtype=float32)
```

```
[ ]  classes = model.predict_classes(img)
     print (classes)
     img = plt.imread(os.path.join(train_dir,train_c5,random_filename_train_c5))
     plt.imshow(img)
```

```
[5]
<matplotlib.image.AxesImage at 0x7f4cee294748>
```

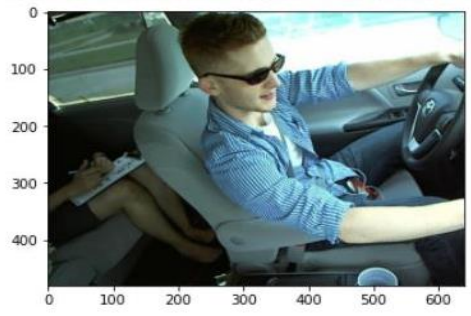

Figure 57: predicting image in c5

It is predicting correctly that the image belongs to class 5 i.e; folder c5 (operating radio)

4.4.1.7 Evaluation on train data – c6 label (drinking)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c6' folder path is given , the randome_filename is chosen from the c6 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ] import random, os
    path = '/content/imgs/train/c6'
    random_filename_train_c6 = random.choice([
        x for x in os.listdir(path)
        if os.path.isfile(os.path.join(path, x))
    ])
    img = plt.imread(os.path.join(train_dir,train_c6,random_filename_train_c6))
    print(type(img))
    img = tf.keras.preprocessing.image.img_to_array(img)
    print(img.shape)
    print(type(img))
    img = tf.image.resize(img,(224,224))
    ## Scaling
    img = img/255
    print(img.shape)
    img = np.expand_dims(img,axis=0)
    print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
[ ] model.predict(img)
```

```
array([[3.5145983e-11, 8.3368333e-11, 6.9514206e-10, 1.2311784e-14,
        3.1039483e-11, 2.1551223e-06, 9.9999774e-01, 1.5948664e-09,
        1.1167253e-08, 4.8694286e-08]], dtype=float32)
```

```
[ ] classes = model.predict_classes(img)
    print (classes)
    img = plt.imread(os.path.join(train_dir,train_c6,random_filename_train_c6))
    plt.imshow(img)
```

```
[6]
<matplotlib.image.AxesImage at 0x7f4cee271940>
```

Figure 58: predicting image in c6

It is predicting correctly that the image belongs to class 6 i.e; folder c6 (drinking)

4.4.1.8 Evaluation on train data – c7 label (reaching behind)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c7' folder path is given , the randome_filename is chosen from the c7 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ]  import random, os
     path = '/content/imgs/train/c7'
     random_filename_train_c7 = random.choice([
         x for x in os.listdir(path)
         if os.path.isfile(os.path.join(path, x))
     ])
     img = plt.imread(os.path.join(train_dir,train_c7,random_filename_train_c7))
     print(type(img))
     img = tf.keras.preprocessing.image.img_to_array(img)
     print(img.shape)
     print(type(img))
     img = tf.image.resize(img,(224,224))
     ## Scaling
     img = img/255
     print(img.shape)
     img = np.expand_dims(img,axis=0)
     print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
]  model.predict(img)
```

```
array([[4.96079889e-14, 2.44301024e-10, 1.02159105e-17, 7.09950260e-17,
        9.49601755e-17, 4.79142140e-20, 7.74991358e-19, 1.00000000e+00,
        7.43653472e-11, 1.32216454e-15]], dtype=float32)
```

```
]  classes = model.predict_classes(img)
   print (classes)
   img = plt.imread(os.path.join(train_dir,train_c7,random_filename_train_c7))
   plt.imshow(img)
```

```
[7]
<matplotlib.image.AxesImage at 0x7f4cee1d9630>
```

Figure 59 : predicting image in c7

It is predicting correctly that the image belongs to class 7 i.e; folder c7 (reaching behind)

4.4.1.9 Evaluation on train data – c8 label (hair and makeup)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c8' folder path is given , the randome_filename is chosen from the c8 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
import random, os
path = '/content/imgs/train/c8'
random_filename_train_c8 = random.choice([
    x for x in os.listdir(path)
    if os.path.isfile(os.path.join(path, x))
])
img = plt.imread(os.path.join(train_dir,train_c8,random_filename_train_c8))
print(type(img))
img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(224,224))
## Scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
model.predict(img)
```

```
array([[9.1126780e-16, 4.2527506e-17, 7.9168960e-17, 1.3779675e-16,
        3.1103898e-12, 3.1615623e-14, 1.2547343e-17, 1.8390773e-12,
        1.0000000e+00, 7.5297529e-14]], dtype=float32)
```

```
classes = model.predict_classes(img)
print (classes)
img = plt.imread(os.path.join(train_dir,train_c8,random_filename_train_c8))
plt.imshow(img)
```

```
[8]
<matplotlib.image.AxesImage at 0x7f4cee1bb3c8>
```

Figure 60 : predicting image in c8

It is predicting correctly that the image belongs to class 8 i.e; folder c8 (hair and makeup)

4.4.1.10 Evaluation on train data – c9 label (talking to the passenger)

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'c9' folder path is given , the randome_filename is chosen from the c9 folder.
Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ] import random, os
    path = '/content/imgs/train/c9'
    random_filename_train_c9 = random.choice([
        x for x in os.listdir(path)
        if os.path.isfile(os.path.join(path, x))
    ])
    img = plt.imread(os.path.join(train_dir,train_c9,random_filename_train_c9))
    print(type(img))
    img = tf.keras.preprocessing.image.img_to_array(img)
    print(img.shape)
    print(type(img))
    img = tf.image.resize(img,(224,224))
    ## Scaling
    img = img/255
    print(img.shape)
    img = np.expand_dims(img,axis=0)
    print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

```
[ ] model.predict(img)
```

```
array([[1.9347329e-08, 8.8398429e-13, 7.6585391e-13, 3.5104835e-13,
        1.4782972e-10, 8.0906125e-07, 3.3566406e-13, 4.9015028e-09,
        2.9455432e-05, 9.9996972e-01]], dtype=float32)
```

```
[ ] classes = model.predict_classes(img)
    print (classes)
    img = plt.imread(os.path.join(train_dir,train_c9,random_filename_train_c9))
    plt.imshow(img)
```

```
[9]
<matplotlib.image.AxesImage at 0x7f4cee11e160>
```

Figure 61: predicting image in c9

It is predicting correctly that the image belongs to class 9 i.e; folder c9 (talking to passenger)

4.4.2 EVALUATING WITH TEST DATA:

Here we are creating a *random_filename* which automatically chooses a random image file . since the 'test' directory path is given , the randome_filename is chosen from the test directory among 79726 image files. Now the image of that randomly generated image file is been imported and is been read and plot.
The image is converted into array and is resized and scaled if necessary and then it is predicted using model.predict().

```
[ ] import random, os
    path = '/content/imgs/test'
    random_filename_test = random.choice([
        x for x in os.listdir(path)
        if os.path.isfile(os.path.join(path, x))
    ])
    from tensorflow.keras.preprocessing import image
    import numpy as np
    img = plt.imread(os.path.join(test_dir,random_filename_test))
    print(type(img))
    img = tf.keras.preprocessing.image.img_to_array(img)
    print(img.shape)
    print(type(img))
    img = tf.image.resize(img,(224,224))
    ## Scaling
    img = img/255
    print(img.shape)
    img = np.expand_dims(img,axis=0)
    print(img.shape)
```

```
<class 'numpy.ndarray'>
(480, 640, 3)
<class 'numpy.ndarray'>
(224, 224, 3)
(1, 224, 224, 3)
```

Figure 62: Reading image from test_dir

Here the model is been predicted and the predicted values in the array and data type is displayed in the output.

```
] model.predict(img)
```

```
array([[8.1827110e-16, 3.1104974e-19, 2.0495501e-16, 1.4424300e-15,
        1.1675642e-13, 1.0000000e+00, 4.5961959e-14, 1.4540821e-14,
        7.5376552e-18, 7.1783173e-16]], dtype=float32)
```

Figure 63:Predicting the image from test_dir

The class to which the randomly generated image belongs to is predicted here .

```
classes = model.predict_classes(img)
print ("class : ",classes)
if (classes == 0):
  print("safely driving")
if (classes == 1):
  print("texting - right")
if (classes == 2):
  print("talking on the phone - right")
if (classes == 3):
  print("texting - left")
if (classes == 4):
  print(" texting - left")
if (classes == 5):
  print(" operating the radio")
if (classes == 6):
  print("drinking")
if (classes == 7):
  print("reaching behind")
if (classes == 8):
  print("hair and makeup")
if (classes == 9):
  print("talking to passenger")
```

```
class :  [5]
 operating the radio
```

Figure 64: Predicting class of the image

For the above input image the predicted class was 5 which means the image belongs to c5 folder in which the driver is operating the radio.

Let us check the image to see whether the driver is operating radio or not . Does the driver really belong to c5 folder.

```
img = plt.imread(os.path.join(test_dir,random_filename_test))
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f4cee0245c0>
```

Figure 65: Displaying and checking the image

After plotting the image we can see that the driver is operating radio thus belongs to c5 folder i.e; class 5 .

Therefore, it classified correctly for the test data . Thus the model is best fit for the given data .

CHAPTER 5

CONCLUSION

The proposed work is designed to develop a model to detect, recognize and classify distracted driver. The softwares used to test the functionality are Anaconda and python 3 and google colaboratory. For driver detection dataset we used convolution neural network model for distracted driver classification. The os , matplotlib, tenserflow and other libraries works in support with python programming. The performance measures are validated with the CNN model designed with an accuracy .
However the results prove that the network architecture designed has better advancements and this application is widely used to achieve distracted driver detection and classification .

In conclusion, we successfully implemented a Convolutional Neural Network model that has great performance on the distracted driver detection task and gives an evaluation accuracy of 99.4%. Still, the Convolutional Neural Network has many advantages on image classification tasks compared to traditional machine learning techniques.Besides, the dataset also provides 77,000 test images without labels, so if these images are labelled or we use semi-supervised learning method, our models could be developed with these images. Finally, we could try out some existing packages from Pytorch or Tensorflow of more complex CNN-based.

CHAPTER 6

REFERENCES:

DATA SET LINK: https://www.kaggle.com/c/state-farm-distracted-driver-detection/data

TENSOR FLOW LINK: https://www.tensorflow.org/install/errors

MACHINE LEARNING LINK: https://en.wikipedia.org/wiki/Machine_learning

PYTHON LINKS: https://www.python.org/

https://en.wikipedia.org/wiki/Python_(programming_language)