CCSP AI Project

Neeraj Appari

1)Topic-Prediction model for Global NCAP

2)Objective-

Using the Car Crash Test reports finding optimal Model

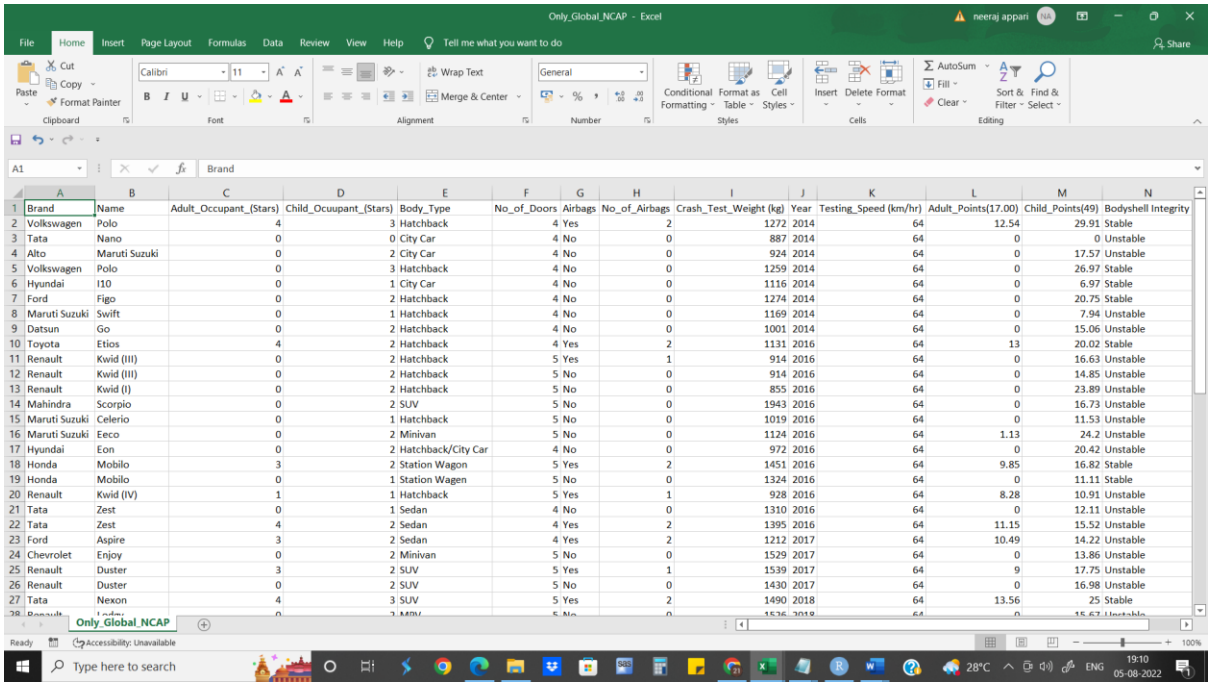
3)Dataset-

Global NCAP of INDIA

It contains

- Brand-Which brand the vehicle belongs to
- Name- Name of the vehicle
- Adult_Occupant(Stars)-Total rating obtained by the vehicle in terms of adults or front passengers
- Child_Occupant(Stars)-Total rating obtained by the vehicle in terms of children or back passengers
- Body_Type-Body type of the particular vehicle
- No_of_Doors-Number of doors a vehicle has
- Airbags-Does the vehicle contain airbags or not
- No_of_Airbags-If the vehicle contains airbags then how many airbags does it have

- Crash_Weight_Test-Weight of the testing vehicle
- Year-Year of production
- Testing_Speed-Speed at which the vehicles are tested
- Adult_Points(17.00)-Total points a vehicle has scored in terms of adults with a max of 17.00
- Child_Points(49.00)-Total points a vehicle has scored in terms of children with a max of 49.00
- Bodyshell_Integrity-Stability of the vehicle after the crash



Brand	Name	Adult_Occupant_(Stars)	Child_Ocuupant_(Stars)	Body_Type	No_of_Doors	Airbags	No_of_Airbags	Crash_Test_Weight (kg)	Year	Testing_Speed (km/hr)	Adult_Points(17.00)	Child_Points(49)	Bodyshell Integrity
Volkswagen	Polo	4	3	Hatchback	4	Yes	2	1272	2014	64	12.54	29.91	Stable
Tata	Nano	0	0	City Car	4	No	0	887	2014	64	0	0	Unstable
Alto	Maruti Suzuki	0	2	City Car	4	No	0	924	2014	64	0	17.57	Unstable
Volkswagen	Polo	0	3	Hatchback	4	No	0	1259	2014	64	0	26.97	Stable
Hyundai	I10	0	1	City Car	4	No	0	1116	2014	64	0	6.97	Stable
Ford	Figo	0	2	Hatchback	4	No	0	1274	2014	64	0	20.75	Stable
Maruti Suzuki	Swift	0	1	Hatchback	4	No	0	1169	2014	64	0	7.94	Unstable
Datsun	Go	0	2	Hatchback	4	No	0	1001	2014	64	0	15.06	Unstable
Toyota	Etios	4	2	Hatchback	4	Yes	2	1131	2016	64	13	20.02	Stable
Renault	Kwid (III)	0	2	Hatchback	5	Yes	1	914	2016	64	0	16.63	Unstable
Renault	Kwid (III)	0	2	Hatchback	5	No	0	914	2016	64	0	14.85	Unstable
Renault	Kwid (I)	0	2	Hatchback	5	No	0	855	2016	64	0	23.89	Unstable
Mahindra	Scorpio	0	2	SUV	5	No	0	1943	2016	64	0	16.73	Unstable
Maruti Suzuki	Celerio	0	1	Hatchback	5	No	0	1019	2016	64	0	11.53	Unstable
Maruti Suzuki	Eeco	0	2	Minivan	5	No	0	1124	2016	64	1.13	24.2	Unstable
Hyundai	Eon	0	2	Hatchback/City Car	4	No	0	972	2016	64	0	20.42	Unstable
Honda	Mobilo	3	2	Station Wagon	5	Yes	2	1451	2016	64	9.85	16.82	Stable
Honda	Mobilo	0	1	Station Wagen	5	No	0	1324	2016	64	0	11.11	Stable
Renault	Kwid (IV)	1	1	Hatchback	5	Yes	1	928	2016	64	8.28	10.91	Unstable
Tata	Zest	0	1	Sedan	4	No	0	1310	2016	64	0	12.11	Unstable
Tata	Zest	4	2	Sedan	4	Yes	2	1395	2016	64	11.15	15.52	Unstable
Ford	Aspire	3	2	Sedan	4	Yes	2	1212	2017	64	10.49	14.22	Unstable
Chevrolet	Enjoy	0	2	Minivan	5	No	0	1529	2017	64	0	13.86	Unstable
Renault	Duster	3	2	SUV	5	Yes	1	1539	2017	64	9	17.75	Unstable
Renault	Duster	0	2	SUV	5	No	0	1430	2017	64	0	16.98	Unstable
Tata	Nexon	4	3	SUV	5	Yes	2	1490	2018	64	13.56	25	Stable

Dataset contains the data taken from Global NCAP with is a non-profit organisation which tests cars for India and Africa

---

*FactSheets

#SAFERCARSFORINDIA

DATSUN GO - NO AIRBAGS

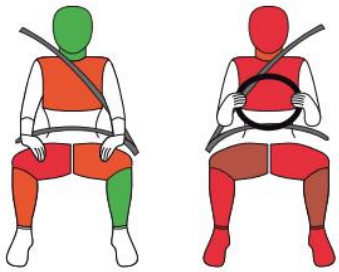
0.00 max. 17.00 Adult

15.06 max. 49.00 Child

Tested at 64 km/h

Bodyshell integrity: UNSTABLE

ADULT OCCUPANT PROTECTION



FRONT PASSENGER DRIVER

- GOOD
- ADEQUATE
- MARGINAL
- WEAK
- POOR

CAR DETAILS

TESTED MODEL	DATSUN GO, RHD
BODY TYPE	4 DOOR HATCHBACK
CRASH TEST WEIGHT	KG 1001
YEAR OF PUBLICATION	2014

GLOBAL NCAP

#SAFERCARSFORINDIA

Honda CITY (4th Gen) - 2 AIRBAGS

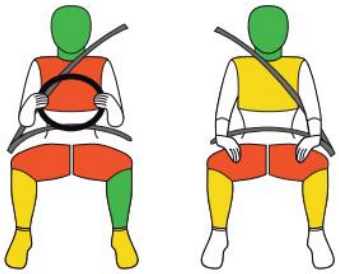
12.03 max. 17.00 Adult

38.27 max. 49.00 Child

Tested at 64 km/h

Bodyshell integrity: UNSTABLE

ADULT OCCUPANT PROTECTION



DRIVER FRONT PASSENGER

- GOOD
- ADEQUATE
- MARGINAL
- WEAK
- POOR

CAR DETAILS

TESTED MODEL	Honda City 4th Gen, RHD
BODY TYPE	4 DOOR SEDAN
CRASH TEST WEIGHT	KG 1260
YEAR OF PUBLICATION	2022

---

*Crash Image

ADAC
OD5016CEN1
GLOBAL NCAP
OD5016CEN1

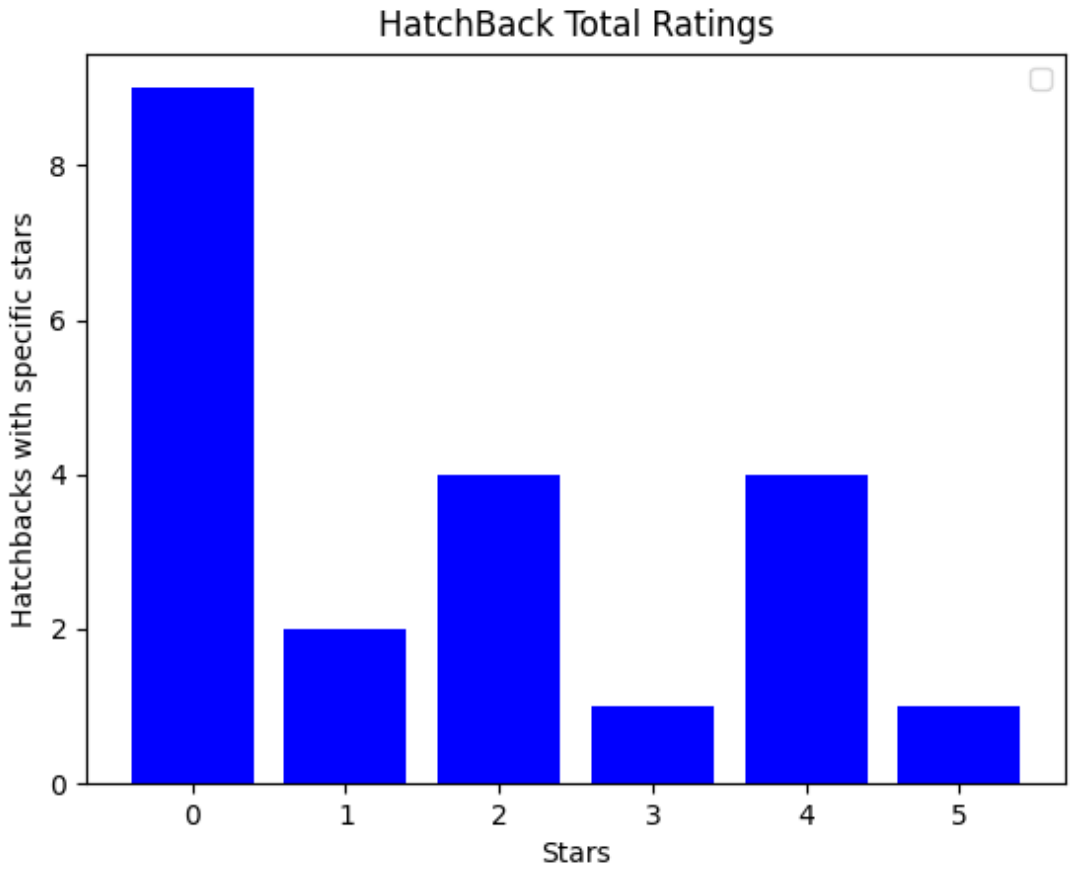
ADAC
FO3813FFI2
NCAP
GLOBAL NCAP
www.globalncap.org
FO3813FFI2
NCAP
FO3813FFI2

---

*Total Performance of Hatchback in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\Hatchback_ASTars.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='b')
plt.title("HatchBack Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Hatchbacks with specific stars")

# Show the plot
plt.show()
```

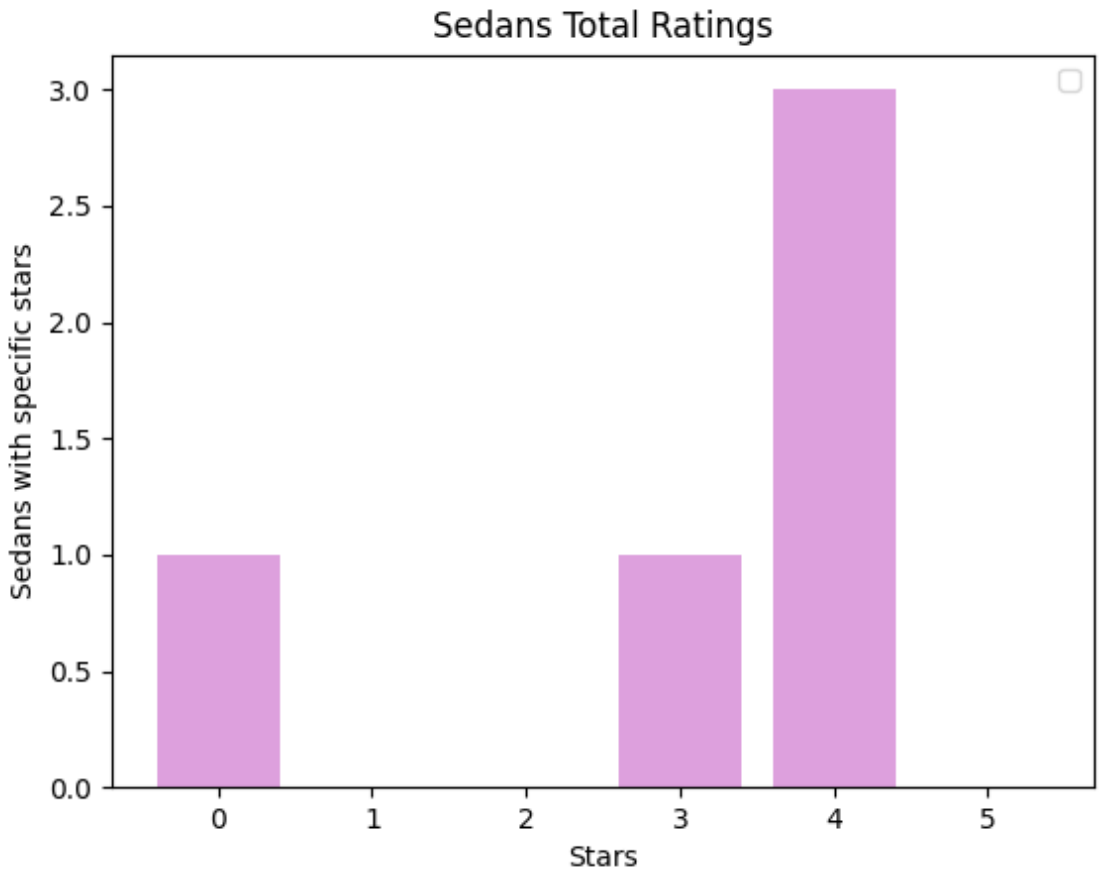


---

*Total Performance of Sedan in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\Sedan_AStars.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='plum')
plt.title("Sedans Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Sedans with specific stars")

# Show the plot
plt.show()
```

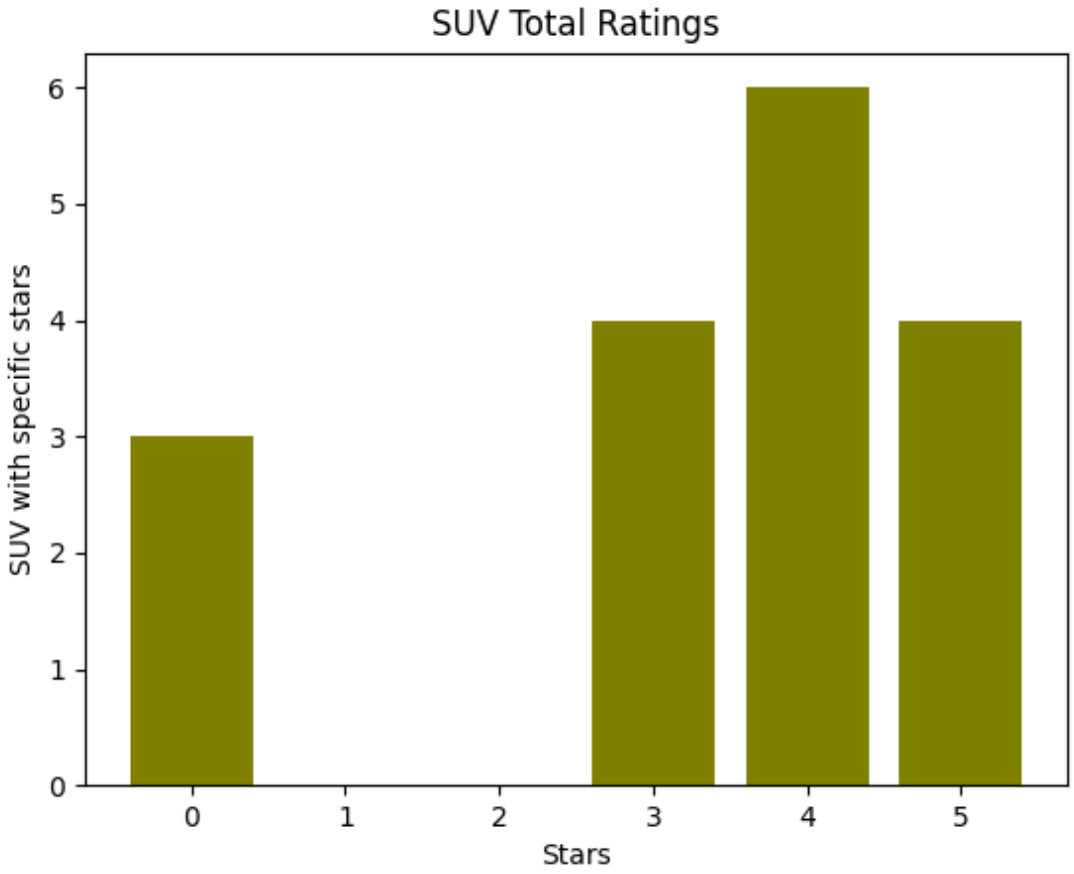


---

*Total Performance of SUV in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\SUV_AStars.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='olive')
plt.title("SUV Total Ratings")
plt.xlabel("Stars")
plt.ylabel("SUV with specific stars")

# Show the plot
plt.show()
```

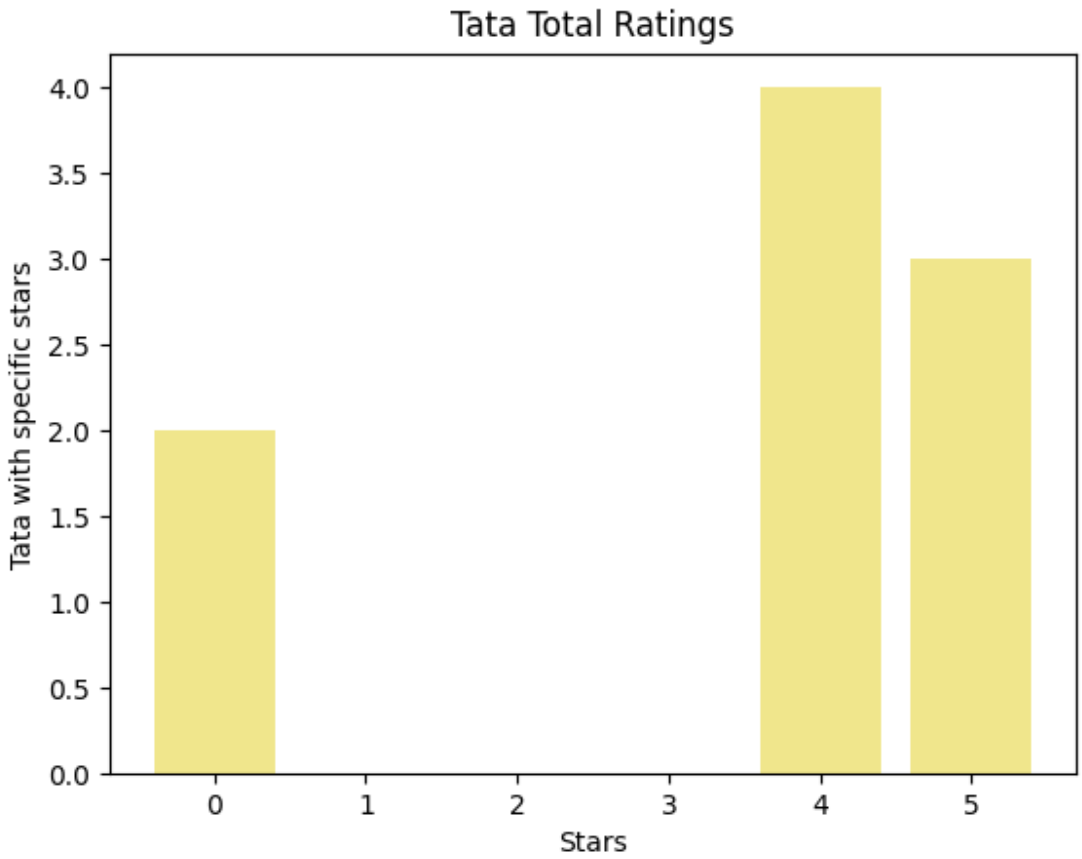


---

*Total Performance of Tata in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\Tata_AStar.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='khaki')
plt.title("Tata Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Tata with specific stars")

# Show the plot
plt.show()
```

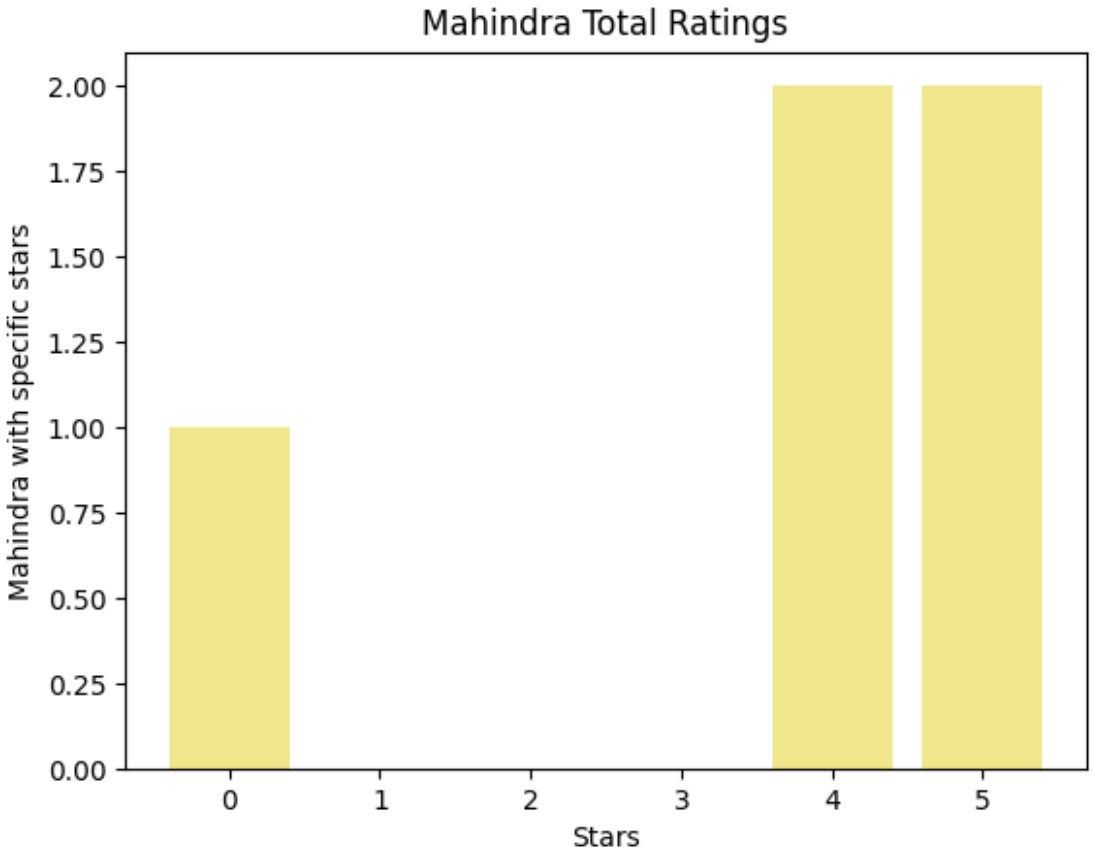


---

*Total Performance of Mahindra in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\Sedan_AStars.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='indianred')
plt.title("Sedans Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Sedans with specific stars")

# Show the plot
plt.show()
```

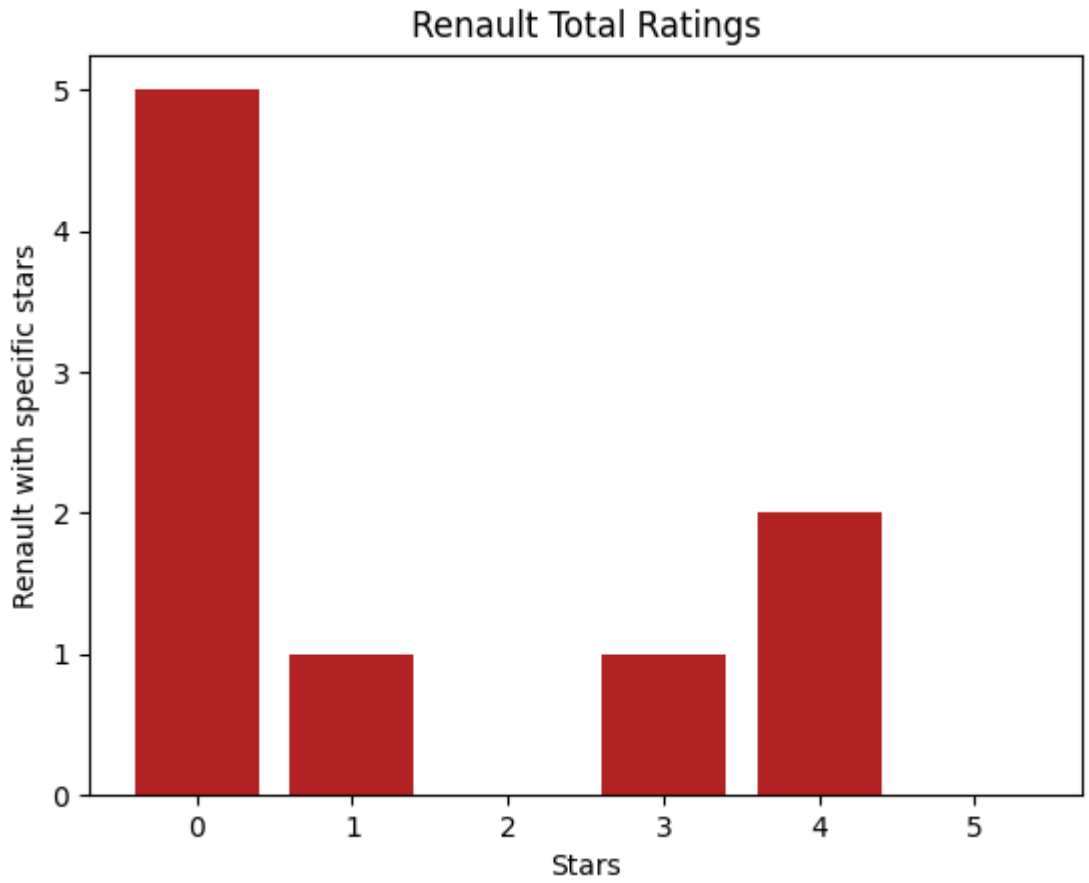


---

*Total Performance of Renault in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\Renault_AStar.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='firebrick')
plt.title("Renault Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Renault with specific stars")

# Show the plot
plt.show()
```

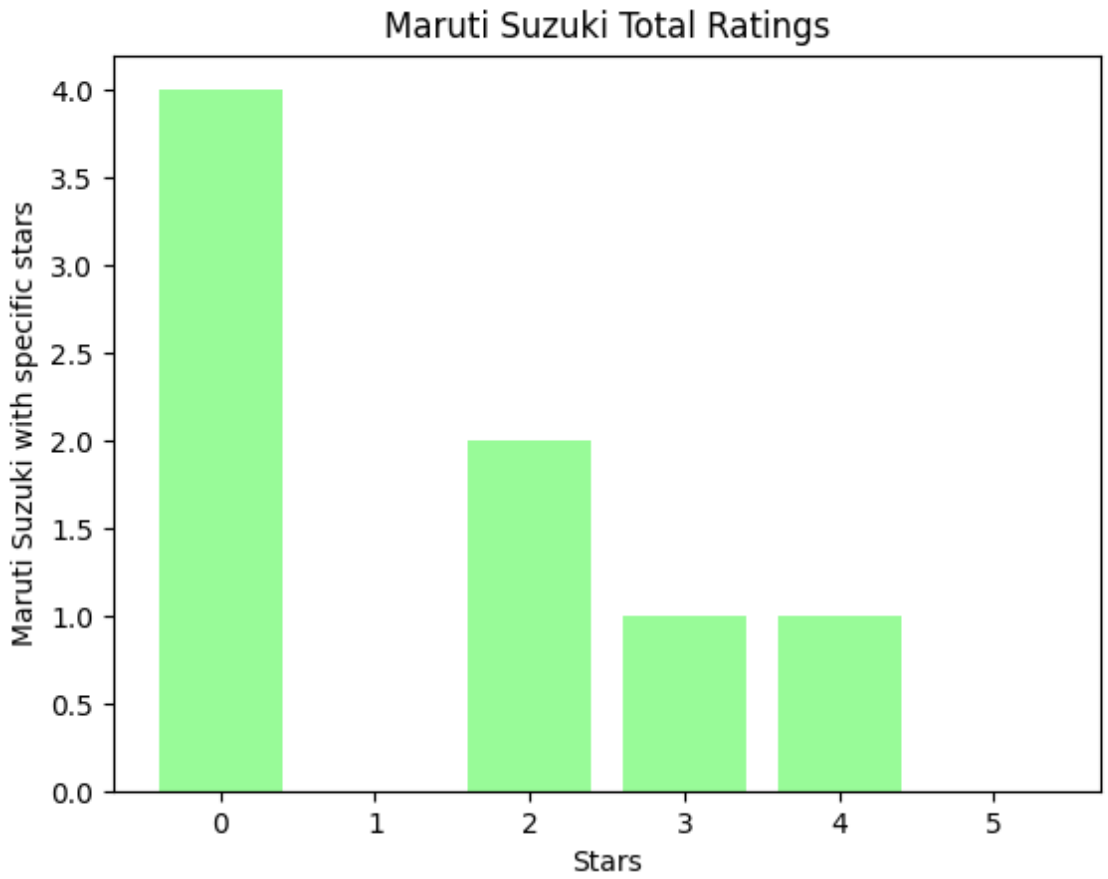


---

*Total Performance of Maruti Suzuki in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\MarutiS_AStar.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='palegreen')
plt.title("Maruti Suzuki Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Maruti Suzuki with specific stars")

# Show the plot
plt.show()
```

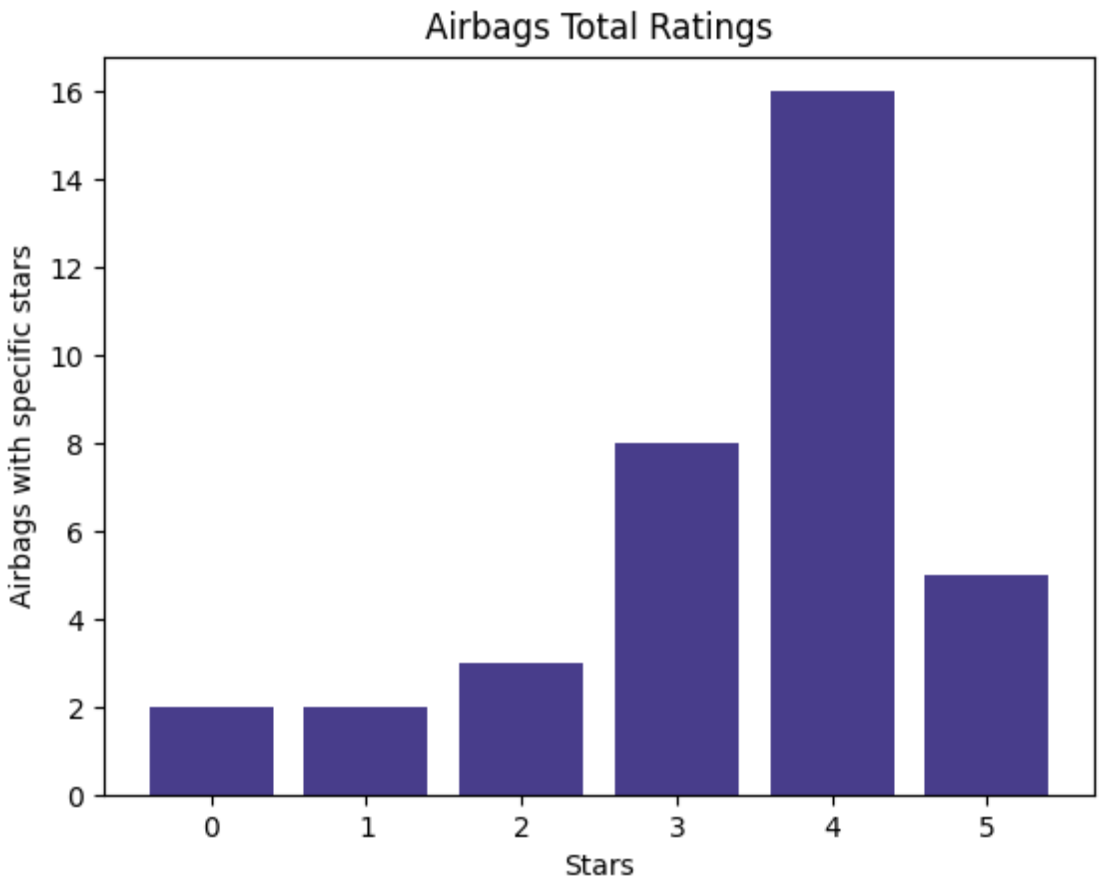


---

*Total Performance of Airbag in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\Airbarg_Astars.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='darkslateblue')
plt.title("Airbags Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Airbags with specific stars")

# Show the plot
plt.show()
```

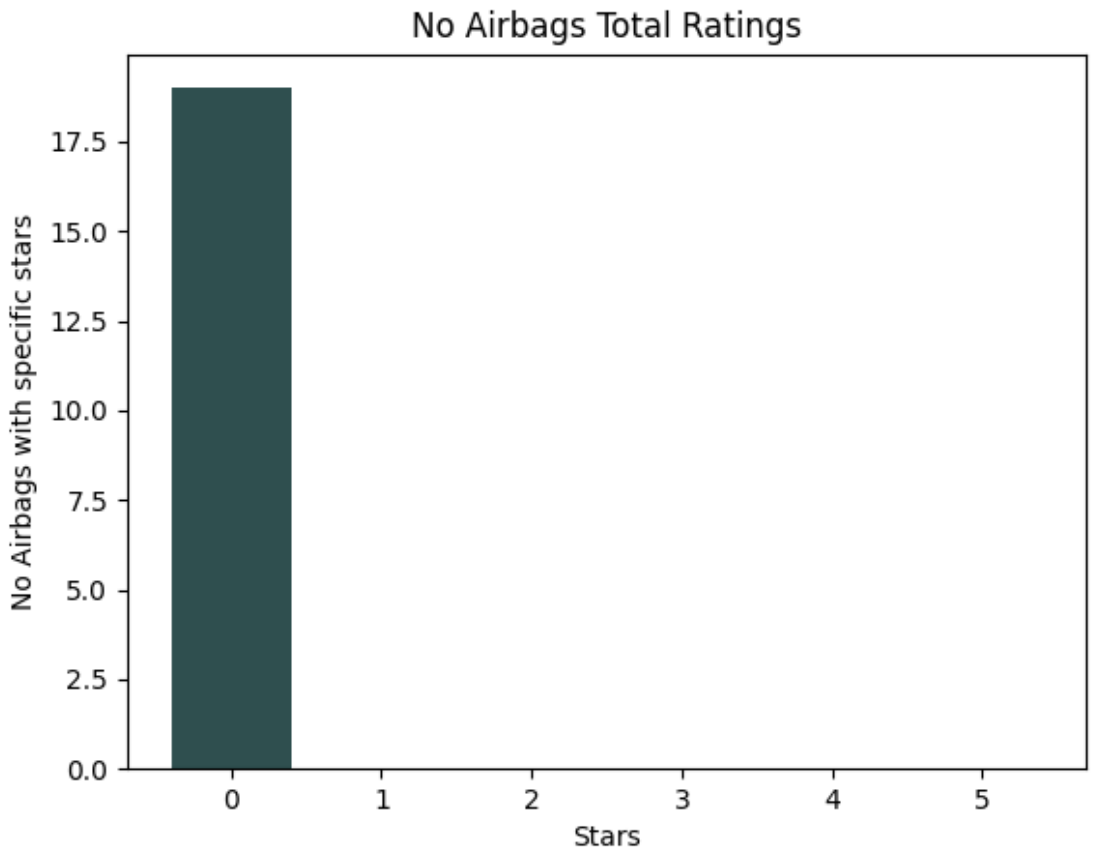


---

*Total Performance of No Airbag in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\NOairbag_Astar.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='darkslategray')
plt.title("No Airbags Total Ratings")
plt.xlabel("Stars")
plt.ylabel("No Airbags with specific stars")

# Show the plot
plt.show()
```



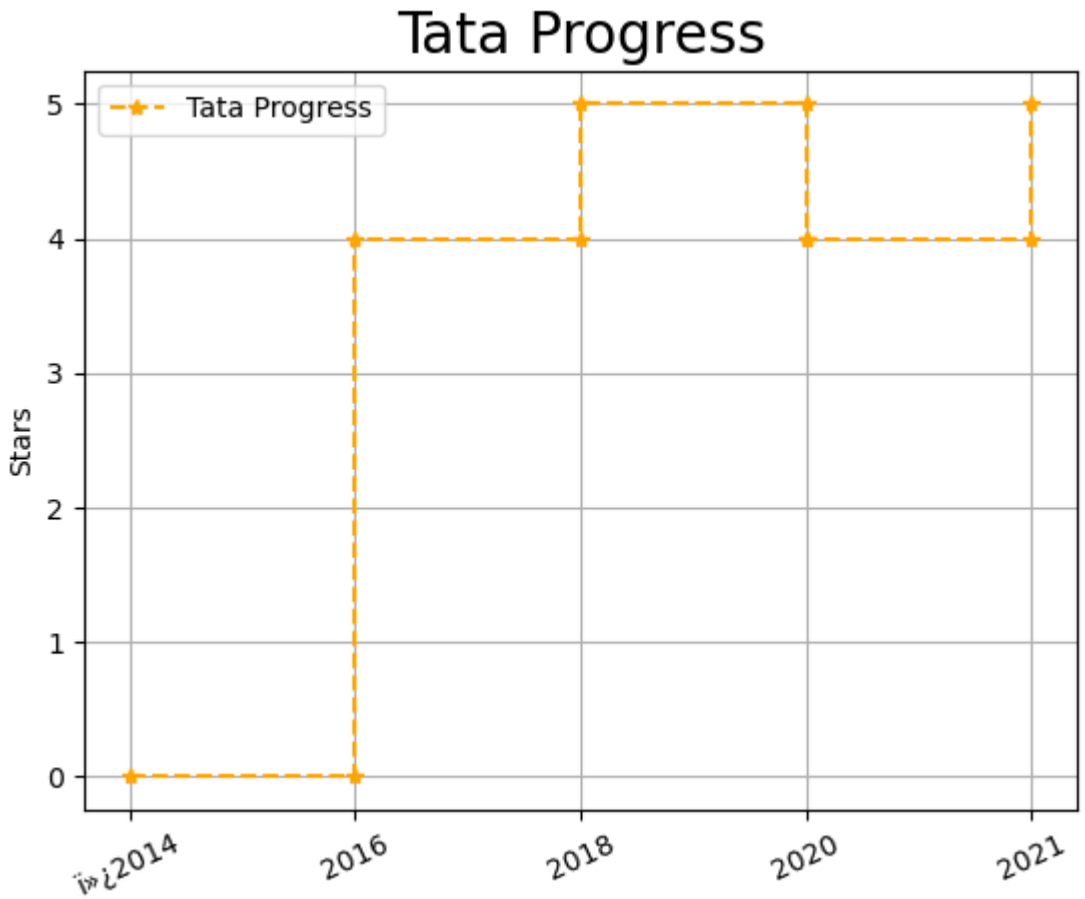
---

* Tata Yearly Progress in terms of rating

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug  5 20:39:49 2022

@author: LAXMINARAYANRO
"""
```

```
import matplotlib.pyplot as plt
import csv

x = []
y = []

with open('E:/fffiiles/college pracs and
projects/Project/Car/CSV/Tata_YearAstar.csv','r') as
csvfile:
    lines = csv.reader(csvfile, delimiter=',')
    for row in lines:
        x.append(row[0])
        y.append(int(row[1]))

plt.plot(x, y, color = 'orange', linestyle = 'dashed',
         marker = '*',label = "Tata Progress")

plt.xticks(rotation = 25)
plt.xlabel('Year')
plt.ylabel('Stars')
plt.title('Tata Progress', fontsize = 20)
```

```
plt.grid()
plt.legend()
plt.show()
```

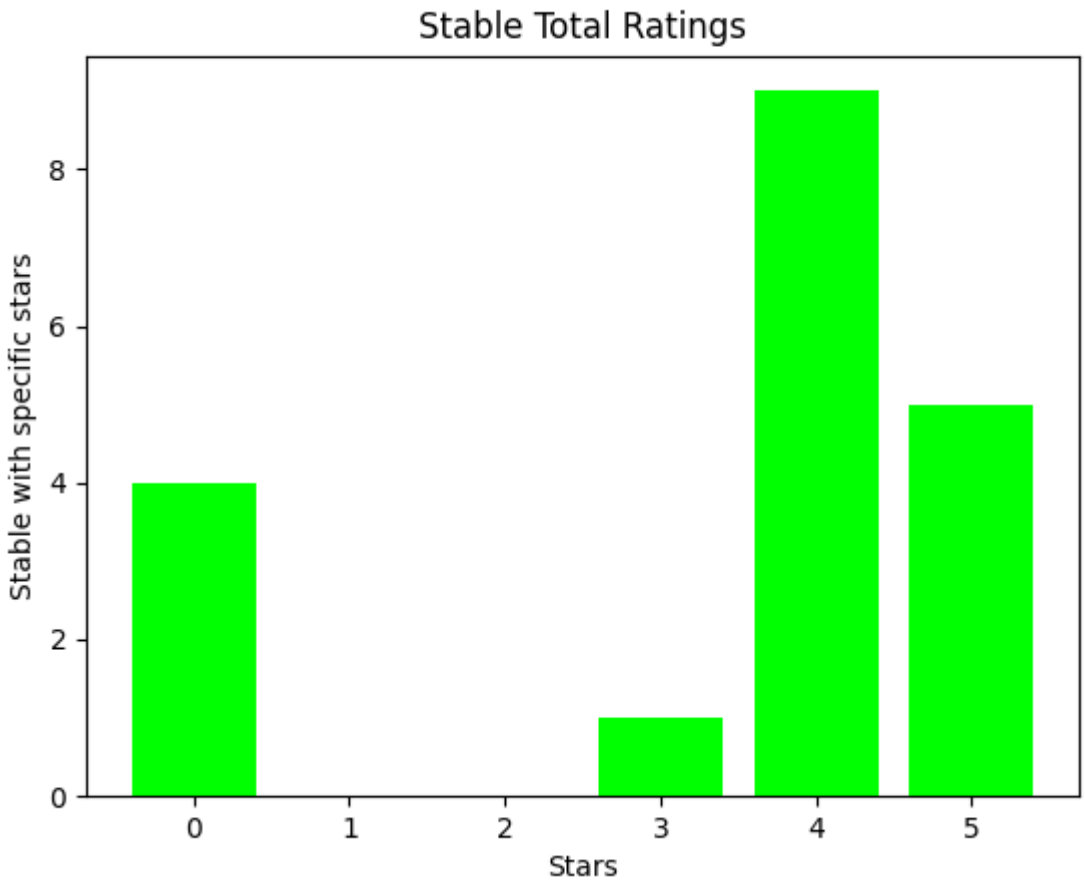


---

*Total Performance of Stable in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Initialize the lists for X and Y
data = pd.read_csv('E:\\fffiiles\\college pracs and projects\\Project\\Car\\CSV\\Stable_Astar.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='lime')
plt.title("Stable Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Stable with specific stars")

# Show the plot
plt.show()
```

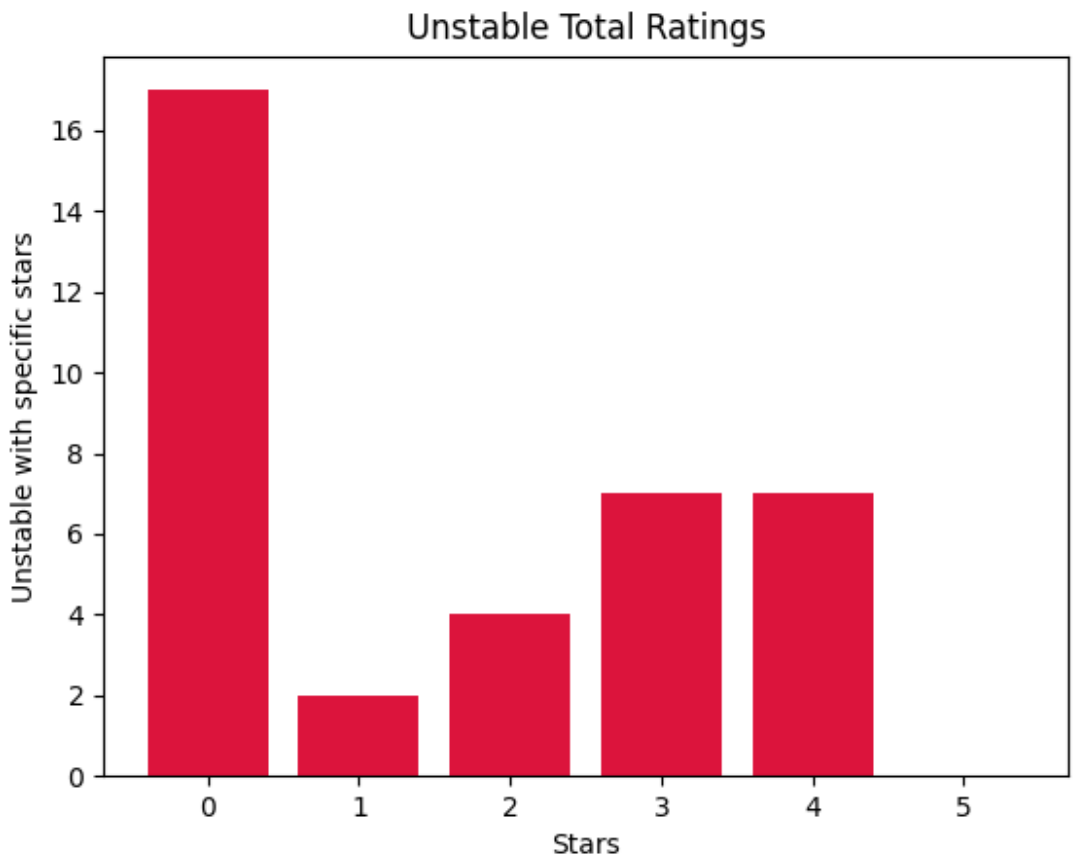


---

*Total Performance of Unstable in terms of rating

```
# Import the necessary modules
import matplotlib.pyplot as plt
import pandas as pd



# Initialize the lists for X and Y
```

```
data = pd.read_csv('E:\\fffiiles\\college pracs and
projects\\Project\\Car\\CSV\\Unstable_Astar.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])

# Plot the data using bar() method
plt.bar(X, Y, color='crimson')
plt.title("Unstable Total Ratings")
plt.xlabel("Stars")
plt.ylabel("Unstable with specific stars")

# Show the plot
plt.show()
```



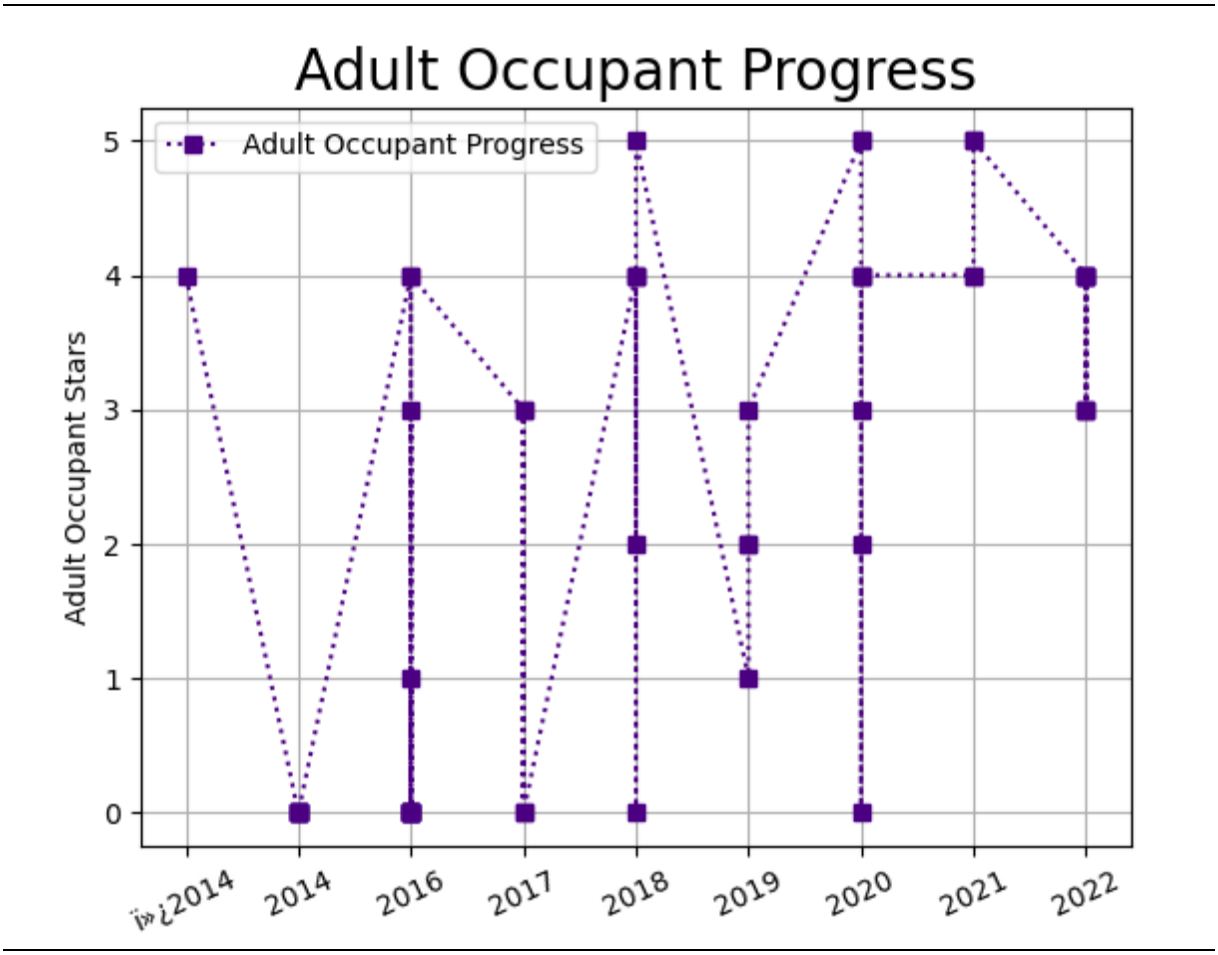
---

*Yearly Progress of Adult Occupant Stars of in terms of rating

---

```
import matplotlib.pyplot as plt
import csv

x = []
y = []
```

```
with open('E:/fffiiles/college pracs and
projects/Project/Car/CSV/Year_Astar.csv','r') as csvfile:
    lines = csv.reader(csvfile, delimiter=',')
    for row in lines:
        x.append(row[0])
        y.append(int(row[1]))

plt.plot(x, y, color = 'indigo', linestyle = 'dotted',
         marker = 's',label = "Adult Occupant Progress")

plt.xticks(rotation = 25)
plt.xlabel('Year')
plt.ylabel('Adult Occupant Stars')
plt.title('Adult Occupant Progress', fontsize = 20)
plt.grid()
plt.legend()
plt.show()
```

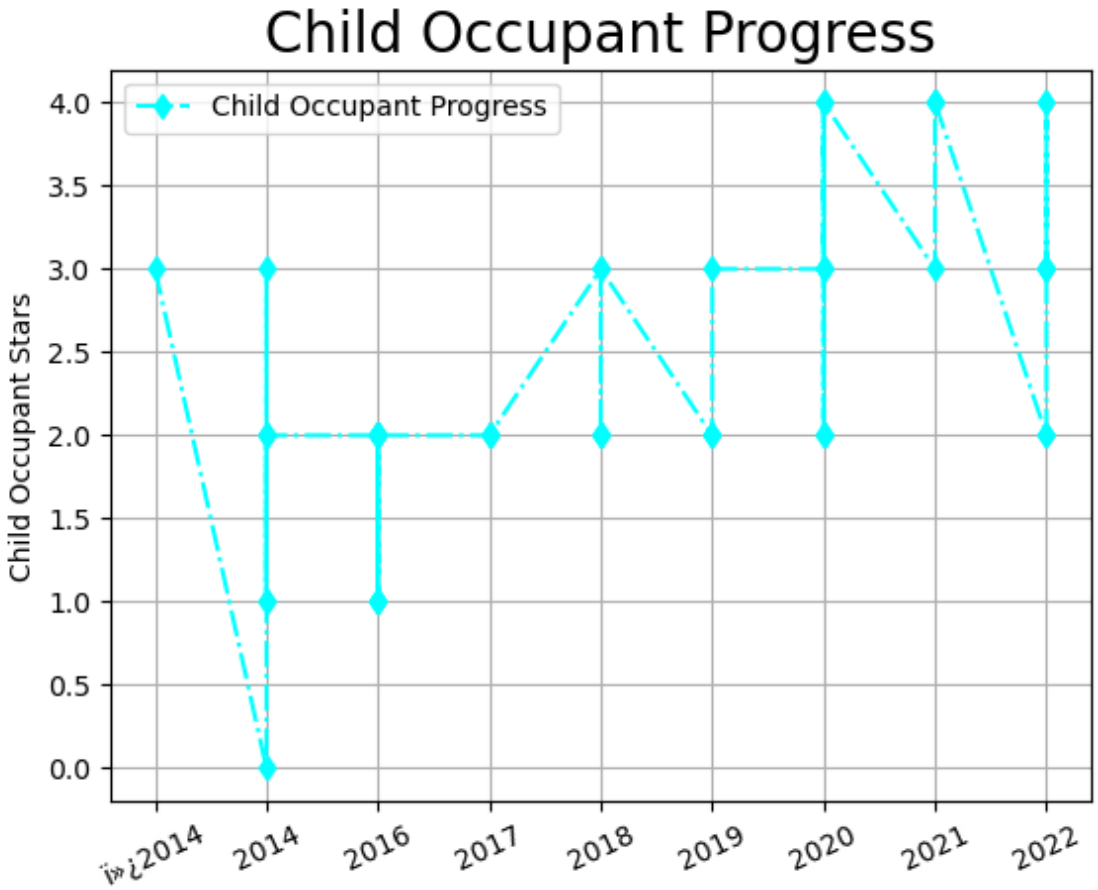


---

* Yearly Progress of Child Occupant Stars in terms of rating

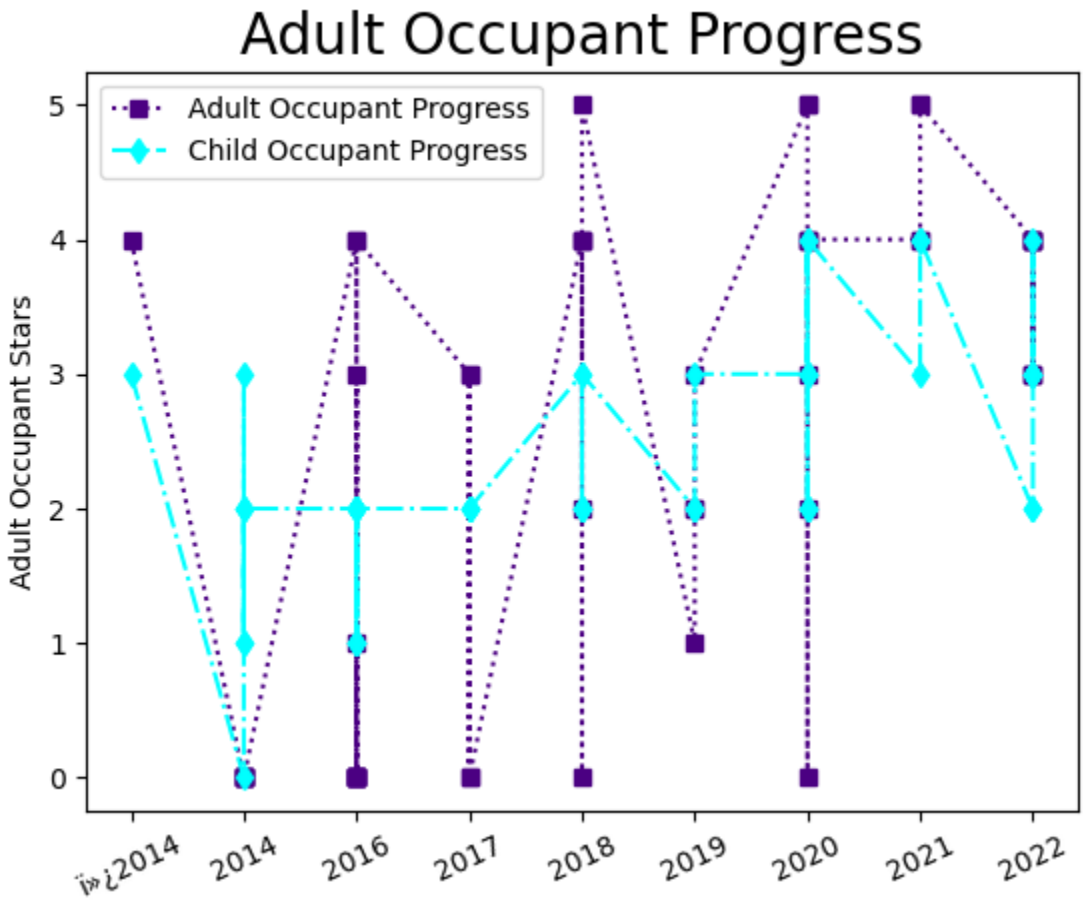
```
import matplotlib.pyplot as plt

import csv


x = []

y = []
```

```
with open('E:/fffiiles/college pracs and
projects/Project/Car/CSV/Year_Cstar.csv','r') as csvfile:
    lines = csv.reader(csvfile, delimiter=',')
    for row in lines:
        x.append(row[0])
        y.append(int(row[1]))

plt.plot(x, y, color = 'cyan', linestyle = 'dashdot',
         marker = 'd',label = "Child Occupant Progress")

plt.xticks(rotation = 25)
plt.xlabel('Year')
plt.ylabel('Adult Occupant Stars')
plt.title('Adult Occupant Progress', fontsize = 20)
plt.grid()
plt.legend()
plt.show()
```



---

*Both Combined



---

*From data it is very certain that SUV body type with high weight, airbags and stable bodyshell are safer than other

* Tata and Mahindra Company seem to provide more safety than others

---

*Cars with same platform

In the given dataset Maruti Suzuki Vitara Brezza and Toyota Urban Cruiser were crash tested

---

BREZZA

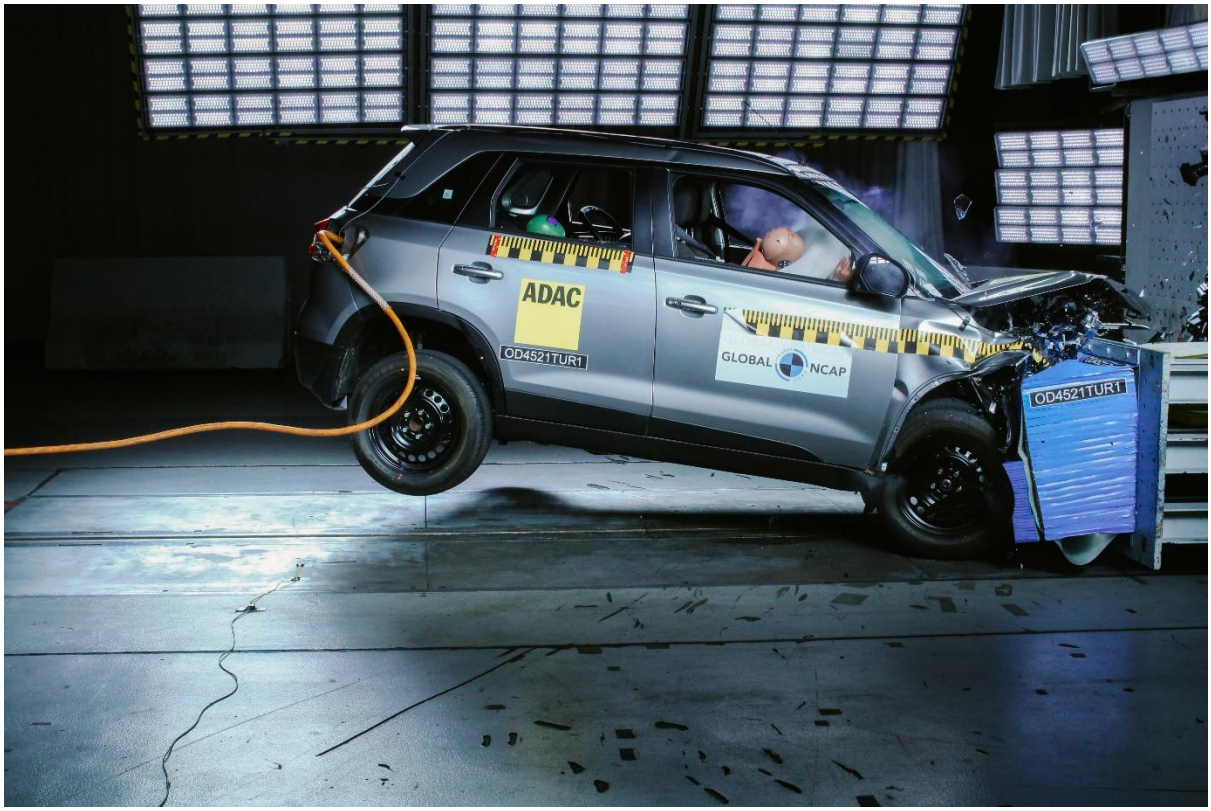
URBAN CRUISER

Both cars have use same platform

A car platform is a shared set of common design, engineering, and production efforts, as well as major components, over a number of outwardly distinct models and even types of cars, often from different, but somewhat related, marques. It is practiced in the automotive industry to reduce the costs associated with the development of products by basing those products on a smaller number of platforms. One more example is Tata Harrier and Land Rover Discovery Sport

---

But Vitara Brezza and Urban Cruiser Scored Different Results


ADAC
OD4521TUR1
GLOBAL NCAP
OD4521TUR1

ADAC
OD3718MVI1
GLOBAL NCAP
OD3718MVI1

	Vitara Brezza	Urban Cruiser
Adult Occupant	4	4
Child Occupant	2	3
Adult Score	12.51	13.52
Child Score	17.93	36.68
Crash Test Weight	1392	1324
Airbags	2	2
Body Shell Integrity	Stable	Stable

Both Cars performed differently even tough literally being the same cars.

The new urban cruiser is not face lifted like Vitara breeza and the Vitara brezza is a diesel car whereas the urban cruiser is a petrol car thus the difference in weight as diesel engines are heavier than petrol engines. The Maruti Suzuki brand has a standard of providing good fuel efficiency by sacrificing some points in safety rating as they facelift the cars which reduce some strength in order to gain boost in fuel efficiency.

---

*Linear Regression

```
#library
library(ggplot2)
library(rgl)
library(readxl)
library(caTools)


#Dataset and Descirption of Dataset
st <- read.csv("E://fffiiles//college pracs and projects//Project//Car//CSV//Only_Global_NCAP.csv")
head(st)
```

```
  ï..Brand         Name Adult_Occupant_.Stars. Child_Ocuupant_.Stars. Body_Type No_of_Doors Airbags No_of_Airbags Crash_Test_Weight..kg. Year
1 Volkswagen         Polo                   4                   3 Hatchback         4     Yes           2                1272 2014
2      Tata         Nano                   0                   0  City Car         4      No           0                 887 2014
3      Alto Maruti Suzuki                   0                   2  City Car         4      No           0                 924 2014
```

4 Volkswagen Polo 0 3 Hatchback 4 No 0 1259 2014

5 Hyundai I10 0 1 City Car 4 No 0 1116 2014

6 Ford Figo 0 2 Hatchback 4 No 0 1274 2014

Testing_Speed..km.hr. Adult_Points.17.00. Child_Points.49. Bodyshell.Integrity

1 64 12.54 29.91 Stable

2 64 0.00 0.00 Unstable

3 64 0.00 17.57 Unstable

4 64 0.00 26.97 Stable

5 64 0.00 6.97 Stable

6 64 0.00 20.75 Stable

summary(st)

ï..Brand Name Adult_Occupant_.Stars. Child_Ocuupant_.Stars. Body_Type No_of_Doors Airbags No_of_Airbags

Tata :10 Duster : 2 Min. :0.000 Min. :0.000 Hatchback:20 Min. :2.000 No :18 Min. :0.000

Renault : 9 Kwid (III): 2 1st Qu.:0.000 1st Qu.:2.000 SUV :14 1st Qu.:4.000 Yes:37 1st Qu.:0.000

Maruti Suzuki: 7 Mobilo : 2 Median :3.000 Median :2.000 Sedan : 6 Median :5.000 Median :2.000

Hyundai : 6 Nexon : 2 Mean :2.236 Mean :2.327 MPV : 4 Mean :4.618 Mean :1.291

Mahindra : 5 Polo : 2 3rd Qu.:4.000 3rd Qu.:3.000 City Car : 3 3rd Qu.:5.000 3rd Qu.:2.000

Honda : 4 Swift : 2 Max. :5.000 Max. :4.000 Hatch SUV: 2 Max. :5.000 Max. :6.000

(Other) :14 (Other) :43 (Other) : 6

Crash_Test_Weight..kg. Year Testing_Speed..km.hr. Adult_Points.17.00. Child_Points.49. Bodyshell.Integrity

Min. : 855 Min. :2014 Min. :64 Min. : 0.00 Min. : 0.00 Stable :19

1st Qu.:1108 1st Qu.:2016 1st Qu.:64 1st Qu.: 0.00 1st Qu.:15.03 Unstable:36

Median :1259 Median :2018 Median :64 Median : 8.36 Median :20.02

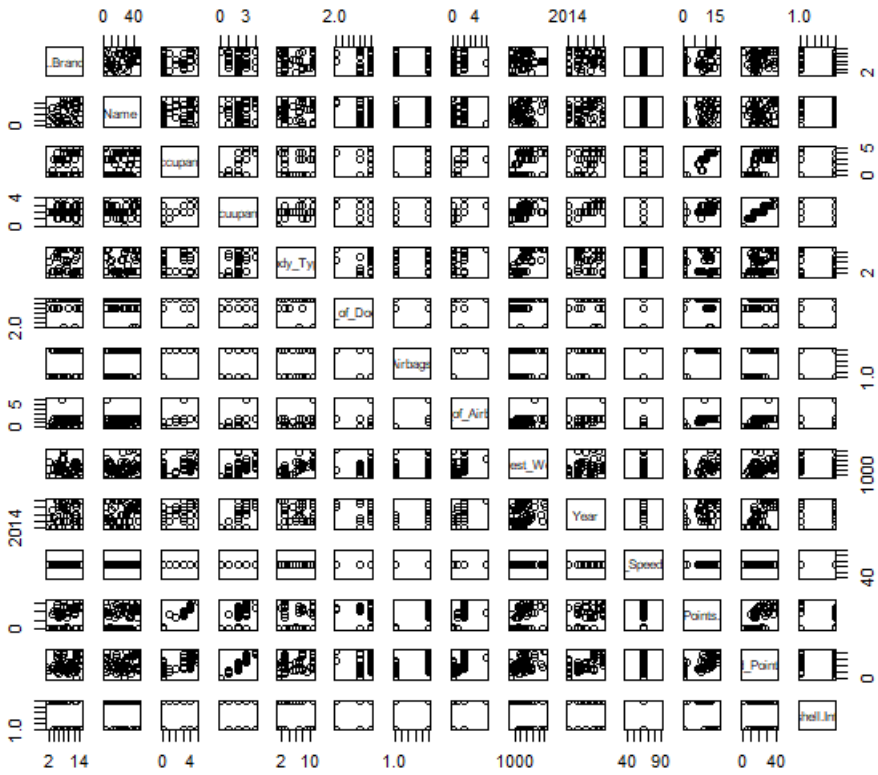
Mean :1282 Mean :2018 Mean :64 Mean : 7.27 Mean :22.04

3rd Qu.:1416 3rd Qu.:2020 3rd Qu.:64 3rd Qu.:12.52 3rd Qu.:28.64

Max. :2014 Max. :2022 Max. :64 Max. :16.45 Max. :41.66

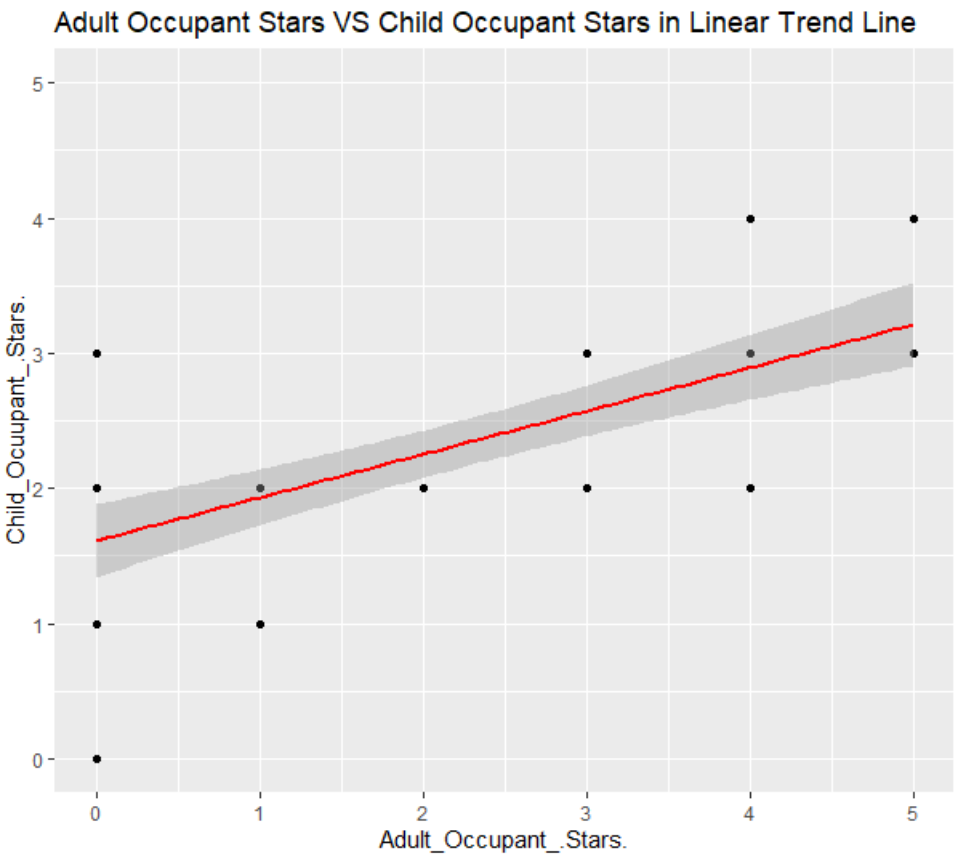
#pairplot

plot(st)

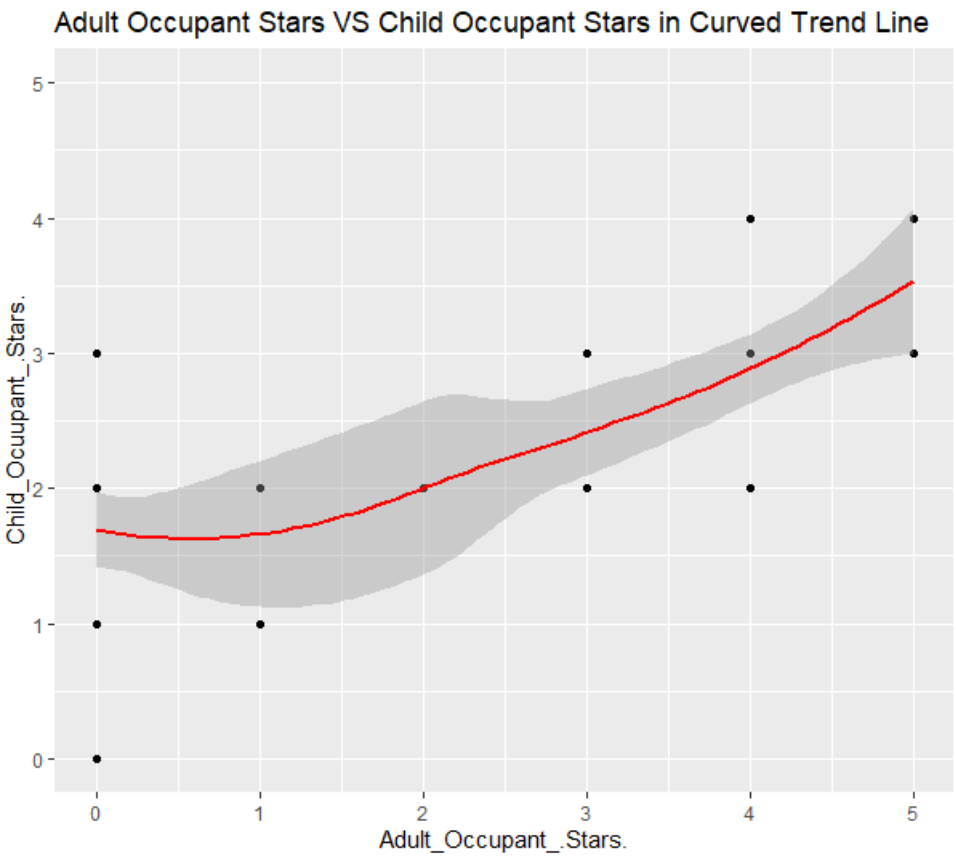


#relations of variables

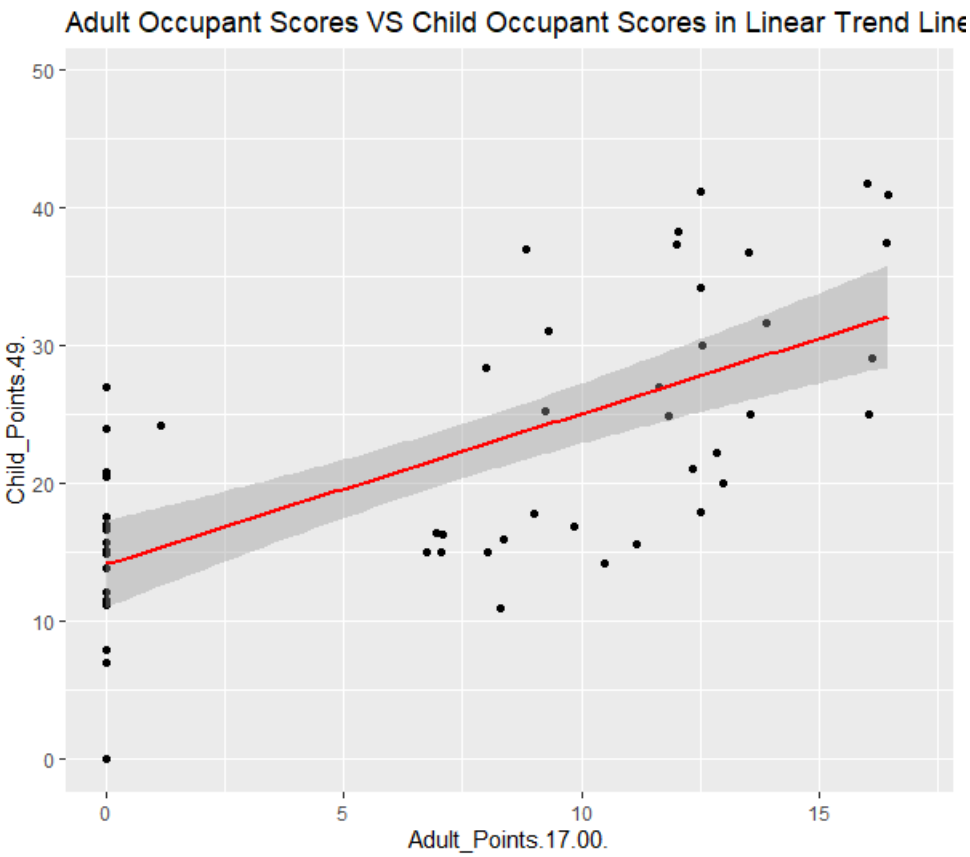
ggplot(st, aes(x=Adult_Occupant_.Stars., y=Child_Ocuupant_.Stars.)) + geom_point() + geom_smooth(color="red",method = lm) +xlim(0,5)+ylim(0,5)+ ggtitle("Adult Occupant Stars VS Child Occupant Stars in Linear Trend Line")



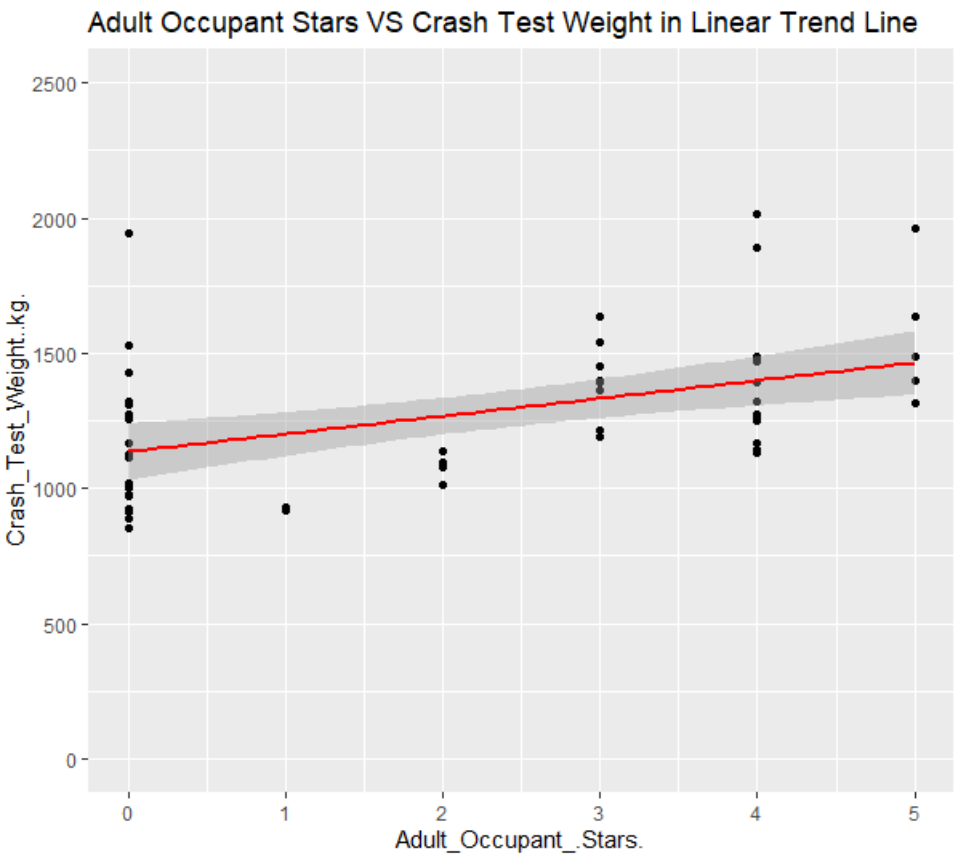

```
ggplot(st, aes(x=Adult_Occupant_.Stars.,
y=Child_Ocuupant_.Stars.)) + geom_point() +
geom_smooth(color="red") +xlim(0,5)+ylim(0,5)+
ggtitle("Adult Occupant Stars VS Child Occupant Stars in
Curved Trend Line")
```



ggplot(st, aes(x=Adult_Points.17.00., y=Child_Points.49.)) + geom_point() + geom_smooth(color="red",method = lm) +xlim(0,17)+ylim(0,49)+ ggtitle("Adult Occupant Scores VS Child Occupant Scores in Linear Trend Line")




```
ggplot(st, aes(x=Adult_Occupant_.Stars.,
y=Crash_Test_Weight..kg.)) + geom_point() +
geom_smooth(color="red",method = lm)
+xlim(0,5)+ylim(0,2500)+ ggtitle("Adult Occupant Stars VS
Crash Test Weight in Linear Trend Line")
```




```
#set seeds
set.seed(2)
function (x, f, drop = FALSE, ...)
UseMethod("split")
<bytecode: 0x00000204f71daee0>
<environment: namespace:base>
#spliting Data
split = sample.split(dataset$Salary, SplitRatio = 0.7)
split
train = subset(st, split = TRUE)
```

```
test = subset(st, split = FALSE)

#training Data

train

      ï..Brand          Name Adult_Occupant_.Stars.
Child_Ocuupant_.Stars.        Body_Type No_of_Doors
Airbags No_of_Airbags

1     Volkswagen         Polo                4                3
Hatchback         4     Yes          2

2          Tata         Nano                0                0        City
Car         4     No          0

3          Alto  Maruti Suzuki                0                2
City Car         4     No          0

4     Volkswagen         Polo                0                3
Hatchback         4     No          0

5        Hyundai         I10                 0                1        City
Car         4     No          0

6          Ford         Figo                0                2
Hatchback         4     No          0

7  Maruti Suzuki         Swift               0                1
Hatchback         4     No          0

8        Datsun          Go                 0                2
Hatchback         4     No          0

9        Toyota         Etios               4                2
Hatchback         4     Yes          2
```

10 Renault Kwid (III) 0 2
Hatchback 5 Yes 1

11 Renault Kwid (III) 0 2
Hatchback 5 No 0

12 Renault Kwid (I) 0 2
Hatchback 5 No 0

13 Mahindra Scorpio 0 2
SUV 5 No 0

14 Maruti Suzuki Celerio 0 1
Hatchback 5 No 0

15 Maruti Suzuki Eeco 0 2
Minivan 5 No 0

16 Hyundai Eon 0 2
Hatchback/City Car 4 No 0

17 Honda Mobilo 3 2
Station Wagon 5 Yes 2

18 Honda Mobilo 0 1
Station Wagen 5 No 0

19 Renault Kwid (IV) 1 1
Hatchback 5 Yes 1

20 Tata Zest 0 1
Sedan 4 No 0

21 Tata Zest 4 2
Sedan 4 Yes 2

22 Ford Aspire 3 2
Sedan 4 Yes 2

23 Chevrolet Enjoy 0 2
Minivan 5 No 0

24 Renault Duster 3 2
SUV 5 Yes 1

25 Renault Duster 0 2
SUV 5 No 0

26 Tata Nexon 4 3
SUV 5 Yes 2

27 Renault Lodgy 0 2
MPV 5 No 0

28 Maruti Suzuki Vitara Brezza 4 2
SUV 5 Yes 2

29 Maruti Suzuki Swift 2 2
Hatchback 5 Yes 2

30 Mahindra Marazzo 4 2
MPV 5 Yes 2

31 Tata Nexon 5 3
SUV 5 Yes 2

32 Datsun Redigo 1 2
Hatchback 5 Yes 1

33 Hyundai Santro 2 2
Hatchback 5 Yes 1

34 Maruti Suzuki WagonR 2 2
Hatchback 5 Yes 1

35 Maruti Suzuki Ertiga 3 3
MPV 5 Yes 2

36 Tata Altroz 5 3
Hatchback 5 Yes 2

37 Mahindra XUV 300 5 4
SUV 5 Yes 2

38 Tata Tigor 4 3
Sedan 5 Yes 2

39 Tata Tiago 4 3
Hatchback 5 Yes 2

40 Kia Seltos 3 2
SUV 5 Yes 2

41 Hyundai Grand I10 Nios 2 2
Hatchback 5 Yes 2

42 Maruti Suzuki S-Presso 0 2
Mini SUV 5 Yes 1

43 Mahindra Thar 4 4
SUV 2 Yes 2

44 Renault Triber 4 3
MPV 2 Yes 2

45 Tata Tigor EV 4 4
Sedan 4 Yes 2

46 Tata Punch 5 4
SUV 5 Yes 2

47 Mahindra XUV 700 5 4
SUV 5 Yes 2

48 Renault Kiger 4 2
Hatch SUV 5 Yes 2

49 Nissan Magnite 4 2
Hatch SUV 5 Yes 2

50 Honda Jazz 4 3
Hatchback 5 Yes 2

51 Honda City (4th Gen) 4 4
Sedan 4 Yes 2

52 Hyundai Creta 3 3
SUV 5 Yes 2

53 Hyundai I20 3 3
Hatchback 5 Yes 2

54 Toyota Urban Cruiser 4 3
SUV 5 Yes 2

55 Kia Carrens 3 3
SUV 5 Yes 6

Crash_Test_Weight..kg. Year Testing_Speed..km.hr.
Adult_Points.17.00. Child_Points.49. Bodyshell.Integrity

1 1272 2014 64 12.54
29.91 Stable

2	887	2014	64	0.00	0.00	Unstable
3	924	2014	64	0.00	17.57	Unstable
4	1259	2014	64	0.00	26.97	Stable
5	1116	2014	64	0.00	6.97	Stable
6	1274	2014	64	0.00	20.75	Stable
7	1169	2014	64	0.00	7.94	Unstable
8	1001	2014	64	0.00	15.06	Unstable
9	1131	2016	64	13.00	20.02	Stable
10	914	2016	64	0.00	16.63	Unstable
11	914	2016	64	0.00	14.85	Unstable
12	855	2016	64	0.00	23.89	Unstable
13	1943	2016	64	0.00	16.73	Unstable

14	1019	2016	64	0.00	11.53	Unstable
15	1124	2016	64	1.13	24.20	Unstable
16	972	2016	64	0.00	20.42	Unstable
17	1451	2016	64	9.85	16.82	Stable
18	1324	2016	64	0.00	11.11	Stable
19	928	2016	64	8.28	10.91	Unstable
20	1310	2016	64	0.00	12.11	Unstable
21	1395	2016	64	11.15	15.52	Unstable
22	1212	2017	64	10.49	14.22	Unstable
23	1529	2017	64	0.00	13.86	Unstable
24	1539	2017	64	9.00	17.75	Unstable
25	1430	2017	64	0.00	16.98	Unstable

26	1490	2018	64	13.56	25.00
Stable					
27	1526	2018	64	0.00	15.67
Unstable					
28	1392	2018	64	12.51	17.93
Stable					
29	1079	2018	64	7.08	16.23
Unstable					
30	1889	2018	64	12.85	22.22
Stable					
31	1490	2018	64	16.06	25.00
Stable					
32	917	2019	64	8.36	15.93
Unstable					
33	1099	2019	64	6.74	15.00
Unstable					
34	1013	2019	64	6.93	16.33
Unstable					
35	1363	2019	64	9.25	25.16
Unstable					
36	1401	2020	64	16.13	29.00
Stable					
37	1637	2020	64	16.42	37.44
Stable					

38	1259	2020	64	12.52	34.15
Unstable					
39	1259	2020	64	12.52	34.15
Unstable					
40	1391	2020	64	8.03	15.00
Unstable					
41	1139	2020	64	7.05	15.00
Unstable					
42	980	2020	64	0.00	13.84
Unstable					
43	2014	2020	64	12.52	41.11
Stable					
44	1166	2021	64	11.62	27.00
Unstable					
45	1470	2021	64	12.00	37.24
Unstable					
46	1313	2021	64	16.45	40.89
Stable					
47	1960	2021	64	16.03	41.66
Stable					
48	1142	2022	64	12.34	21.05
Unstable					
49	1167	2022	64	11.85	24.88
Stable					

50 1252 2022 64 13.89
31.54 Stable

51 1260 2022 64 12.03
38.27 Unstable

52 1398 2022 64 8.00
28.29 Unstable

53 1193 2022 64 8.84
36.89 Unstable

54 1324 2022 64 13.52
36.68 Stable

55 1636 2022 64 9.30
30.99 Unstable

#Testing Data

test

ï..Brand Name Adult_Occupant_.Stars.
Child_Ocuupant_.Stars. Body_Type No_of_Doors
Airbags No_of_Airbags

1 Volkswagen Polo 4 3
Hatchback 4 Yes 2

2 Tata Nano 0 0 City
Car 4 No 0

3 Alto Maruti Suzuki 0 2
City Car 4 No 0

4 Volkswagen Polo 0 3
Hatchback 4 No 0

5 Hyundai I10 0 1 City Car 4 No 0

6 Ford Figo 0 2 Hatchback 4 No 0

7 Maruti Suzuki Swift 0 1 Hatchback 4 No 0

8 Datsun Go 0 2 Hatchback 4 No 0

9 Toyota Etios 4 2 Hatchback 4 Yes 2

10 Renault Kwid (III) 0 2 Hatchback 5 Yes 1

11 Renault Kwid (III) 0 2 Hatchback 5 No 0

12 Renault Kwid (I) 0 2 Hatchback 5 No 0

13 Mahindra Scorpio 0 2 SUV 5 No 0

14 Maruti Suzuki Celerio 0 1 Hatchback 5 No 0

15 Maruti Suzuki Eeco 0 2 Minivan 5 No 0

16 Hyundai Eon 0 2 Hatchback/City Car 4 No 0

17 Honda Mobilo 3 2
Station Wagon 5 Yes 2

18 Honda Mobilo 0 1
Station Wagen 5 No 0

19 Renault Kwid (IV) 1 1
Hatchback 5 Yes 1

20 Tata Zest 0 1
Sedan 4 No 0

21 Tata Zest 4 2
Sedan 4 Yes 2

22 Ford Aspire 3 2
Sedan 4 Yes 2

23 Chevrolet Enjoy 0 2
Minivan 5 No 0

24 Renault Duster 3 2
SUV 5 Yes 1

25 Renault Duster 0 2
SUV 5 No 0

26 Tata Nexon 4 3
SUV 5 Yes 2

27 Renault Lodgy 0 2
MPV 5 No 0

28 Maruti Suzuki Vitara Brezza 4 2
SUV 5 Yes 2

29 Maruti Suzuki Swift 2 2
Hatchback 5 Yes 2

30 Mahindra Marazzo 4 2
MPV 5 Yes 2

31 Tata Nexon 5 3
SUV 5 Yes 2

32 Datsun Redigo 1 2
Hatchback 5 Yes 1

33 Hyundai Santro 2 2
Hatchback 5 Yes 1

34 Maruti Suzuki WagonR 2 2
Hatchback 5 Yes 1

35 Maruti Suzuki Ertiga 3 3
MPV 5 Yes 2

36 Tata Altroz 5 3
Hatchback 5 Yes 2

37 Mahindra XUV 300 5 4
SUV 5 Yes 2

38 Tata Tigor 4 3
Sedan 5 Yes 2

39 Tata Tiago 4 3
Hatchback 5 Yes 2

40 Kia Seltos 3 2
SUV 5 Yes 2

41 Hyundai Grand I10 Nios 2 2
Hatchback 5 Yes 2

42 Maruti Suzuki S-Presso 0 2
Mini SUV 5 Yes 1

43 Mahindra Thar 4 4
SUV 2 Yes 2

44 Renault Triber 4 3
MPV 2 Yes 2

45 Tata Tigor EV 4 4
Sedan 4 Yes 2

46 Tata Punch 5 4
SUV 5 Yes 2

47 Mahindra XUV 700 5 4
SUV 5 Yes 2

48 Renault Kiger 4 2
Hatch SUV 5 Yes 2

49 Nissan Magnite 4 2
Hatch SUV 5 Yes 2

50 Honda Jazz 4 3
Hatchback 5 Yes 2

51 Honda City (4th Gen) 4 4
Sedan 4 Yes 2

52 Hyundai Creta 3 3
SUV 5 Yes 2

53 Hyundai I20 3 3
Hatchback 5 Yes 2

54 Toyota Urban Cruiser 4 3
SUV 5 Yes 2

55 Kia Carrens 3 3
SUV 5 Yes 6

Crash_Test_Weight..kg. Year Testing_Speed..km.hr.
Adult_Points.17.00. Child_Points.49. Bodyshell.Integrity

1 1272 2014 64 12.54
29.91 Stable

2 887 2014 64 0.00 0.00
Unstable

3 924 2014 64 0.00 17.57
Unstable

4 1259 2014 64 0.00
26.97 Stable

5 1116 2014 64 0.00 6.97
Stable

6 1274 2014 64 0.00
20.75 Stable

7 1169 2014 64 0.00 7.94
Unstable

8 1001 2014 64 0.00
15.06 Unstable

9	1131	2016	64	13.00	20.02
Stable					
10	914	2016	64	0.00	16.63
Unstable					
11	914	2016	64	0.00	14.85
Unstable					
12	855	2016	64	0.00	23.89
Unstable					
13	1943	2016	64	0.00	16.73
Unstable					
14	1019	2016	64	0.00	11.53
Unstable					
15	1124	2016	64	1.13	24.20
Unstable					
16	972	2016	64	0.00	20.42
Unstable					
17	1451	2016	64	9.85	16.82
Stable					
18	1324	2016	64	0.00	11.11
Stable					
19	928	2016	64	8.28	10.91
Unstable					
20	1310	2016	64	0.00	12.11
Unstable					

21	1395	2016	64	11.15	15.52
Unstable					
22	1212	2017	64	10.49	14.22
Unstable					
23	1529	2017	64	0.00	13.86
Unstable					
24	1539	2017	64	9.00	17.75
Unstable					
25	1430	2017	64	0.00	16.98
Unstable					
26	1490	2018	64	13.56	25.00
Stable					
27	1526	2018	64	0.00	15.67
Unstable					
28	1392	2018	64	12.51	17.93
Stable					
29	1079	2018	64	7.08	16.23
Unstable					
30	1889	2018	64	12.85	22.22
Stable					
31	1490	2018	64	16.06	25.00
Stable					
32	917	2019	64	8.36	15.93
Unstable					

33	1099	2019	64	6.74	15.00
Unstable					
34	1013	2019	64	6.93	16.33
Unstable					
35	1363	2019	64	9.25	25.16
Unstable					
36	1401	2020	64	16.13	29.00
Stable					
37	1637	2020	64	16.42	37.44
Stable					
38	1259	2020	64	12.52	34.15
Unstable					
39	1259	2020	64	12.52	34.15
Unstable					
40	1391	2020	64	8.03	15.00
Unstable					
41	1139	2020	64	7.05	15.00
Unstable					
42	980	2020	64	0.00	13.84
Unstable					
43	2014	2020	64	12.52	41.11
Stable					
44	1166	2021	64	11.62	27.00
Unstable					

45	1470	2021	64	12.00	37.24	Unstable
46	1313	2021	64	16.45	40.89	Stable
47	1960	2021	64	16.03	41.66	Stable
48	1142	2022	64	12.34	21.05	Unstable
49	1167	2022	64	11.85	24.88	Stable
50	1252	2022	64	13.89	31.54	Stable
51	1260	2022	64	12.03	38.27	Unstable
52	1398	2022	64	8.00	28.29	Unstable
53	1193	2022	64	8.84	36.89	Unstable
54	1324	2022	64	13.52	36.68	Stable
55	1636	2022	64	9.30	30.99	Unstable

```
#creating the model

Model <-
lm(Adult_Occupant_.Stars.~Child_Ocuupant_.Stars.+Adult_P
```

```
oints.17.00.+Child_Points.49.+Crash_Test_Weight..kg.,data=t
rain)

summary(Model)

Call:

lm(formula = Adult_Occupant_.Stars. ~
Child_Ocuupant_.Stars. +

   Adult_Points.17.00. + Child_Points.49. +
Crash_Test_Weight..kg.,

   data = train)


Residuals:

    Min      1Q   Median      3Q     Max

-1.39047 -0.11756  0.03444  0.13092  0.62105


Coefficients:

                      Estimate Std. Error t value Pr(>|t|)

(Intercept)          -0.5950983  0.2336693  -2.547   0.0140 *

Child_Ocuupant_.Stars.  0.0721907  0.1594797   0.453
0.6527

Adult_Points.17.00.     0.2960508  0.0108917  27.181   <2e-16
***

Child_Points.49.       0.0032572  0.0142243   0.229   0.8198
```

```
Crash_Test_Weight..kg.  0.0003428  0.0001974   1.736
0.0887 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3483 on 50 degrees of freedom
Multiple R-squared:  0.969,    Adjusted R-squared:  0.9666
F-statistic: 391.4 on 4 and 50 DF,  p-value: < 2.2e-16
#predicting Values
pred <- predict(Model,test)
pred
        1          2          3          4          5          6          7          8
9         10         11         12
 3.86736641 -0.29106834 -0.07677605  0.14085747 -
0.11768257  0.05354853 -0.09635671 -0.05855890
3.85081608 -0.08326542 -0.08906321 -0.07984122
        13         14         15         16         17         18         19
20         21         22         23         24
 0.26976248 -0.13607772  0.34790890 -0.05104049
3.01751696 -0.03290331  2.28201201 -0.03444479
3.37895397  3.11660065  0.11851085  2.79906603
        25         26         27         28         29         30         31
32         33         34         35         36
```

```
 0.09473983  4.22806756  0.12337807  3.78840457
2.06802704  4.07338792  4.96819454  2.39046744
1.97021867  2.00132284  2.90907903  4.97144101

       37        38        39        40        41        42        43
44        45        46        47        48

 5.23786891  3.87079995  3.87079995  2.45221072
2.07570490 -0.06973067  4.22444601  3.54918849
3.87143172  5.14693281  5.24686658  3.66254784

       49        50        51        52        53        54        55

 3.53852703  4.26548894  3.81168810  2.56120720
2.76763544  4.19737096  3.03644501
```

```
prediction=data.frame(Year=
test$Year,Adult_Occupant_.Stars. =
test$Adult_Occupant_.Stars., stock_Predict =  pred)

prediction

Year Adult_Occupant_.Stars. stock_Predict

1  2014                  4    3.86736641

2  2014                  0   -0.29106834

3  2014                  0   -0.07677605

4  2014                  0    0.14085747

5  2014                  0   -0.11768257

6  2014                  0    0.05354853
```

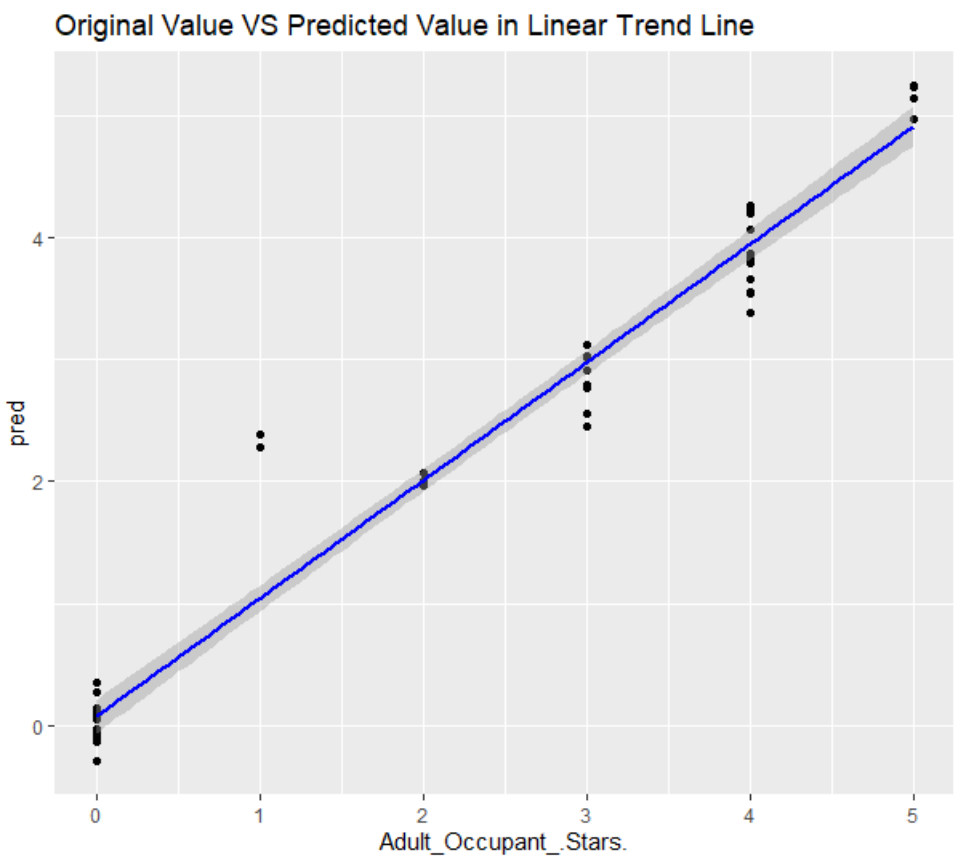
7	2014	0	-0.09635671
8	2014	0	-0.05855890
9	2016	4	3.85081608
10	2016	0	-0.08326542
11	2016	0	-0.08906321
12	2016	0	-0.07984122
13	2016	0	0.26976248
14	2016	0	-0.13607772
15	2016	0	0.34790890
16	2016	0	-0.05104049
17	2016	3	3.01751696
18	2016	0	-0.03290331
19	2016	1	2.28201201
20	2016	0	-0.03444479
21	2016	4	3.37895397
22	2017	3	3.11660065
23	2017	0	0.11851085
24	2017	3	2.79906603
25	2017	0	0.09473983
26	2018	4	4.22806756
27	2018	0	0.12337807
28	2018	4	3.78840457

29	2018	2	2.06802704
30	2018	4	4.07338792
31	2018	5	4.96819454
32	2019	1	2.39046744
33	2019	2	1.97021867
34	2019	2	2.00132284
35	2019	3	2.90907903
36	2020	5	4.97144101
37	2020	5	5.23786891
38	2020	4	3.87079995
39	2020	4	3.87079995
40	2020	3	2.45221072
41	2020	2	2.07570490
42	2020	0	-0.06973067
43	2020	4	4.22444601
44	2021	4	3.54918849
45	2021	4	3.87143172
46	2021	5	5.14693281
47	2021	5	5.24686658
48	2022	4	3.66254784
49	2022	4	3.53852703
50	2022	4	4.26548894

51	2022	4	3.81168810
52	2022	3	2.56120720
53	2022	3	2.76763544
54	2022	4	4.19737096
55	2022	3	3.03644501

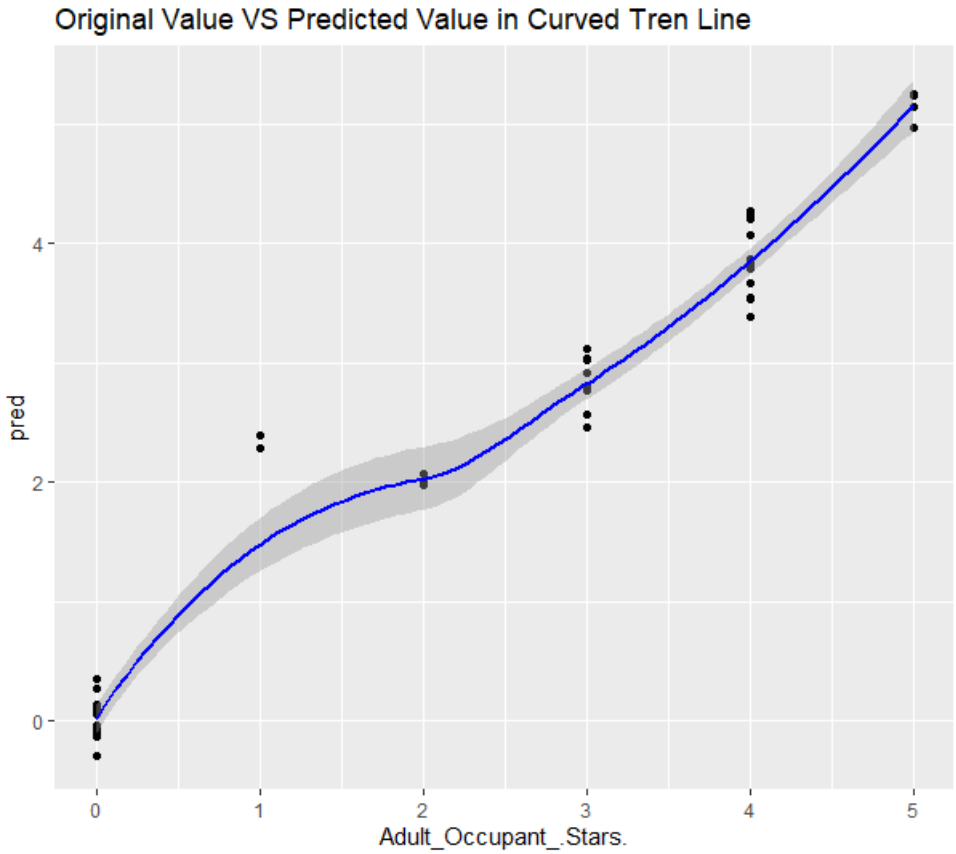
#creating Graph

```
ggplot(st, aes(x=Adult_Occupant_.Stars., y=pred)) +
geom_point() + geom_smooth(color="blue",method = lm) +
ggtitle("Original Value VS Predicted Value in Linear Trend
Line")
```




```
ggplot(st, aes(x=Adult_Occupant_.Stars., y=pred)) +
geom_point() + geom_smooth(color="blue") +
```

ggtitle("Original Value VS Predicted Value in Curved Tren Line")

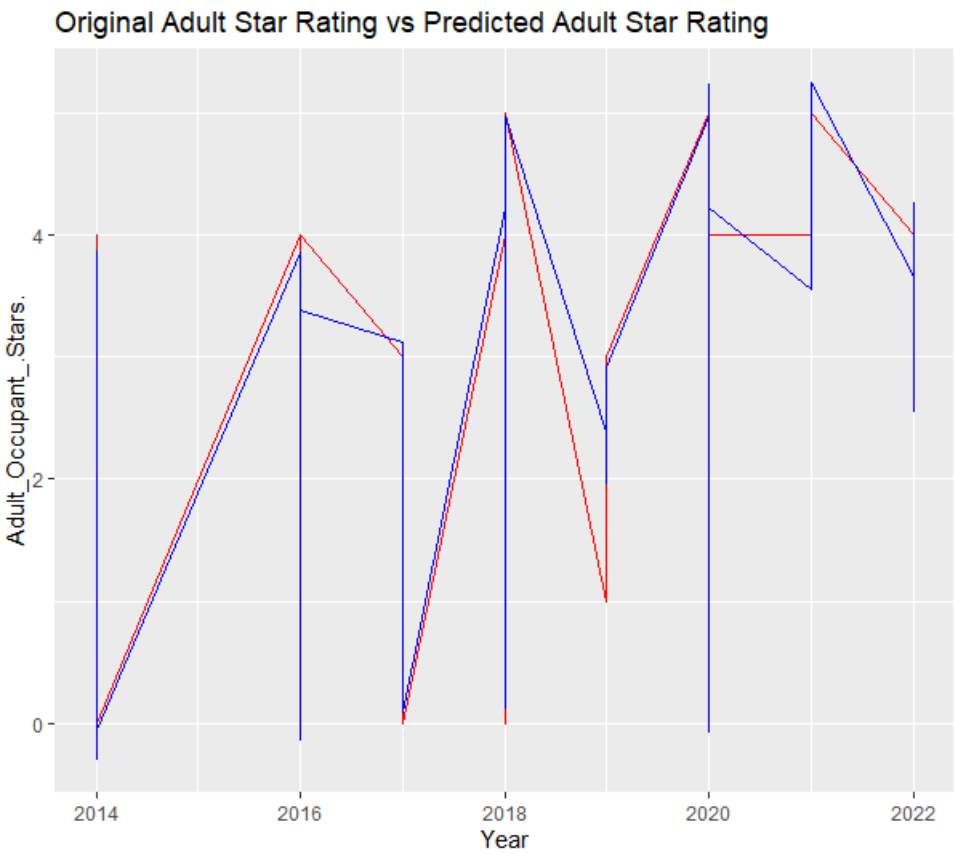


ggplot()+

geom_line(data=prediction,mapping=aes(x=Year,y=Adult_Occupant_.Stars.),color="red")+

geom_line(data=prediction,mapping=aes(x=Year,y=pred),color="blue")+

ggtitle("Original Adult Star Rating vs Predicted Adult Star Rating")



#accuracy

acc <- sqrt(mean(pred-st$Adult_Occupant_.Stars.)^2)

acc

[1] 3.199442e-16

---

*Time Series Decomposition

---

library(forecast)

library(smooth)

library(ggplot2)

library(fpp2)

```
library(tseries)
library(imputeTS)

st <- read.csv("E://fffiiles//college pracs and projects//Project//Car//CSV//Only_Global_NCAP.csv")
dim(st)
[1] 55 14
names(st)
[1] "ï..Brand"            "Name"
"Adult_Occupant_.Stars." "Child_Ocuupant_.Stars."
"Body_Type"
 [6] "No_of_Doors"         "Airbags"             "No_of_Airbags"
"Crash_Test_Weight..kg." "Year"
[11] "Testing_Speed..km.hr."  "Adult_Points.17.00."
"Child_Points.49."      "Bodyshell.Integrity"
#transforming raw data then plotting as well
st.ts=ts(st[,12],start = c(2014,1),end=c(2022,4),frequency=12)
st.ts
     Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov
Dec
2014 12.54  0.00  0.00  0.00  0.00  0.00  0.00  0.00 13.00  0.00
0.00  0.00
2015  0.00  0.00  1.13  0.00  9.85  0.00  8.28  0.00 11.15 10.49
0.00  9.00
```

2016 0.00 13.56 0.00 12.51 7.08 12.85 16.06 8.36 6.74 6.93 9.25 16.13

2017 16.42 12.52 12.52 8.03 7.05 0.00 12.52 11.62 12.00 16.45 16.03 12.34

2018 11.85 13.89 12.03 8.00 8.84 13.52 9.30 12.54 0.00 0.00 0.00 0.00

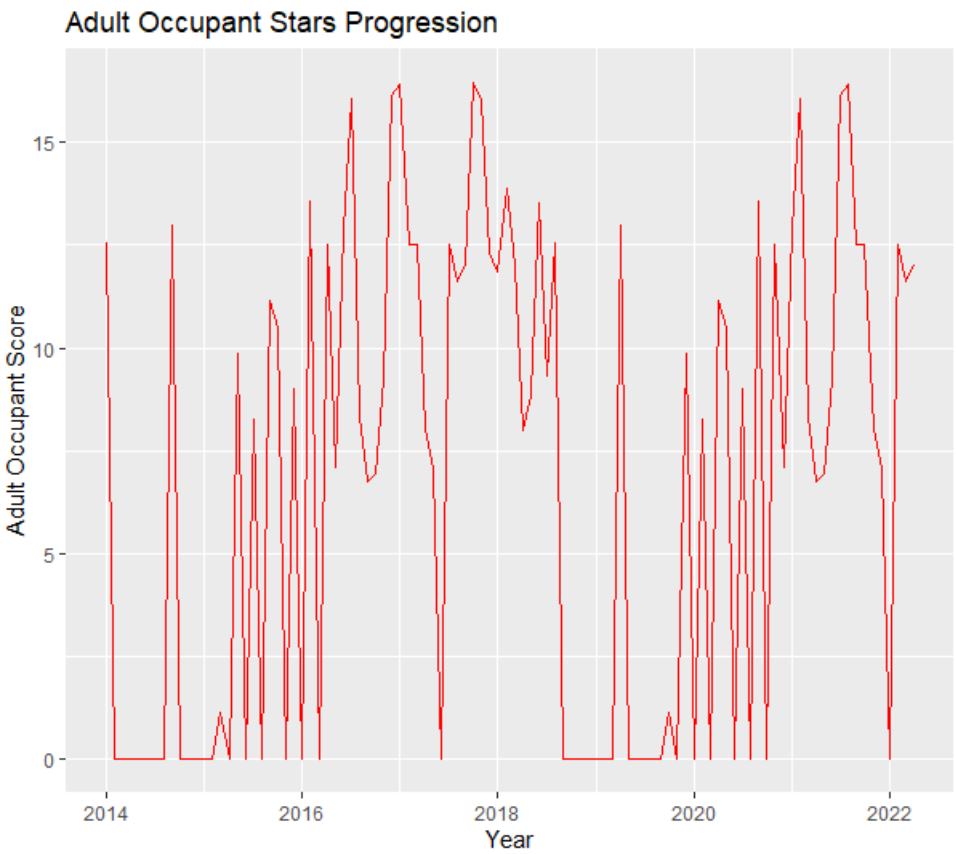
2019 0.00 0.00 0.00 13.00 0.00 0.00 0.00 0.00 0.00 1.13 0.00 9.85

2020 0.00 8.28 0.00 11.15 10.49 0.00 9.00 0.00 13.56 0.00 12.51 7.08

2021 12.85 16.06 8.36 6.74 6.93 9.25 16.13 16.42 12.52 12.52 8.03 7.05

2022 0.00 12.52 11.62 12.00

```
#here is the plot

autoplot(st.ts)+ ggtitle("Adult Occupant Stars Progression")+xlab("Year")+ylab("Adult Occupant Score")+geom_line(color="red")
```



#seeing frequency

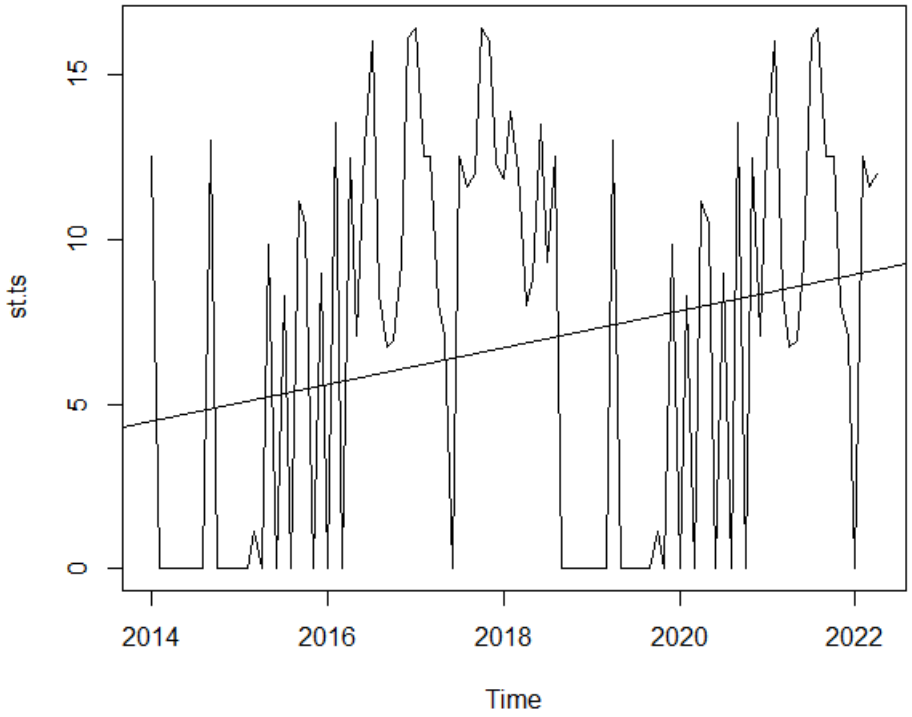
frequency(st)

[1] 1

#with linear regression

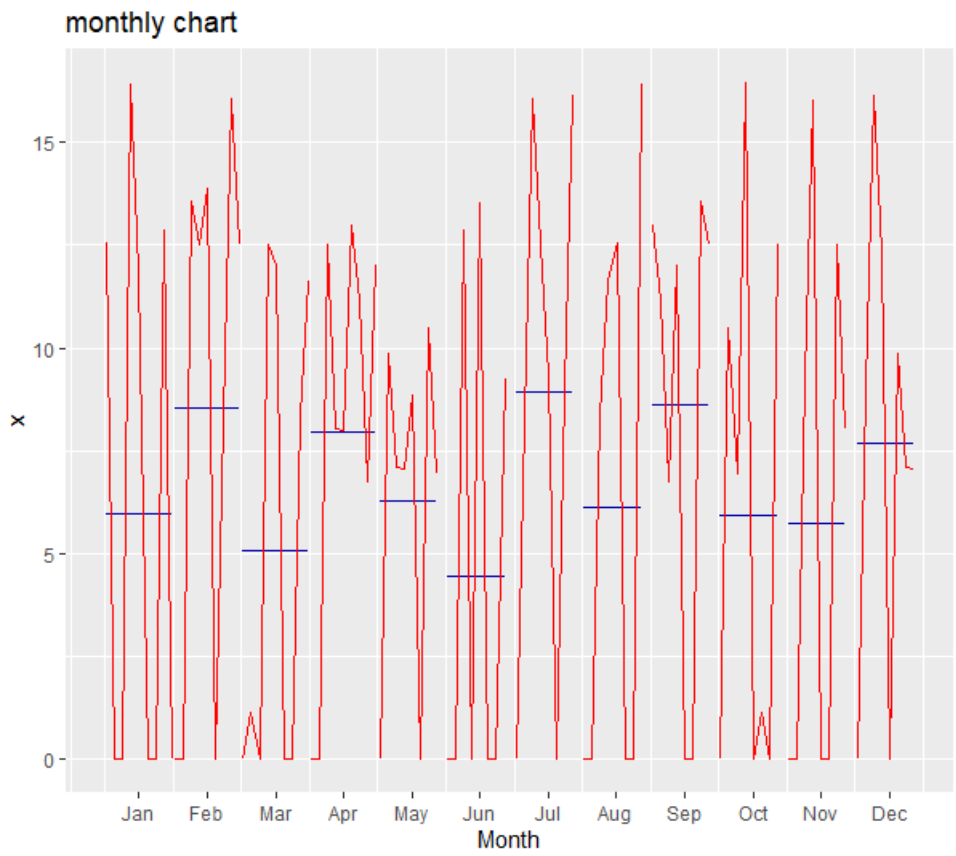
plot(st.ts)

abline(reg=lm(st.ts~time(st.ts)))

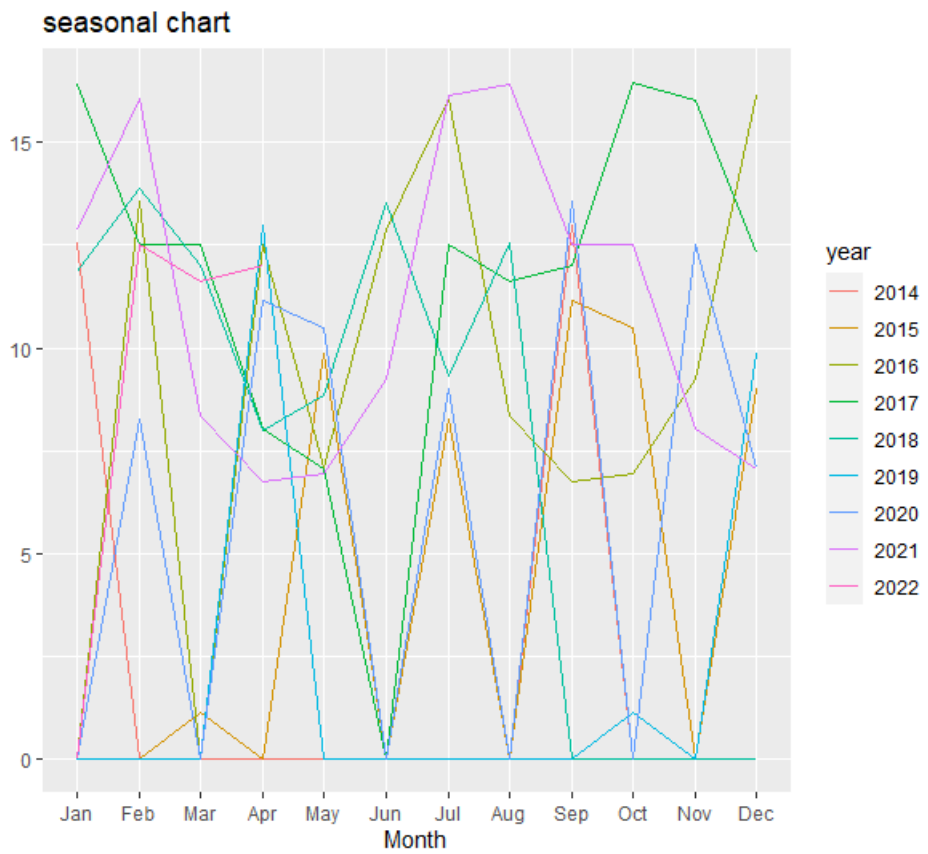


#monthly plot and seasonal plot

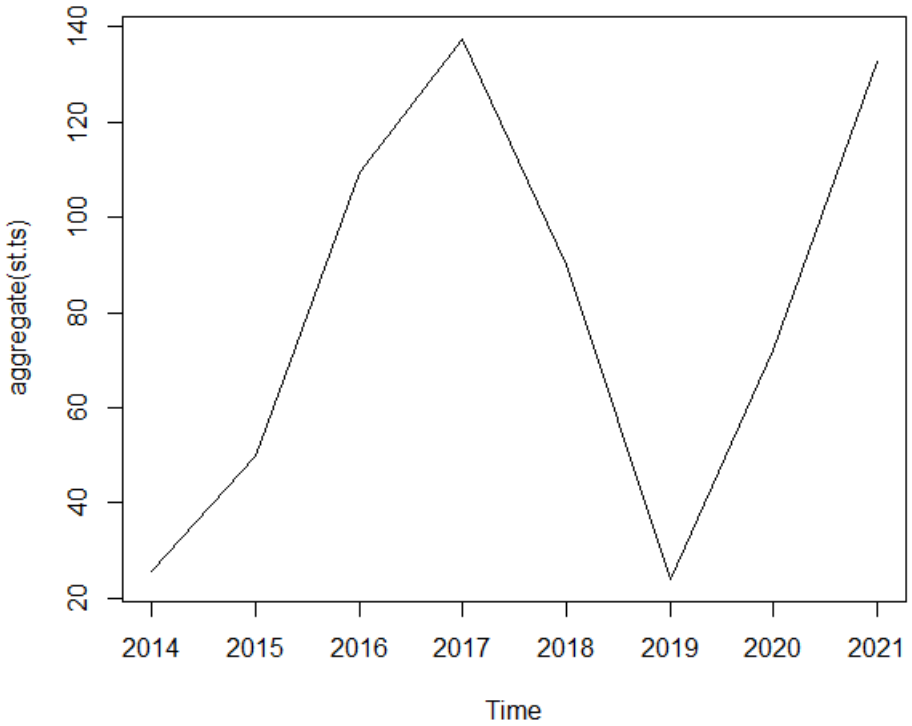
ggmonthplot(st.ts)+geom_line(color="red")+ggtitle("monthly chart")




```
frequency(st.ts)
[1] 12
which.max(st.ts)
[1] 46
ggseasonplot(st.ts)+ggtitle("seasonal chart")
```



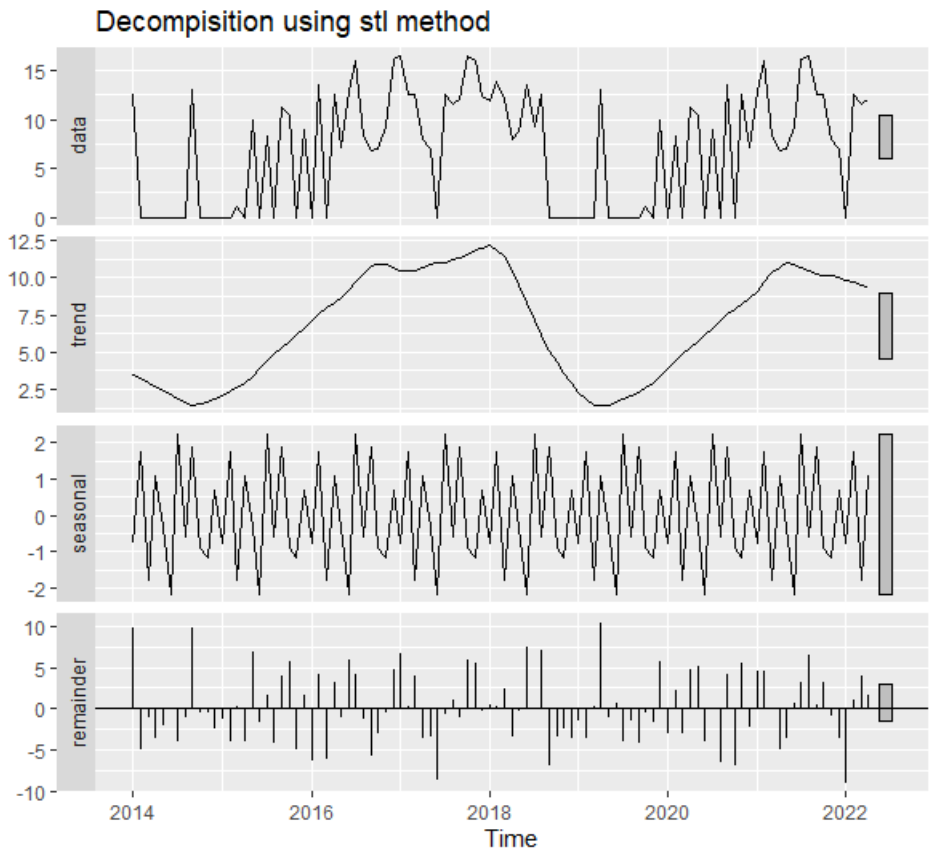
plot(aggregate(st.ts), FUN=mean)



#decomposition

st.ts.decomposition =stl(st.ts,s.window="periodic")

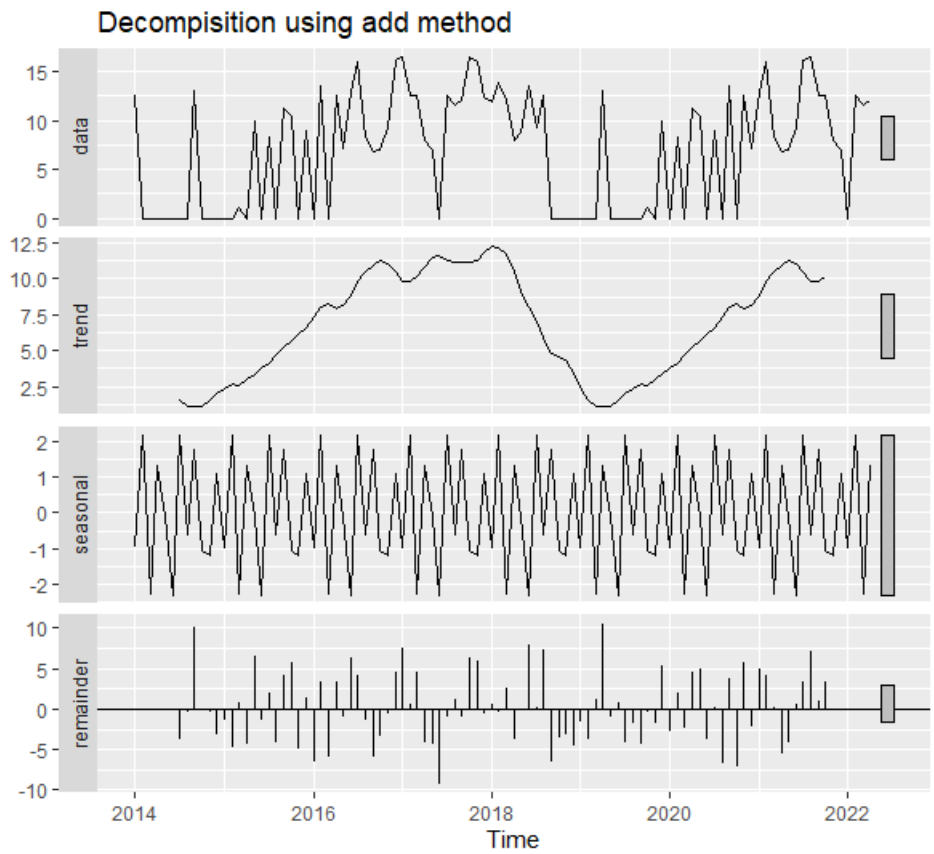
autoplot(st.ts.decomposition,main="Decompisition using stl method")



#additive

st.ts.add=decompose(st.ts,type = "additive")

autoplot(st.ts.add,main="Decompisition using add method")



#values being formed

st.ts.decomposition$time.series

	seasonal	trend	remainder
Jan 2014	-0.7495660	3.476219	9.81334738
Feb 2014	1.7656326	3.207746	-4.97337817
Mar 2014	-1.7569494	2.939273	-1.18232314
Apr 2014	1.0798847	2.676804	-3.75668822
May 2014	-0.3288203	2.414334	-2.08551414
Jun 2014	-2.1814940	2.136618	0.04487635

Jul 2014	2.2520816	1.858901	-4.11098241
Aug 2014	-0.5823203	1.625973	-1.04365319
Sep 2014	1.8807730	1.393046	9.72618093
Oct 2014	-0.8807587	1.509932	-0.62917298
Nov 2014	-1.1735434	1.626817	-0.45327378
Dec 2014	0.6750805	1.859381	-2.53446196
Jan 2015	-0.7495660	2.091946	-1.34237959
Feb 2015	1.7656326	2.347973	-4.11360566
Mar 2015	-1.7569494	2.604001	0.28294884
Apr 2015	1.0798847	2.983719	-4.06360403
May 2015	-0.3288203	3.363438	6.81538226
Jun 2015	-2.1814940	3.882796	-1.70130193
Jul 2015	2.2520816	4.402154	1.62576462
Aug 2015	-0.5823203	4.850158	-4.26783799
Sep 2015	1.8807730	5.298163	3.97106429
Oct 2015	-0.8807587	5.713172	5.65758671
Nov 2015	-1.1735434	6.128181	-4.95463777
Dec 2015	0.6750805	6.639855	1.68506409
Jan 2016	-0.7495660	7.151530	-6.40196351
Feb 2016	1.7656326	7.568316	4.22605174
Mar 2016	-1.7569494	7.985102	-6.22815243
Apr 2016	1.0798847	8.288942	3.14117337

May 2016	-0.3288203	8.592782	-1.18396168
Jun 2016	-2.1814940	9.143392	5.88810185
Jul 2016	2.2520816	9.694002	4.11391612
Aug 2016	-0.5823203	10.227199	-1.28487902
Sep 2016	1.8807730	10.760396	-5.90116927
Oct 2016	-0.8807587	10.843800	-3.03304160
Nov 2016	-1.1735434	10.927204	-0.50366082
Dec 2016	0.6750805	10.689779	4.76514093
Jan 2017	-0.7495660	10.452353	6.71721323
Feb 2017	1.7656326	10.431877	0.32249015
Mar 2017	-1.7569494	10.411402	3.86554765
Apr 2017	1.0798847	10.633448	-3.68333244
May 2017	-0.3288203	10.855494	-3.47667338
Jun 2017	-2.1814940	10.946619	-8.76512535
Jul 2017	2.2520816	11.037745	-0.76982658
Aug 2017	-0.5823203	11.163898	1.03842240
Sep 2017	1.8807730	11.290051	-1.17082374
Oct 2017	-0.8807587	11.545710	5.78504842
Nov 2017	-1.1735434	11.801370	5.40217369
Dec 2017	0.6750805	11.992479	-0.32755948
Jan 2018	-0.7495660	12.183588	0.41597790
Feb 2018	1.7656326	11.809297	0.31507003

Mar 2018	-1.7569494	11.435007	2.35194273
Apr 2018	1.0798847	10.462325	-3.54221016
May 2018	-0.3288203	9.489644	-0.32082390
Jun 2018	-2.1814940	8.309898	7.39159630
Jul 2018	2.2520816	7.130151	-0.08223277
Aug 2018	-0.5823203	6.120273	7.00204748
Sep 2018	1.8807730	5.110394	-6.99116739
Oct 2018	-0.8807587	4.369131	-3.48837255
Nov 2018	-1.1735434	3.627868	-2.45432460
Dec 2018	0.6750805	2.984931	-3.66001165
Jan 2019	-0.7495660	2.341994	-1.59242814
Feb 2019	1.7656326	1.911487	-3.67711907
Mar 2019	-1.7569494	1.480979	0.27597056
Apr 2019	1.0798847	1.461513	10.45860230
May 2019	-0.3288203	1.442047	-1.11322681
Jun 2019	-2.1814940	1.649684	0.53180973
Jul 2019	2.2520816	1.857321	-4.10940300
Aug 2019	-0.5823203	2.074372	-1.49205127
Sep 2019	1.8807730	2.291422	-4.17219467
Oct 2019	-0.8807587	2.602799	-0.59204010
Nov 2019	-1.1735434	2.914176	-1.74063241
Dec 2019	0.6750805	3.408506	5.76641322

Jan 2020	-0.7495660	3.902837	-3.15327059
Feb 2020	1.7656326	4.401135	2.11323273
Mar 2020	-1.7569494	4.899433	-3.14248338
Apr 2020	1.0798847	5.304989	4.76512606
May 2020	-0.3288203	5.710546	5.10827466
Jun 2020	-2.1814940	6.162816	-3.98132236
Jul 2020	2.2520816	6.615087	0.13283136
Aug 2020	-0.5823203	7.105919	-6.52359902
Sep 2020	1.8807730	7.596752	4.08247550
Oct 2020	-0.8807587	7.919648	-7.03888903
Nov 2020	-1.1735434	8.242544	5.44099955
Dec 2020	0.6750805	8.676781	-2.27186179
Jan 2021	-0.7495660	9.111019	4.48854742
Feb 2021	1.7656326	9.714594	4.57977386
Mar 2021	-1.7569494	10.318169	-0.20121913
Apr 2021	1.0798847	10.655305	-4.99518945
May 2021	-0.3288203	10.992441	-3.73362063
Jun 2021	-2.1814940	10.835409	0.59608472
Jul 2021	2.2520816	10.678378	3.19954080
Aug 2021	-0.5823203	10.470628	6.53169202
Sep 2021	1.8807730	10.262879	0.37634812
Oct 2021	-0.8807587	10.180526	3.22023258

Nov 2021 -1.1735434 10.098173 -0.89462985

Dec 2021 0.6750805 9.976252 -3.60133222

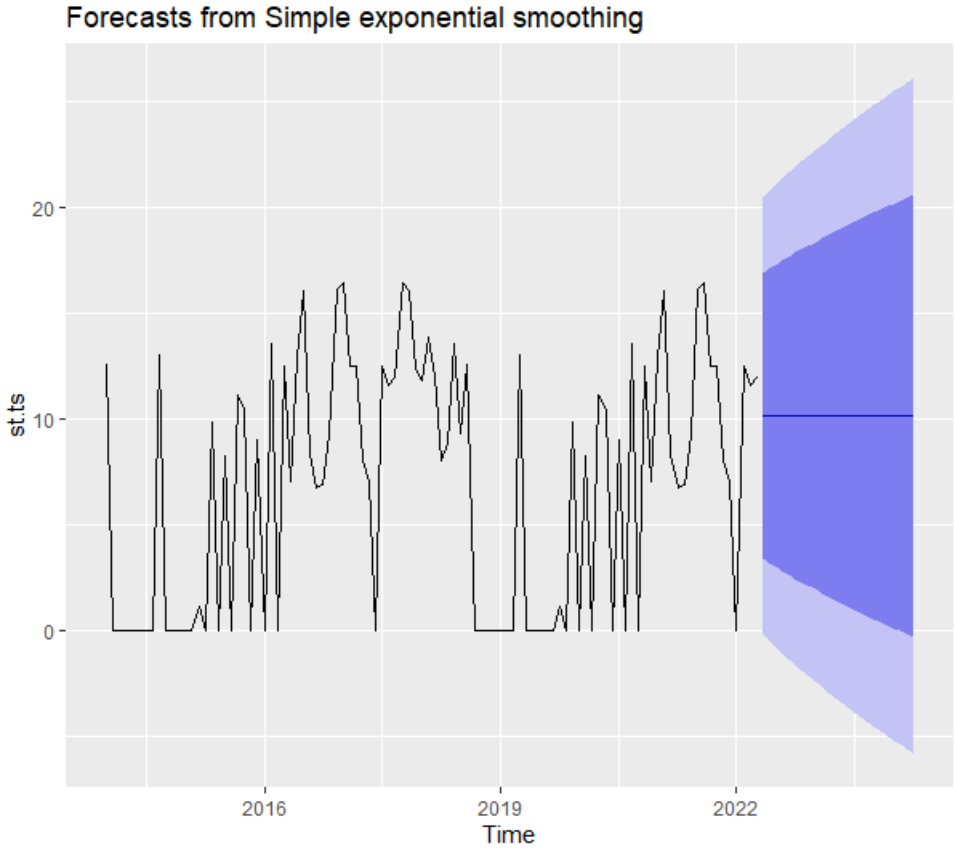
Jan 2022 -0.7495660 9.854330 -9.10476405

Feb 2022 1.7656326 9.688761 1.06560630

Mar 2022 -1.7569494 9.523192 3.85375721

Apr 2022 1.0798847 9.353445 1.56666988

```
#single expo smoothing
stseas= ses(st.ts,h=24)
autoplot(stseas)
```




```
stseas$model
```

```
Simple exponential smoothing

Call:
 ses(y = st.ts, h = 24)

  Smoothing parameters:
    alpha = 0.2473

  Initial states:
    l = 3.5038

  sigma:  5.2506

     AIC     AICc      BIC
796.1639 796.4139 803.9795
```

```
summary(stseas)
```

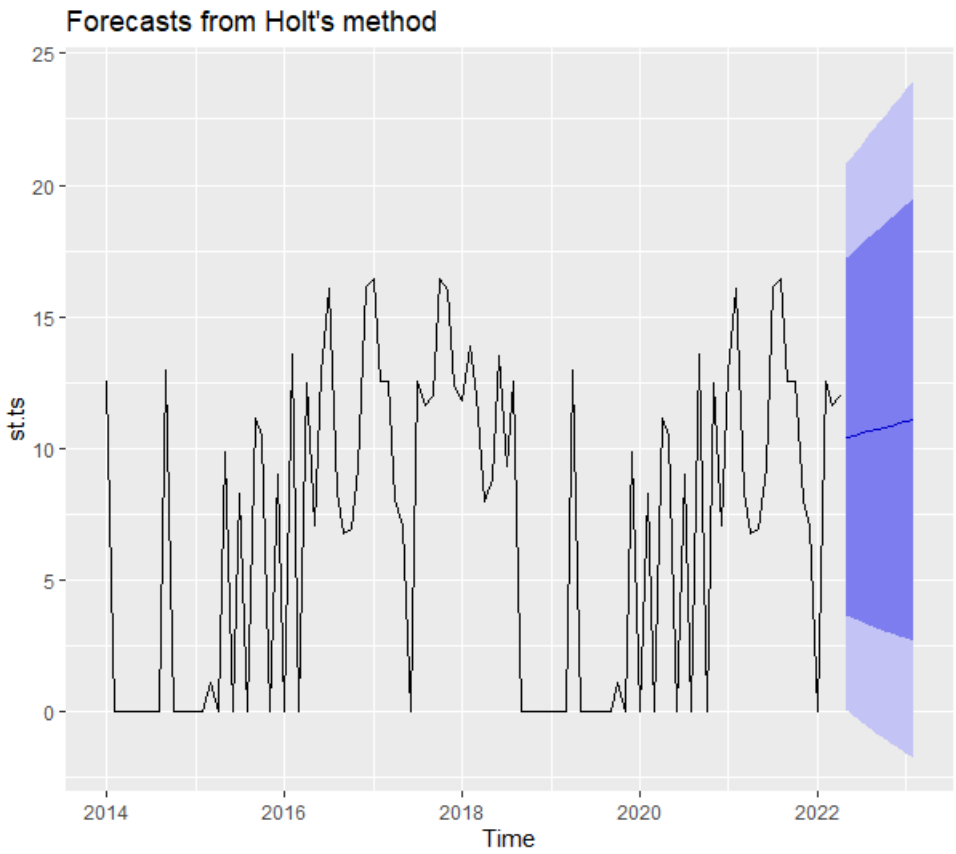
```
Forecast method: Simple exponential smoothing

Model Information:
Simple exponential smoothing
```

```
Call:
 ses(y = st.ts, h = 24)

  Smoothing parameters:
    alpha = 0.2473

  Initial states:
    l = 3.5038

  sigma:  5.2506

     AIC     AICc      BIC
796.1639 796.4139 803.9795

Error measures:
                    ME     RMSE    MAE  MPE MAPE      MASE       ACF1
Training set 0.2686109 5.197796 4.2755 -Inf  Inf 0.6612954 -0.08156403

Forecasts:
         Point Forecast      Lo 80    Hi 80      Lo 95    Hi 95
```

May 2022	10.14679	3.41791752	16.87566	-0.1441320	20.43771
Jun 2022	10.14679	3.21519636	17.07838	-0.4541673	20.74775
Jul 2022	10.14679	3.01823783	17.27534	-0.7553894	21.04897
Aug 2022	10.14679	2.82657678	17.46700	-1.0485097	21.34209
Sep 2022	10.14679	2.63980744	17.65377	-1.3341487	21.62773
Oct 2022	10.14679	2.45757335	17.83601	-1.6128517	21.90643
Nov 2022	10.14679	2.27955934	18.01402	-1.8851006	22.17868
Dec 2022	10.14679	2.10548514	18.18809	-2.1513242	22.44490
Jan 2023	10.14679	1.93510019	18.35848	-2.4119054	22.70548
Feb 2023	10.14679	1.76817943	18.52540	-2.6671887	22.96077
Mar 2023	10.14679	1.60451978	18.68906	-2.9174846	23.21106
Apr 2023	10.14679	1.44393724	18.84964	-3.1630744	23.45665

May 2023	10.14679	1.28626453	19.00731	-3.4042140	23.69779
Jun 2023	10.14679	1.13134896	19.16223	-3.6411369	23.93472
Jul 2023	10.14679	0.97905076	19.31453	-3.8740569	24.16764
Aug 2023	10.14679	0.82924160	19.46434	-4.1031702	24.39675
Sep 2023	10.14679	0.68180327	19.61178	-4.3286577	24.62224
Oct 2023	10.14679	0.53662666	19.75695	-4.5506861	24.84426
Nov 2023	10.14679	0.39361078	19.89997	-4.7694101	25.06299
Dec 2023	10.14679	0.25266192	20.04092	-4.9849727	25.27855
Jan 2024	10.14679	0.11369298	20.17989	-5.1975074	25.49109
Feb 2024	10.14679	-0.02337723	20.31696	-5.4071382	25.70072
Mar 2024	10.14679	-0.15862445	20.45220	-5.6139810	25.90756
Apr 2024	10.14679	-0.29211954	20.58570	-5.8181441	26.11172

```
#with holt
stholt= holt(st.ts,t.h=24)
autoplot(stholt)
```




```
stholt$model
```

```
Holt's method

Call:
 holt(y = st.ts, t.h = 24)

 Smoothing parameters:
```

```
  alpha = 0.2413
  beta  = 1e-04

 Initial states:
  l = 3.0461
  b = 0.0725

 sigma:  5.2982

    AIC     AICc      BIC
799.9084 800.5467 812.9343
summary(stholt)

Forecast method: Holt's method

Model Information:
Holt's method

Call:
 holt(y = st.ts, t.h = 24)

 Smoothing parameters:
```

alpha = 0.2413

beta = 1e-04

Initial states:

l = 3.0461

b = 0.0725

sigma: 5.2982

AIC AICc BIC

799.9084 800.5467 812.9343

Error measures:

ME RMSE MAE MPE MAPE MASE ACF1

Training set 0.002629141 5.19116 4.2988 -Inf Inf 0.6648991 -0.07778537

Forecasts:

Point Forecast Lo 80 Hi 80 Lo 95 Hi 95

May 2022 10.43494 3.645020 17.22487 0.05065233 20.81923

Jun 2022 10.50748 3.522573 17.49238 -0.17501109 21.18996

Jul 2022 10.58001 3.405268 17.75475 -0.39281082 21.55283

Aug 2022 10.65254 3.292706 18.01238 -0.60335547 21.90844

Sep 2022 10.72508 3.184539 18.26561 -0.80717942 22.25733

Oct 2022 10.79761 3.080458 18.51476 -1.00475486 22.59997

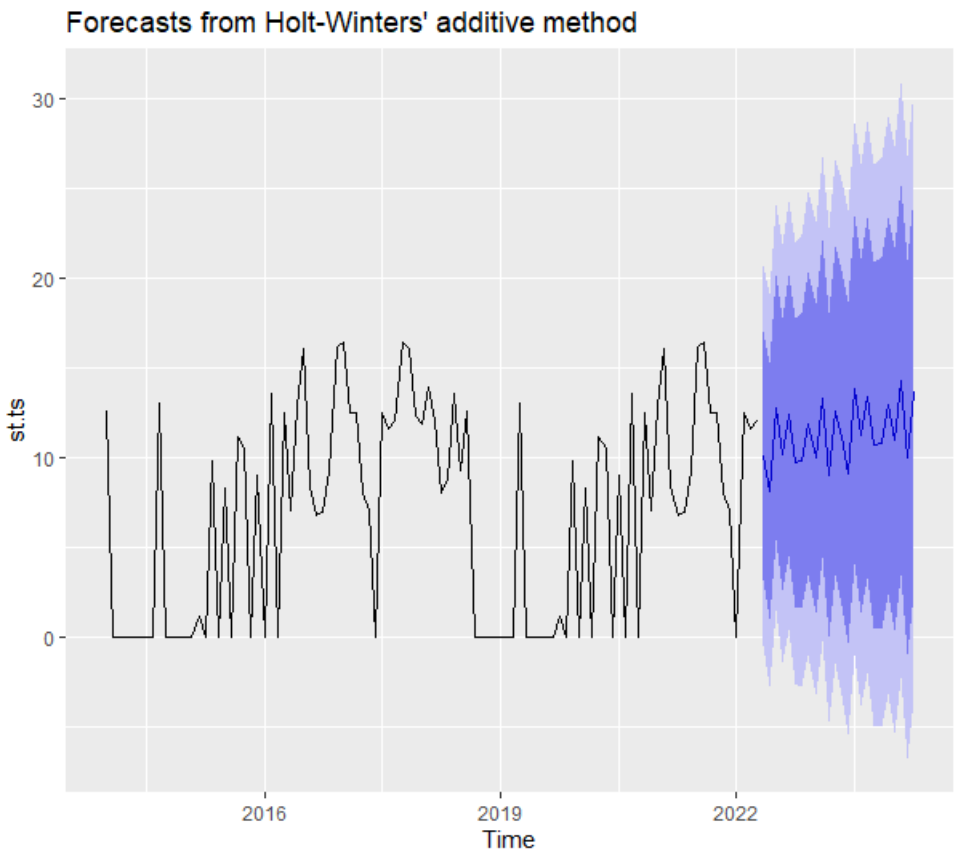
Nov 2022 10.87014 2.980188 18.76009 -1.19650150 22.93678

Dec 2022 10.94267 2.883483 19.00186 -1.38279439 23.26814

Jan 2023 11.01521 2.790125 19.24029 -1.56397025 23.59438

Feb 2023 11.08774 2.699914 19.47557 -1.74033277 23.91581

```
#with holtwinter

stholtwinter= hw(st.ts,t.h=24,seasonal = "a")

autoplot(stholtwinter)
```



stholtwinter$model

Holt-Winters' additive method

Call:

hw(y = st.ts, seasonal = "a", t.h = 24)

Smoothing parameters:

alpha = 0.2613

beta = 1e-04

gamma = 1e-04

```
  Initial states:
    l = 1.0004
    b = 0.0838
    s = 0.9545 -1.0383 -1.0895 1.6675 -0.4129 2.2658
          -2.3232 -0.2587 1.319 -2.2032 2.1825 -1.0632

  sigma:  5.4089

     AIC     AICc      BIC
814.6921 822.1555 858.9800
summary(stholtwinter)
Forecast method: Holt-Winters' additive method

Model Information:
Holt-Winters' additive method

Call:
 hw(y = st.ts, seasonal = "a", t.h = 24)

  Smoothing parameters:
    alpha = 0.2613
```

```
  beta  = 1e-04
  gamma = 1e-04

 Initial states:
  l = 1.0004
  b = 0.0838
  s = 0.9545 -1.0383 -1.0895 1.6675 -0.4129 2.2658
       -2.3232 -0.2587 1.319 -2.2032 2.1825 -1.0632

 sigma:  5.4089

   AIC     AICc      BIC
814.6921 822.1555 858.9800

Error measures:
                   ME     RMSE      MAE MPE MAPE      MASE       ACF1
Training set 0.0321409 4.957371 4.098729 NaN  Inf
0.6339541 -0.04502809

Forecasts:
       Point Forecast       Lo 80    Hi 80      Lo 95    Hi 95
```

May 2022	10.098993	3.167163877	17.03082	-0.5023246	20.70031
Jun 2022	8.118694	0.953894280	15.28349	-2.8389217	19.07631
Jul 2022	12.791548	5.400946605	20.18215	1.4885987	24.09450
Aug 2022	10.196787	2.586916885	17.80666	-1.4415053	21.83508
Sep 2022	12.361619	4.538461512	20.18478	0.3971320	24.32611
Oct 2022	9.688665	1.657725893	17.71960	-2.5935964	21.97093
Nov 2022	9.823653	1.590021722	18.05728	-2.7685995	22.41591
Dec 2022	11.900473	3.468871059	20.33208	-0.9945496	24.79550
Jan 2023	9.967767	1.342590167	18.59294	-3.2233026	23.15884
Feb 2023	13.296671	4.482026665	22.11131	-0.1841641	26.77751
Mar 2023	8.995988	-0.004276179	17.99625	-4.7687282	22.76070
Apr 2023	12.601339	3.419069202	21.78361	-1.4417310	26.64441

May 2023	11.108083	1.747063168	20.46910	-3.2083615	25.42453
Jun 2023	9.127785	-0.408623503	18.66419	-5.4568931	23.71246
Jul 2023	13.800638	4.091879345	23.50940	-1.0476272	28.64890
Aug 2023	11.205878	1.327647036	21.08411	-3.9015726	26.31333
Sep 2023	13.370710	3.325739516	23.41568	-1.9917465	28.73317
Oct 2023	10.697755	0.488644961	20.90687	-4.9157317	26.31124
Nov 2023	10.832744	0.461968505	21.20352	-5.0279883	26.69348
Dec 2023	12.909564	2.379485211	23.43964	-3.1948018	29.01393
Jan 2024	10.976857	0.289730892	21.66398	-5.3676922	27.32141
Feb 2024	14.305761	3.463744807	25.14778	-2.2756721	30.88719
Mar 2024	10.005079	-0.989761589	20.99992	-6.8100784	26.82024
Apr 2024	13.610430	2.464747609	24.75611	-3.4354203	30.65628

---

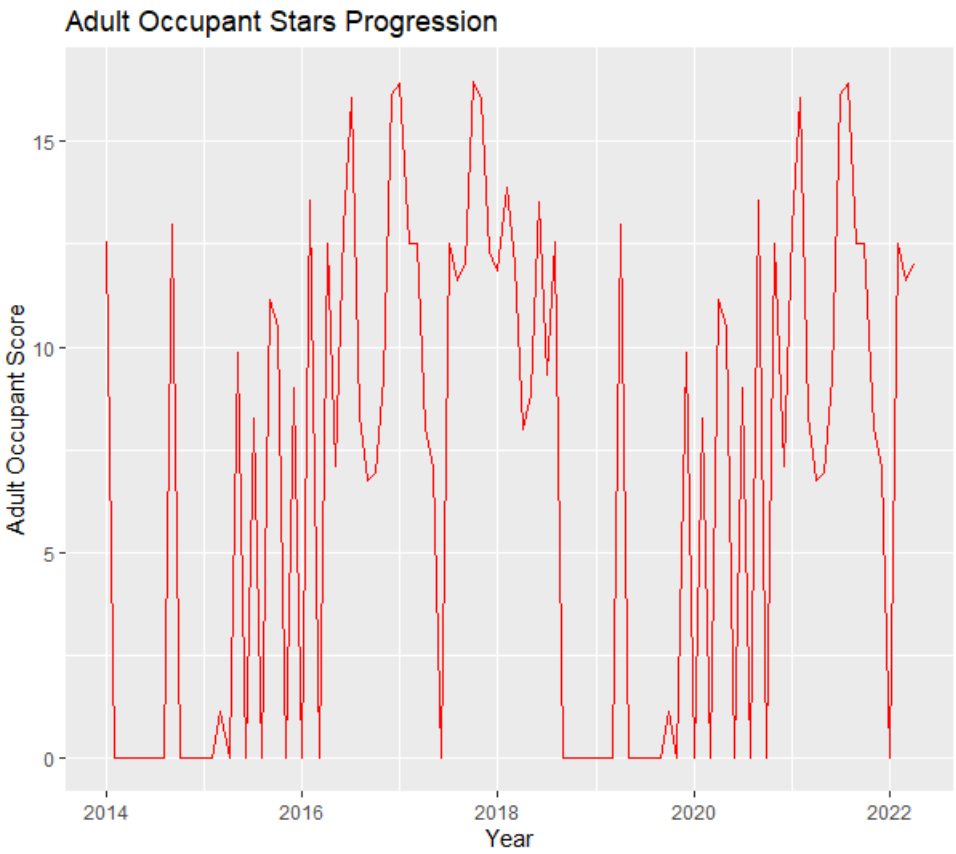
*Arima

```
library(forecast)
library(smooth)
library(ggplot2)
library(fpp2)
library(tseries)
library(imputeTS)

st.arima <- read.csv("E://fffiiles//college pracs and projects//Project//Car//CSV//Only_Global_NCAP.csv")
dim(st.arima)
[1] 55 14
names(st.arima)
[1] "ï..Brand"            "Name"
"Adult_Occupant_.Stars." "Child_Ocuupant_.Stars."
"Body_Type"
 [6] "No_of_Doors"         "Airbags"             "No_of_Airbags"
"Crash_Test_Weight..kg." "Year"
[11] "Testing_Speed..km.hr."  "Adult_Points.17.00."
"Child_Points.49."      "Bodyshell.Integrity"
attach(st.arima)
```

```
#transforming raw data then plotting as well
st.ts.arima=ts(st.arima[,12],start =
c(2014,1),end=c(2022,4),frequency=12)
st.ts.arima
Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov
Dec
2014 12.54  0.00  0.00  0.00  0.00  0.00  0.00  0.00 13.00  0.00
0.00  0.00
2015  0.00  0.00  1.13  0.00  9.85  0.00  8.28  0.00 11.15 10.49
0.00  9.00
2016  0.00 13.56  0.00 12.51  7.08 12.85 16.06  8.36  6.74
6.93  9.25 16.13
2017 16.42 12.52 12.52  8.03  7.05  0.00 12.52 11.62 12.00
16.45 16.03 12.34
2018 11.85 13.89 12.03  8.00  8.84 13.52  9.30 12.54  0.00
0.00  0.00  0.00
2019  0.00  0.00  0.00 13.00  0.00  0.00  0.00  0.00  0.00  1.13
0.00  9.85
2020  0.00  8.28  0.00 11.15 10.49  0.00  9.00  0.00 13.56
0.00 12.51  7.08
2021 12.85 16.06  8.36  6.74  6.93  9.25 16.13 16.42 12.52
12.52  8.03  7.05
2022  0.00 12.52 11.62 12.00
```

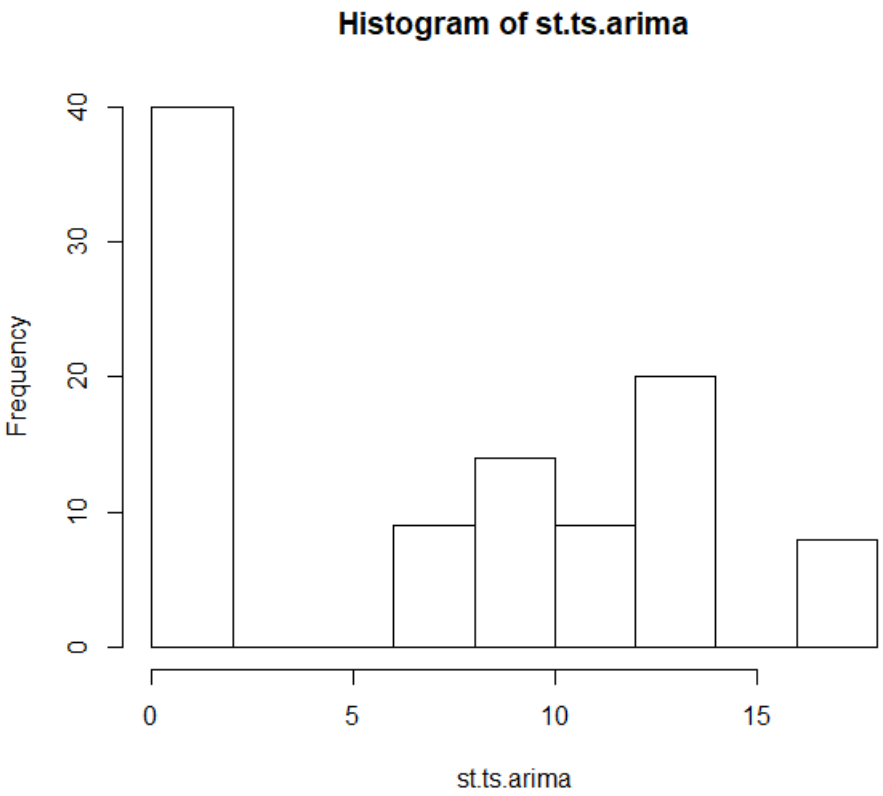
#here is the plot for arima

autoplot(st.ts.arima)+ ggtitle("Adult Occupant Stars Progression")+xlab("Year")+ylab("Adult Occupant Score")+geom_line(color="red")

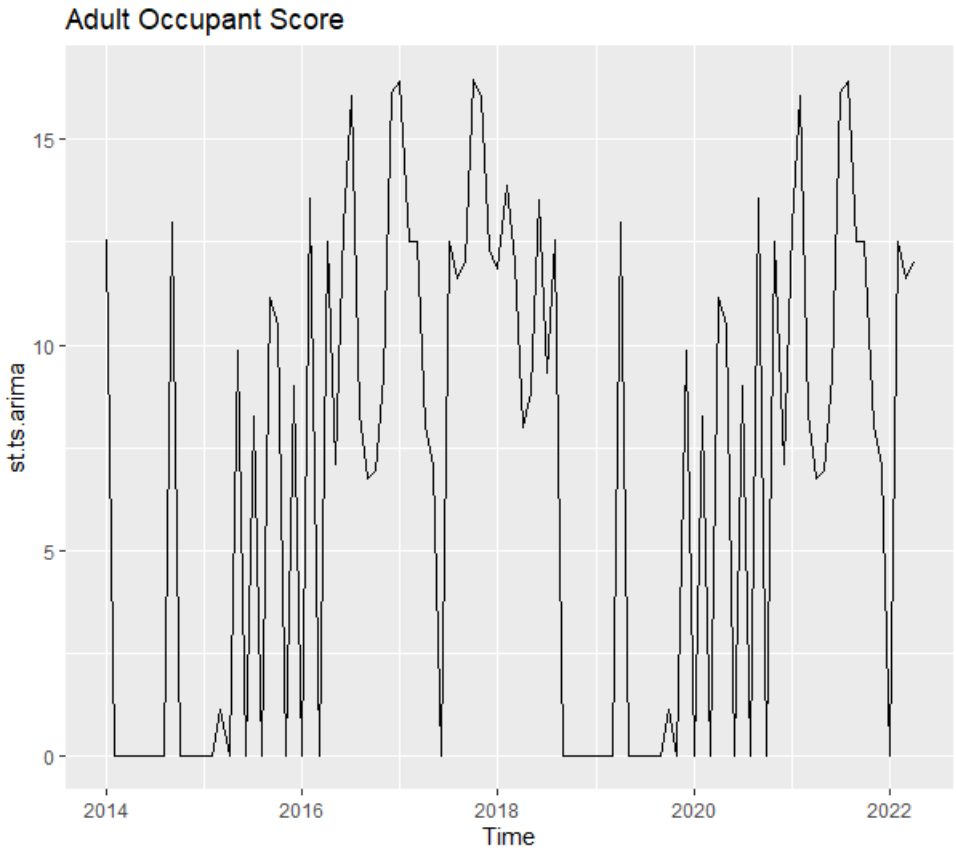


#histogram fot stationary data

hist(st.ts.arima)



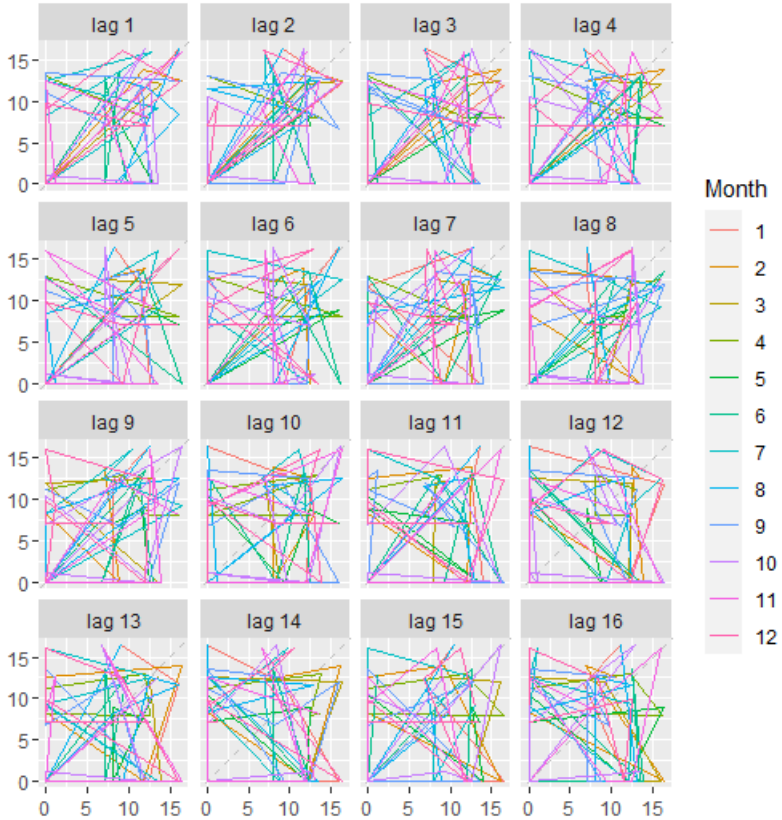
autoplot(st.ts.arima,main="Adult Occupant Score")



#to see lag

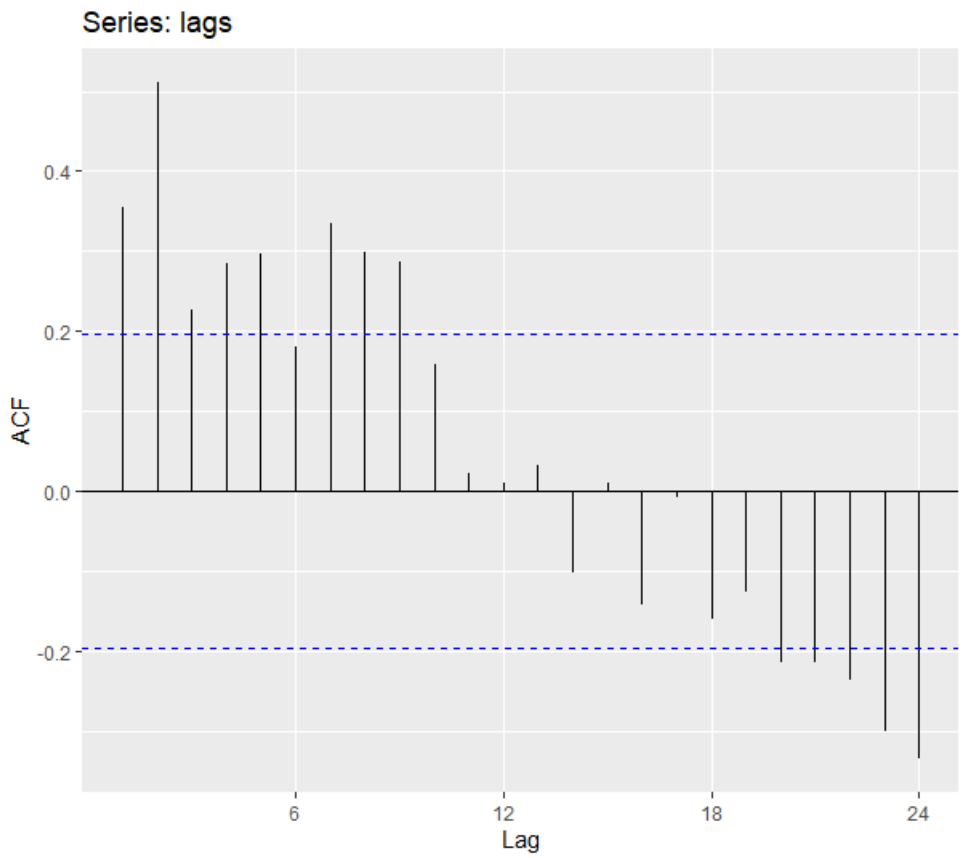
lags=window(st.ts.arima,start=c(2014,1),end=c(2022,4))

gglagplot(lags)



ggAcf(lags)

#stattistcial test



adf.test(st.ts.arima)

Augmented Dickey-Fuller Test

data: st.ts.arima

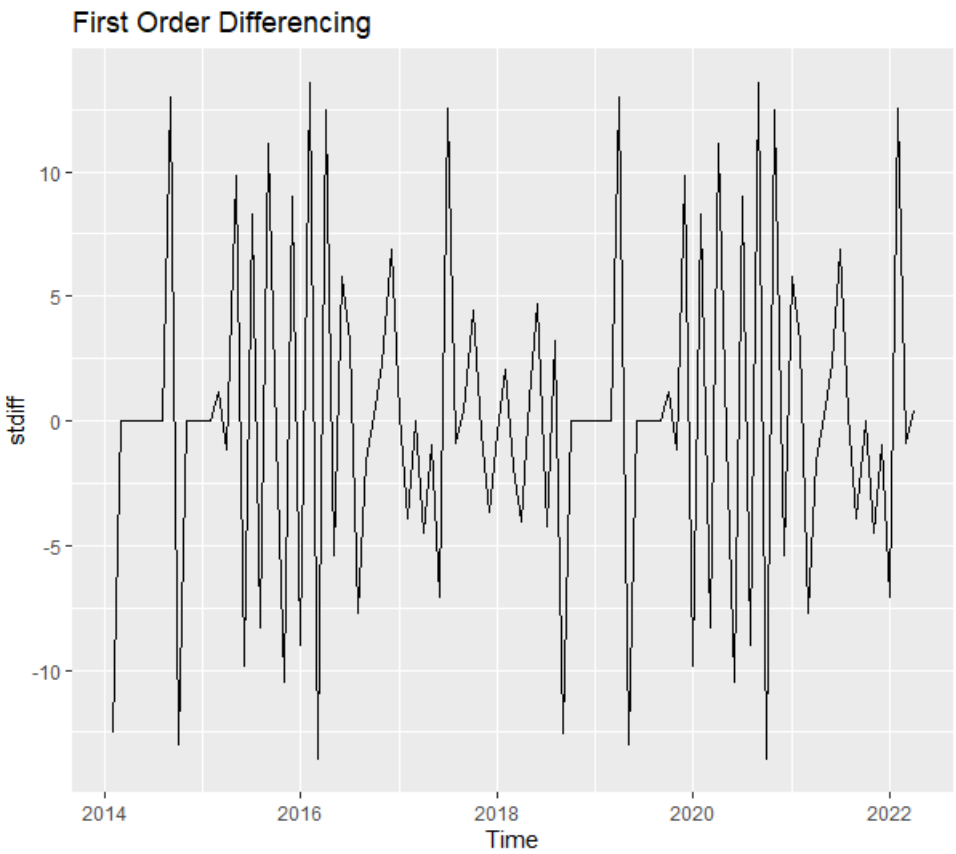
Dickey-Fuller = -2.3857, Lag order = 4, p-value = 0.4168

alternative hypothesis: stationary

#making stationary data

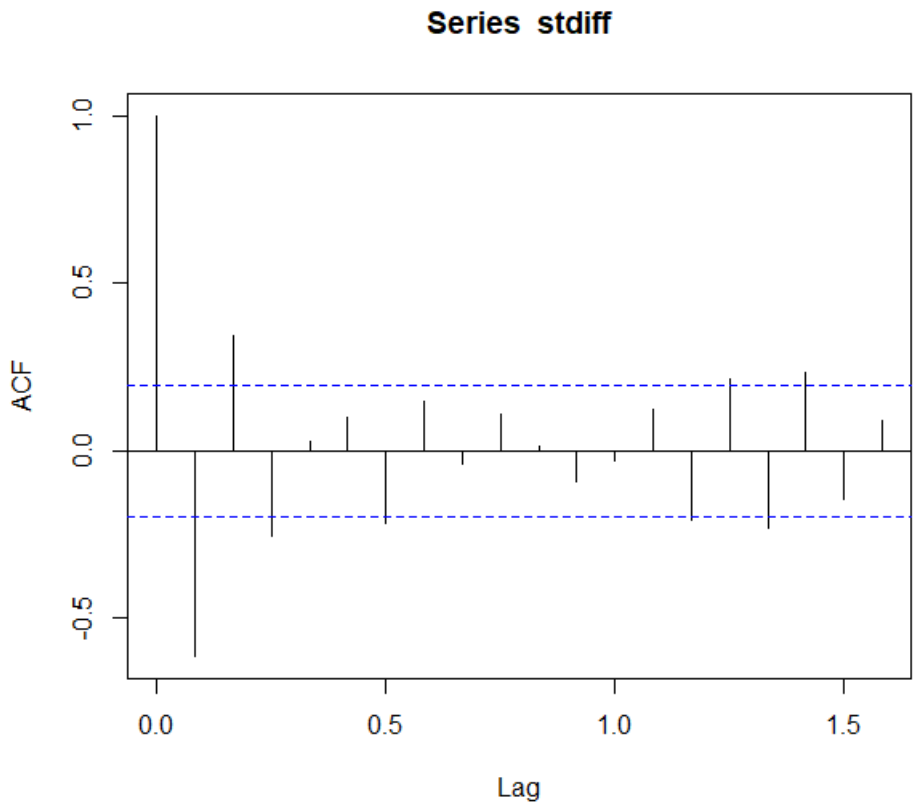
stdiff=diff(st.ts.arima)

autoplot(stdiff,main="First Order Differencing")

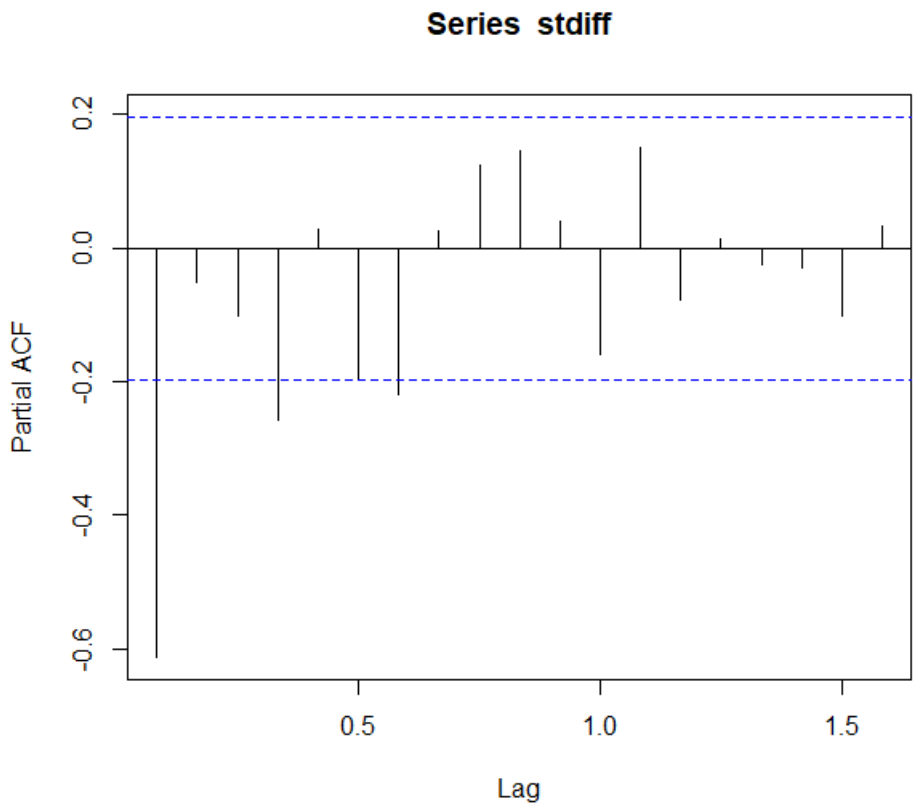


#to see corelation using acf and pacf

acf(stdiff)


Series stdiff
ACF
1.0
0.5
0.0
-0.5
0.0
0.5
1.0
1.5
Lag

pacf(stdiff)


Series stdiff
Partial ACF
0.2
0.0
-0.2
-0.4
-0.6
0.5
1.0
1.5
Lag

```
#split data
sttrain = window(st.ts.arima,end=c(2018,1),frequency=12)
sttest = window(st.ts.arima,start=c(2018,2),frequency=12)


strain
     Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov
Dec
2014 12.54  0.00  0.00  0.00  0.00  0.00  0.00  0.00 13.00  0.00
0.00  0.00
2015  0.00  0.00  1.13  0.00  9.85  0.00  8.28  0.00 11.15 10.49
0.00  9.00
2016  0.00 13.56  0.00 12.51  7.08 12.85 16.06  8.36  6.74
6.93  9.25 16.13
2017 16.42 12.52 12.52  8.03  7.05  0.00 12.52 11.62 12.00
16.45 16.03 12.34
2018 11.85
sttest
     Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov
Dec
2018       13.89 12.03  8.00  8.84 13.52  9.30 12.54  0.00  0.00
0.00  0.00
2019  0.00  0.00  0.00 13.00  0.00  0.00  0.00  0.00  0.00  1.13
0.00  9.85
```

```
2020  0.00  8.28  0.00 11.15 10.49  0.00  9.00  0.00 13.56
0.00 12.51  7.08
2021 12.85 16.06  8.36  6.74  6.93  9.25 16.13 16.42 12.52
12.52  8.03  7.05
2022  0.00 12.52 11.62 12.00
```

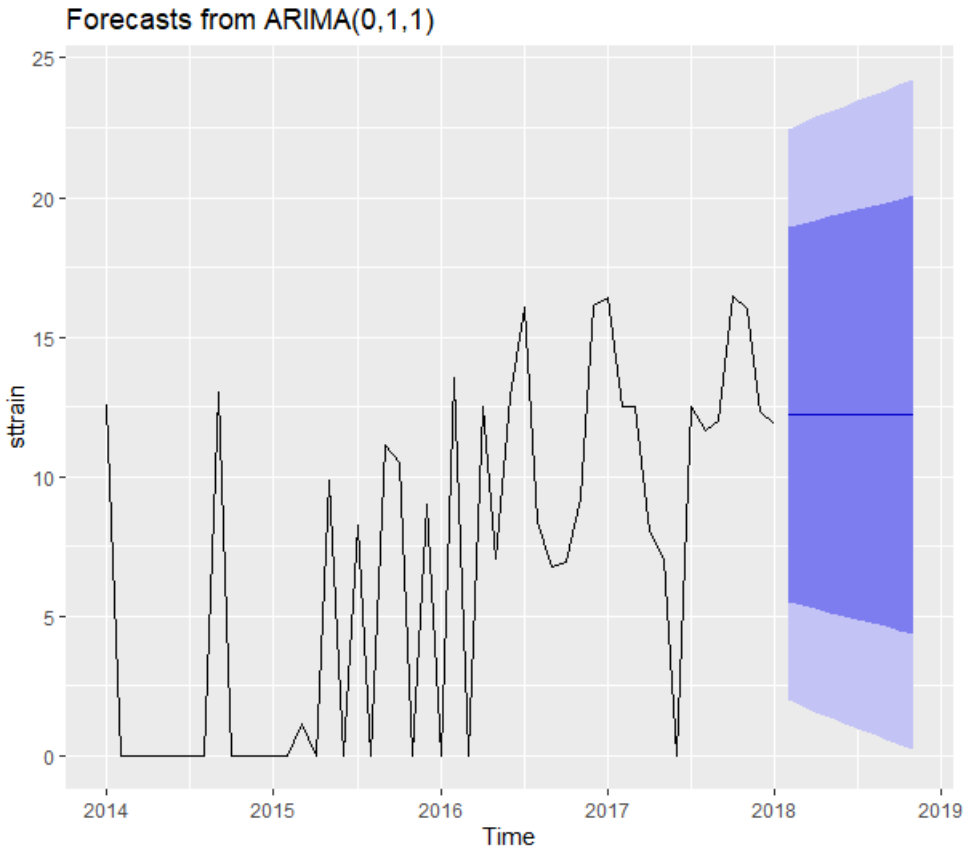
#auto arima model

```
autoarima.train=auto.arima(sttrain,seasonal = TRUE, ic="bic")
autoarima.train
Series: sttrain
ARIMA(0,1,1)

Coefficients:
         ma1
     -0.7970
s.e.   0.0788

sigma^2 estimated as 27.31:  log likelihood=-147.48
AIC=298.96   AICc=299.23   BIC=302.71
```

#camparing values to see accuracy levels

```
autoarima.train.forecast=forecast(autoarima.train,h=10)
```

autoplot(autoarima.train.forecast)



#accuracy

rmse=sqrt(mean(sttest-autoarima.train.forecast$mean)^2)

rmse

[1] 4.405236

accuracy(autoarima.train.forecast,sttest)

```
                    ME     RMSE      MAE  MPE MAPE      MASE
ACF1 Theil's U
Training set  0.5858504 5.118131 4.116792 -Inf  Inf
0.6907682 -0.04412059       NA
```

Test set -4.4052358 7.000844 5.064894 -Inf Inf 0.8498530 0.44970841 0.8970646

#residuals

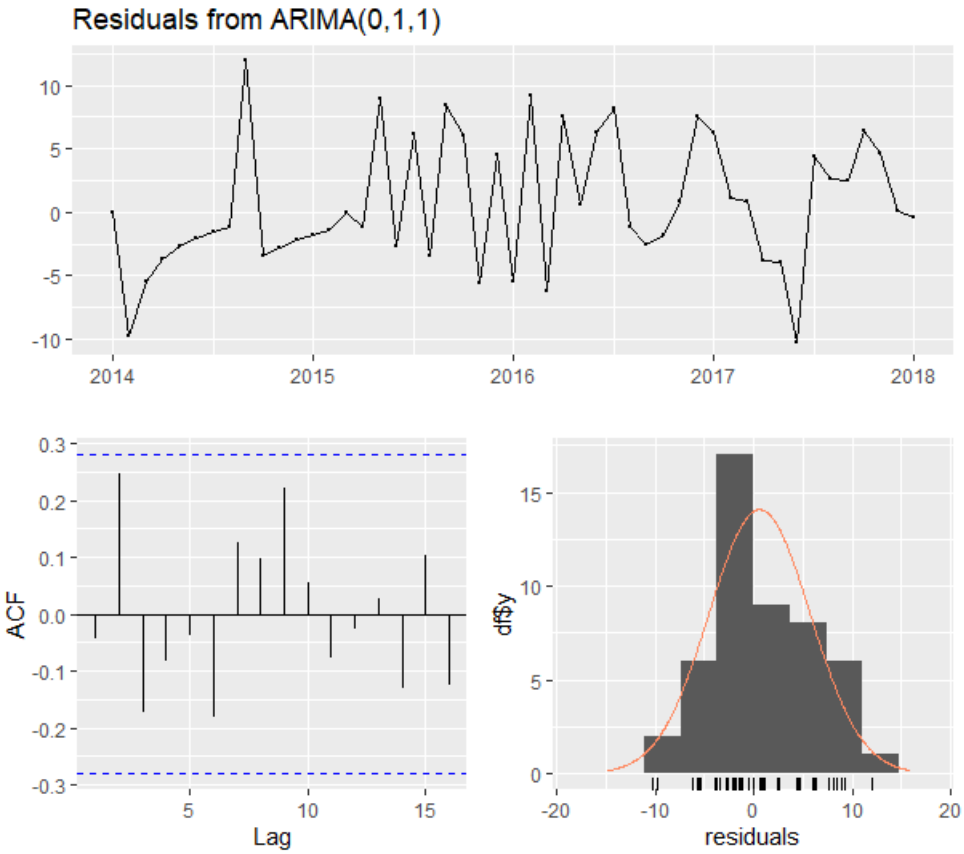
checkresiduals(autoarima.train.forecast)

Ljung-Box test

data: Residuals from ARIMA(0,1,1)

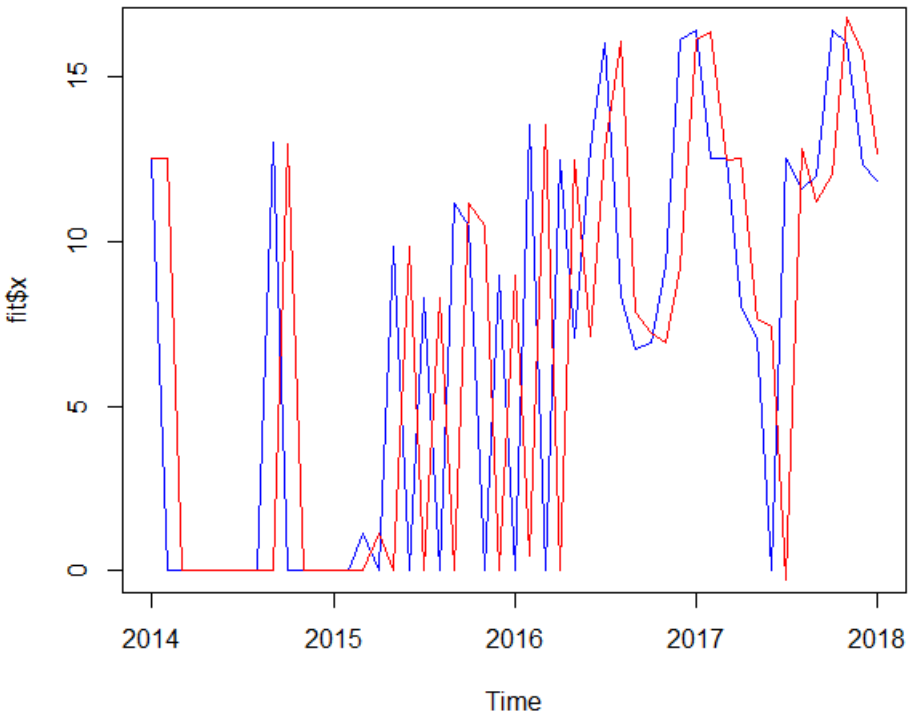
Q* = 12.137, df = 9, p-value = 0.2057

Model df: 1. Total lags used: 10

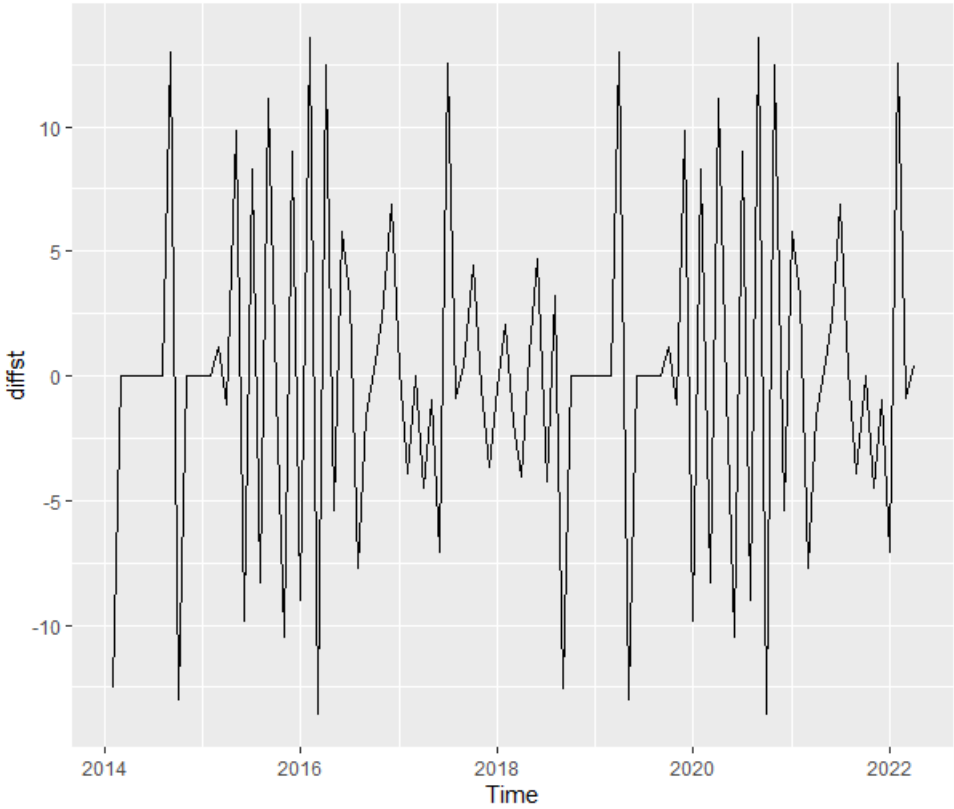


#fitarimamodel

```
fit=Arima(sttrain, c(0,1,0),seasonal =
list(order=c(2,0,0),period=12))
plot(fit$x,col="blue")
lines(fit$fitted,col="red",main="Actucal VS forecast")
```

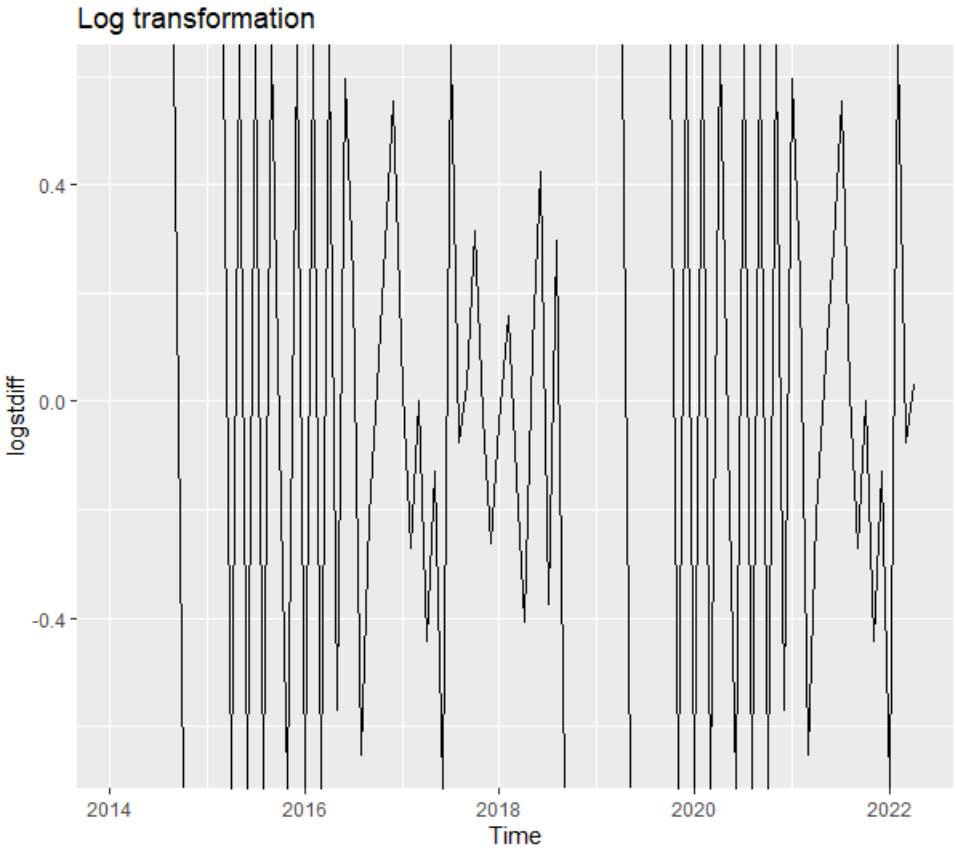



```
#TO see variance and remove

diffst= diff(st.ts.arima)
autoplot(diffst)
```

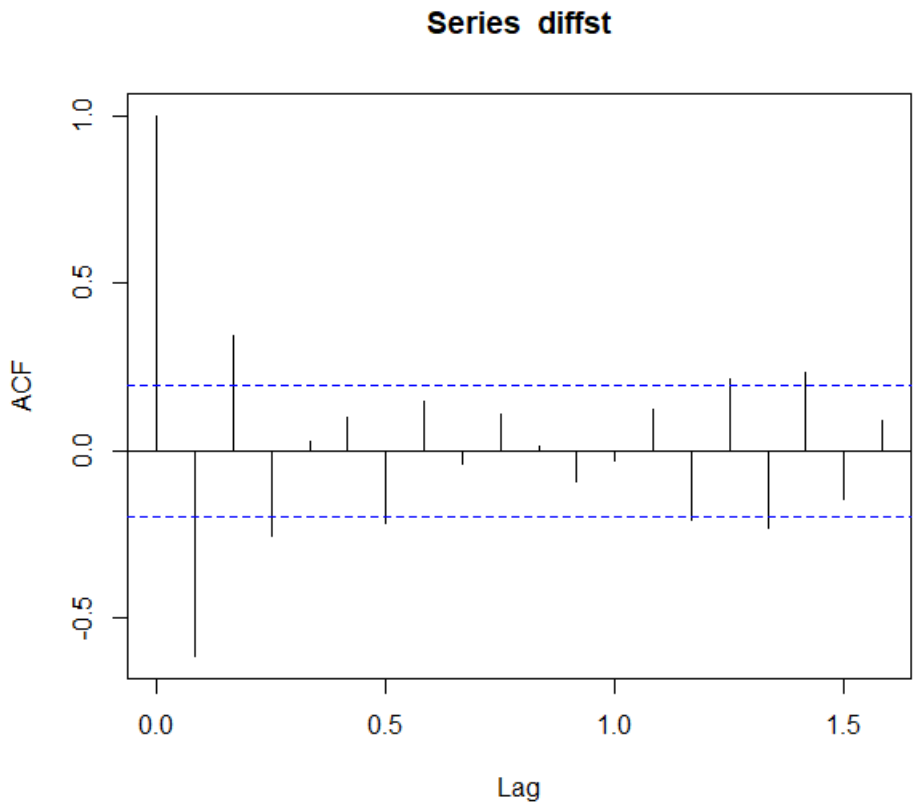


#log

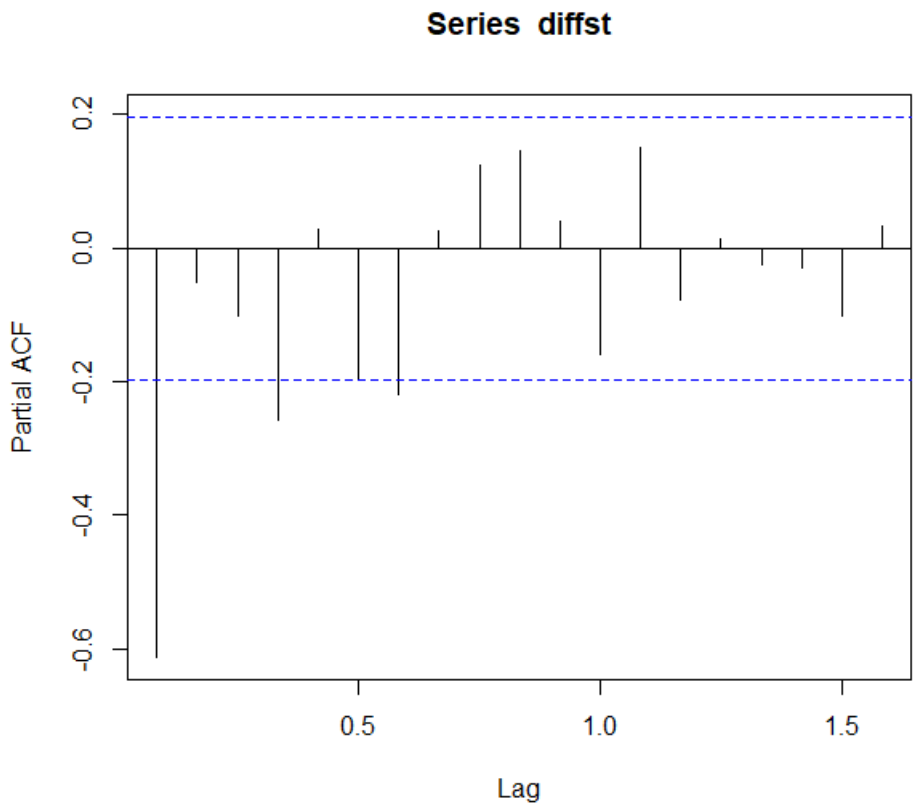
```
logstdiff=diff(log(st.ts.arima))
autoplot(logstdiff,main="Log transformation")
```


Log transformation
0.4
0.0
-0.4
logstdiff
2014
2016
2018
2020
2022
Time

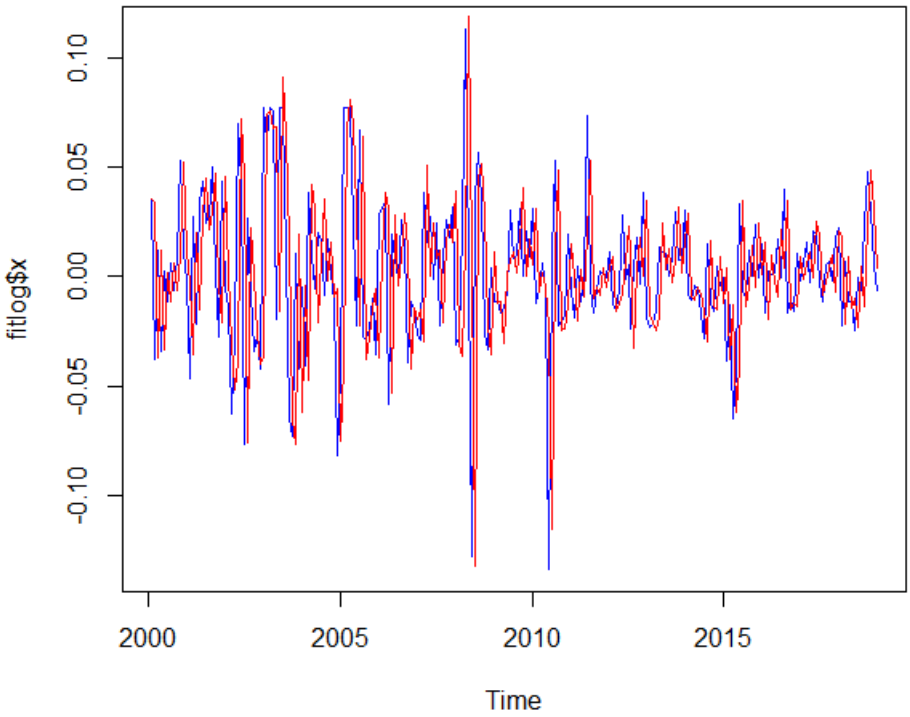
acf(diffst)


Series diffst
ACF
1.0
0.5
0.0
-0.5
0.0
0.5
1.0
1.5
Lag

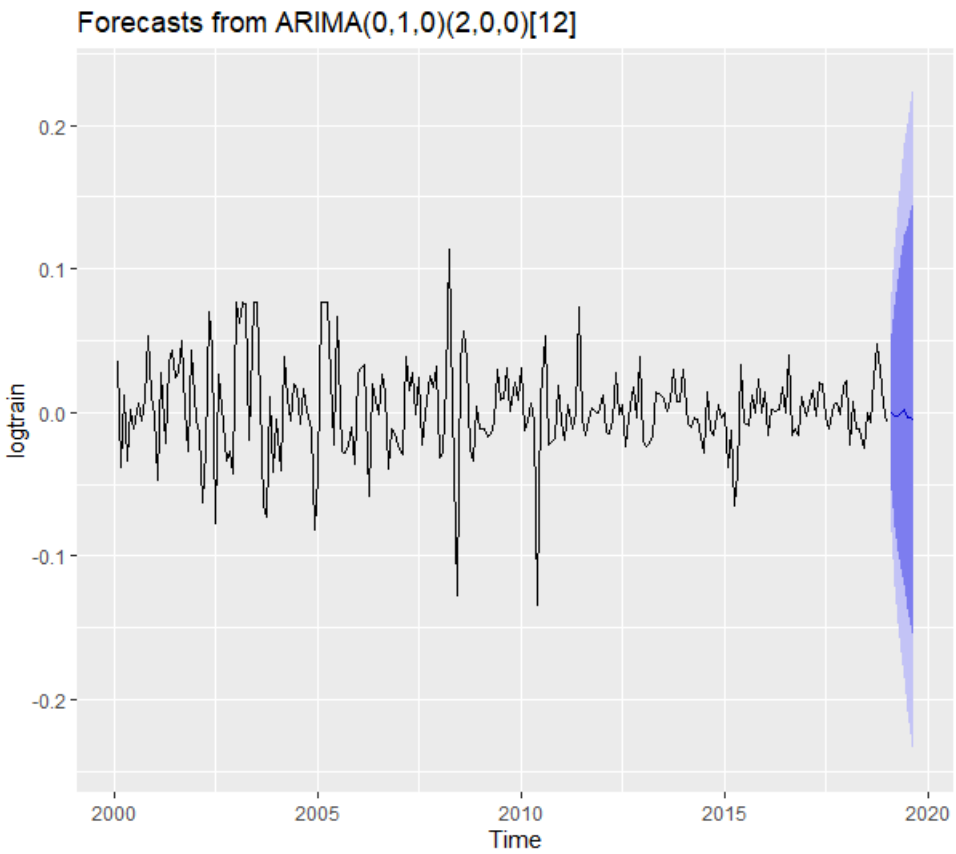
pacf(diffst)


Series diffst
Partial ACF
0.2
0.0
-0.2
-0.4
-0.6
0.5
1.0
1.5
Lag

```
#arima fit for log data
plot(fitlog$x,col="blue")
lines(fitlog$fitted,col="red",main="Actucal VS forecast with log trandormation")
```




```
logforecast= forecast(fitlog,h=8)
autoplot(logforecast)
```



accuracy(fitlog)

ME RMSE MAE MPE MAPE MASE ACF1

Training set -0.0001748147 0.04218236 0.02991583 -Inf Inf 0.843532 -0.3276072

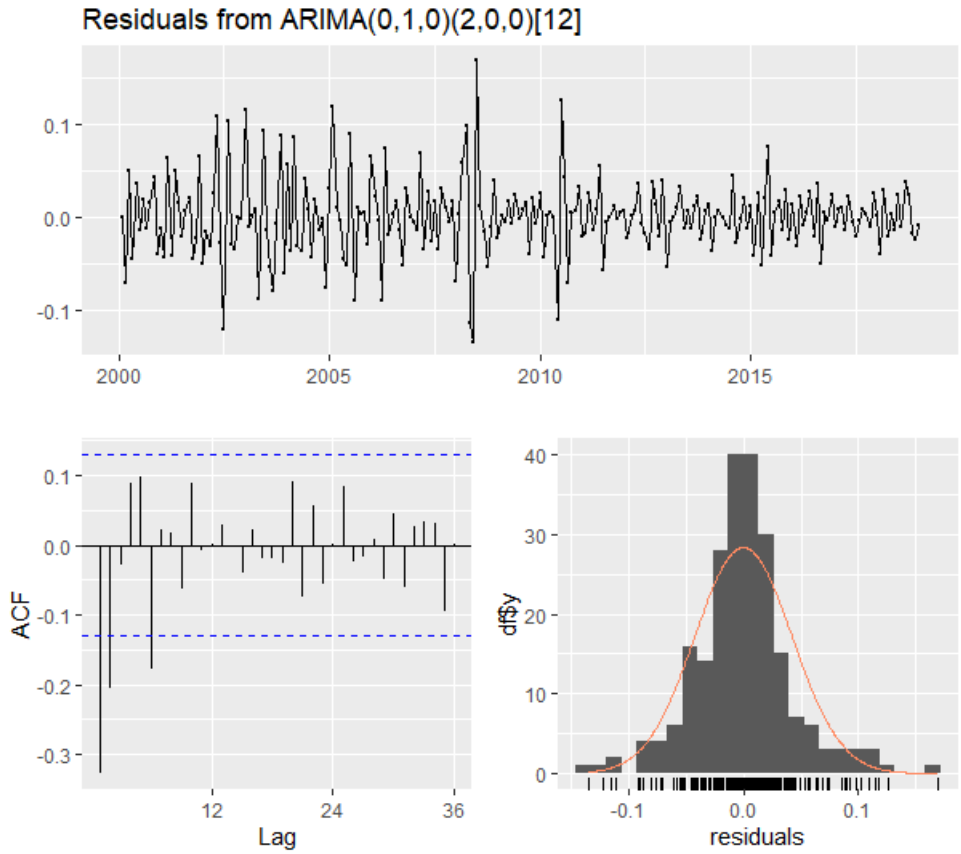
checkresiduals(fitlog)

Ljung-Box test

data: Residuals from ARIMA(0,1,0)(2,0,0)[12]

Q* = 55.638, df = 22, p-value = 9.635e-05

Model df: 2. Total lags used: 24



VGG16 With XGBoost

-*- coding: utf-8 -*-

"""

Created on Thu Aug 11 15:14:23 2022

@author: LAXMINARAYANRO

"""

```
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
import os
import seaborn as sns
from tensorflow.keras.applications.vgg16 import VGG16



# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 256
#Channel=3  #Resize images
```

```
#lists
train_images = []
train_labels = []

for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)
```

```
# test/validation into lists


test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

```
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded


# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0


#Load model
VGG_model = VGG16(weights='imagenet', include_top=False,
input_shape=(SIZE, SIZE, 3))

#Make loaded layers as non-trainable.
for layer in VGG_model.layers:
    layer.trainable = False
```

VGG_model.summary()

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 256, 256, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 256, 256, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 128, 128, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 128, 128, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 128, 128, 128)     147584
```

_____

block2_pool (MaxPooling2D) (None, 64, 64, 128) 0

_____

block3_conv1 (Conv2D) (None, 64, 64, 256) 295168

_____

block3_conv2 (Conv2D) (None, 64, 64, 256) 590080

_____

block3_conv3 (Conv2D) (None, 64, 64, 256) 590080

_____

block3_pool (MaxPooling2D) (None, 32, 32, 256) 0

_____

block4_conv1 (Conv2D) (None, 32, 32, 512) 1180160

_____

block4_conv2 (Conv2D) (None, 32, 32, 512) 2359808

_____

block4_conv3 (Conv2D) (None, 32, 32, 512) 2359808

_____

_____

block4_pool (MaxPooling2D) (None, 16, 16, 512) 0

_____

_____

block5_conv1 (Conv2D) (None, 16, 16, 512) 2359808

_____

_____

block5_conv2 (Conv2D) (None, 16, 16, 512) 2359808

_____

_____

block5_conv3 (Conv2D) (None, 16, 16, 512) 2359808

_____

_____

block5_pool (MaxPooling2D) (None, 8, 8, 512) 0

==========

==========

Total params: 14,714,688

Trainable params: 0

Non-trainable params: 14,714,688

_____

convolutional network for RF

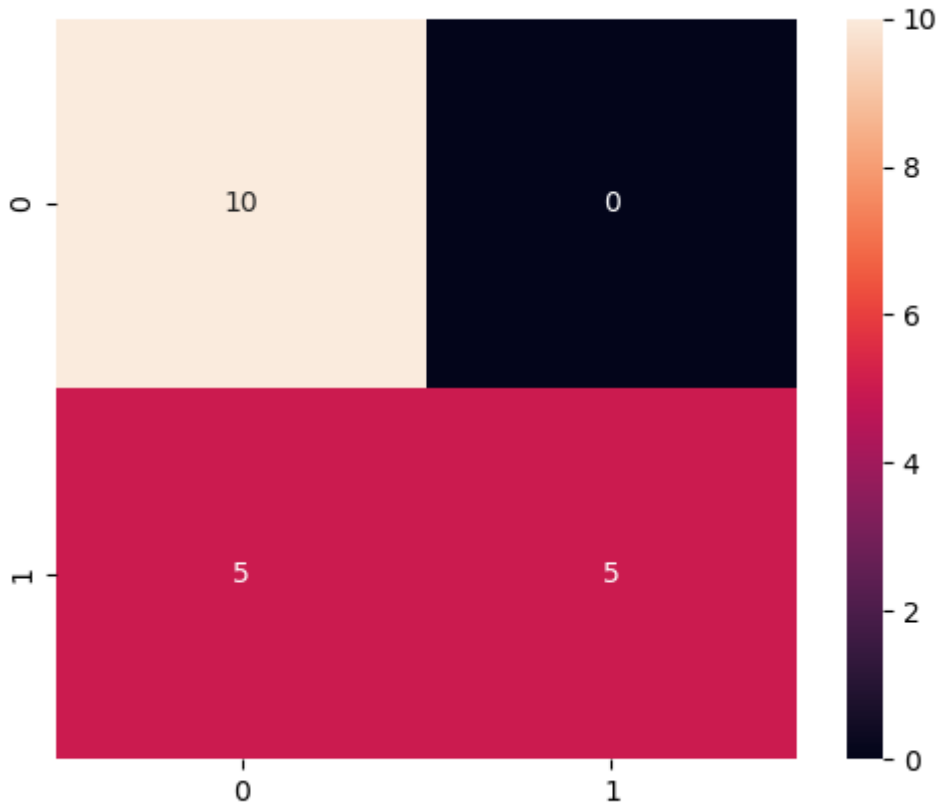
feature_extractor=VGG_model.predict(x_train)

```
features =
feature_extractor.reshape(feature_extractor.shape[0], -1)


X_for_training = features



#XGBOOST
import xgboost as xgb
model = xgb.XGBClassifier()
model.fit(X_for_training, y_train)
GBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
            colsample_bynode=1, colsample_bytree=1,
enable_categorical=False,
            gamma=0, gpu_id=-1, importance_type=None,
            interaction_constraints='',
learning_rate=0.300000012,
            max_delta_step=0, max_depth=6,
min_child_weight=1, missing=nan,
            monotone_constraints='()', n_estimators=100,
n_jobs=4,
            num_parallel_tree=1, predictor='auto',
random_state=0,
```

```
            reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
            tree_method='exact', validate_parameters=1, verbosity=None)
#Send test data through same feature extractor process
X_test_feature = VGG_model.predict(x_test)
X_test_features = X_test_feature.reshape(X_test_feature.shape[0], -1)

# predict using RF model.
prediction = model.predict(X_test_features)
#Inverse le transform to get original label back.
prediction = le.inverse_transform(prediction)

#accuracy
from sklearn import metrics
#from sklearn.metrics import classification_report
#from sklearn.metrics import average_precision_score
print ("Accuracy = ", metrics.accuracy_score(test_labels, prediction))
#print (classification_report(test_labels, prediction))
#print ("Precision = ", average_precision_score('fish', 'phone'))
```

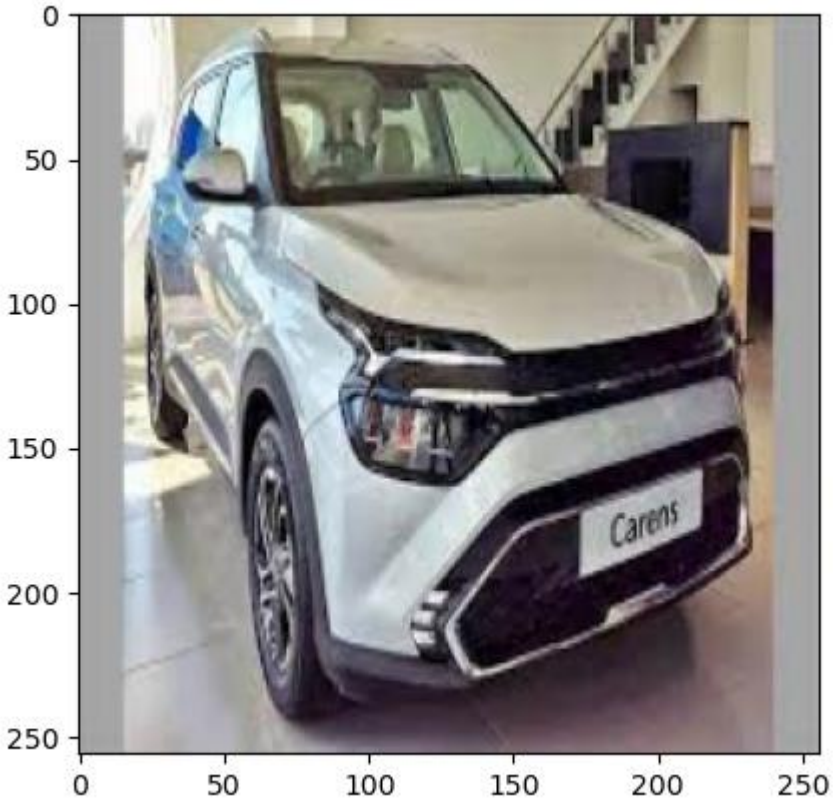
Accuracy = 0.75

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_labels, prediction)
#print(cm)
sns.heatmap(cm, annot=True)
```



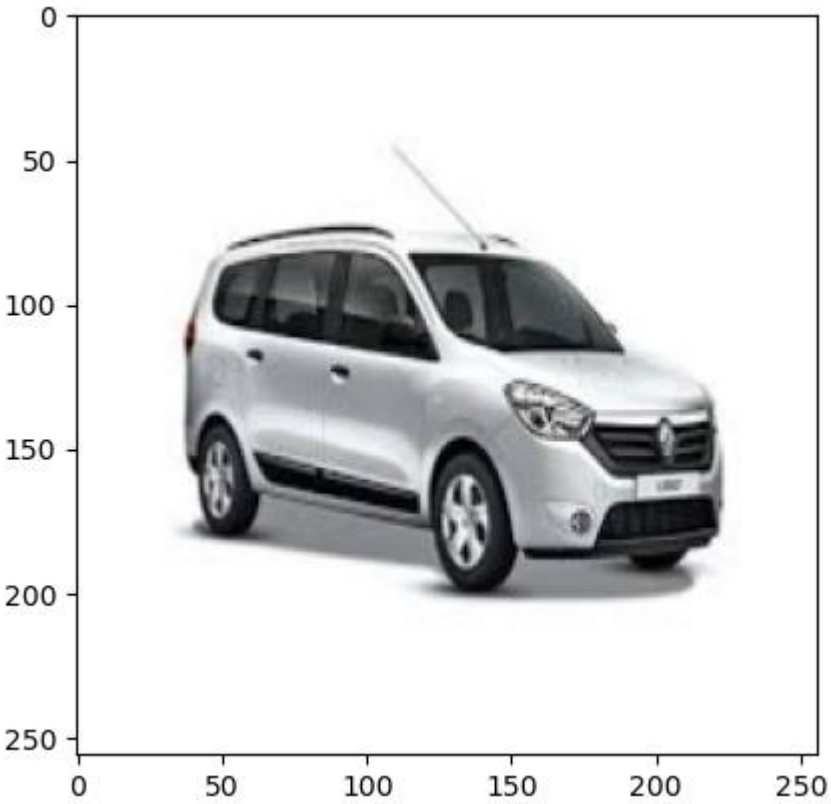

```
#Check results on a few select images
n=np.random.randint(0, x_test.shape[0])
```

```
img = x_test[n]

plt.imshow(img)

input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)

input_img_feature=VGG_model.predict(input_img)

input_img_features=input_img_feature.reshape(input_img_feature.shape[0], -1)

prediction = model.predict(input_img_features)[0]

prediction = le.inverse_transform([prediction])  #Reverse the label encoder to original name

print("The prediction for this image is: ", prediction)

print("The actual label for this image is: ", test_labels[n])
```



The prediction for this image is: ['Safe']

The actual label for this image is: Safe



The prediction for this image is: ['Unsafe']

The actual label for this image is: Unsafe

CNN

```
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 14 19:30:03 2022


@author: LAXMINARAYANRO
"""
```

```
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
#from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
import os
import seaborn as sns
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten, Dropout, Conv2D, Dense
)


print(os.listdir("E:/fffiiles/college pracs and projects/Project/Car"))

SIZE = 128
```

```
train_images = []
train_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

train_images = np.array(train_images)
train_labels = np.array(train_labels)


# test
test_images = []
test_labels = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)
```

```
#Split data into test and train datasets (already split but assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images, train_labels_encoded, test_images, test_labels_encoded

##################################################################
# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

#One hot encode y values for neural network.
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

##############################

activation = 'sigmoid'

feature_extractor = Sequential()
feature_extractor.add(Conv2D(32, 3, activation = activation, padding = 'same', input_shape = (SIZE, SIZE, 3)))
feature_extractor.add(BatchNormalization())
```

```
feature_extractor.add(Conv2D(32, 3, activation = activation, padding = 'same', kernel_initializer = 'he_uniform'))
feature_extractor.add(BatchNormalization())
feature_extractor.add(MaxPooling2D())

feature_extractor.add(Conv2D(64, 3, activation = activation, padding = 'same', kernel_initializer = 'he_uniform'))
feature_extractor.add(BatchNormalization())

feature_extractor.add(Conv2D(64, 3, activation = activation, padding = 'same', kernel_initializer = 'he_uniform'))
feature_extractor.add(BatchNormalization())
feature_extractor.add(MaxPooling2D())

feature_extractor.add(Flatten())

#Add layers for deep learning prediction
x = feature_extractor.output
x = Dense(128, activation = activation, kernel_initializer = 'he_uniform')(x)
prediction_layer = Dense(2, activation = 'softmax')(x)
```

```
# Make a new model combining both feature extractor and x
cnn_model = Model(inputs=feature_extractor.input, outputs=prediction_layer)
cnn_model.compile(optimizer='rmsprop',loss = 'categorical_crossentropy', metrics = ['accuracy'])
print(cnn_model.summary())
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_input (InputLayer)    [(None, 128, 128, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 128, 128, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 128, 32)      128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 32)      9248
_________________________________________________________________
```

batch_normalization_1 (Batch (None, 128, 128, 32) 128

_____

max_pooling2d (MaxPooling2D) (None, 64, 64, 32) 0

_____

conv2d_2 (Conv2D) (None, 64, 64, 64) 18496

_____

batch_normalization_2 (Batch (None, 64, 64, 64) 256

_____

conv2d_3 (Conv2D) (None, 64, 64, 64) 36928

_____

batch_normalization_3 (Batch (None, 64, 64, 64) 256

_____

max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64) 0

_____

flatten (Flatten) (None, 65536) 0

_____

```
dense (Dense)                (None, 128)               8388736
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 8,455,330
Trainable params: 8,454,946
Non-trainable params: 384
```

```
##########################################
#Train the CNN model
history = cnn_model.fit(x_train, y_train_one_hot, epochs=50, validation_data = (x_test, y_test_one_hot))
```

```
Epoch 1/50
16/16 [==============================] - 43s 3s/step - loss: 2.4519 - accuracy: 0.6165 - val_loss: 2.7376 - val_accuracy: 0.5000
Epoch 2/50
16/16 [==============================] - 42s 3s/step - loss: 1.7722 - accuracy: 0.6124 - val_loss: 1.7730 - val_accuracy: 0.5000
Epoch 3/50
```

```
16/16 [==============================] - 41s 3s/step - loss: 1.1353 - accuracy: 0.6124 - val_loss: 1.0966 - val_accuracy: 0.5000

Epoch 4/50

16/16 [==============================] - 43s 3s/step - loss: 0.7626 - accuracy: 0.6104 - val_loss: 0.7769 - val_accuracy: 0.5000

Epoch 5/50

16/16 [==============================] - 43s 3s/step - loss: 0.7060 - accuracy: 0.5723 - val_loss: 0.7223 - val_accuracy: 0.5000

Epoch 6/50

16/16 [==============================] - 40s 3s/step - loss: 0.6678 - accuracy: 0.6124 - val_loss: 0.7104 - val_accuracy: 0.5000

Epoch 7/50

16/16 [==============================] - 49s 3s/step - loss: 0.6946 - accuracy: 0.6004 - val_loss: 0.7665 - val_accuracy: 0.5000

Epoch 8/50

16/16 [==============================] - 43s 3s/step - loss: 0.6536 - accuracy: 0.6225 - val_loss: 0.7106 - val_accuracy: 0.5000

Epoch 9/50
```

16/16 [==========] - 45s 3s/step - loss: 0.6606 - accuracy: 0.6124 - val_loss: 0.7171 - val_accuracy: 0.5000

Epoch 10/50

16/16 [==========] - 39s 2s/step - loss: 0.6444 - accuracy: 0.6426 - val_loss: 0.7021 - val_accuracy: 0.5000

Epoch 11/50

16/16 [==========] - 41s 3s/step - loss: 0.6563 - accuracy: 0.6386 - val_loss: 0.7376 - val_accuracy: 0.5000

Epoch 12/50

16/16 [==========] - 47s 3s/step - loss: 0.6308 - accuracy: 0.6667 - val_loss: 0.7318 - val_accuracy: 0.5000

Epoch 13/50

16/16 [==========] - 38s 2s/step - loss: 0.6136 - accuracy: 0.6727 - val_loss: 0.8044 - val_accuracy: 0.5000

Epoch 14/50

16/16 [==========] - 36s 2s/step - loss: 0.6116 - accuracy: 0.6747 - val_loss: 0.7830 - val_accuracy: 0.5000

Epoch 15/50

```
16/16 [==============================] - 40s 3s/step - loss: 0.6188 - accuracy: 0.6707 - val_loss: 0.8597 - val_accuracy: 0.5000

Epoch 16/50

16/16 [==============================] - 38s 2s/step - loss: 0.6069 - accuracy: 0.6606 - val_loss: 0.6725 - val_accuracy: 0.6000

Epoch 17/50

16/16 [==============================] - 40s 2s/step - loss: 0.5823 - accuracy: 0.6847 - val_loss: 0.7239 - val_accuracy: 0.6000

Epoch 18/50

16/16 [==============================] - 38s 2s/step - loss: 0.5654 - accuracy: 0.7209 - val_loss: 0.7651 - val_accuracy: 0.5500

Epoch 19/50

16/16 [==============================] - 40s 3s/step - loss: 0.5523 - accuracy: 0.7209 - val_loss: 0.6413 - val_accuracy: 0.6000

Epoch 20/50

16/16 [==============================] - 43s 3s/step - loss: 0.5422 - accuracy: 0.7289 - val_loss: 0.8310 - val_accuracy: 0.5000

Epoch 21/50
```

```
16/16 [==============================] - 41s 3s/step - loss: 0.5495 - accuracy: 0.7309 - val_loss: 0.6574 - val_accuracy: 0.6500

Epoch 22/50

16/16 [==============================] - 41s 3s/step - loss: 0.5276 - accuracy: 0.7510 - val_loss: 0.7374 - val_accuracy: 0.5500

Epoch 23/50

16/16 [==============================] - 45s 3s/step - loss: 0.5122 - accuracy: 0.7631 - val_loss: 0.7876 - val_accuracy: 0.5000

Epoch 24/50

16/16 [==============================] - 40s 3s/step - loss: 0.5252 - accuracy: 0.7651 - val_loss: 0.8318 - val_accuracy: 0.5000

Epoch 25/50

16/16 [==============================] - 40s 2s/step - loss: 0.4869 - accuracy: 0.7651 - val_loss: 0.7637 - val_accuracy: 0.5000

Epoch 26/50

16/16 [==============================] - 37s 2s/step - loss: 0.5050 - accuracy: 0.7811 - val_loss: 0.7345 - val_accuracy: 0.5000

Epoch 27/50
```

16/16 [==========] - 41s 3s/step - loss: 0.4902 - accuracy: 0.7490 - val_loss: 0.7086 - val_accuracy: 0.5500

Epoch 28/50

16/16 [==========] - 39s 2s/step - loss: 0.4478 - accuracy: 0.8012 - val_loss: 0.6788 - val_accuracy: 0.6500

Epoch 29/50

16/16 [==========] - 39s 2s/step - loss: 0.4418 - accuracy: 0.8193 - val_loss: 0.7535 - val_accuracy: 0.6000

Epoch 30/50

16/16 [==========] - 40s 2s/step - loss: 0.4516 - accuracy: 0.8032 - val_loss: 0.5901 - val_accuracy: 0.5500

Epoch 31/50

16/16 [==========] - 39s 2s/step - loss: 0.4575 - accuracy: 0.8193 - val_loss: 0.6662 - val_accuracy: 0.6500

Epoch 32/50

16/16 [==========] - 44s 3s/step - loss: 0.4260 - accuracy: 0.8173 - val_loss: 0.8218 - val_accuracy: 0.5000

Epoch 33/50

16/16 [==========] - 40s 3s/step - loss: 0.4066 - accuracy: 0.8353 - val_loss: 0.8072 - val_accuracy: 0.5500

Epoch 34/50

16/16 [==========] - 39s 2s/step - loss: 0.4110 - accuracy: 0.8193 - val_loss: 0.7160 - val_accuracy: 0.6500

Epoch 35/50

16/16 [==========] - 37s 2s/step - loss: 0.3707 - accuracy: 0.8414 - val_loss: 0.7861 - val_accuracy: 0.6000

Epoch 36/50

16/16 [==========] - 38s 2s/step - loss: 0.3587 - accuracy: 0.8454 - val_loss: 0.6788 - val_accuracy: 0.7000

Epoch 37/50

16/16 [==========] - 40s 2s/step - loss: 0.3526 - accuracy: 0.8534 - val_loss: 0.6974 - val_accuracy: 0.5500

Epoch 38/50

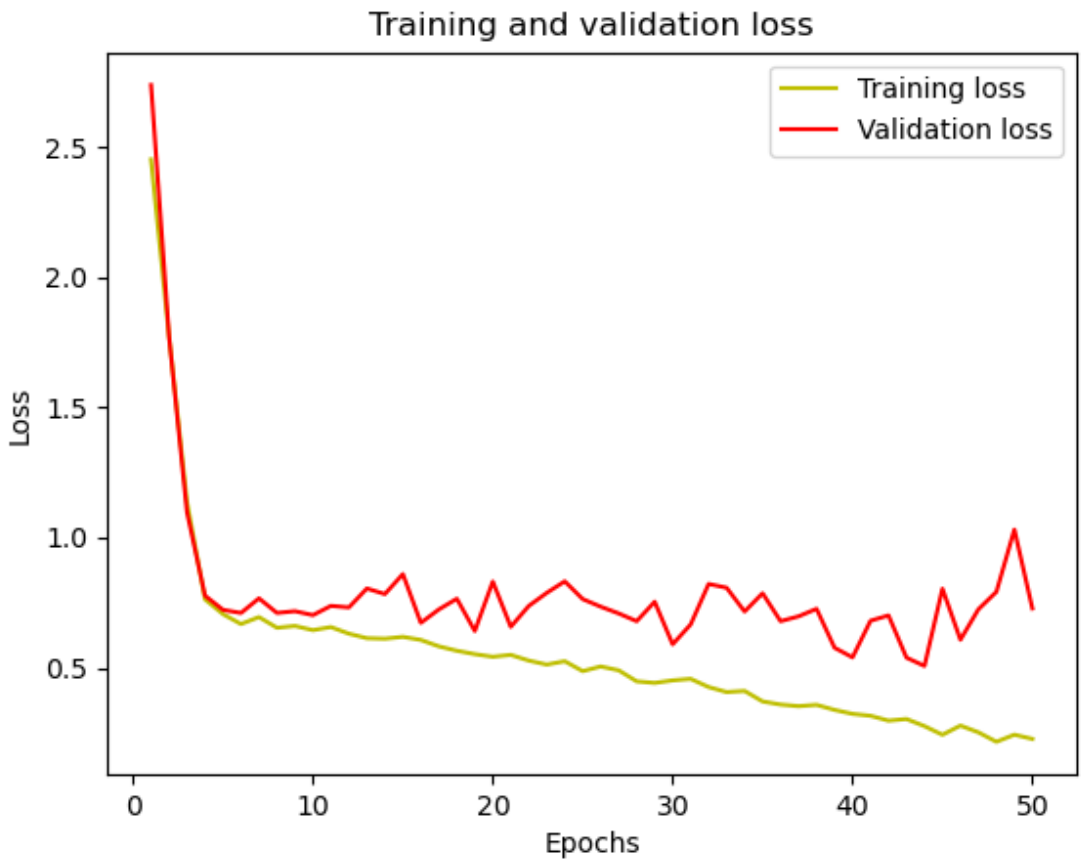
16/16 [==========] - 38s 2s/step - loss: 0.3576 - accuracy: 0.8655 - val_loss: 0.7263 - val_accuracy: 0.5500

Epoch 39/50

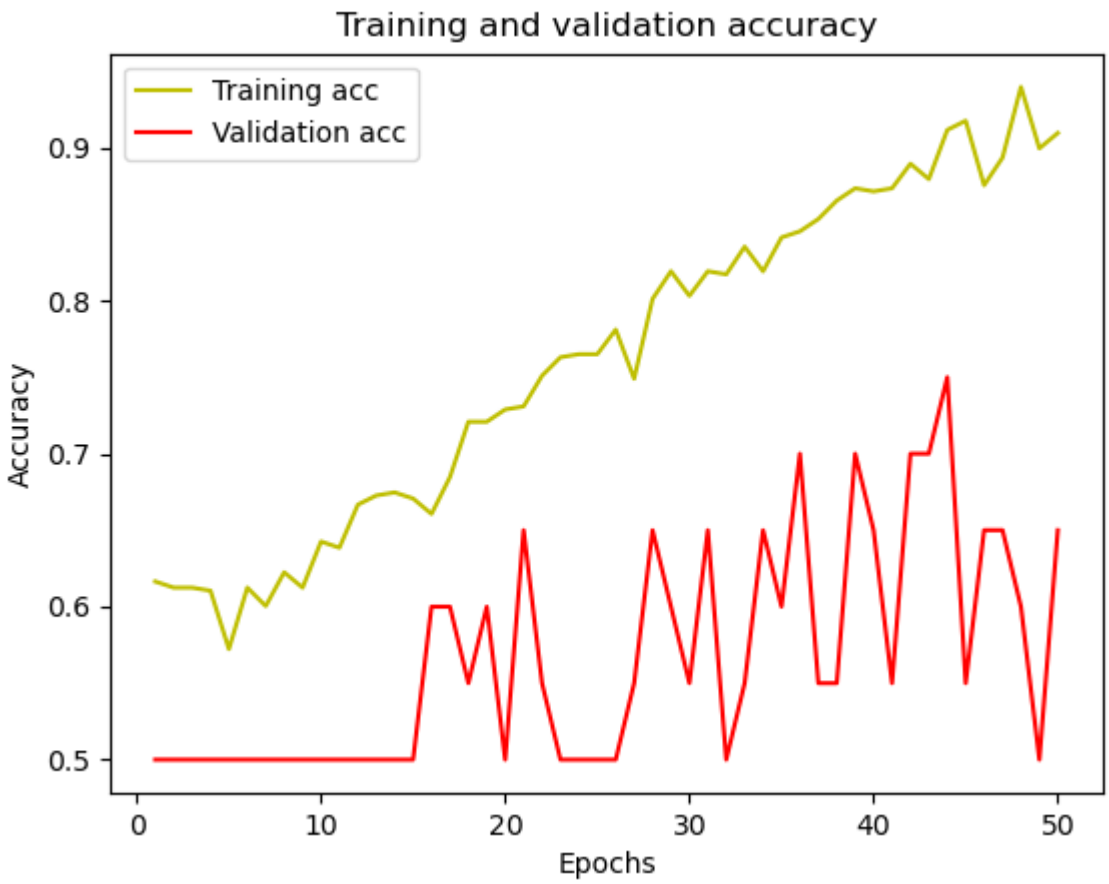
```
16/16 [==============================] - 40s 2s/step - loss: 0.3382 - accuracy: 0.8735 - val_loss: 0.5769 - val_accuracy: 0.7000

Epoch 40/50

16/16 [==============================] - 39s 2s/step - loss: 0.3230 - accuracy: 0.8715 - val_loss: 0.5399 - val_accuracy: 0.6500

Epoch 41/50

16/16 [==============================] - 35s 2s/step - loss: 0.3156 - accuracy: 0.8735 - val_loss: 0.6805 - val_accuracy: 0.5500

Epoch 42/50

16/16 [==============================] - 40s 2s/step - loss: 0.2970 - accuracy: 0.8896 - val_loss: 0.7009 - val_accuracy: 0.7000

Epoch 43/50

16/16 [==============================] - 38s 2s/step - loss: 0.3032 - accuracy: 0.8795 - val_loss: 0.5389 - val_accuracy: 0.7000

Epoch 44/50

16/16 [==============================] - 40s 2s/step - loss: 0.2760 - accuracy: 0.9116 - val_loss: 0.5069 - val_accuracy: 0.7500

Epoch 45/50
```

```
16/16 [==============================] - 39s 2s/step - loss: 0.2426 - accuracy: 0.9177 - val_loss: 0.8036 - val_accuracy: 0.5500

Epoch 46/50

16/16 [==============================] - 35s 2s/step - loss: 0.2781 - accuracy: 0.8755 - val_loss: 0.6076 - val_accuracy: 0.6500

Epoch 47/50

16/16 [==============================] - 38s 2s/step - loss: 0.2521 - accuracy: 0.8936 - val_loss: 0.7246 - val_accuracy: 0.6500

Epoch 48/50

16/16 [==============================] - 42s 3s/step - loss: 0.2158 - accuracy: 0.9398 - val_loss: 0.7917 - val_accuracy: 0.6000

Epoch 49/50

16/16 [==============================] - 36s 2s/step - loss: 0.2430 - accuracy: 0.8996 - val_loss: 1.0306 - val_accuracy: 0.5000

Epoch 50/50

16/16 [==============================] - 37s 2s/step - loss: 0.2268 - accuracy: 0.9096 - val_loss: 0.7278 - val_accuracy: 0.6500
```

```
#plot the training and validation accuracy and loss at each epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



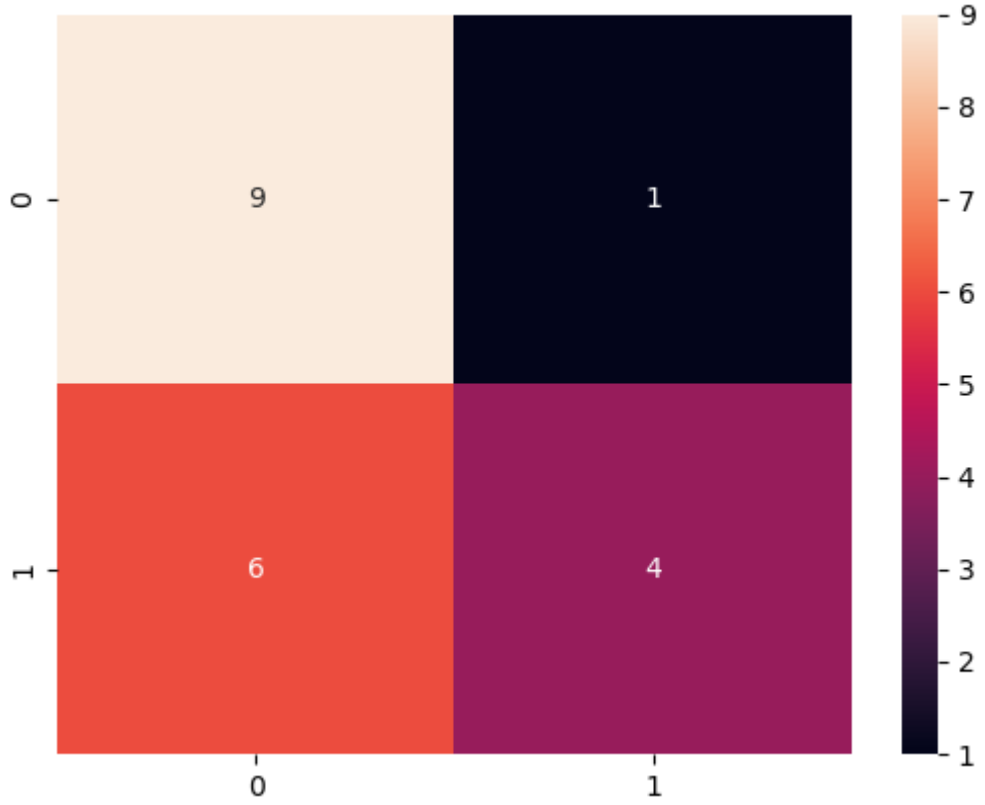

```
acc = history.history['acc']
val_acc = history.history['val_acc']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
prediction_NN = cnn_model.predict(x_test)
prediction_NN = np.argmax(prediction_NN, axis=-1)
prediction_NN = le.inverse_transform(prediction_NN)
#Confusion Matrix - verify accuracy of each class
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_labels, prediction_NN)
print(cm)
```

```
[[9 1]
 [6 4]]
```

sns.heatmap(cm, annot=True)



#Check results on a few select images

n=9 #Select the index of image to be loaded for testing

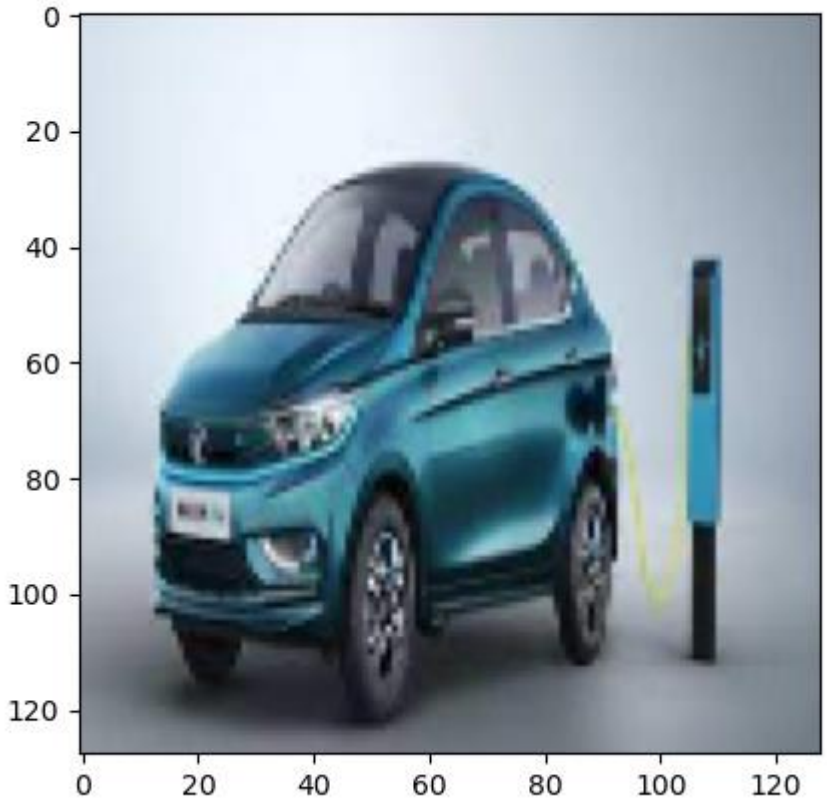
img = x_test[n]

plt.imshow(img)

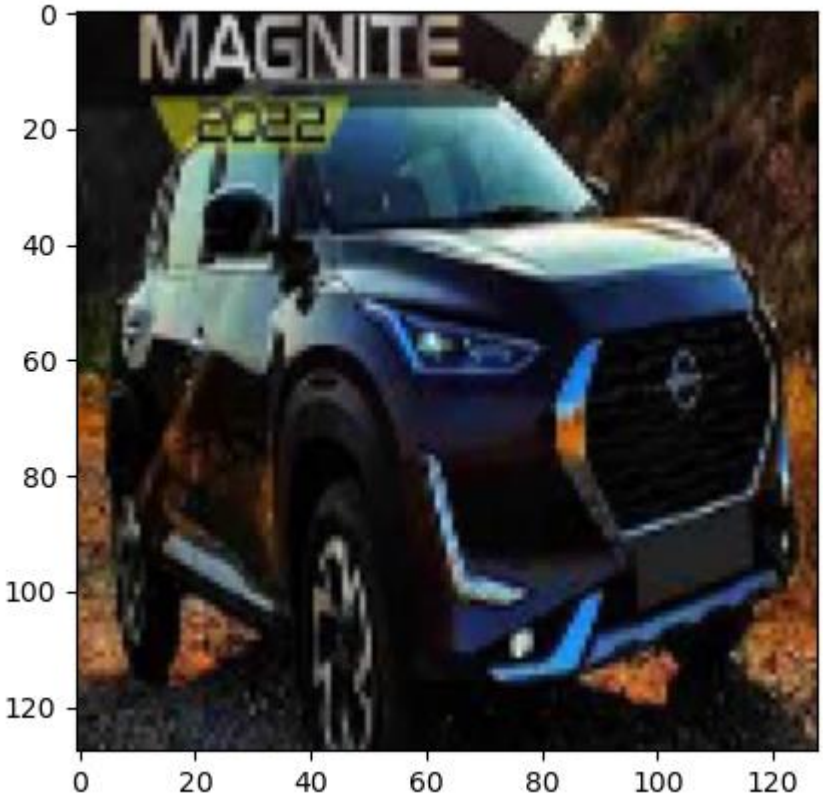
input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)

prediction = np.argmax(cnn_model.predict(input_img)) #argmax to convert categorical back to original

```
prediction = le.inverse_transform([prediction])  #Reverse the label encoder to original name
print("The prediction for this image is: ", prediction)
print("The actual label for this image is: ", test_labels[n])
```




```
The prediction for this image is:  ['Safe']
The actual label for this image is:  Safe
```



The prediction for this image is: ['Unsafe']

The actual label for this image is: Safe

CNN With Random Forest

-*- coding: utf-8 -*-

"""

Created on Sun Aug 14 19:30:03 2022

@author: LAXMINARAYANRO

```
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
#from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
import os
import seaborn as sns
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten, Dropout, Conv2D, Dense
)


print(os.listdir("E:/fffiiles/college pracs and projects/Project/Car"))

SIZE = 128
```

```
train_images = []
train_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

train_images = np.array(train_images)
train_labels = np.array(train_labels)


# test
test_images = []
```

```
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)
```

```
#Split data into test and train datasets (already split but assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images, train_labels_encoded, test_images, test_labels_encoded

##################################################################
# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0



##############################

activation = 'sigmoid'

feature_extractor = Sequential()
feature_extractor.add(Conv2D(32, 3, activation = activation, padding = 'same', input_shape = (SIZE, SIZE, 3)))
feature_extractor.add(BatchNormalization())

feature_extractor.add(Conv2D(32, 3, activation = activation, padding = 'same', kernel_initializer = 'he_uniform'))
```

```
feature_extractor.add(BatchNormalization())
feature_extractor.add(MaxPooling2D())

feature_extractor.add(Conv2D(64, 3, activation = activation,
padding = 'same', kernel_initializer = 'he_uniform'))
feature_extractor.add(BatchNormalization())

feature_extractor.add(Conv2D(64, 3, activation = activation,
padding = 'same', kernel_initializer = 'he_uniform'))
feature_extractor.add(BatchNormalization())
feature_extractor.add(MaxPooling2D())

feature_extractor.add(Flatten())



#Now, let us use features from convolutional network for RF
X_for_RF = feature_extractor.predict(x_train) #This is out X
input to RF

#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier
```

```
RF_model = RandomForestClassifier(n_estimators = 50,
random_state = 42)

# Train the model on training data

RF_model.fit(X_for_RF, y_train) #For sklearn no one hot
encoding


#Send test data through same feature extractor process

X_test_feature = feature_extractor.predict(x_test)

#Now predict using the trained RF model.

prediction_RF = RF_model.predict(X_test_feature)

#Inverse le transform to get original label back.

prediction_RF = le.inverse_transform(prediction_RF)


#Print overall accuracy

from sklearn import metrics

print ("Accuracy = ", metrics.accuracy_score(test_labels,
prediction_RF))
```

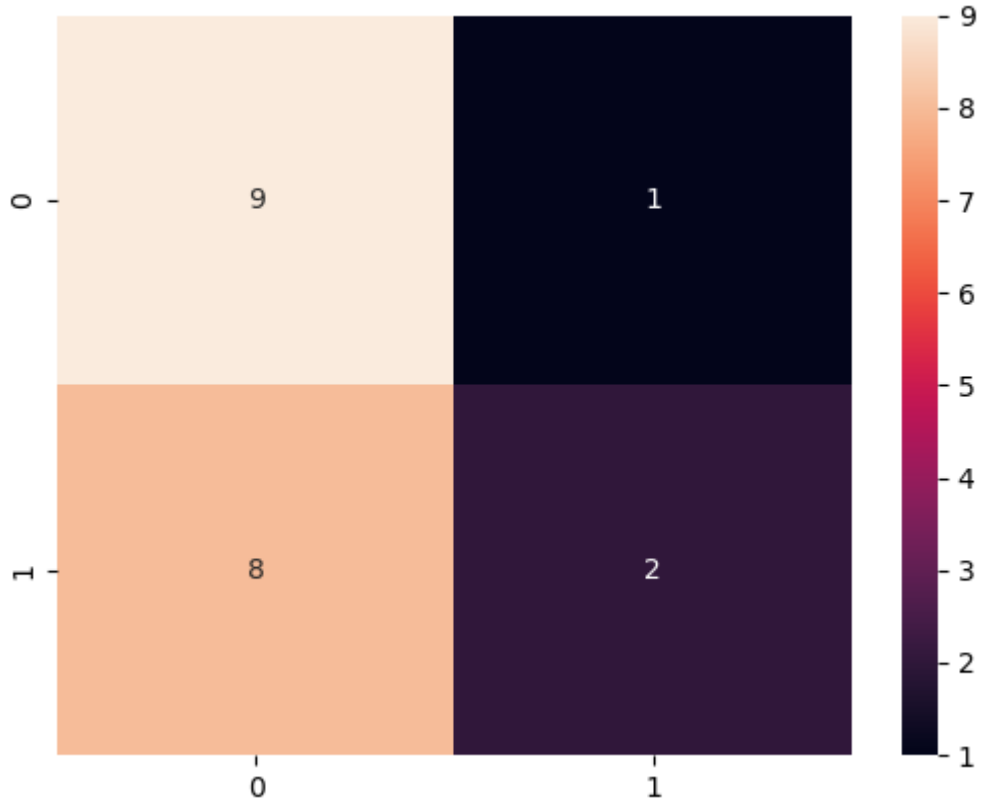
```
Accuracy =  0.55
```

```
#Confusion Matrix - verify accuracy of each class
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_labels, prediction_RF)
#print(cm)
```

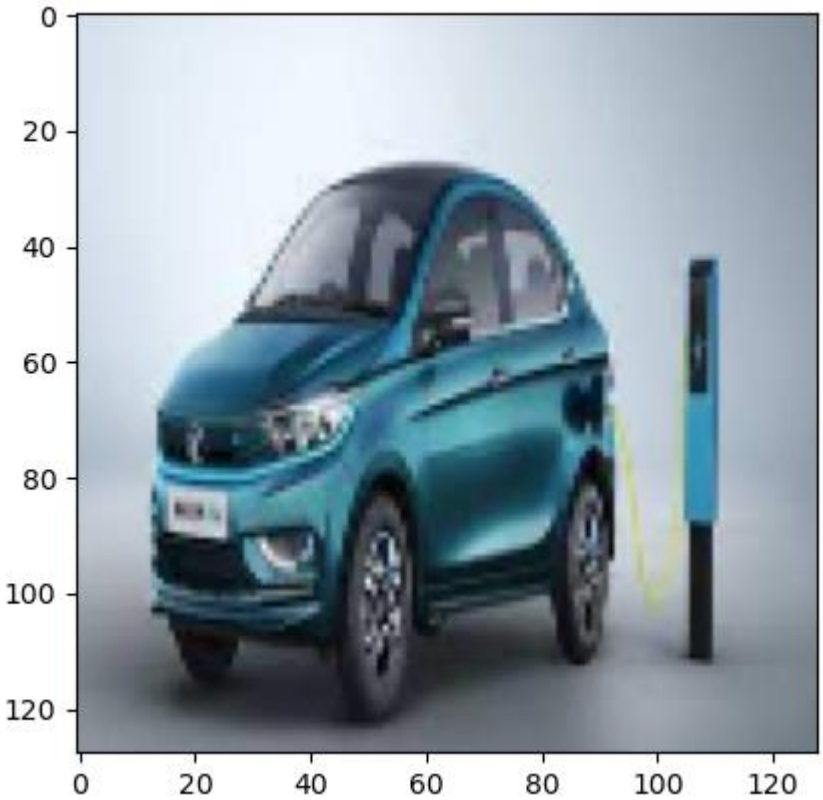
<AxesSubplot:>

```
sns.heatmap(cm, annot=True)
```



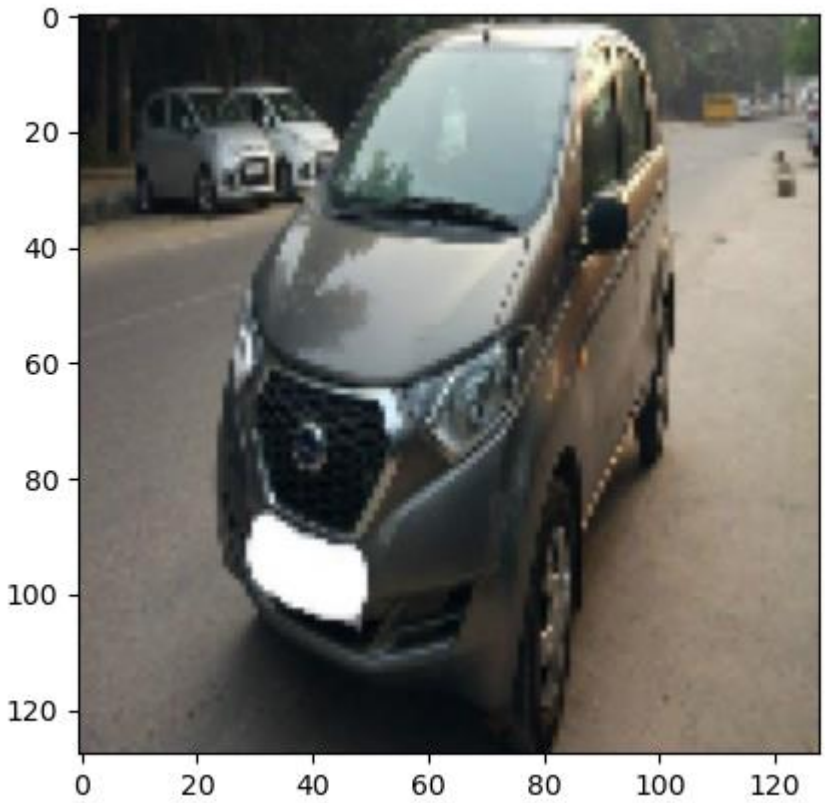

```
#Check results on a few select images
#n=5
n=9 #Select the index of image to be loaded for testing
img = x_test[n]
plt.imshow(img)
```

```
input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)

input_img_features=feature_extractor.predict(input_img)

prediction_RF = RF_model.predict(input_img_features)[0]

prediction_RF = le.inverse_transform([prediction_RF]) #Reverse the label encoder to original name

print("The prediction for this image is: ", prediction_RF)

print("The actual label for this image is: ", test_labels[n])
```



The prediction for this image is: ['Safe']

The actual label for this image is: Safe



The prediction for this image is: ['Safe']

The actual label for this image is: Unsafe

VGG16 With Random Forest

-*- coding: utf-8 -*-

"""

```
Created on Tue Aug 16 12:57:30 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
#from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
import os
import seaborn as sns
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.vgg16 import VGG16
```

```
# Read input images and assign labels based on folder names
print(os.listdir("E:/fffiiles/college pracs and projects/Project/Car"))

SIZE = 256  #Resize images

#Capture training data and labels into respective lists
train_images = []
train_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)
```

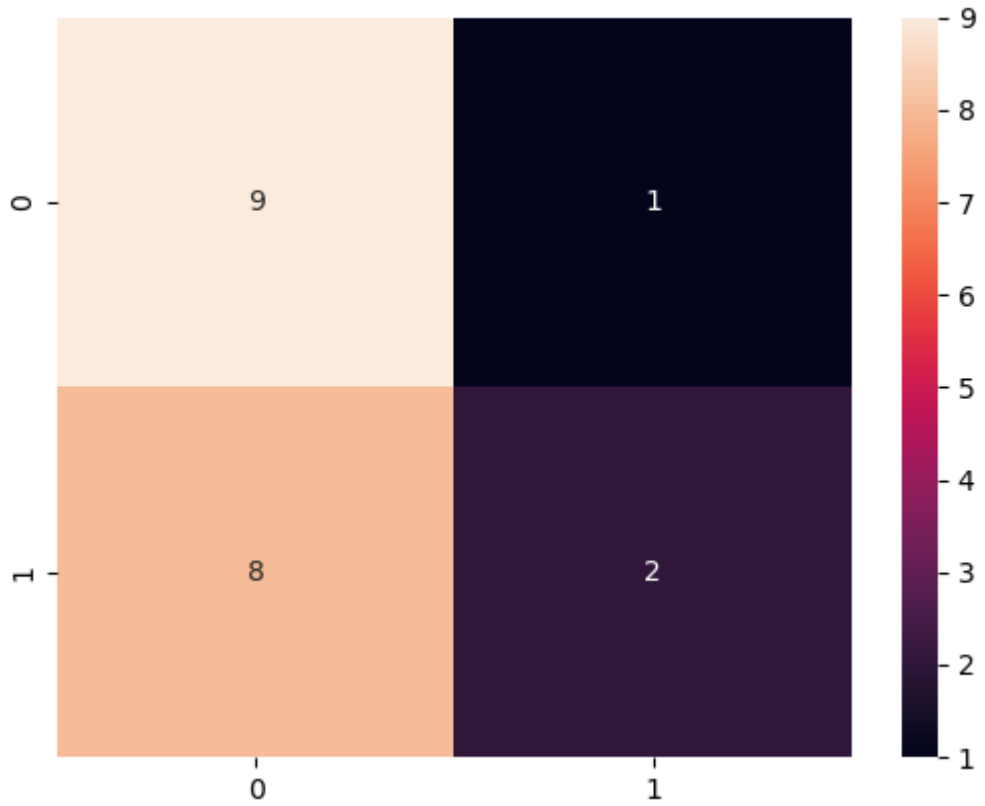
```
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data into test and train datasets (already split but
assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded

##################################################
#################
# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

#One hot encode y values for neural network.
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

##############################
```

```
#Load model wothout classifier/fully connected layers
VGG_model = VGG16(weights='imagenet', include_top=False,
input_shape=(SIZE, SIZE, 3))

#Make loaded layers as non-trainable. This is important as
we want to work with pre-trained weights
for layer in VGG_model.layers:
    layer.trainable = False

VGG_model.summary()  #Trainable parameters will be 0



#Now, let us use features from convolutional network for RF
feature_extractor=VGG_model.predict(x_train)

features =
feature_extractor.reshape(feature_extractor.shape[0], -1)

X_for_RF = features #This is our X input to RF

#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier
```

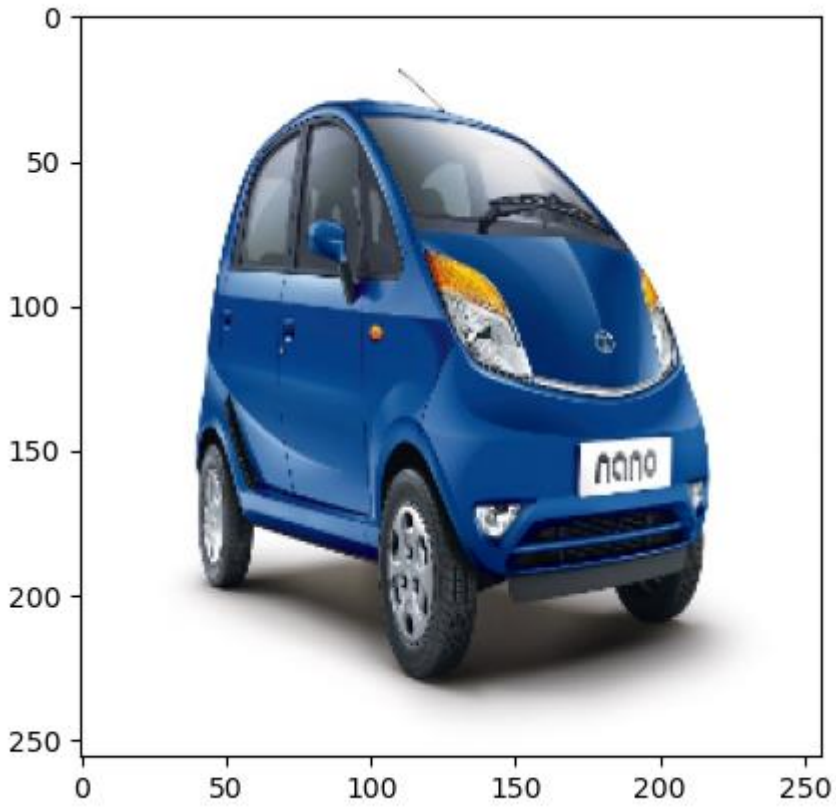
```
RF_model = RandomForestClassifier(n_estimators = 50,
random_state = 42)

# Train the model on training data
RF_model.fit(X_for_RF, y_train) #For sklearn no one hot
encoding

#Send test data through same feature extractor process
X_test_feature = VGG_model.predict(x_test)
X_test_features =
X_test_feature.reshape(X_test_feature.shape[0], -1)

#Now predict using the trained RF model.
prediction_RF = RF_model.predict(X_test_features)
#Inverse le transform to get original label back.
prediction_RF = le.inverse_transform(prediction_RF)

#Print overall accuracy
from sklearn import metrics
print ("Accuracy = ", metrics.accuracy_score(test_labels,
prediction_RF))
Accuracy =  0.55
#Confusion Matrix - verify accuracy of each class
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_labels, prediction_RF)
#print(cm)
sns.heatmap(cm, annot=True)
```



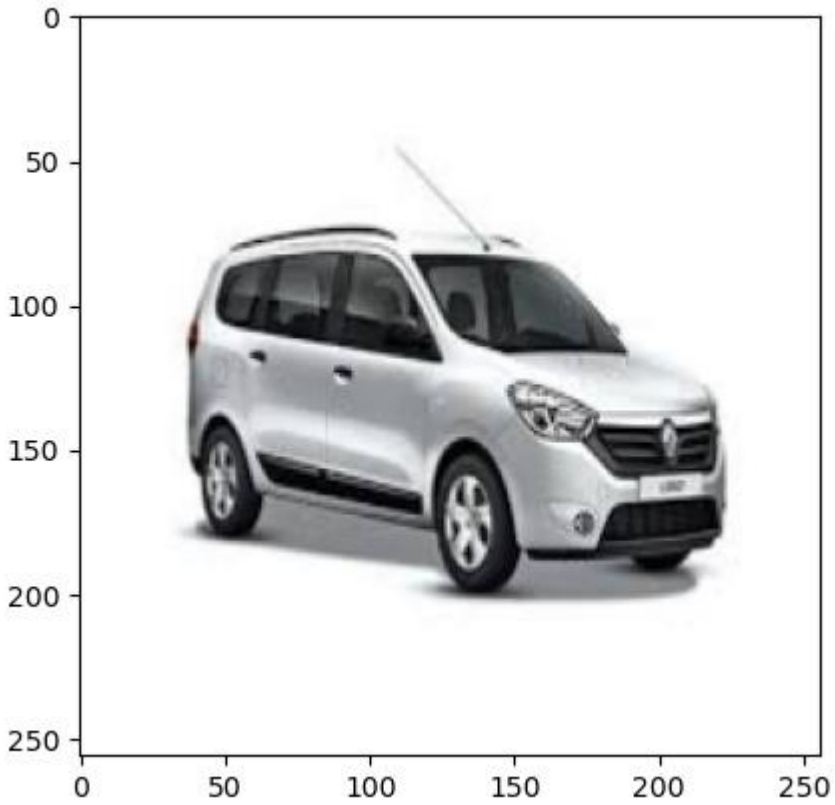

```
#Check results on a few select images
n=np.random.randint(0, x_test.shape[0])
img = x_test[n]
plt.imshow(img)
input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)
```

```
input_img_feature=VGG_model.predict(input_img)

input_img_features=input_img_feature.reshape(input_img_feature.shape[0], -1)

prediction_RF = RF_model.predict(input_img_features)[0]

prediction_RF = le.inverse_transform([prediction_RF]) #Reverse the label encoder to original name

print("The prediction for this image is: ", prediction_RF)

print("The actual label for this image is: ", test_labels[n])
```



The prediction for this image is: ['Unsafe']

The actual label for this image is: Unsafe



The prediction for this image is: ['Safe']

The actual label for this image is: Unsafe

Resnet50 with Adaboost

-*- coding: utf-8 -*-

"""

Created on Wed Aug 17 19:07:29 2022

@author: LAXMINARAYANRO

```
"""

# -*- coding: utf-8 -*-
"""
Created on Tue Aug 16 12:57:30 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
#from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
import os
import seaborn as sns
from tensorflow.keras.layers import (
```

```
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.resnet50 import
ResNet50



# Read input images and assign labels based on folder names
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))


SIZE = 256  #Resize images


#Capture training data and labels into respective lists
train_images = []
train_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
```

```
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
```

```
    test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data into test and train datasets (already split but
assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded

##################################################################
# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
#One hot encode y values for neural network.
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

##############################
#Load model wothout classifier/fully connected layers
Resnet50_model = ResNet50(weights='imagenet',
include_top=False, input_shape=(SIZE, SIZE, 3))

#Make loaded layers as non-trainable. This is important as
we want to work with pre-trained weights
for layer in Resnet50_model.layers:
    layer.trainable = False

Resnet50_model.summary()  #Trainable parameters will be 0


#Now, let us use features from convolutional network for RF
feature_extractor=Resnet50_model.predict(x_train)

features =
feature_extractor.reshape(feature_extractor.shape[0], -1)
```

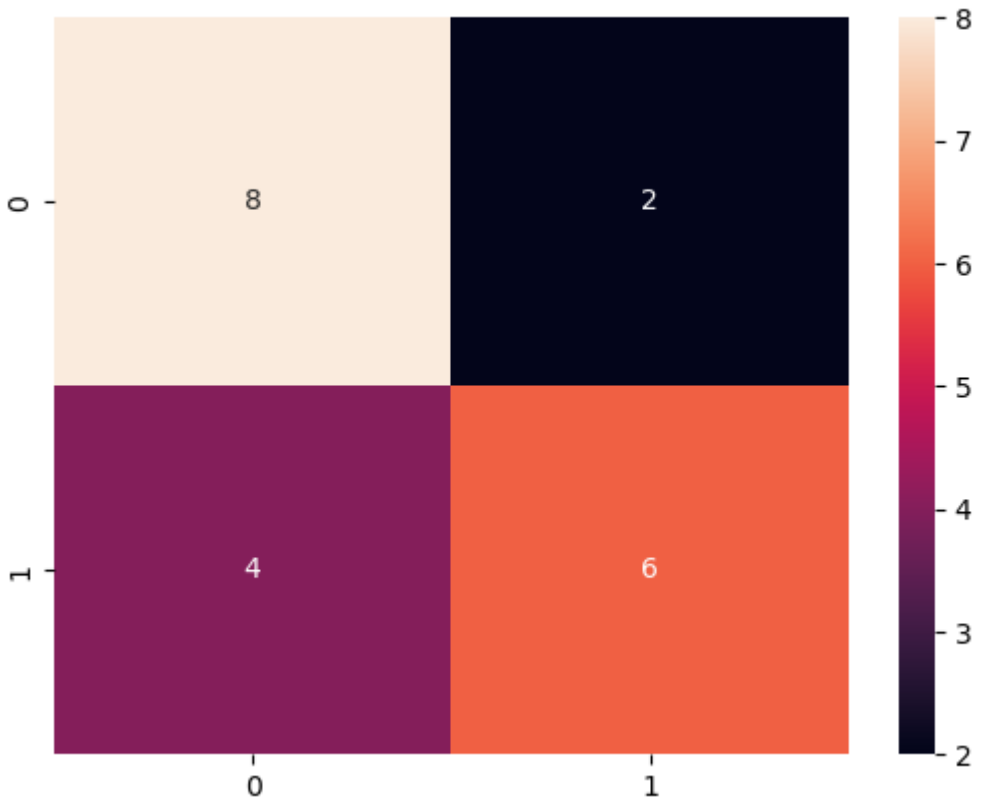
```
X_for_AD = features #This is our X input to RF

#RANDOM FOREST
from sklearn.ensemble import AdaBoostClassifier
AD_model = AdaBoostClassifier(n_estimators = 50,
random_state = 42)

# Train the model on training data
AD_model.fit(X_for_AD, y_train) #For sklearn no one hot
encoding

#Send test data through same feature extractor process
X_test_feature = Resnet50_model.predict(x_test)
X_test_features =
X_test_feature.reshape(X_test_feature.shape[0], -1)

#Now predict using the trained RF model.
prediction_AD = AD_model.predict(X_test_features)
#Inverse le transform to get original label back.
prediction_AD = le.inverse_transform(prediction_AD)

#Print overall accuracy
```

```
from sklearn import metrics

print ("Accuracy = ", metrics.accuracy_score(test_labels, prediction_AD))
```

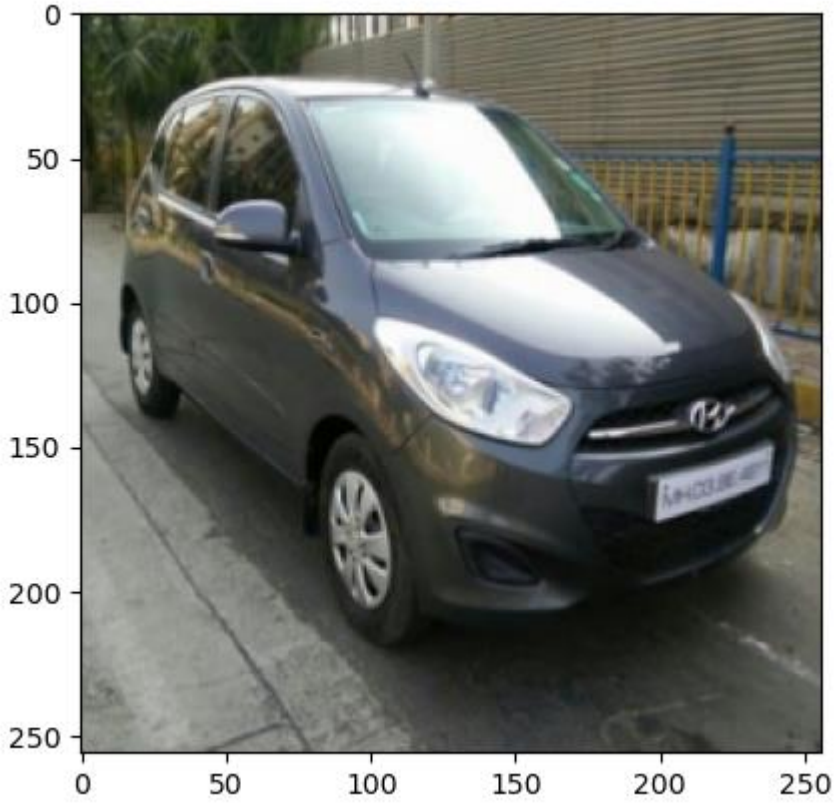
Accuracy = 0.7

```
#Confusion Matrix - verify accuracy of each class

from sklearn.metrics import confusion_matrix


cm = confusion_matrix(test_labels, prediction_AD)

#print(cm)

sns.heatmap(cm, annot=True)
```

<AxesSubplot:>

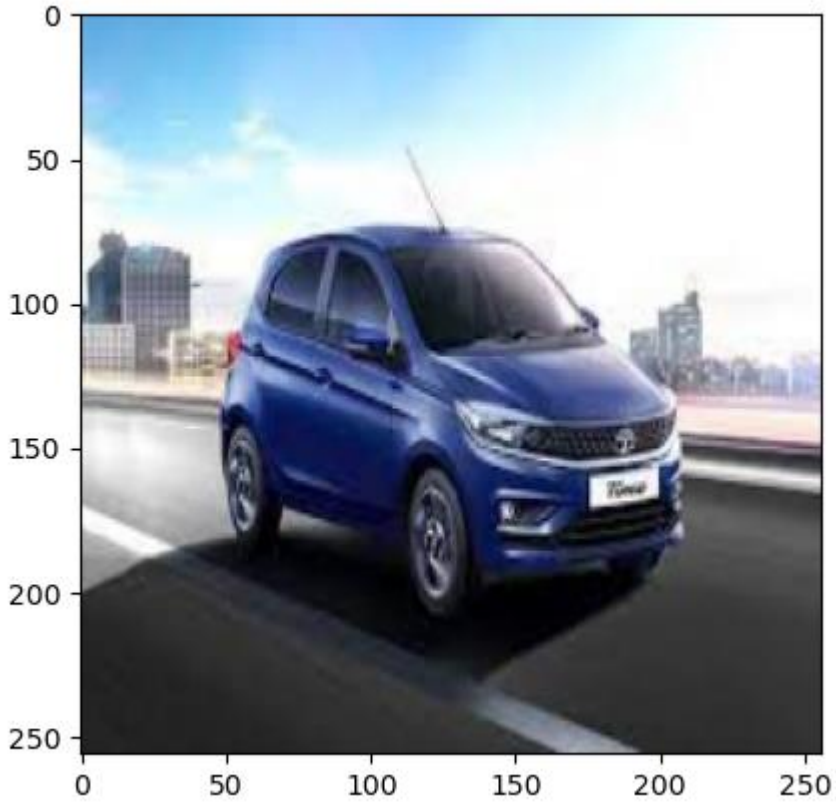


```
#Check results on a few select images
n=np.random.randint(0, x_test.shape[0])
img = x_test[n]
plt.imshow(img)
input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)
input_img_feature=Resnet50_model.predict(input_img)
input_img_features=input_img_feature.reshape(input_img_feature.shape[0], -1)
prediction_AD = AD_model.predict(input_img_features)[0]
prediction_AD = le.inverse_transform([prediction_AD]) #Reverse the label encoder to original name
print("The prediction for this image is: ", prediction_AD)
print("The actual label for this image is: ", test_labels[n])
```



The prediction for this image is: ['Unsafe']

The actual label for this image is: Unsafe



The prediction for this image is: ['Safe']

The actual label for this image is: Safe

InceptionV3 with Support Vector Machine

-*- coding: utf-8 -*-

"""

Created on Wed Aug 17 20:41:13 2022

@author: LAXMINARAYANRO

```
"""

# -*- coding: utf-8 -*-
"""
Created on Wed Aug 17 19:07:29 2022

@author: LAXMINARAYANRO
"""

# -*- coding: utf-8 -*-
"""
Created on Tue Aug 16 12:57:30 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
```

```
#from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
import os
import seaborn as sns
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.inception_v3 import InceptionV3


# Read input images and assign labels based on folder names
print(os.listdir("E:/fffiiles/college pracs and projects/Project/Car"))

SIZE = 256  #Resize images

#Capture training data and labels into respective lists
train_images = []
train_labels = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

train_images = np.array(train_images)
train_labels = np.array(train_labels)


# test
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
```

```
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data into test and train datasets (already split but
assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded
```

```
##################################################
#################
# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

#One hot encode y values for neural network.
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

##############################
#Load model wothout classifier/fully connected layers
InceptionV3_model = InceptionV3(weights='imagenet',
include_top=False, input_shape=(SIZE, SIZE, 3))

#Make loaded layers as non-trainable. This is important as
we want to work with pre-trained weights
for layer in InceptionV3_model.layers:
    layer.trainable = False

InceptionV3_model.summary()  #Trainable parameters will
be 0
```

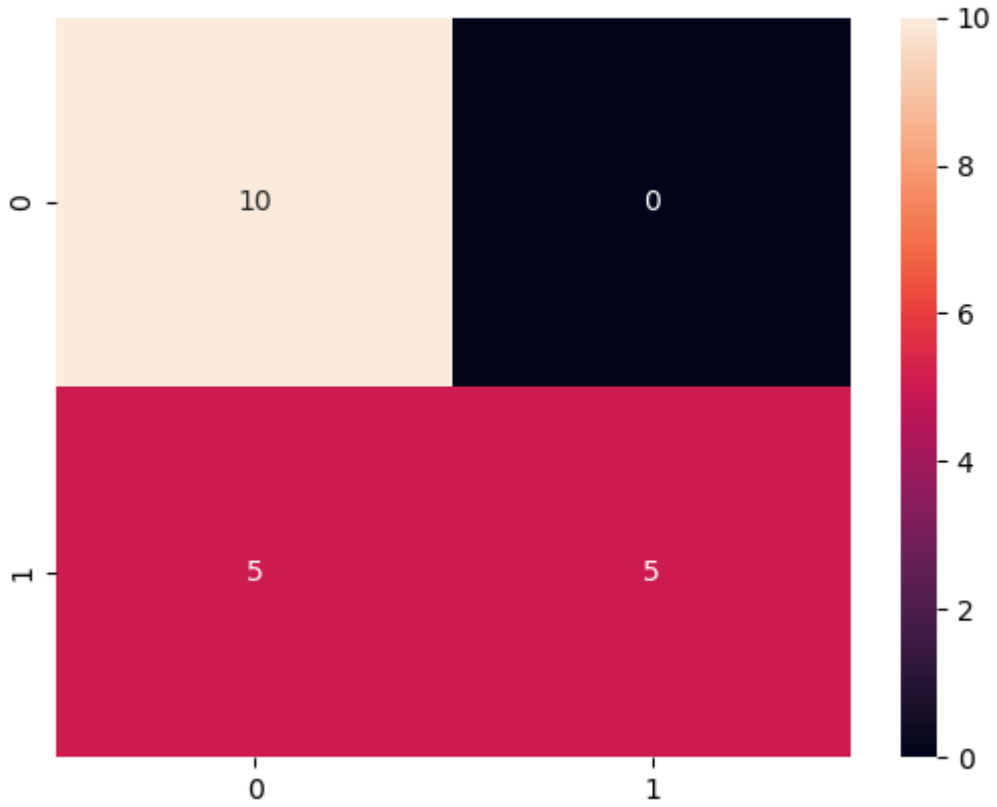
```
#Now, let us use features from convolutional network for RF
feature_extractor=InceptionV3_model.predict(x_train)

features =
feature_extractor.reshape(feature_extractor.shape[0], -1)

X_for_SVM = features #This is our X input to RF

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC
SVM_model = make_pipeline(StandardScaler(),
SVC(gamma='auto'))

# Train the model on training data
SVM_model.fit(X_for_SVM, y_train) #For sklearn no one hot
encoding

#Send test data through same feature extractor process
X_test_feature = InceptionV3_model.predict(x_test)
```

```
X_test_features =
X_test_feature.reshape(X_test_feature.shape[0], -1)

#Now predict using the trained RF model.
prediction_SVM = SVM_model.predict(X_test_features)
#Inverse le transform to get original label back.
prediction_SVM = le.inverse_transform(prediction_SVM)

#Print overall accuracy
from sklearn import metrics
print ("Accuracy = ", metrics.accuracy_score(test_labels,
prediction_SVM))
```

```
Accuracy =  0.75
```

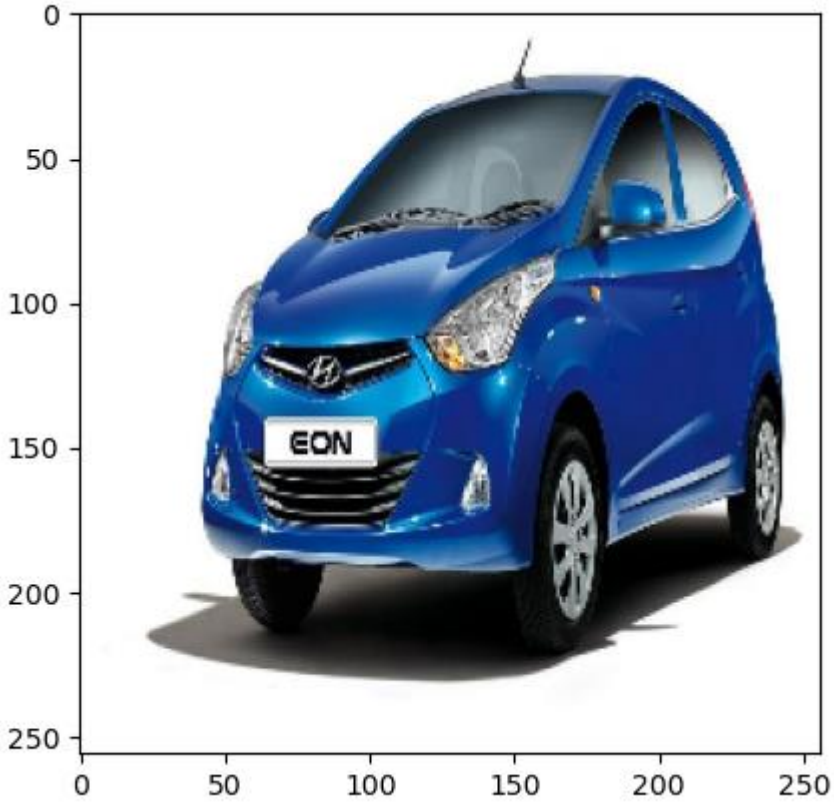
```
#Confusion Matrix - verify accuracy of each class
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_labels, prediction_SVM)
#print(cm)
sns.heatmap(cm, annot=True)
```

```
<AxesSubplot:>
```

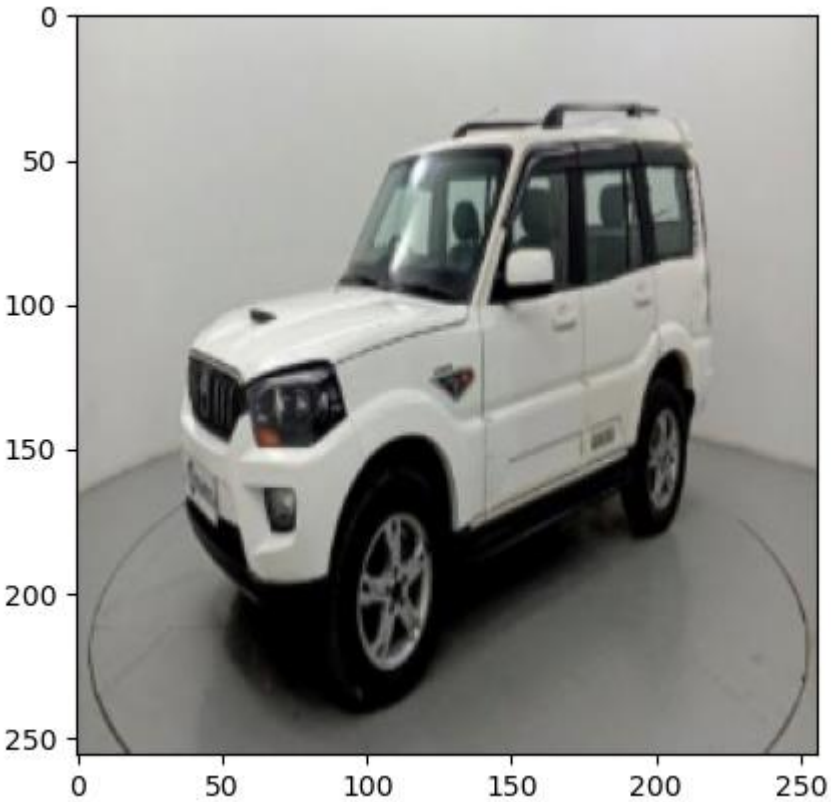



```
#Check results on a few select images

n=np.random.randint(0, x_test.shape[0])

img = x_test[n]

plt.imshow(img)

input_img = np.expand_dims(img, axis=0) #Expand dims so
the input is (num images, x, y, c)

input_img_feature=InceptionV3_model.predict(input_img)

input_img_features=input_img_feature.reshape(input_img_f
eature.shape[0], -1)

prediction_SVM =
SVM_model.predict(input_img_features)[0]
```

```
prediction_SVM = le.inverse_transform([prediction_SVM])
#Reverse the label encoder to original name

print("The prediction for this image is: ", prediction_SVM)

print("The actual label for this image is: ", test_labels[n])
```




```
The prediction for this image is:  ['Unsafe']

The actual label for this image is:  Unsafe
```



The prediction for this image is: ['Safe']

The actual label for this image is: Unsafe

GLCM with Light BGM

-*- coding: utf-8 -*-

"""

Created on Fri Aug 19 15:54:01 2022

@author: LAXMINARAYANRO

```
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
import os
import seaborn as sns
import pandas as pd
from skimage.filters import sobel
from skimage.feature import greycomatrix, greycoprops
from skimage.measure import shannon_entropy

print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

#Resize images to
SIZE = 128

#Capture images and labels into arrays.
#Start by creating empty lists.
train_images = []
```

```
train_labels = []
#for directory_path in glob.glob("cell_images/train/*"):
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, 0) #Reading color images
        img = cv2.resize(img, (SIZE, SIZE)) #Resize images
        train_images.append(img)
        train_labels.append(label)

train_images = np.array(train_images)
train_labels = np.array(train_labels)

#Do exactly the same for test/validation images
# test
test_images = []
test_labels = []
#for directory_path in glob.glob("cell_images/test/*"):
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, 0)
        img = cv2.resize(img, (SIZE, SIZE))
        test_images.append(img)
        test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text (folder names) to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data into test and train datasets (already split but
assigning to meaningful convention)
```

```
#If you only have one dataset then split here
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded


# Normalize pixel values to between 0 and 1
#x_train, x_test = x_train / 255.0, x_test / 255.0


##################################################
#################
# FEATURE EXTRACTOR function
# input shape is (n, x, y, c) - number of images, x, y, and
channels
def feature_extractor(dataset):
    image_dataset = pd.DataFrame()
    for image in range(dataset.shape[0]):  #iterate through
each file
        #print(image)


        df = pd.DataFrame()  #Temporary data frame to capture
information for each loop.
        #Reset dataframe to blank after each loop.


        img = dataset[image, :,:]
```

```
##################################################
##############
   #START ADDING DATA TO THE DATAFRAME



        #Full image
       #GLCM = greycomatrix(img, [1], [0, np.pi/4, np.pi/2,
3*np.pi/4])
       GLCM = greycomatrix(img, [1], [0])
       GLCM_Energy = greycoprops(GLCM, 'energy')[0]
       df['Energy'] = GLCM_Energy
       GLCM_corr = greycoprops(GLCM, 'correlation')[0]
       df['Corr'] = GLCM_corr
       GLCM_diss = greycoprops(GLCM, 'dissimilarity')[0]
       df['Diss_sim'] = GLCM_diss
       GLCM_hom = greycoprops(GLCM, 'homogeneity')[0]
       df['Homogen'] = GLCM_hom
       GLCM_contr = greycoprops(GLCM, 'contrast')[0]
       df['Contrast'] = GLCM_contr



       GLCM2 = greycomatrix(img, [3], [0])
```

```
GLCM_Energy2 = greycoprops(GLCM2, 'energy')[0]
df['Energy2'] = GLCM_Energy2
GLCM_corr2 = greycoprops(GLCM2, 'correlation')[0]
df['Corr2'] = GLCM_corr2
GLCM_diss2 = greycoprops(GLCM2, 'dissimilarity')[0]
df['Diss_sim2'] = GLCM_diss2
GLCM_hom2 = greycoprops(GLCM2, 'homogeneity')[0]
df['Homogen2'] = GLCM_hom2
GLCM_contr2 = greycoprops(GLCM2, 'contrast')[0]
df['Contrast2'] = GLCM_contr2

GLCM3 = greycomatrix(img, [5], [0])
GLCM_Energy3 = greycoprops(GLCM3, 'energy')[0]
df['Energy3'] = GLCM_Energy3
GLCM_corr3 = greycoprops(GLCM3, 'correlation')[0]
df['Corr3'] = GLCM_corr3
GLCM_diss3 = greycoprops(GLCM3, 'dissimilarity')[0]
df['Diss_sim3'] = GLCM_diss3
GLCM_hom3 = greycoprops(GLCM3, 'homogeneity')[0]
df['Homogen3'] = GLCM_hom3
GLCM_contr3 = greycoprops(GLCM3, 'contrast')[0]
df['Contrast3'] = GLCM_contr3
```

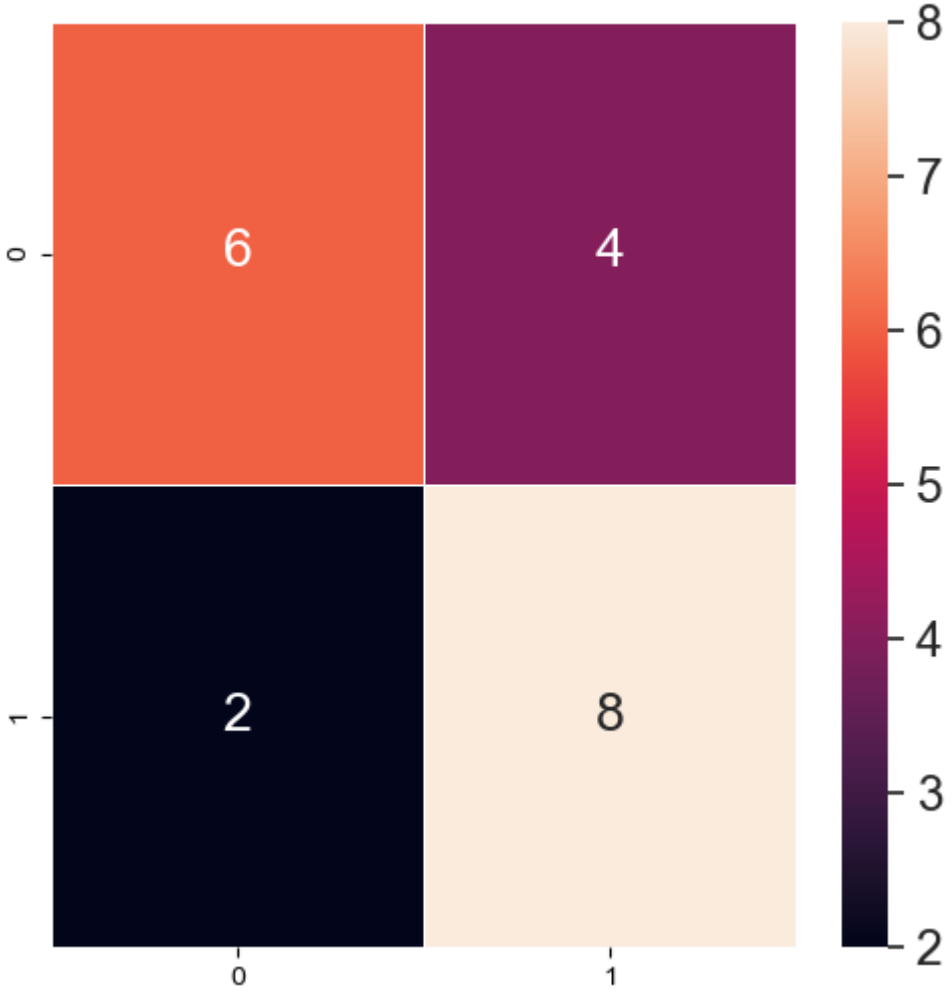
```
GLCM4 = greycomatrix(img, [0], [np.pi/4])
GLCM_Energy4 = greycoprops(GLCM4, 'energy')[0]
df['Energy4'] = GLCM_Energy4
GLCM_corr4 = greycoprops(GLCM4, 'correlation')[0]
df['Corr4'] = GLCM_corr4
GLCM_diss4 = greycoprops(GLCM4, 'dissimilarity')[0]
df['Diss_sim4'] = GLCM_diss4
GLCM_hom4 = greycoprops(GLCM4, 'homogeneity')[0]
df['Homogen4'] = GLCM_hom4
GLCM_contr4 = greycoprops(GLCM4, 'contrast')[0]
df['Contrast4'] = GLCM_contr4

GLCM5 = greycomatrix(img, [0], [np.pi/2])
GLCM_Energy5 = greycoprops(GLCM5, 'energy')[0]
df['Energy5'] = GLCM_Energy5
GLCM_corr5 = greycoprops(GLCM5, 'correlation')[0]
df['Corr5'] = GLCM_corr5
GLCM_diss5 = greycoprops(GLCM5, 'dissimilarity')[0]
df['Diss_sim5'] = GLCM_diss5
GLCM_hom5 = greycoprops(GLCM5, 'homogeneity')[0]
df['Homogen5'] = GLCM_hom5
```

```
        GLCM_contr5 = greycoprops(GLCM5, 'contrast')[0]
        df['Contrast5'] = GLCM_contr5

        #Add more filters as needed
        #entropy = shannon_entropy(img)
        #df['Entropy'] = entropy



        #Append features from current image to the dataset
        image_dataset = image_dataset.append(df)

    return image_dataset
##################################################################
#Extract features from training images
image_features = feature_extractor(x_train)
X_for_ML =image_features
#Reshape to a vector for Random Forest / SVM training
#n_features = image_features.shape[1]
#image_features = np.expand_dims(image_features, axis=0)
#X_for_ML = np.reshape(image_features, (x_train.shape[0], -1))  #Reshape to #images, features
```

```
#Define the classifier

# from sklearn.ensemble import RandomForestClassifier

# RF_model = RandomForestClassifier(n_estimators = 50,
random_state = 42)


#Can also use SVM but RF is faster and may be more
accurate.

#from sklearn import svm

#SVM_model = svm.SVC(decision_function_shape='ovo')
#For multiclass classification

#SVM_model.fit(X_for_ML, y_train)


# Fit the model on training data

# RF_model.fit(X_for_ML, y_train) #For sklearn no one hot
encoding




import lightgbm as lgb

 #Class names for LGBM start at 0 so reassigning labels from
1,2,3,4 to 0,1,2,3

d_train = lgb.Dataset(X_for_ML, label=y_train)
```

```
#
https://lightgbm.readthedocs.io/en/latest/Parameters.html
lgbm_params = {'learning_rate':0.05, 'boosting_type':'dart',
            'objective':'multiclass',
            'metric': 'multi_logloss',
            'num_leaves':100,
            'max_depth':10,
            'num_class':4}  #no.of unique values in the target
class not inclusive of the end value



lgb_model = lgb.train(lgbm_params, d_train, 100) #50
iterations. Increase iterations for small learning rates




#Predict on Test data
#Extract features from test data and reshape, just like
training data
test_features = feature_extractor(x_test)
test_features = np.expand_dims(test_features, axis=0)
test_for_RF = np.reshape(test_features, (x_test.shape[0], -1))
```

```
#Predict on test
test_prediction = lgb_model.predict(test_for_RF)
test_prediction=np.argmax(test_prediction, axis=1)
#Inverse le transform to get original label back.
test_prediction = le.inverse_transform(test_prediction)

#Print overall accuracy
from sklearn import metrics
print ("Accuracy = ", metrics.accuracy_score(test_labels,
test_prediction))
Accuracy =  0.7
#Print confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_labels, test_prediction)

fig, ax = plt.subplots(figsize=(6,6))        # Sample figsize in
inches
sns.set(font_scale=1.6)
sns.heatmap(cm, annot=True, linewidths=.5, ax=ax)
```



#Check results on a few random images

import random

n=random.randint(0, x_test.shape[0]-1) #Select the index of image to be loaded for testing

img = x_test[n]

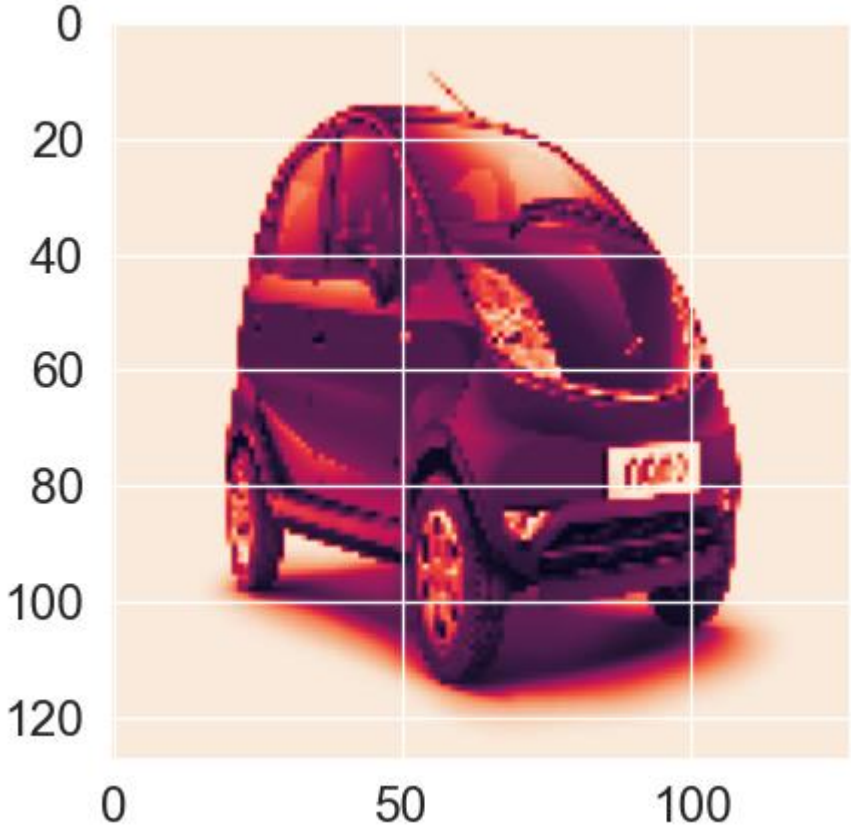
plt.imshow(img)

#Extract features and reshape to right dimensions

```
input_img = np.expand_dims(img, axis=0) #Expand dims so
the input is (num images, x, y, c)

input_img_features=feature_extractor(input_img)

input_img_features = np.expand_dims(input_img_features,
axis=0)

input_img_for_RF = np.reshape(input_img_features,
(input_img.shape[0], -1))

#Predict

img_prediction = lgb_model.predict(input_img_for_RF)

img_prediction=np.argmax(img_prediction, axis=1)

img_prediction = le.inverse_transform([img_prediction])
#Reverse the label encoder to original name

print("The prediction for this image is: ", img_prediction)

print("The actual label for this image is: ", test_labels[n])
```

The prediction for this image is: ['Safe']

The actual label for this image is: Safe



The prediction for this image is: ['Unsafe']

The actual label for this image is: Unsafe

Bag of Visual Words-KNN Clustering with LinearSVC

Train.py

-*- coding: utf-8 -*-

"""

Created on Fri Aug 19 21:08:43 2022

```
@author: LAXMINARAYANRO
"""

import cv2
import numpy as np
import os


# Get the training classes names and store them in a list
#Here we use folder names for class names

#train_path = 'dataset/train'  # Names are Aeroplane, Bicycle, Car
train_path = 'E:/fffiiles/college pracs and projects/Project/Car/Model_images/train'  # Folder Names are Parasitized and Uninfected
training_names = os.listdir(train_path)

# Get path to all images and save them in a list
# image_paths and the corresponding label in image_paths
image_paths = []
image_classes = []
class_id = 0
```

```
#To make it easy to list all file names in a directory let us
define a function
#
def imglist(path):
    return [os.path.join(path, f) for f in os.listdir(path)]

#Fill the placeholder empty lists with image path, classes, and
add class ID number
#

for training_name in training_names:
    dir = os.path.join(train_path, training_name)
    class_path = imglist(dir)
    image_paths+=class_path
    image_classes+=[class_id]*len(class_path)
    class_id+=1

# Create feature extraction and keypoint detector objects
    #SIFT is not available anymore in openCV
# Create List where all the descriptors will be stored
des_list = []
```

```
#BRISK is a good replacement to SIFT. ORB also works but
didn;t work well for this example
brisk = cv2.BRISK_create(30)

for image_path in image_paths:
    im = cv2.imread(image_path)
    kpts, des = brisk.detectAndCompute(im, None)
    des_list.append((image_path, des))

# Stack all the descriptors vertically in a numpy array
descriptors = des_list[0][1]
for image_path, descriptor in des_list[1:]:
    descriptors = np.vstack((descriptors, descriptor))

#kmeans works only on float, so convert integers to float
descriptors_float = descriptors.astype(float)

# Perform k-means clustering and vector quantization
from scipy.cluster.vq import kmeans, vq
```

```
k = 200  #k means with 100 clusters gives lower accuracy for
the aeroplane example
voc, variance = kmeans(descriptors_float, k, 1)


# Calculate the histogram of features and represent them as
vector
#vq Assigns codes from a code book to observations.
im_features = np.zeros((len(image_paths), k), "float32")
for i in range(len(image_paths)):
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        im_features[i][w] += 1


# Perform Tf-Idf vectorization
nbr_occurences = np.sum( (im_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) /
(1.0*nbr_occurences + 1)), 'float32')


# Scaling the words
#Standardize features by removing the mean and scaling to
unit variance
#In a way normalization
from sklearn.preprocessing import StandardScaler
```

```
stdSlr = StandardScaler().fit(im_features)
im_features = stdSlr.transform(im_features)

#Train an algorithm to discriminate vectors corresponding to positive and negative training images
# Train the Linear SVM
from sklearn.svm import LinearSVC
clf = LinearSVC(max_iter=10000)  #Default of 100 is not converging
clf.fit(im_features, np.array(image_classes))

#Train Random forest to compare how it does against SVM
#from sklearn.ensemble import RandomForestClassifier
#clf = RandomForestClassifier(n_estimators = 100, random_state=30)
#clf.fit(im_features, np.array(image_classes))


# Save the SVM
#Joblib dumps Python object into one file
# from sklearn.externals import joblib
import joblib
```

```
joblib.dump((clf, training_names, stdSlr, k, voc), "bovw.pkl",
compress=3)
```

Test.py

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 19 21:09:02 2022

@author: LAXMINARAYANRO
"""

import cv2
import numpy as np
import os
import pylab as pl
import joblib
import seaborn as sns

# Load the classifier, class names, scaler, number of clusters
and vocabulary
#from stored pickle file (generated during training)
clf, classes_names, stdSlr, k, voc = joblib.load("bovw.pkl")
```

```
# Get the path of the testing image(s) and store them in a list
#test_path = 'dataset/test' # Names are Aeroplane, Bicycle, Car
test_path = 'E:/fffiiles/college pracs and projects/Project/Car/Model_images/test'  # Folder Names are Parasitized and Uninfected
#instead of test if you use train then we get great accuracy

testing_names = os.listdir(test_path)

# Get path to all images and save them in a list
# image_paths and the corresponding label in image_paths
image_paths = []
image_classes = []
class_id = 0

#To make it easy to list all file names in a directory let us define a function
#
def imglist(path):
    return [os.path.join(path, f) for f in os.listdir(path)]
```

```
#Fill the placeholder empty lists with image path, classes, and add class ID number

for testing_name in testing_names:
    dir = os.path.join(test_path, testing_name)
    class_path = imglist(dir)
    image_paths+=class_path
    image_classes+=[class_id]*len(class_path)
    class_id+=1

# Create feature extraction and keypoint detector objects
    #SIFT is not available anymore in openCV
# Create List where all the descriptors will be stored
des_list = []

#BRISK is a good replacement to SIFT. ORB also works but didn;t work well for this example
brisk = cv2.BRISK_create(30)

for image_path in image_paths:
    im = cv2.imread(image_path)
    kpts, des = brisk.detectAndCompute(im, None)
```

```
    des_list.append((image_path, des))

# Stack all the descriptors vertically in a numpy array
descriptors = des_list[0][1]
for image_path, descriptor in des_list[0:]:
    descriptors = np.vstack((descriptors, descriptor))

# Calculate the histogram of features
#vq Assigns codes from a code book to observations.
from scipy.cluster.vq import vq
test_features = np.zeros((len(image_paths), k), "float32")
for i in range(len(image_paths)):
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        test_features[i][w] += 1

# Perform Tf-Idf vectorization
nbr_occurences = np.sum( (test_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) /
(1.0*nbr_occurences + 1)), 'float32')

# Scale the features
```

```
#Standardize features by removing the mean and scaling to
unit variance

#Scaler (stdSlr comes from the pickled file we imported)

test_features = stdSlr.transform(test_features)


#######Until here most of the above code is similar to Train
except for kmeans clustering####


#Report true class names so they can be compared with
predicted classes

true_class =  [classes_names[i] for i in image_classes]

# Perform the predictions and report predicted class names.

predictions =  [classes_names[i] for i in
clf.predict(test_features)]



#Print the true class and Predictions

print ("true_class ="  + str(true_class))

print ("prediction ="  + str(predictions))
```

```
true_class =['Safe', 'Safe', 'Safe', 'Safe', 'Safe', 'Safe', 'Safe',
'Safe', 'Safe', 'Safe', 'Unsafe', 'Unsafe', 'Unsafe', 'Unsafe',
'Unsafe', 'Unsafe', 'Unsafe', 'Unsafe', 'Unsafe', 'Unsafe']
```

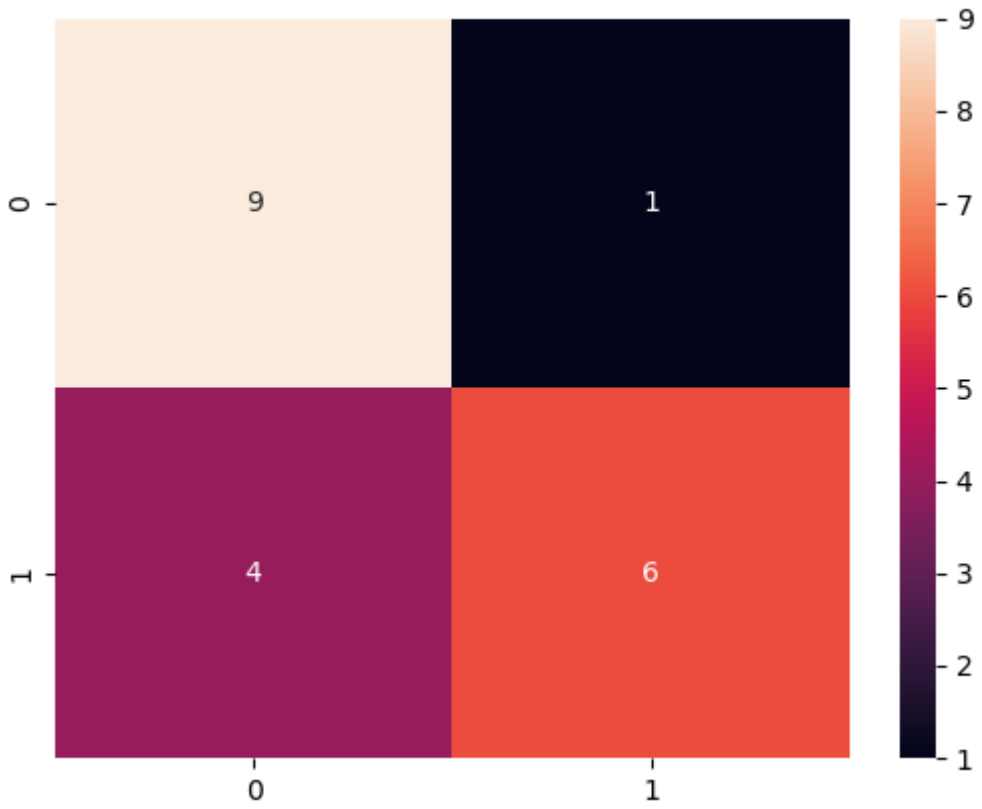
```
prediction =['Safe', 'Safe', 'Safe', 'Unsafe', 'Safe', 'Safe', 'Safe',
'Safe', 'Safe', 'Safe', 'Unsafe', 'Unsafe', 'Unsafe', 'Unsafe',
'Unsafe', 'Unsafe', 'Safe', 'Safe', 'Safe', 'Safe']
```

```
##################################################
#To make it easy to understand the accuracy let us print the
confusion matrix


# def showconfusionmatrix(cm):
#     pl.matshow(cm)
#     pl.title('Confusion matrix')
#     pl.colorbar()
#     pl.show()


from sklearn import metrics
print ("Accuracy = ", metrics.accuracy_score(true_class,
predictions))
```

```
Accuracy =  0.75
```

```
# accuracy = accuracy_score(true_class, predictions)
# print ("accuracy = ", accuracy)
# # cm = confusion_matrix(true_class, predictions)
# print (cm)
#Confusion Matrix - verify accuracy of each class
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(true_class, predictions)
#print(cm)
sns.heatmap(cm, annot=True)
```



EfficeinetB3 with PCA

```
import numpy as np
import matplotlib.pyplot as plt
import glob
```

```
import cv2

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
import os
import seaborn as sns
from keras.applications.efficientnet import EfficientNetB3
from tensorflow.keras.layers import Input

# Read input images and assign labels based on folder names
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 256  #Resize images

#Capture training data and labels into respective lists
train_images = []
train_labels = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#Convert lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)


# Capture test/validation data and labels into respective lists

test_images = []
test_labels = []
```

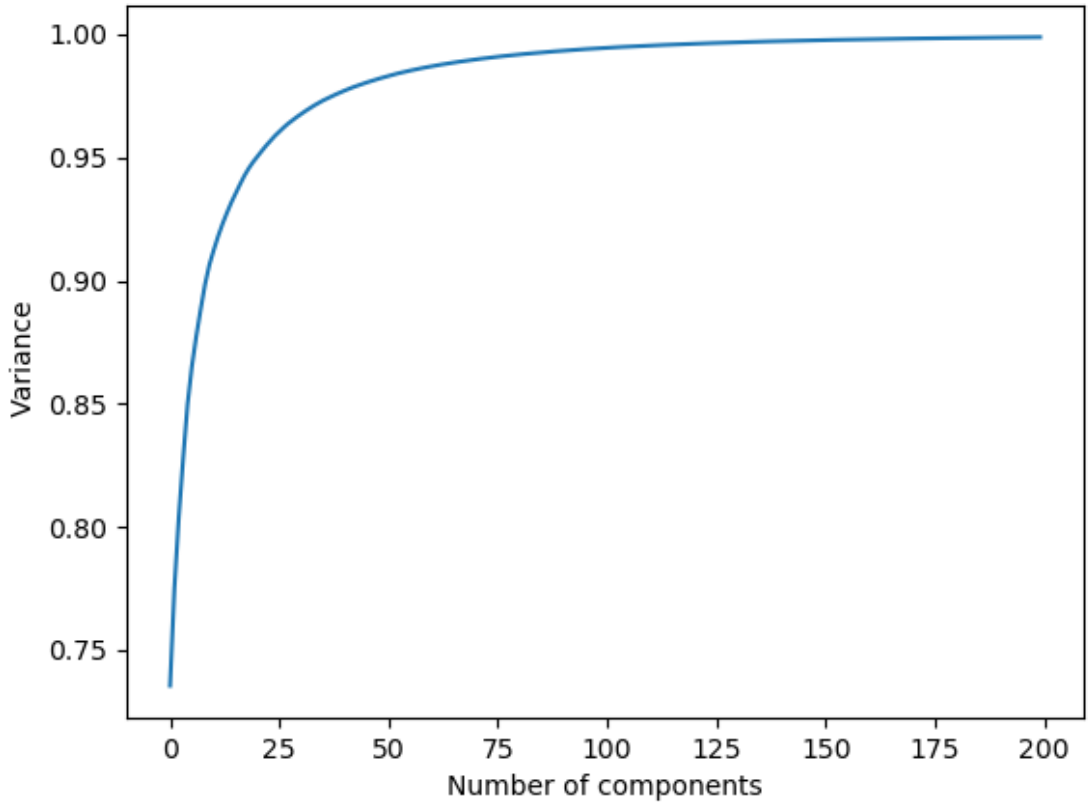
```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#Convert lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)
```

```
#Split data into test and train datasets (already split but assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images, train_labels_encoded, test_images, test_labels_encoded

##################################################################
# Scale pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

#One hot encode y values for neural network. Not needed for Random Forest
from keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

################################
#Load VGG model with imagenet trained weights and without classifier/fully connected layers
#We will use this as feature extractor.
EfficientNetB3_model = EfficientNetB3(weights='imagenet', include_top=False, input_shape=(SIZE, SIZE, 3))
```

```
#Make loaded layers as non-trainable. This is important as
we want to work with pre-trained weights
for layer in EfficientNetB3_model.layers:
    layer.trainable = False

EfficientNetB3_model.summary()  #Trainable parameters will
be 0


#Now, let us extract features using VGG imagenet weights
#Train features
train_feature_extractor=EfficientNetB3_model.predict(x_trai
n)
train_features =
train_feature_extractor.reshape(train_feature_extractor.sha
pe[0], -1)
#test features
test_feature_extractor=EfficientNetB3_model.predict(x_test)
test_features =
test_feature_extractor.reshape(test_feature_extractor.shape
[0], -1)

# Reduce dimensions using PCA
```

```
from sklearn.decomposition import PCA


#First verfiy the ideal number of PCA components to not lose much information.

#Try to retain 90% information, so look where the curve starts to flatten.

#Remember that the n_components must be lower than the number of rows or columns (features)

pca_test = PCA(n_components=200) #

pca_test.fit(train_features)

plt.plot(np.cumsum(pca_test.explained_variance_ratio_))

plt.xlabel("Number of components")

plt.ylabel(" Variance")
```

```
Text(0, 0.5, ' Variance')
```



#Pick the optimal number of components. This is how many features we will have

#for our machine learning

n_PCA_components = 200

pca = PCA(n_components=n_PCA_components)

train_PCA = pca.fit_transform(train_features)

test_PCA = pca.transform(test_features) #Make sure you are just transforming, not fitting.

#If we want 90% information captured we can also try ...

#pca=PCA(0.9)

#principalComponents = pca.fit_transform(X_for_RF)

```
############### Neural Network Approach
###################

##Add hidden dense layers and final output/classifier layer.
model = Sequential()
inputs = Input(shape=(n_PCA_components,)) #Shape =
n_components
hidden = Dense(256, activation='relu')(inputs)
#hidden1 = Dense(512, activation='relu')(inputs)
#hidden2 = Dense(256, activation='relu')(hidden1)
output = Dense(4, activation='softmax')(hidden)
model = Model(inputs=inputs, outputs=output)

print(model.summary())
#
Model: "model_1"
_____________________________________________________________
_________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 200)]             0
```

```
 dense_2 (Dense)             (None, 256)               51456

 dense_3 (Dense)             (None, 2)                 514

=================================================================
Total params: 51,970
Trainable params: 51,970
Non-trainable params: 0
_________________________________________________________________
None
```

```
model.compile(loss='categorical_crossentropy',
optimizer='rmsprop', metrics=['categorical_accuracy'])
```

```
Epoch 1/20
16/16 [==============================] - 3s 7ms/step -
loss: 0.6804 - categorical_accuracy: 0.5944
Epoch 2/20
16/16 [==============================] - 0s 2ms/step -
loss: 0.6131 - categorical_accuracy: 0.6908
Epoch 3/20
```

```
16/16 [==============================] - 0s 2ms/step - loss: 0.5706 - categorical_accuracy: 0.7249

Epoch 4/20

16/16 [==============================] - 0s 3ms/step - loss: 0.5421 - categorical_accuracy: 0.7490

Epoch 5/20

16/16 [==============================] - 0s 2ms/step - loss: 0.5076 - categorical_accuracy: 0.7871

Epoch 6/20

16/16 [==============================] - 0s 2ms/step - loss: 0.4781 - categorical_accuracy: 0.8052

Epoch 7/20

16/16 [==============================] - 0s 3ms/step - loss: 0.4460 - categorical_accuracy: 0.8112

Epoch 8/20

16/16 [==============================] - 0s 2ms/step - loss: 0.4193 - categorical_accuracy: 0.8373

Epoch 9/20

16/16 [==============================] - 0s 2ms/step - loss: 0.3934 - categorical_accuracy: 0.8534

Epoch 10/20

16/16 [==============================] - 0s 2ms/step - loss: 0.3669 - categorical_accuracy: 0.8614

Epoch 11/20
```

```
16/16 [==============================] - 0s 2ms/step - loss: 0.3439 - categorical_accuracy: 0.8835

Epoch 12/20

16/16 [==============================] - 0s 2ms/step - loss: 0.3237 - categorical_accuracy: 0.8775

Epoch 13/20

16/16 [==============================] - 0s 3ms/step - loss: 0.2997 - categorical_accuracy: 0.9036

Epoch 14/20

16/16 [==============================] - 0s 2ms/step - loss: 0.2855 - categorical_accuracy: 0.9036

Epoch 15/20

16/16 [==============================] - 0s 2ms/step - loss: 0.2693 - categorical_accuracy: 0.9137

Epoch 16/20

16/16 [==============================] - 0s 3ms/step - loss: 0.2519 - categorical_accuracy: 0.9277

Epoch 17/20

16/16 [==============================] - 0s 2ms/step - loss: 0.2372 - categorical_accuracy: 0.9277

Epoch 18/20

16/16 [==============================] - 0s 2ms/step - loss: 0.2187 - categorical_accuracy: 0.9518

Epoch 19/20
```

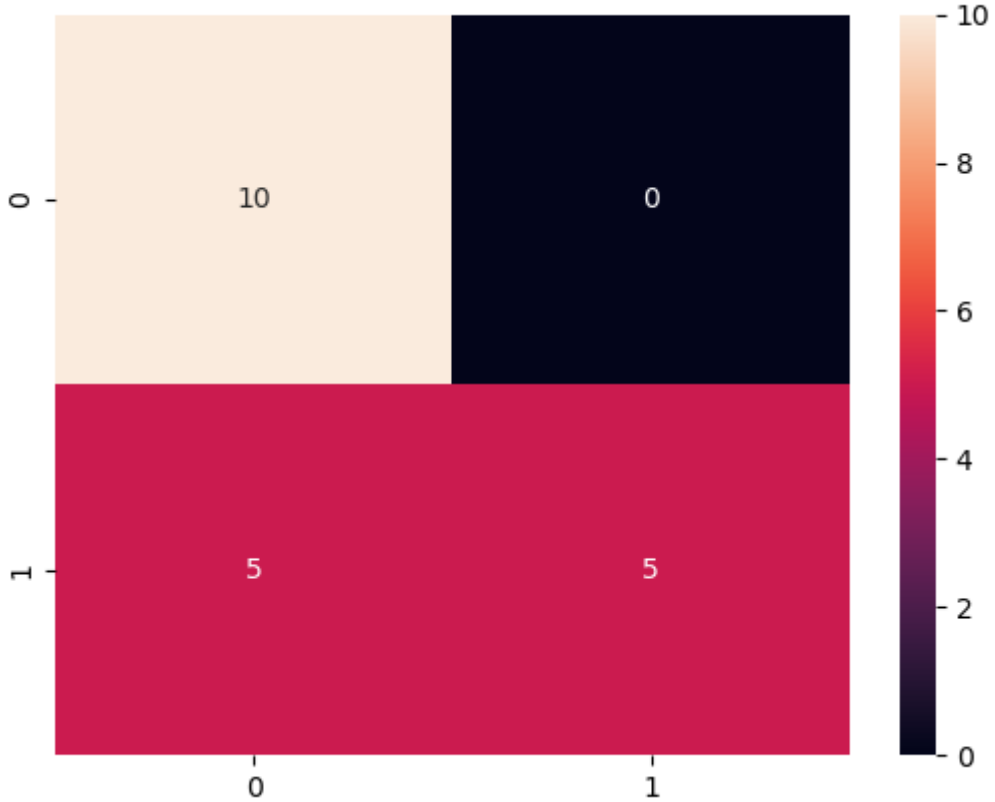
```
16/16 [==============================] - 0s 3ms/step - loss: 0.2102 - categorical_accuracy: 0.9498

Epoch 20/20

16/16 [==============================] - 0s 2ms/step - loss: 0.1941 - categorical_accuracy: 0.9498

Total execution time with PCA is:  0:00:03.749183
```

```
import datetime

start = datetime.datetime.now()

#Fit the model. Do not forget to use on-hot-encoded Y values.

model.fit(train_PCA, y_train_one_hot, epochs=20, verbose=1)


end = datetime.datetime.now()

print("Total execution time with PCA is: ", end-start)


##Predict on test dataset

predict_test = model.predict(test_PCA)

predict_test = np.argmax(predict_test, axis=1)

predict_test = le.inverse_transform(predict_test)

#

#

##Print overall accuracy
```

```
from sklearn import metrics

print ("Accuracy = ", metrics.accuracy_score(test_labels, predict_test))
```

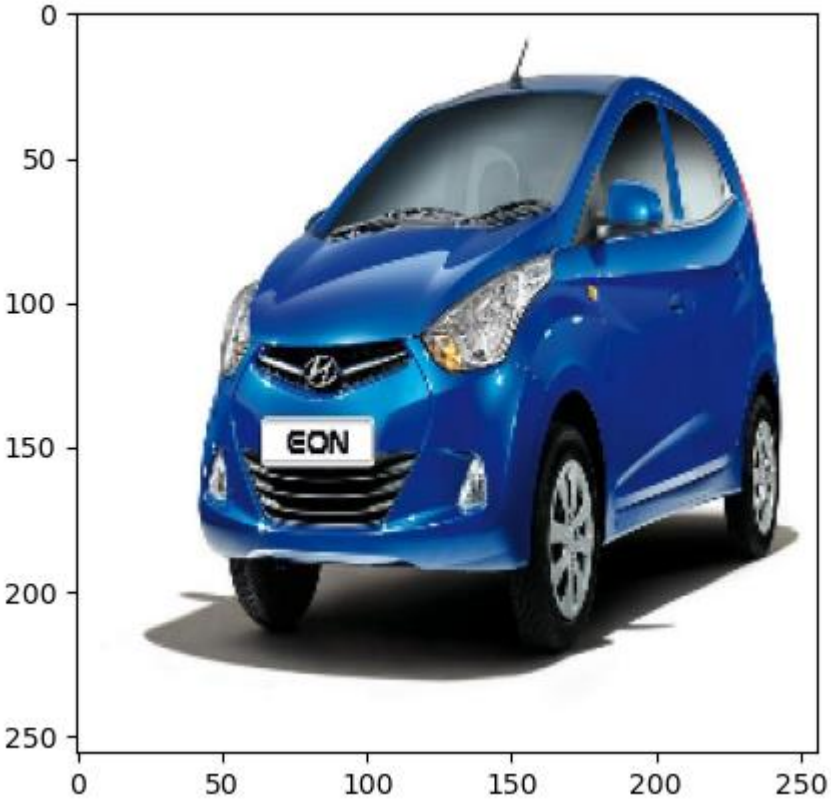
Accuracy = 0.75

```
#Confusion Matrix - verify accuracy of each class

from sklearn.metrics import confusion_matrix


cm = confusion_matrix(test_labels, predict_test)

#print(cm)

sns.heatmap(cm, annot=True)
```




```
#Check results on a few select images
```

```
n=np.random.randint(0, x_test.shape[0])
img = x_test[n]
plt.imshow(img)

input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)
input_img_feature=EfficientNetB3_model.predict(input_img)
input_img_features=input_img_feature.reshape(input_img_feature.shape[0], -1)
input_img_PCA = pca.transform(input_img_features)
prediction_img = model.predict(input_img_PCA)
prediction_img = np.argmax(prediction_img, axis=1)
prediction_img = le.inverse_transform(prediction_img) #Reverse the label encoder to original name
print("The prediction for this image is: ", prediction_img)
print("The actual label for this image is: ", test_labels[n])
```

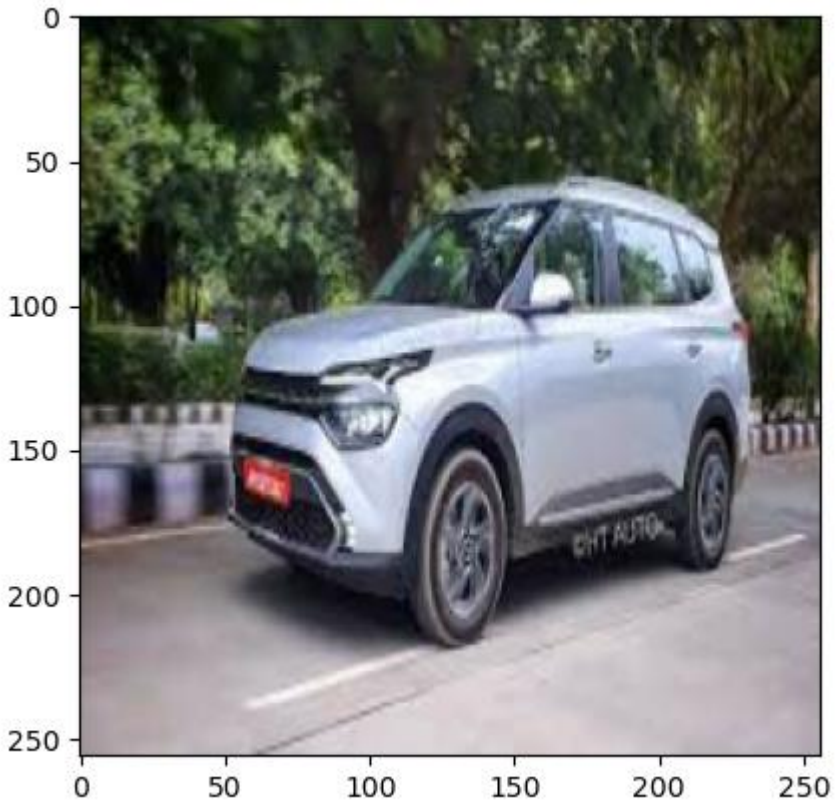


1/1 [==========] - 0s 330ms/step

1/1 [==========] - 0s 47ms/step

The prediction for this image is: ['Safe']

The actual label for this image is: Unsafe



1/1 [==========] - 0s 250ms/step

1/1 [==========] - 0s 31ms/step

The prediction for this image is: ['Safe']

The actual label for this image is: Safe

U-Net with K Nearest Neighbours

---

import tensorflow as tf

import os

import random

```
import numpy as np

from tqdm import tqdm

from skimage.io import imread, imshow
from skimage.transform import resize
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
import os
import seaborn as sns
```

```
seed = 42
np.random.seed = seed

IMG_WIDTH = 256
IMG_HEIGHT = 256
IMG_CHANNELS = 3



train_images = []
train_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_HEIGHT, IMG_WIDTH))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)
```

```
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_HEIGHT, IMG_WIDTH))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)
```

```
# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded


# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

#Build the model
inputs = tf.keras.layers.Input((IMG_HEIGHT, IMG_WIDTH,
IMG_CHANNELS))
s = tf.keras.layers.Lambda(lambda x: x / 255)(inputs)
```

```
#Contraction path
c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(s)
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c1)
p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p1)
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p2)
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p3)
```

```
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c4)
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p4)
c5 = tf.keras.layers.Dropout(0.3)(c5)
c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c5)

#Expansive path
u6 = tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2,
2), padding='same')(c5)
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c6)

u7 = tf.keras.layers.Conv2DTranspose(64, (2, 2), strides=(2,
2), padding='same')(c6)
u7 = tf.keras.layers.concatenate([u7, c3])
```

```
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u7)

c7 = tf.keras.layers.Dropout(0.2)(c7)

c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c7)


u8 = tf.keras.layers.Conv2DTranspose(32, (2, 2), strides=(2,
2), padding='same')(c7)

u8 = tf.keras.layers.concatenate([u8, c2])

c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u8)

c8 = tf.keras.layers.Dropout(0.1)(c8)

c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c8)


u9 = tf.keras.layers.Conv2DTranspose(16, (2, 2), strides=(2,
2), padding='same')(c8)

u9 = tf.keras.layers.concatenate([u9, c1], axis=3)

c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u9)

c9 = tf.keras.layers.Dropout(0.1)(c9)

c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c9)
```

```
outputs = tf.keras.layers.Conv2D(1, (1, 1),
activation='sigmoid')(c9)


model = tf.keras.Model(inputs=[inputs], outputs=[outputs])

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

model.summary()
```

```
 Layer (type)                Output Shape         Param #
Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3  0           []
                             )]

 lambda (Lambda)             (None, 256, 256, 3)  0
['input_1[0][0]']

 conv2d (Conv2D)             (None, 256, 256, 16  448
['lambda[0][0]']
                             )

 dropout (Dropout)           (None, 256, 256, 16  0
['conv2d[0][0]']
```

)

conv2d_1 (Conv2D) (None, 256, 256, 16 2320
['dropout[0][0]']
)

max_pooling2d (MaxPooling2D) (None, 128, 128, 16 0
['conv2d_1[0][0]']
)

conv2d_2 (Conv2D) (None, 128, 128, 32 4640
['max_pooling2d[0][0]']
)

dropout_1 (Dropout) (None, 128, 128, 32 0
['conv2d_2[0][0]']
)

conv2d_3 (Conv2D) (None, 128, 128, 32 9248
['dropout_1[0][0]']
)

max_pooling2d_1 (MaxPooling2D) (None, 64, 64, 32) 0
['conv2d_3[0][0]']

conv2d_4 (Conv2D) (None, 64, 64, 64) 18496
['max_pooling2d_1[0][0]']

dropout_2 (Dropout) (None, 64, 64, 64) 0
['conv2d_4[0][0]']

conv2d_5 (Conv2D) (None, 64, 64, 64) 36928
['dropout_2[0][0]']

max_pooling2d_2 (MaxPooling2D) (None, 32, 32, 64) 0
['conv2d_5[0][0]']

conv2d_6 (Conv2D) (None, 32, 32, 128) 73856
['max_pooling2d_2[0][0]']

dropout_3 (Dropout) (None, 32, 32, 128) 0
['conv2d_6[0][0]']

conv2d_7 (Conv2D) (None, 32, 32, 128) 147584
['dropout_3[0][0]']

max_pooling2d_3 (MaxPooling2D) (None, 16, 16, 128) 0
['conv2d_7[0][0]']

conv2d_8 (Conv2D) (None, 16, 16, 256) 295168
['max_pooling2d_3[0][0]']

dropout_4 (Dropout) (None, 16, 16, 256) 0
['conv2d_8[0][0]']

conv2d_9 (Conv2D) (None, 16, 16, 256) 590080
['dropout_4[0][0]']

conv2d_transpose (Conv2DTransp (None, 32, 32, 128)
131200 ['conv2d_9[0][0]']
ose)

concatenate (Concatenate) (None, 32, 32, 256) 0
['conv2d_transpose[0][0]',
'conv2d_7[0][0]']

conv2d_10 (Conv2D) (None, 32, 32, 128) 295040
['concatenate[0][0]']

```
 dropout_5 (Dropout)            (None, 32, 32, 128)  0
['conv2d_10[0][0]']

 conv2d_11 (Conv2D)             (None, 32, 32, 128)  147584
['dropout_5[0][0]']

 conv2d_transpose_1 (Conv2DTran  (None, 64, 64, 64)  32832
['conv2d_11[0][0]']
 spose)

 concatenate_1 (Concatenate)    (None, 64, 64, 128)  0
['conv2d_transpose_1[0][0]',
                                                     'conv2d_5[0][0]']

 conv2d_12 (Conv2D)             (None, 64, 64, 64)   73792
['concatenate_1[0][0]']

 dropout_6 (Dropout)            (None, 64, 64, 64)   0
['conv2d_12[0][0]']

 conv2d_13 (Conv2D)             (None, 64, 64, 64)   36928
['dropout_6[0][0]']
```

conv2d_transpose_2 (Conv2DTran (None, 128, 128, 32 8224 ['conv2d_13[0][0]']
spose))

concatenate_2 (Concatenate) (None, 128, 128, 64 0 ['conv2d_transpose_2[0][0]',
) 'conv2d_3[0][0]']

conv2d_14 (Conv2D) (None, 128, 128, 32 18464 ['concatenate_2[0][0]']
)

dropout_7 (Dropout) (None, 128, 128, 32 0 ['conv2d_14[0][0]']
)

conv2d_15 (Conv2D) (None, 128, 128, 32 9248 ['dropout_7[0][0]']
)

conv2d_transpose_3 (Conv2DTran (None, 256, 256, 16 2064 ['conv2d_15[0][0]']
spose))

```
 concatenate_3 (Concatenate)    (None, 256, 256, 32  0
['conv2d_transpose_3[0][0]',
                                                     )                                  'conv2d_1[0][0]']

 conv2d_16 (Conv2D)             (None, 256, 256, 16  4624
['concatenate_3[0][0]']
                                                     )

 dropout_8 (Dropout)            (None, 256, 256, 16  0
['conv2d_16[0][0]']
                                                     )

 conv2d_17 (Conv2D)             (None, 256, 256, 16  2320
['dropout_8[0][0]']
                                                     )

 conv2d_18 (Conv2D)             (None, 256, 256, 1)  17
['conv2d_17[0][0]']

==================================================================================================
Total params: 1,941,105
```

```
Trainable params: 1,941,105
Non-trainable params: 0
```

```
feature_extractor=model.predict(x_train)


features =
feature_extractor.reshape(feature_extractor.shape[0], -1)


X_for_KNN = features
```

```
1/16 [>.............................] - ETA: 2:26
2022-08-21 22:00:03.382674: W
tensorflow/stream_executor/platform/default/dso_loader.cc
:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror:
cudnn64_8.dll not found
2022-08-21 22:00:03.406017: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:1850]
Cannot dlopen some GPU libraries. Please make sure the
missing libraries mentioned above are installed properly if
you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download
and setup the required libraries for your platform.
Skipping registering GPU devices...
2022-08-21 22:00:03.433319: I
tensorflow/core/platform/cpu_feature_guard.cc:193] This
```

TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-08-21 22:07:46.400007: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-08-21 22:07:49.586641: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

3/16 [====>.....] - ETA: 40s

2022-08-21 22:00:03.382674: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-08-21 22:00:03.406017: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-08-21 22:00:03.433319: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-08-21 22:07:46.400007: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-08-21 22:07:49.586641: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-08-21 22:07:52.694507: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

4/16 [======>.....] - ETA: 37s

2022-08-21 22:00:03.382674: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-08-21 22:00:03.406017: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the

missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-08-21 22:00:03.433319: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-08-21 22:07:46.400007: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-08-21 22:07:49.586641: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-08-21 22:07:52.694507: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-08-21 22:07:55.756143: W tensorflow/core/framework/cpu_allocator_impl.cc:82]

Allocation of 268435456 exceeds 10% of free system memory.

2022-08-21 22:00:03.382674: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-08-21 22:00:03.406017: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-08-21 22:00:03.433319: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

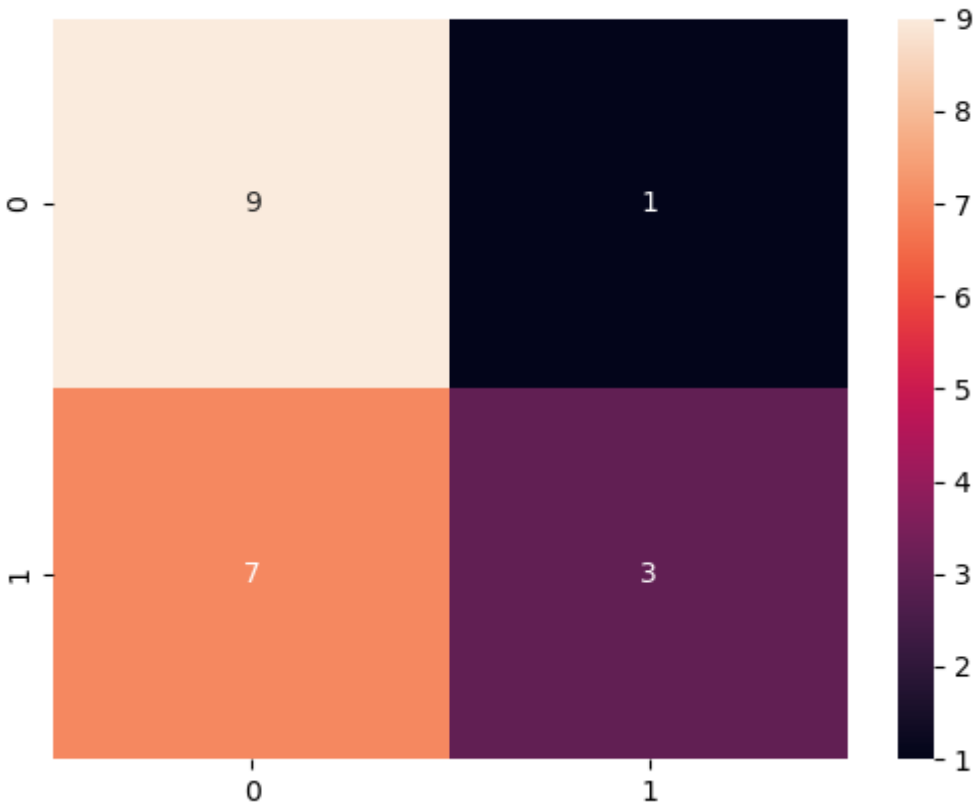
2022-08-21 22:07:46.400007: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

```
2022-08-21 22:07:49.586641: W
tensorflow/core/framework/cpu_allocator_impl.cc:82]
Allocation of 268435456 exceeds 10% of free system
memory.
2022-08-21 22:07:52.694507: W
tensorflow/core/framework/cpu_allocator_impl.cc:82]
Allocation of 268435456 exceeds 10% of free system
memory.
2022-08-21 22:07:55.756143: W
tensorflow/core/framework/cpu_allocator_impl.cc:82]
Allocation of 268435456 exceeds 10% of free system
memory.
2022-08-21 22:07:59.825308: W
tensorflow/core/framework/cpu_allocator_impl.cc:82]
Allocation of 268435456 exceeds 10% of free system
memory.
16/16 [==============================] - 59s 3s/step
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
KNN_model =
BaggingClassifier(KNeighborsClassifier(),max_samples=0.5,
max_features=0.5)

# Train the model on training data
```

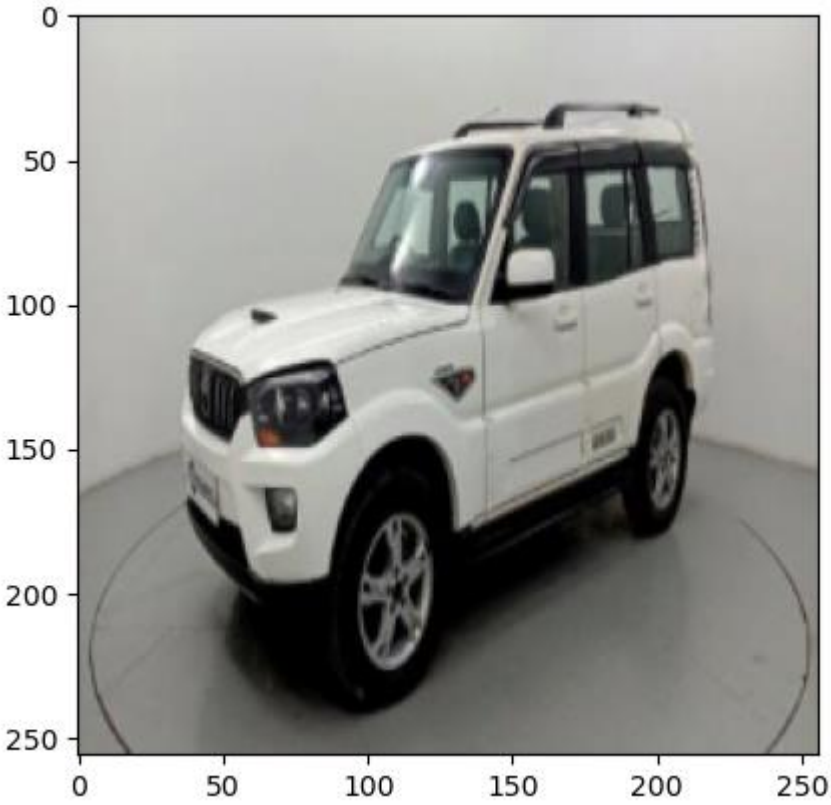
```
KNN_model.fit(X_for_RF, y_train)

#Send test data through same feature extractor process
X_test_feature = model.predict(x_test)
X_test_features =
X_test_feature.reshape(X_test_feature.shape[0], -1)

#Now predict using the trained RF model.
prediction_KNN = KNN_model.predict(X_test_features)
#Inverse le transform to get original label back.
prediction_KNN = le.inverse_transform(prediction_KNN)

#Print overall accuracy
from sklearn import metrics
print ("Accuracy = ", metrics.accuracy_score(test_labels,
prediction_KNN))
Accuracy =  0.6
#Confusion Matrix - verify accuracy of each class
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_labels, prediction_KNN)
#print(cm)
```

```
sns.heatmap(cm, annot=True)
```



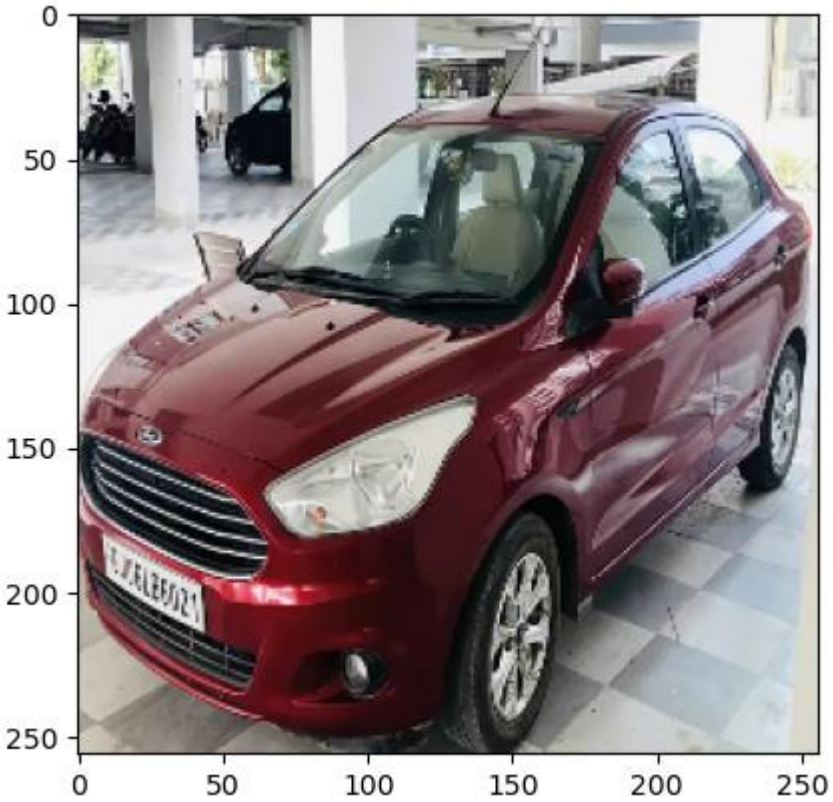

```
#Check results on a few select images

n=np.random.randint(0, x_test.shape[0])

img = x_test[n]

plt.imshow(img)

input_img = np.expand_dims(img, axis=0) #Expand dims so
the input is (num images, x, y, c)

input_img_feature=model.predict(input_img)

input_img_features=input_img_feature.reshape(input_img_f
eature.shape[0], -1)

prediction_KNN =
KNN_model.predict(input_img_features)[0]
```

```
prediction_KNN = le.inverse_transform([prediction_KNN])
#Reverse the label encoder to original name

print("The prediction for this image is: ", prediction_KNN)

print("The actual label for this image is: ", test_labels[n])
```



he prediction for this image is: ['Unsafe']

The actual label for this image is: Unsafe



The prediction for this image is: ['Safe']

The actual label for this image is: Safe

Linknet with Resnet Backbone and logicsti regression

---

ds# -*- coding: utf-8 -*-

"""

Created on Fri Sep 9 20:21:51 2022

@author: LAXMINARAYANRO

"""

```
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
import os
import seaborn as sns
from tensorflow.keras.applications.vgg16 import VGG16
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 224
```

```
#Channel=3  #Resize images


#lists
train_images = []
train_labels = []


for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)


#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)
```

```
# test/validation into lists


test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)


# text to integers.
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded


# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

from segmentation_models import Linknet
from segmentation_models import get_preprocessing
from segmentation_models.losses import bce_jaccard_loss
```

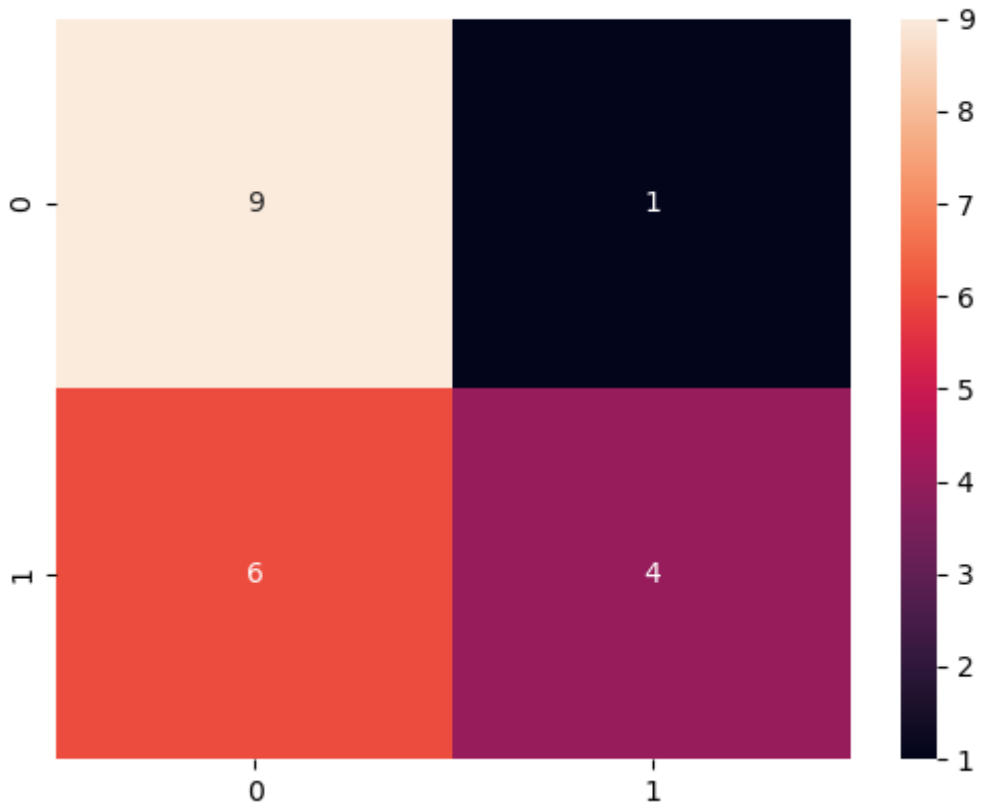
```
from segmentation_models.metrics import iou_score

BACKBONE = 'resnet34'
preprocess_input = get_preprocessing(BACKBONE)




# define model
model = Linknet(BACKBONE, input_shape=(224, 224, 3),
encoder_weights=None)
model.compile('Adam', loss=bce_jaccard_loss,
metrics=[iou_score])

#Now, let us use features from convolutional network for RF
feature_extractor=model.predict(x_train)

features =
feature_extractor.reshape(feature_extractor.shape[0], -1)

X_for_Logreg = features #This is our X input to RF
from sklearn.linear_model import LogisticRegression
```

```
Logreg_model = LogisticRegression(random_state=1)

# #RANDOM FOREST
# from sklearn.ensemble import RandomForestClassifier
# RF_model = RandomForestClassifier(n_estimators = 50,
random_state = 42)

# Train the model on training data
Logreg_model.fit(X_for_Logreg, y_train) #For sklearn no one
hot encoding

#Send test data through same feature extractor process
X_test_feature = model.predict(x_test)
X_test_features =
X_test_feature.reshape(X_test_feature.shape[0], -1)

#Now predict using the trained RF model.
prediction_Logreg = Logreg_model.predict(X_test_features)
#Inverse le transform to get original label back.
prediction_Logreg = le.inverse_transform(prediction_Logreg)

#Print overall accuracy
```

```
from sklearn import metrics

print ("Accuracy = ", metrics.accuracy_score(test_labels, prediction_Logreg))
```

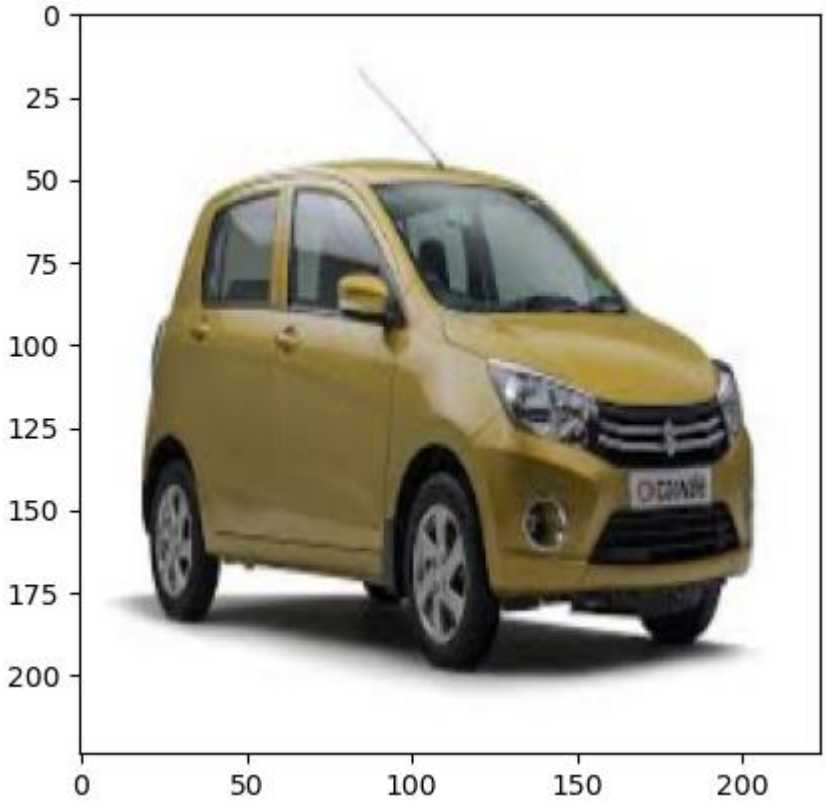
Accuracy = 0.65

```
#Confusion Matrix - verify accuracy of each class

from sklearn.metrics import confusion_matrix


cm = confusion_matrix(test_labels, prediction_Logreg)

#print(cm)

sns.heatmap(cm, annot=True)
```




```
#Check results on a few select images
```

```
n=np.random.randint(0, x_test.shape[0])

img = x_test[n]

plt.imshow(img)

input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)

input_img_feature=model.predict(input_img)

input_img_features=input_img_feature.reshape(input_img_feature.shape[0], -1)

prediction_Logreg = Logreg_model.predict(input_img_features)[0]

prediction_Logreg = le.inverse_transform([prediction_Logreg])  #Reverse the label encoder to original name

print("The prediction for this image is: ", prediction_Logreg)

print("The actual label for this image is: ", test_labels[n])
```



The prediction for this image is: ['Safe']

The actual label for this image is: Unsafe

The prediction for this image is: ['Safe']

The actual label for this image is: Safe

PSPNet with Inceptionv3 backbone and GausianNB

MobileNetV2 with Histogram Gradient Boosting

-*- coding: utf-8 -*-

"""

```
Created on Tue Aug 23 15:11:46 2022

@author: LAXMINARAYANRO
"""

# -*- coding: utf-8 -*-
"""
Created on Mon Aug 22 22:28:55 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
#from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
import os
```

```
import seaborn as sns
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2



# Read input images and assign labels based on folder names
print(os.listdir("E:/fffiiles/college pracs and projects/Project/Car"))

SIZE = 256 #Resize images

#Capture training data and labels into respective lists
train_images = []
train_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
```

```
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
```

```
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    test_images.append(img)
    test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data into test and train datasets (already split but
assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded

##################################################
#################
# Normalize pixel values to between 0 and 1
```

```
x_train, x_test = x_train / 255.0, x_test / 255.0

#One hot encode y values for neural network.
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

##############################
#Load model wothout classifier/fully connected layers
MobileNetV2_model = MobileNetV2(weights='imagenet',
include_top=False, input_shape=(SIZE, SIZE, 3))

#Make loaded layers as non-trainable. This is important as
we want to work with pre-trained weights
for layer in MobileNetV2_model.layers:
    layer.trainable = False

MobileNetV2_model.summary()  #Trainable parameters will
be 0
_________________________________________________________________
__________________________________________________________
 Layer (type)                Output Shape         Param #
Connected to
```

```
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 Conv1 (Conv2D)                 (None, 128, 128, 32  864         ['input_1[0][0]']
                                )

 bn_Conv1 (BatchNormalization)  (None, 128, 128, 32  128         ['Conv1[0][0]']
                                )

 Conv1_relu (ReLU)              (None, 128, 128, 32  0           ['bn_Conv1[0][0]']
                                )

 expanded_conv_depthwise (Depth  (None, 128, 128, 32  288        ['Conv1_relu[0][0]']
 wiseConv2D)                    )

 expanded_conv_depthwise_BN (Ba  (None, 128, 128, 32  128        ['expanded_conv_depthwise[0][0]']
```

tchNormalization))

expanded_conv_depthwise_relu ((None, 128, 128, 32 0
['expanded_conv_depthwise_BN[0][0
ReLU))]']

expanded_conv_project (Conv2D) (None, 128, 128, 16 512
['expanded_conv_depthwise_relu[0]
) [0]']

expanded_conv_project_BN (Batc (None, 128, 128, 16 64
['expanded_conv_project[0][0]']
hNormalization))

block_1_expand (Conv2D) (None, 128, 128, 96 1536
['expanded_conv_project_BN[0][0]'
)]

block_1_expand_BN (BatchNormal (None, 128, 128, 96 384
['block_1_expand[0][0]']
ization))

block_1_expand_relu (ReLU) (None, 128, 128, 96 0
['block_1_expand_BN[0][0]']
)

block_1_pad (ZeroPadding2D) (None, 129, 129, 96 0
['block_1_expand_relu[0][0]']
)

block_1_depthwise (DepthwiseCo (None, 64, 64, 96) 864
['block_1_pad[0][0]']
nv2D)

block_1_depthwise_BN (BatchNor (None, 64, 64, 96) 384
['block_1_depthwise[0][0]']
malization)

block_1_depthwise_relu (ReLU) (None, 64, 64, 96) 0
['block_1_depthwise_BN[0][0]']

block_1_project (Conv2D) (None, 64, 64, 24) 2304
['block_1_depthwise_relu[0][0]']

block_1_project_BN (BatchNorma (None, 64, 64, 24) 96
['block_1_project[0][0]']

lization)

block_2_expand (Conv2D) (None, 64, 64, 144) 3456
['block_1_project_BN[0][0]']

block_2_expand_BN (BatchNormal (None, 64, 64, 144) 576
['block_2_expand[0][0]']

ization)

block_2_expand_relu (ReLU) (None, 64, 64, 144) 0
['block_2_expand_BN[0][0]']

block_2_depthwise (DepthwiseCo (None, 64, 64, 144) 1296
['block_2_expand_relu[0][0]']

nv2D)

block_2_depthwise_BN (BatchNor (None, 64, 64, 144) 576
['block_2_depthwise[0][0]']

malization)

block_2_depthwise_relu (ReLU) (None, 64, 64, 144) 0
['block_2_depthwise_BN[0][0]']

block_2_project (Conv2D) (None, 64, 64, 24) 3456
['block_2_depthwise_relu[0][0]']

block_2_project_BN (BatchNorma (None, 64, 64, 24) 96
['block_2_project[0][0]']
lization)

block_2_add (Add) (None, 64, 64, 24) 0
['block_1_project_BN[0][0]',

'block_2_project_BN[0][0]']

block_3_expand (Conv2D) (None, 64, 64, 144) 3456
['block_2_add[0][0]']

block_3_expand_BN (BatchNormal (None, 64, 64, 144) 576
['block_3_expand[0][0]']
ization)

block_3_expand_relu (ReLU) (None, 64, 64, 144) 0
['block_3_expand_BN[0][0]']

block_3_pad (ZeroPadding2D) (None, 65, 65, 144) 0
['block_3_expand_relu[0][0]']

block_3_depthwise (DepthwiseCo (None, 32, 32, 144) 1296
['block_3_pad[0][0]']

nv2D)

block_3_depthwise_BN (BatchNor (None, 32, 32, 144) 576
['block_3_depthwise[0][0]']

malization)

block_3_depthwise_relu (ReLU) (None, 32, 32, 144) 0
['block_3_depthwise_BN[0][0]']

block_3_project (Conv2D) (None, 32, 32, 32) 4608
['block_3_depthwise_relu[0][0]']

block_3_project_BN (BatchNorma (None, 32, 32, 32) 128
['block_3_project[0][0]']

lization)

block_4_expand (Conv2D) (None, 32, 32, 192) 6144
['block_3_project_BN[0][0]']

block_4_expand_BN (BatchNormal (None, 32, 32, 192) 768
['block_4_expand[0][0]']

ization)

block_4_expand_relu (ReLU) (None, 32, 32, 192) 0
['block_4_expand_BN[0][0]']

block_4_depthwise (DepthwiseCo (None, 32, 32, 192) 1728
['block_4_expand_relu[0][0]']

nv2D)

block_4_depthwise_BN (BatchNor (None, 32, 32, 192) 768
['block_4_depthwise[0][0]']

malization)

block_4_depthwise_relu (ReLU) (None, 32, 32, 192) 0
['block_4_depthwise_BN[0][0]']

block_4_project (Conv2D) (None, 32, 32, 32) 6144
['block_4_depthwise_relu[0][0]']

block_4_project_BN (BatchNorma (None, 32, 32, 32) 128
['block_4_project[0][0]']

lization)

block_4_add (Add) (None, 32, 32, 32) 0
['block_3_project_BN[0][0]',

'block_4_project_BN[0][0]']

block_5_expand (Conv2D) (None, 32, 32, 192) 6144
['block_4_add[0][0]']

block_5_expand_BN (BatchNormal (None, 32, 32, 192) 768
['block_5_expand[0][0]']

ization)

block_5_expand_relu (ReLU) (None, 32, 32, 192) 0
['block_5_expand_BN[0][0]']

block_5_depthwise (DepthwiseCo (None, 32, 32, 192) 1728
['block_5_expand_relu[0][0]']

nv2D)

block_5_depthwise_BN (BatchNor (None, 32, 32, 192) 768
['block_5_depthwise[0][0]']

malization)

block_5_depthwise_relu (ReLU) (None, 32, 32, 192) 0
['block_5_depthwise_BN[0][0]']

block_5_project (Conv2D) (None, 32, 32, 32) 6144
['block_5_depthwise_relu[0][0]']

block_5_project_BN (BatchNorma (None, 32, 32, 32) 128
['block_5_project[0][0]']
lization)

block_5_add (Add) (None, 32, 32, 32) 0
['block_4_add[0][0]',

'block_5_project_BN[0][0]']

block_6_expand (Conv2D) (None, 32, 32, 192) 6144
['block_5_add[0][0]']

block_6_expand_BN (BatchNormal (None, 32, 32, 192) 768
['block_6_expand[0][0]']
ization)

block_6_expand_relu (ReLU) (None, 32, 32, 192) 0
['block_6_expand_BN[0][0]']

block_6_pad (ZeroPadding2D) (None, 33, 33, 192) 0
['block_6_expand_relu[0][0]']

block_6_depthwise (DepthwiseCo (None, 16, 16, 192) 1728
['block_6_pad[0][0]']

nv2D)

block_6_depthwise_BN (BatchNor (None, 16, 16, 192) 768
['block_6_depthwise[0][0]']

malization)

block_6_depthwise_relu (ReLU) (None, 16, 16, 192) 0
['block_6_depthwise_BN[0][0]']

block_6_project (Conv2D) (None, 16, 16, 64) 12288
['block_6_depthwise_relu[0][0]']

block_6_project_BN (BatchNorma (None, 16, 16, 64) 256
['block_6_project[0][0]']

lization)

block_7_expand (Conv2D) (None, 16, 16, 384) 24576
['block_6_project_BN[0][0]']

block_7_expand_BN (BatchNormal (None, 16, 16, 384) 1536 ['block_7_expand[0][0]']

ization)

block_7_expand_relu (ReLU) (None, 16, 16, 384) 0 ['block_7_expand_BN[0][0]']

block_7_depthwise (DepthwiseCo (None, 16, 16, 384) 3456 ['block_7_expand_relu[0][0]']

nv2D)

block_7_depthwise_BN (BatchNor (None, 16, 16, 384) 1536 ['block_7_depthwise[0][0]']

malization)

block_7_depthwise_relu (ReLU) (None, 16, 16, 384) 0 ['block_7_depthwise_BN[0][0]']

block_7_project (Conv2D) (None, 16, 16, 64) 24576 ['block_7_depthwise_relu[0][0]']

block_7_project_BN (BatchNorma (None, 16, 16, 64) 256 ['block_7_project[0][0]']

lization)

block_7_add (Add) (None, 16, 16, 64) 0
['block_6_project_BN[0][0]',

'block_7_project_BN[0][0]']

block_8_expand (Conv2D) (None, 16, 16, 384) 24576
['block_7_add[0][0]']

block_8_expand_BN (BatchNormal (None, 16, 16, 384)
1536 ['block_8_expand[0][0]']

ization)

block_8_expand_relu (ReLU) (None, 16, 16, 384) 0
['block_8_expand_BN[0][0]']

block_8_depthwise (DepthwiseCo (None, 16, 16, 384) 3456
['block_8_expand_relu[0][0]']

nv2D)

block_8_depthwise_BN (BatchNor (None, 16, 16, 384) 1536
['block_8_depthwise[0][0]']

malization)

block_8_depthwise_relu (ReLU) (None, 16, 16, 384) 0
['block_8_depthwise_BN[0][0]']

block_8_project (Conv2D) (None, 16, 16, 64) 24576
['block_8_depthwise_relu[0][0]']

block_8_project_BN (BatchNorma (None, 16, 16, 64) 256
['block_8_project[0][0]']
lization)

block_8_add (Add) (None, 16, 16, 64) 0
['block_7_add[0][0]',

'block_8_project_BN[0][0]']

block_9_expand (Conv2D) (None, 16, 16, 384) 24576
['block_8_add[0][0]']

block_9_expand_BN (BatchNormal (None, 16, 16, 384)
1536 ['block_9_expand[0][0]']
ization)

block_9_expand_relu (ReLU) (None, 16, 16, 384) 0
['block_9_expand_BN[0][0]']

block_9_depthwise (DepthwiseCo (None, 16, 16, 384) 3456
['block_9_expand_relu[0][0]']

nv2D)

block_9_depthwise_BN (BatchNor (None, 16, 16, 384) 1536
['block_9_depthwise[0][0]']

malization)

block_9_depthwise_relu (ReLU) (None, 16, 16, 384) 0
['block_9_depthwise_BN[0][0]']

block_9_project (Conv2D) (None, 16, 16, 64) 24576
['block_9_depthwise_relu[0][0]']

block_9_project_BN (BatchNorma (None, 16, 16, 64) 256
['block_9_project[0][0]']

lization)

block_9_add (Add) (None, 16, 16, 64) 0
['block_8_add[0][0]',

'block_9_project_BN[0][0]']

block_10_expand (Conv2D) (None, 16, 16, 384) 24576
['block_9_add[0][0]']

block_10_expand_BN (BatchNorma (None, 16, 16, 384)
1536 ['block_10_expand[0][0]']

lization)

block_10_expand_relu (ReLU) (None, 16, 16, 384) 0
['block_10_expand_BN[0][0]']

block_10_depthwise (DepthwiseC (None, 16, 16, 384) 3456
['block_10_expand_relu[0][0]']

onv2D)

block_10_depthwise_BN (BatchNo (None, 16, 16, 384) 1536
['block_10_depthwise[0][0]']

rmalization)

block_10_depthwise_relu (ReLU) (None, 16, 16, 384) 0
['block_10_depthwise_BN[0][0]']

block_10_project (Conv2D) (None, 16, 16, 96) 36864
['block_10_depthwise_relu[0][0]']

block_10_project_BN (BatchNorm (None, 16, 16, 96) 384
['block_10_project[0][0]']

alization)

block_11_expand (Conv2D) (None, 16, 16, 576) 55296
['block_10_project_BN[0][0]']

block_11_expand_BN (BatchNorma (None, 16, 16, 576)
2304 ['block_11_expand[0][0]']

lization)

block_11_expand_relu (ReLU) (None, 16, 16, 576) 0
['block_11_expand_BN[0][0]']

block_11_depthwise (DepthwiseC (None, 16, 16, 576) 5184
['block_11_expand_relu[0][0]']

onv2D)

block_11_depthwise_BN (BatchNo (None, 16, 16, 576) 2304
['block_11_depthwise[0][0]']

rmalization)

block_11_depthwise_relu (ReLU) (None, 16, 16, 576) 0
['block_11_depthwise_BN[0][0]']

block_11_project (Conv2D) (None, 16, 16, 96) 55296
['block_11_depthwise_relu[0][0]']

block_11_project_BN (BatchNorm (None, 16, 16, 96) 384
['block_11_project[0][0]']
alization)

block_11_add (Add) (None, 16, 16, 96) 0
['block_10_project_BN[0][0]',

'block_11_project_BN[0][0]']

block_12_expand (Conv2D) (None, 16, 16, 576) 55296
['block_11_add[0][0]']

block_12_expand_BN (BatchNorma (None, 16, 16, 576)
2304 ['block_12_expand[0][0]']
lization)

block_12_expand_relu (ReLU) (None, 16, 16, 576) 0
['block_12_expand_BN[0][0]']

block_12_depthwise (DepthwiseC (None, 16, 16, 576) 5184
['block_12_expand_relu[0][0]']

onv2D)

block_12_depthwise_BN (BatchNo (None, 16, 16, 576) 2304
['block_12_depthwise[0][0]']

rmalization)

block_12_depthwise_relu (ReLU) (None, 16, 16, 576) 0
['block_12_depthwise_BN[0][0]']

block_12_project (Conv2D) (None, 16, 16, 96) 55296
['block_12_depthwise_relu[0][0]']

block_12_project_BN (BatchNorm (None, 16, 16, 96) 384
['block_12_project[0][0]']

alization)

block_12_add (Add) (None, 16, 16, 96) 0
['block_11_add[0][0]',

'block_12_project_BN[0][0]']

block_13_expand (Conv2D) (None, 16, 16, 576) 55296
['block_12_add[0][0]']

block_13_expand_BN (BatchNorma (None, 16, 16, 576)
2304 ['block_13_expand[0][0]']
lization)

block_13_expand_relu (ReLU) (None, 16, 16, 576) 0
['block_13_expand_BN[0][0]']

block_13_pad (ZeroPadding2D) (None, 17, 17, 576) 0
['block_13_expand_relu[0][0]']

block_13_depthwise (DepthwiseC (None, 8, 8, 576) 5184
['block_13_pad[0][0]']
onv2D)

block_13_depthwise_BN (BatchNo (None, 8, 8, 576) 2304
['block_13_depthwise[0][0]']
rmalization)

block_13_depthwise_relu (ReLU) (None, 8, 8, 576) 0
['block_13_depthwise_BN[0][0]']

block_13_project (Conv2D) (None, 8, 8, 160) 92160
['block_13_depthwise_relu[0][0]']

block_13_project_BN (BatchNorm (None, 8, 8, 160) 640
['block_13_project[0][0]']

alization)

block_14_expand (Conv2D) (None, 8, 8, 960) 153600
['block_13_project_BN[0][0]']

block_14_expand_BN (BatchNorma (None, 8, 8, 960) 3840
['block_14_expand[0][0]']

lization)

block_14_expand_relu (ReLU) (None, 8, 8, 960) 0
['block_14_expand_BN[0][0]']

block_14_depthwise (DepthwiseC (None, 8, 8, 960) 8640
['block_14_expand_relu[0][0]']

onv2D)

block_14_depthwise_BN (BatchNo (None, 8, 8, 960) 3840
['block_14_depthwise[0][0]']

rmalization)

block_14_depthwise_relu (ReLU) (None, 8, 8, 960) 0
['block_14_depthwise_BN[0][0]']

block_14_project (Conv2D) (None, 8, 8, 160) 153600
['block_14_depthwise_relu[0][0]']

block_14_project_BN (BatchNorm (None, 8, 8, 160) 640
['block_14_project[0][0]']

alization)

block_14_add (Add) (None, 8, 8, 160) 0
['block_13_project_BN[0][0]',

'block_14_project_BN[0][0]']

block_15_expand (Conv2D) (None, 8, 8, 960) 153600
['block_14_add[0][0]']

block_15_expand_BN (BatchNorma (None, 8, 8, 960) 3840
['block_15_expand[0][0]']

lization)

block_15_expand_relu (ReLU) (None, 8, 8, 960) 0
['block_15_expand_BN[0][0]']

block_15_depthwise (DepthwiseC (None, 8, 8, 960) 8640
['block_15_expand_relu[0][0]']

onv2D)

block_15_depthwise_BN (BatchNo (None, 8, 8, 960) 3840
['block_15_depthwise[0][0]']

rmalization)

block_15_depthwise_relu (ReLU) (None, 8, 8, 960) 0
['block_15_depthwise_BN[0][0]']

block_15_project (Conv2D) (None, 8, 8, 160) 153600
['block_15_depthwise_relu[0][0]']

block_15_project_BN (BatchNorm (None, 8, 8, 160) 640
['block_15_project[0][0]']

alization)

block_15_add (Add) (None, 8, 8, 160) 0
['block_14_add[0][0]',

'block_15_project_BN[0][0]']

block_16_expand (Conv2D) (None, 8, 8, 960) 153600
['block_15_add[0][0]']

block_16_expand_BN (BatchNorma (None, 8, 8, 960) 3840
['block_16_expand[0][0]']

lization)

block_16_expand_relu (ReLU) (None, 8, 8, 960) 0
['block_16_expand_BN[0][0]']

block_16_depthwise (DepthwiseC (None, 8, 8, 960) 8640
['block_16_expand_relu[0][0]']

onv2D)

block_16_depthwise_BN (BatchNo (None, 8, 8, 960) 3840
['block_16_depthwise[0][0]']

rmalization)

block_16_depthwise_relu (ReLU) (None, 8, 8, 960) 0
['block_16_depthwise_BN[0][0]']

block_16_project (Conv2D) (None, 8, 8, 320) 307200
['block_16_depthwise_relu[0][0]']

block_16_project_BN (BatchNorm (None, 8, 8, 320) 1280
['block_16_project[0][0]']
alization)

Conv_1 (Conv2D) (None, 8, 8, 1280) 409600
['block_16_project_BN[0][0]']

Conv_1_bn (BatchNormalization) (None, 8, 8, 1280) 5120
['Conv_1[0][0]']

out_relu (ReLU) (None, 8, 8, 1280) 0
['Conv_1_bn[0][0]']

==========
==========

Total params: 2,257,984

Trainable params: 0

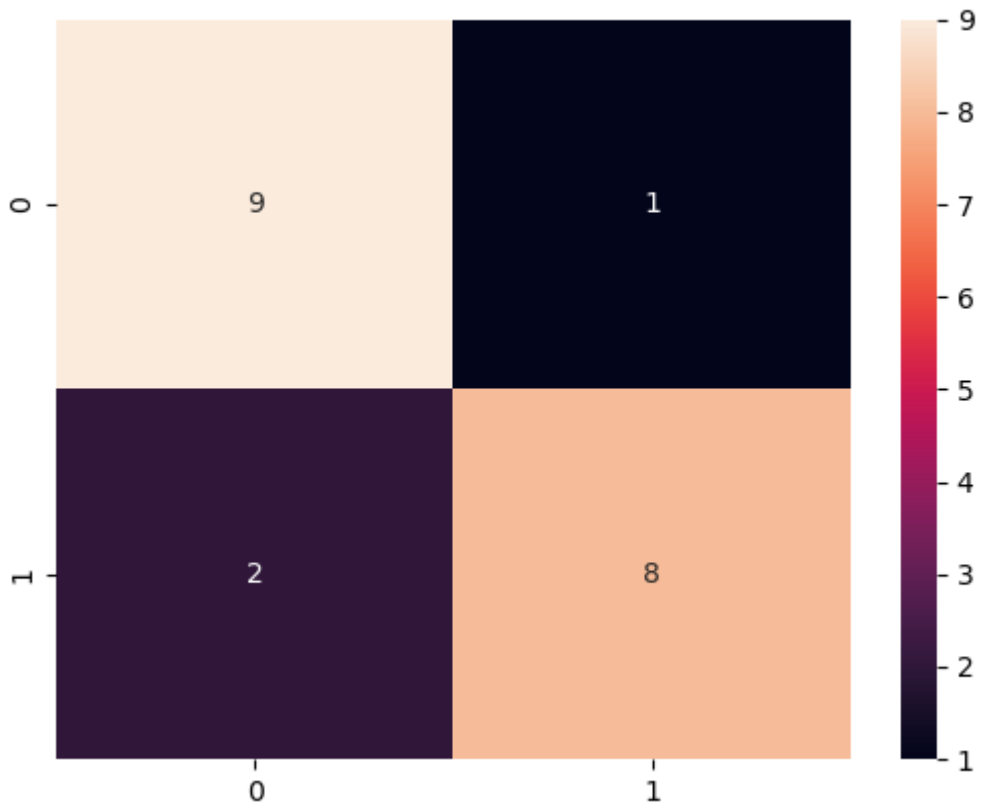
Non-trainable params: 2,257,984

```
#Now, let us use features from convolutional network for RF
feature_extractor=MobileNetV2_model.predict(x_train)

features =
feature_extractor.reshape(feature_extractor.shape[0], -1)

X_for_HistGradient = features #This is our X input to RF

#RANDOM FOREST
from sklearn.experimental import
enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier
HistGradient_model =
HistGradientBoostingClassifier(max_iter=100)

# Train the model on training data
HistGradient_model.fit(X_for_HistGradient, y_train) #For
sklearn no one hot encoding

#Send test data through same feature extractor process
```

```
X_test_feature = MobileNetV2_model.predict(x_test)

X_test_features =
X_test_feature.reshape(X_test_feature.shape[0], -1)


#Now predict using the trained RF model.

prediction_HistGradient =
HistGradient_model.predict(X_test_features)

#Inverse le transform to get original label back.

prediction_HistGradient =
le.inverse_transform(prediction_HistGradient)


#Print overall accuracy

from sklearn import metrics

print ("Accuracy = ", metrics.accuracy_score(test_labels,
prediction_HistGradient))
```

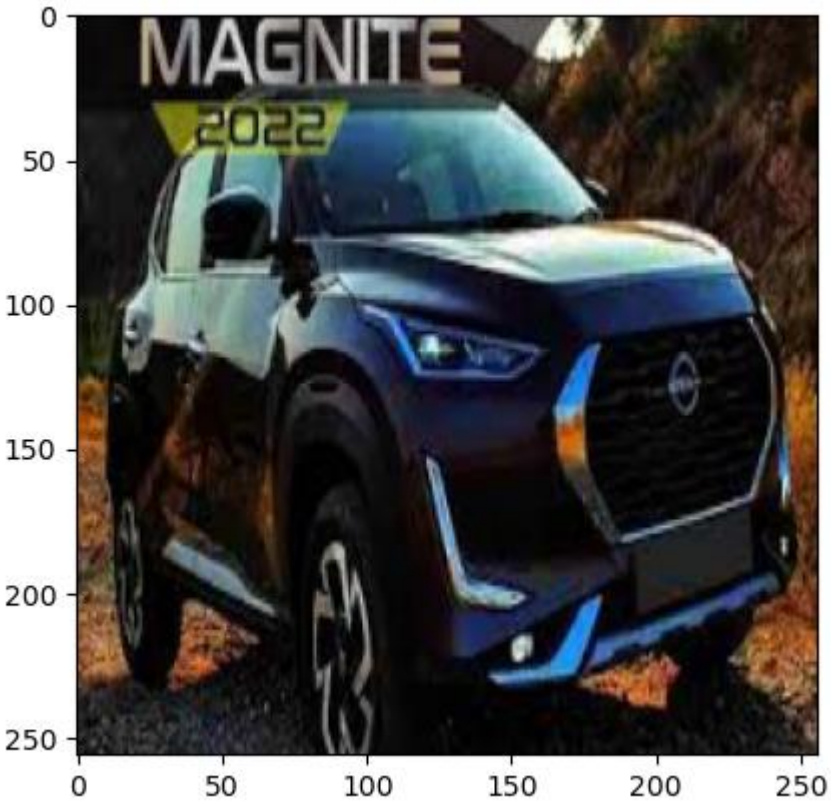
```
Accuracy =  0.85
```

```
#Confusion Matrix - verify accuracy of each class

from sklearn.metrics import confusion_matrix


cm = confusion_matrix(test_labels, prediction_HistGradient)

#print(cm)

sns.heatmap(cm, annot=True)
```



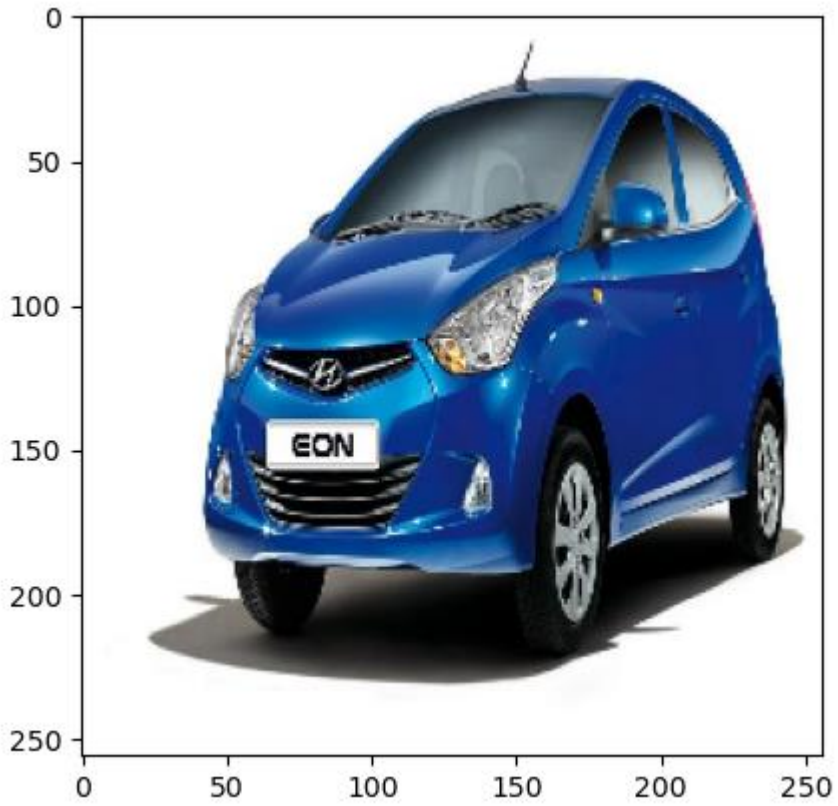

```
#Check results on a few select images

n=np.random.randint(0, x_test.shape[0])

img = x_test[n]

plt.imshow(img)

input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)

input_img_feature=MobileNetV2_model.predict(input_img)

input_img_features=input_img_feature.reshape(input_img_feature.shape[0], -1)

prediction_HistGradient = HistGradient_model.predict(input_img_features)[0]
```

```
prediction_HistGradient = le.inverse_transform([prediction_HistGradient])  #Reverse the label encoder to original name

print("The prediction for this image is: ", prediction_HistGradient)

print("The actual label for this image is: ", test_labels[n])
```



The prediction for this image is: ['Safe']

The actual label for this image is: Safe



The prediction for this image is: ['Unsafe']

The actual label for this image is: Unsafe

Xception with Gradient Boosting

-*- coding: utf-8 -*-

"""

Created on Tue Aug 23 21:01:51 2022

@author: LAXMINARAYANRO

```
"""

# -*- coding: utf-8 -*-
"""
Created on Tue Aug 16 12:57:30 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

from tensorflow.keras.models import Model, Sequential
#from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
import os
import seaborn as sns
from tensorflow.keras.layers import (
```

```
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.xception import Xception



# Read input images and assign labels based on folder names
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 256  #Resize images

#Capture training data and labels into respective lists
train_images = []
train_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
```

```
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)
```

```
test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data into test and train datasets (already split but
assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded

##################################################################
# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

#One hot encode y values for neural network.
```

```
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

##############################
#Load model wothout classifier/fully connected layers
Xception_model = Xception(weights='imagenet',
include_top=False, input_shape=(SIZE, SIZE, 3))

#Make loaded layers as non-trainable. This is important as
we want to work with pre-trained weights
for layer in Xception_model.layers:
    layer.trainable = False

Xception_model.summary()  #Trainable parameters will be 0
```

```
Layer (type)                    Output Shape         Param #
Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]
```

block1_conv1 (Conv2D) (None, 127, 127, 32 864 ['input_1[0][0]']
)

block1_conv1_bn (BatchNormaliz (None, 127, 127, 32 128 ['block1_conv1[0][0]']
ation))

block1_conv1_act (Activation) (None, 127, 127, 32 0 ['block1_conv1_bn[0][0]']
)

block1_conv2 (Conv2D) (None, 125, 125, 64 18432 ['block1_conv1_act[0][0]']
)

block1_conv2_bn (BatchNormaliz (None, 125, 125, 64 256 ['block1_conv2[0][0]']
ation))

block1_conv2_act (Activation) (None, 125, 125, 64 0 ['block1_conv2_bn[0][0]']
)

block2_sepconv1 (SeparableConv (None, 125, 125, 12 8768
['block1_conv2_act[0][0]']

2D) 8)

block2_sepconv1_bn (BatchNorma (None, 125, 125, 12 512
['block2_sepconv1[0][0]']

lization) 8)

block2_sepconv2_act (Activatio (None, 125, 125, 12 0
['block2_sepconv1_bn[0][0]']

n) 8)

block2_sepconv2 (SeparableConv (None, 125, 125, 12
17536 ['block2_sepconv2_act[0][0]']

2D) 8)

block2_sepconv2_bn (BatchNorma (None, 125, 125, 12 512
['block2_sepconv2[0][0]']

lization) 8)

conv2d (Conv2D) (None, 63, 63, 128) 8192
['block1_conv2_act[0][0]']

block2_pool (MaxPooling2D) (None, 63, 63, 128) 0
['block2_sepconv2_bn[0][0]']

batch_normalization (BatchNorm (None, 63, 63, 128) 512
['conv2d[0][0]']
alization)

add (Add) (None, 63, 63, 128) 0
['block2_pool[0][0]',

'batch_normalization[0][0]']

block3_sepconv1_act (Activatio (None, 63, 63, 128) 0
['add[0][0]']
n)

block3_sepconv1 (SeparableConv (None, 63, 63, 256) 33920
['block3_sepconv1_act[0][0]']
2D)

block3_sepconv1_bn (BatchNorma (None, 63, 63, 256)
1024 ['block3_sepconv1[0][0]']
lization)

block3_sepconv2_act (Activatio (None, 63, 63, 256) 0
['block3_sepconv1_bn[0][0]']
n)

block3_sepconv2 (SeparableConv (None, 63, 63, 256) 67840
['block3_sepconv2_act[0][0]']
2D)

block3_sepconv2_bn (BatchNorma (None, 63, 63, 256)
1024 ['block3_sepconv2[0][0]']
lization)

conv2d_1 (Conv2D) (None, 32, 32, 256) 32768
['add[0][0]']

block3_pool (MaxPooling2D) (None, 32, 32, 256) 0
['block3_sepconv2_bn[0][0]']

batch_normalization_1 (BatchNo (None, 32, 32, 256) 1024
['conv2d_1[0][0]']
rmalization)

add_1 (Add) (None, 32, 32, 256) 0
['block3_pool[0][0]',

'batch_normalization_1[0][0]']

block4_sepconv1_act (Activatio (None, 32, 32, 256) 0
['add_1[0][0]']
n)

block4_sepconv1 (SeparableConv (None, 32, 32, 728)
188672 ['block4_sepconv1_act[0][0]']
2D)

block4_sepconv1_bn (BatchNorma (None, 32, 32, 728)
2912 ['block4_sepconv1[0][0]']
lization)

block4_sepconv2_act (Activatio (None, 32, 32, 728) 0
['block4_sepconv1_bn[0][0]']
n)

block4_sepconv2 (SeparableConv (None, 32, 32, 728)
536536 ['block4_sepconv2_act[0][0]']
2D)

block4_sepconv2_bn (BatchNorma (None, 32, 32, 728) 2912 ['block4_sepconv2[0][0]']
lization)

conv2d_2 (Conv2D) (None, 16, 16, 728) 186368 ['add_1[0][0]']

block4_pool (MaxPooling2D) (None, 16, 16, 728) 0 ['block4_sepconv2_bn[0][0]']

batch_normalization_2 (BatchNo (None, 16, 16, 728) 2912 ['conv2d_2[0][0]']
rmalization)

add_2 (Add) (None, 16, 16, 728) 0 ['block4_pool[0][0]',

'batch_normalization_2[0][0]']

block5_sepconv1_act (Activatio (None, 16, 16, 728) 0 ['add_2[0][0]']
n)

block5_sepconv1 (SeparableConv (None, 16, 16, 728)
536536 ['block5_sepconv1_act[0][0]']

2D)

block5_sepconv1_bn (BatchNorma (None, 16, 16, 728)
2912 ['block5_sepconv1[0][0]']

lization)

block5_sepconv2_act (Activatio (None, 16, 16, 728) 0
['block5_sepconv1_bn[0][0]']

n)

block5_sepconv2 (SeparableConv (None, 16, 16, 728)
536536 ['block5_sepconv2_act[0][0]']

2D)

block5_sepconv2_bn (BatchNorma (None, 16, 16, 728)
2912 ['block5_sepconv2[0][0]']

lization)

block5_sepconv3_act (Activatio (None, 16, 16, 728) 0
['block5_sepconv2_bn[0][0]']

n)

block5_sepconv3 (SeparableConv (None, 16, 16, 728)
536536 ['block5_sepconv3_act[0][0]']
2D)

block5_sepconv3_bn (BatchNorma (None, 16, 16, 728)
2912 ['block5_sepconv3[0][0]']
lization)

add_3 (Add) (None, 16, 16, 728) 0
['block5_sepconv3_bn[0][0]',
'add_2[0][0]']

block6_sepconv1_act (Activatio (None, 16, 16, 728) 0
['add_3[0][0]']
n)

block6_sepconv1 (SeparableConv (None, 16, 16, 728)
536536 ['block6_sepconv1_act[0][0]']
2D)

block6_sepconv1_bn (BatchNorma (None, 16, 16, 728)
2912 ['block6_sepconv1[0][0]']

lization)

block6_sepconv2_act (Activatio (None, 16, 16, 728) 0
['block6_sepconv1_bn[0][0]']

n)

block6_sepconv2 (SeparableConv (None, 16, 16, 728)
536536 ['block6_sepconv2_act[0][0]']

2D)

block6_sepconv2_bn (BatchNorma (None, 16, 16, 728)
2912 ['block6_sepconv2[0][0]']

lization)

block6_sepconv3_act (Activatio (None, 16, 16, 728) 0
['block6_sepconv2_bn[0][0]']

n)

block6_sepconv3 (SeparableConv (None, 16, 16, 728)
536536 ['block6_sepconv3_act[0][0]']

2D)

block6_sepconv3_bn (BatchNorma (None, 16, 16, 728)
2912 ['block6_sepconv3[0][0]']
lization)

add_4 (Add) (None, 16, 16, 728) 0
['block6_sepconv3_bn[0][0]',
'add_3[0][0]']

block7_sepconv1_act (Activatio (None, 16, 16, 728) 0
['add_4[0][0]']
n)

block7_sepconv1 (SeparableConv (None, 16, 16, 728)
536536 ['block7_sepconv1_act[0][0]']
2D)

block7_sepconv1_bn (BatchNorma (None, 16, 16, 728)
2912 ['block7_sepconv1[0][0]']
lization)

block7_sepconv2_act (Activatio (None, 16, 16, 728) 0
['block7_sepconv1_bn[0][0]']
n)

block7_sepconv2 (SeparableConv (None, 16, 16, 728)
536536 ['block7_sepconv2_act[0][0]']

2D)

block7_sepconv2_bn (BatchNorma (None, 16, 16, 728)
2912 ['block7_sepconv2[0][0]']

lization)

block7_sepconv3_act (Activatio (None, 16, 16, 728) 0
['block7_sepconv2_bn[0][0]']

n)

block7_sepconv3 (SeparableConv (None, 16, 16, 728)
536536 ['block7_sepconv3_act[0][0]']

2D)

block7_sepconv3_bn (BatchNorma (None, 16, 16, 728)
2912 ['block7_sepconv3[0][0]']

lization)

add_5 (Add) (None, 16, 16, 728) 0
['block7_sepconv3_bn[0][0]',

'add_4[0][0]']

block8_sepconv1_act (Activatio (None, 16, 16, 728) 0
['add_5[0][0]']
n)

block8_sepconv1 (SeparableConv (None, 16, 16, 728)
536536 ['block8_sepconv1_act[0][0]']
2D)

block8_sepconv1_bn (BatchNorma (None, 16, 16, 728)
2912 ['block8_sepconv1[0][0]']
lization)

block8_sepconv2_act (Activatio (None, 16, 16, 728) 0
['block8_sepconv1_bn[0][0]']
n)

block8_sepconv2 (SeparableConv (None, 16, 16, 728)
536536 ['block8_sepconv2_act[0][0]']
2D)

block8_sepconv2_bn (BatchNorma (None, 16, 16, 728)
2912 ['block8_sepconv2[0][0]']
lization)

block8_sepconv3_act (Activatio (None, 16, 16, 728) 0
['block8_sepconv2_bn[0][0]']
n)

block8_sepconv3 (SeparableConv (None, 16, 16, 728)
536536 ['block8_sepconv3_act[0][0]']
2D)

block8_sepconv3_bn (BatchNorma (None, 16, 16, 728)
2912 ['block8_sepconv3[0][0]']
lization)

add_6 (Add) (None, 16, 16, 728) 0
['block8_sepconv3_bn[0][0]',
'add_5[0][0]']

block9_sepconv1_act (Activatio (None, 16, 16, 728) 0
['add_6[0][0]']
n)

block9_sepconv1 (SeparableConv (None, 16, 16, 728)
536536 ['block9_sepconv1_act[0][0]']

2D)

block9_sepconv1_bn (BatchNorma (None, 16, 16, 728)
2912 ['block9_sepconv1[0][0]']

lization)

block9_sepconv2_act (Activatio (None, 16, 16, 728) 0
['block9_sepconv1_bn[0][0]']

n)

block9_sepconv2 (SeparableConv (None, 16, 16, 728)
536536 ['block9_sepconv2_act[0][0]']

2D)

block9_sepconv2_bn (BatchNorma (None, 16, 16, 728)
2912 ['block9_sepconv2[0][0]']

lization)

block9_sepconv3_act (Activatio (None, 16, 16, 728) 0
['block9_sepconv2_bn[0][0]']

n)

block9_sepconv3 (SeparableConv (None, 16, 16, 728)
536536 ['block9_sepconv3_act[0][0]']

2D)

block9_sepconv3_bn (BatchNorma (None, 16, 16, 728)
2912 ['block9_sepconv3[0][0]']

lization)

add_7 (Add) (None, 16, 16, 728) 0
['block9_sepconv3_bn[0][0]',

'add_6[0][0]']

block10_sepconv1_act (Activati (None, 16, 16, 728) 0
['add_7[0][0]']

on)

block10_sepconv1 (SeparableCon (None, 16, 16, 728)
536536 ['block10_sepconv1_act[0][0]']

v2D)

block10_sepconv1_bn (BatchNorm (None, 16, 16, 728)
2912 ['block10_sepconv1[0][0]']

alization)

block10_sepconv2_act (Activati (None, 16, 16, 728) 0
['block10_sepconv1_bn[0][0]']

on)

block10_sepconv2 (SeparableCon (None, 16, 16, 728)
536536 ['block10_sepconv2_act[0][0]']

v2D)

block10_sepconv2_bn (BatchNorm (None, 16, 16, 728)
2912 ['block10_sepconv2[0][0]']

alization)

block10_sepconv3_act (Activati (None, 16, 16, 728) 0
['block10_sepconv2_bn[0][0]']

on)

block10_sepconv3 (SeparableCon (None, 16, 16, 728)
536536 ['block10_sepconv3_act[0][0]']

v2D)

block10_sepconv3_bn (BatchNorm (None, 16, 16, 728) 2912 ['block10_sepconv3[0][0]']
alization)

add_8 (Add) (None, 16, 16, 728) 0 ['block10_sepconv3_bn[0][0]',
'add_7[0][0]']

block11_sepconv1_act (Activati (None, 16, 16, 728) 0 ['add_8[0][0]']
on)

block11_sepconv1 (SeparableCon (None, 16, 16, 728) 536536 ['block11_sepconv1_act[0][0]']
v2D)

block11_sepconv1_bn (BatchNorm (None, 16, 16, 728) 2912 ['block11_sepconv1[0][0]']
alization)

block11_sepconv2_act (Activati (None, 16, 16, 728) 0 ['block11_sepconv1_bn[0][0]']

on)

block11_sepconv2 (SeparableCon (None, 16, 16, 728)
536536 ['block11_sepconv2_act[0][0]']

v2D)

block11_sepconv2_bn (BatchNorm (None, 16, 16, 728)
2912 ['block11_sepconv2[0][0]']

alization)

block11_sepconv3_act (Activati (None, 16, 16, 728) 0
['block11_sepconv2_bn[0][0]']

on)

block11_sepconv3 (SeparableCon (None, 16, 16, 728)
536536 ['block11_sepconv3_act[0][0]']

v2D)

block11_sepconv3_bn (BatchNorm (None, 16, 16, 728)
2912 ['block11_sepconv3[0][0]']

alization)

add_9 (Add) (None, 16, 16, 728) 0
['block11_sepconv3_bn[0][0]',
'add_8[0][0]']

block12_sepconv1_act (Activati (None, 16, 16, 728) 0
['add_9[0][0]']
on)

block12_sepconv1 (SeparableCon (None, 16, 16, 728)
536536 ['block12_sepconv1_act[0][0]']
v2D)

block12_sepconv1_bn (BatchNorm (None, 16, 16, 728)
2912 ['block12_sepconv1[0][0]']
alization)

block12_sepconv2_act (Activati (None, 16, 16, 728) 0
['block12_sepconv1_bn[0][0]']
on)

block12_sepconv2 (SeparableCon (None, 16, 16, 728)
536536 ['block12_sepconv2_act[0][0]']
v2D)

block12_sepconv2_bn (BatchNorm (None, 16, 16, 728)
2912 ['block12_sepconv2[0][0]']
alization)

block12_sepconv3_act (Activati (None, 16, 16, 728) 0
['block12_sepconv2_bn[0][0]']
on)

block12_sepconv3 (SeparableCon (None, 16, 16, 728)
536536 ['block12_sepconv3_act[0][0]']
v2D)

block12_sepconv3_bn (BatchNorm (None, 16, 16, 728)
2912 ['block12_sepconv3[0][0]']
alization)

add_10 (Add) (None, 16, 16, 728) 0
['block12_sepconv3_bn[0][0]',
'add_9[0][0]']

block13_sepconv1_act (Activati (None, 16, 16, 728) 0
['add_10[0][0]']

on)

block13_sepconv1 (SeparableCon (None, 16, 16, 728)
536536 ['block13_sepconv1_act[0][0]']

v2D)

block13_sepconv1_bn (BatchNorm (None, 16, 16, 728)
2912 ['block13_sepconv1[0][0]']

alization)

block13_sepconv2_act (Activati (None, 16, 16, 728) 0
['block13_sepconv1_bn[0][0]']

on)

block13_sepconv2 (SeparableCon (None, 16, 16, 1024
752024 ['block13_sepconv2_act[0][0]']

v2D))

block13_sepconv2_bn (BatchNorm (None, 16, 16, 1024
4096 ['block13_sepconv2[0][0]']

alization))

conv2d_3 (Conv2D) (None, 8, 8, 1024) 745472
['add_10[0][0]']

block13_pool (MaxPooling2D) (None, 8, 8, 1024) 0
['block13_sepconv2_bn[0][0]']

batch_normalization_3 (BatchNo (None, 8, 8, 1024) 4096
['conv2d_3[0][0]']

rmalization)

add_11 (Add) (None, 8, 8, 1024) 0
['block13_pool[0][0]',

'batch_normalization_3[0][0]']

block14_sepconv1 (SeparableCon (None, 8, 8, 1536)
1582080 ['add_11[0][0]']

v2D)

block14_sepconv1_bn (BatchNorm (None, 8, 8, 1536) 6144
['block14_sepconv1[0][0]']

alization)

block14_sepconv1_act (Activati (None, 8, 8, 1536) 0 ['block14_sepconv1_bn[0][0]']

on)

block14_sepconv2 (SeparableCon (None, 8, 8, 2048) 3159552 ['block14_sepconv1_act[0][0]']

v2D)

block14_sepconv2_bn (BatchNorm (None, 8, 8, 2048) 8192 ['block14_sepconv2[0][0]']

alization)

block14_sepconv2_act (Activati (None, 8, 8, 2048) 0 ['block14_sepconv2_bn[0][0]']

on)

==========

Total params: 20,861,480

Trainable params: 0

Non-trainable params: 20,861,480

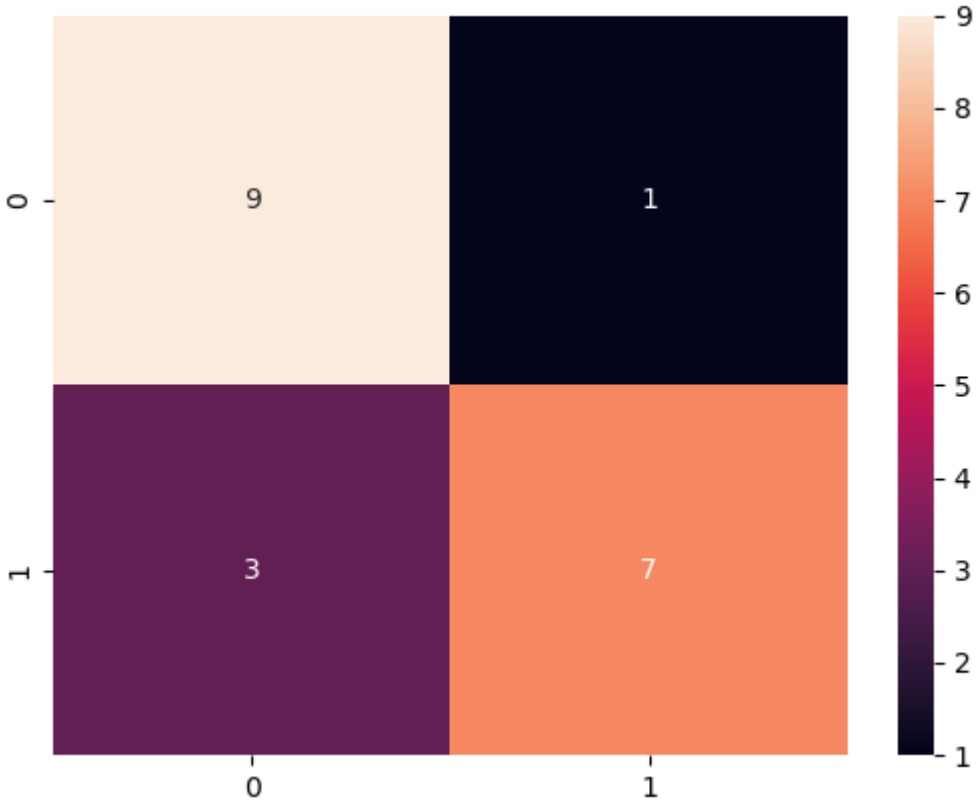
____

```
#Now, let us use features from convolutional network for RF
feature_extractor=Xception_model.predict(x_train)

features =
feature_extractor.reshape(feature_extractor.shape[0], -1)

X_for_Grad = features #This is our X input to RF

#RANDOM FOREST
from sklearn.ensemble import GradientBoostingClassifier
Grad_model = GradientBoostingClassifier(n_estimators=50,
learning_rate=1.0,max_depth=1, random_state=42)

# Train the model on training data
Grad_model.fit(X_for_Grad, y_train) #For sklearn no one hot
encoding

#Send test data through same feature extractor process
X_test_feature = Xception_model.predict(x_test)
X_test_features =
X_test_feature.reshape(X_test_feature.shape[0], -1)

#Now predict using the trained RF model.
```

```
prediction_Grad = Grad_model.predict(X_test_features)
#Inverse le transform to get original label back.
prediction_Grad = le.inverse_transform(prediction_Grad)

#Print overall accuracy
from sklearn import metrics
print ("Accuracy = ", metrics.accuracy_score(test_labels,
prediction_Grad))
```

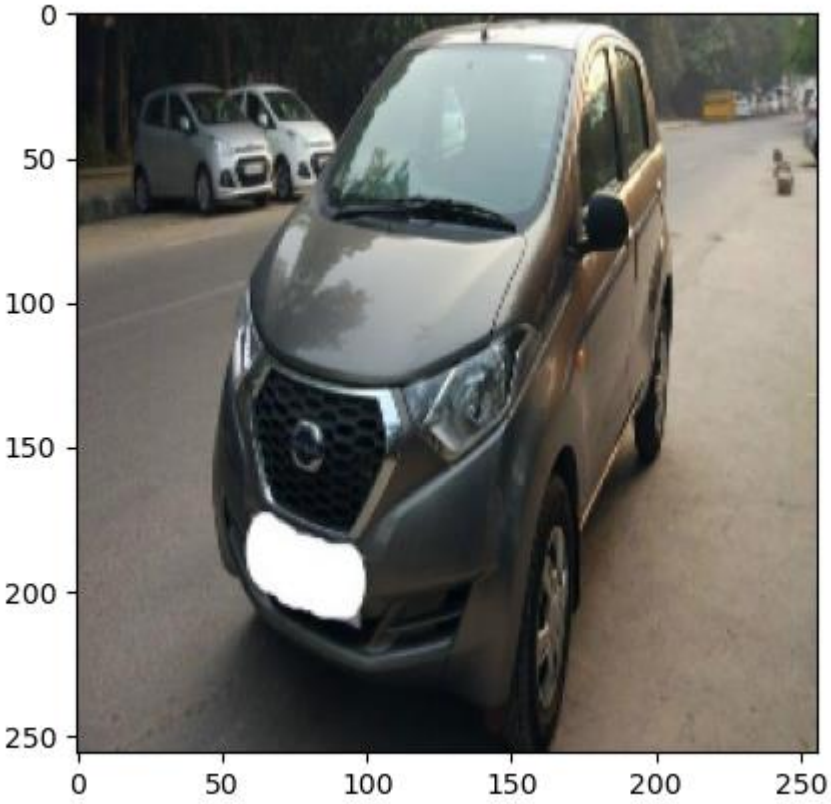
Accuracy = 0.8

```
#Confusion Matrix - verify accuracy of each class
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_labels, prediction_Grad)
#print(cm)
sns.heatmap(cm, annot=True)
```



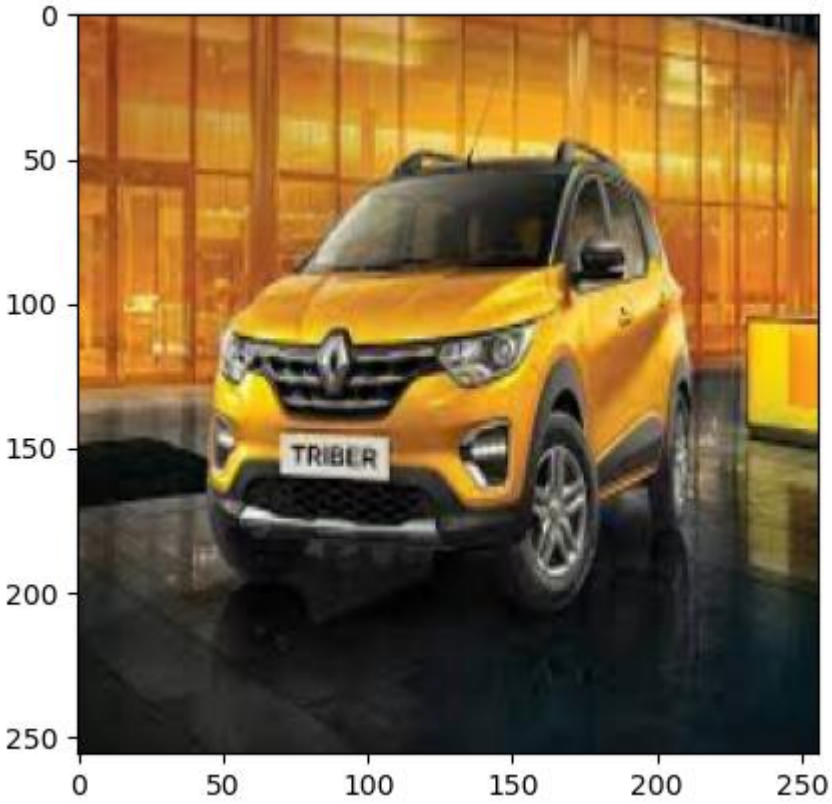

```
#Check results on a few select images

n=np.random.randint(0, x_test.shape[0])

img = x_test[n]

plt.imshow(img)

input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)

input_img_feature=Xception_model.predict(input_img)

input_img_features=input_img_feature.reshape(input_img_feature.shape[0], -1)

prediction_Grad = Grad_model.predict(input_img_features)[0]
```

```
prediction_Grad = le.inverse_transform([prediction_Grad])
#Reverse the label encoder to original name
print("The prediction for this image is: ", prediction_Grad)
print("The actual label for this image is: ", test_labels[n])
```



The prediction for this image is: ['Unsafe']

The actual label for this image is: Unsafe



The prediction for this image is: ['Safe']

The actual label for this image is: Safe

VGG16

```
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
```

```
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
import os
import seaborn as sns
from tensorflow.keras.applications.vgg16 import VGG16
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 224
#Channel=3  #Resize images

#lists
train_images = []
train_labels = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists


test_images = []
test_labels = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)
```

```
#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded



# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

vgg_model = Sequential()

pretrained_model=
tf.keras.applications.VGG16(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False
```

```
vgg_model.add(pretrained_model)

vgg_model.add(Flatten())
vgg_model.add(Dense(512, activation='relu'))
vgg_model.add(Dense(2, activation='softmax'))

vgg_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 512)               14714688

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 512)               262656

 dense_1 (Dense)             (None, 2)                 1026
```

==========
==========

Total params: 14,978,370

Trainable params: 263,682

Non-trainable params: 14,714,688

_____
_____

```
vgg_model.compile(optimizer=Adam(learning_rate=0.001),loss='categorical_crossentropy',metrics=['accuracy'])


#history = model.fit(lung_train, infect_train, epochs = 20, validation_data = (lung_test, infect_test))


#epochs=10
history=vgg_model.fit(x_train, y_train_one_hot, epochs=40, validation_data = (x_test, y_test_one_hot))
```

Epoch 1/40

2022-08-26 20:11:37.238541: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-08-26 20:11:37.297446: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850]

Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-08-26 20:11:37.717506: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-08-26 20:12:16.053674: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:11:37.238541: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-08-26 20:11:37.297446: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at

https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-08-26 20:11:37.717506: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-08-26 20:12:16.053674: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:12:16.743009: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:11:37.238541: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-08-26 20:11:37.297446: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the

missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-08-26 20:11:37.717506: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-08-26 20:12:16.053674: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:12:16.743009: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:12:32.054077: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 205520896 exceeds 10% of free system memory.

2022-08-26 20:11:37.238541: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-08-26 20:11:37.297446: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-08-26 20:11:37.717506: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-08-26 20:12:16.053674: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:12:16.743009: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:12:32.054077: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 205520896 exceeds 10% of free system memory.

2022-08-26 20:12:33.885567: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 205520896 exceeds 10% of free system memory.

1/16 [>.....] - ETA: 7:57 - loss: 0.6695 - accuracy: 0.6250

2022-08-26 20:11:37.238541: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-08-26 20:11:37.297446: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-08-26 20:11:37.717506: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-08-26 20:12:16.053674: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:12:16.743009: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

2022-08-26 20:12:32.054077: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 205520896 exceeds 10% of free system memory.

2022-08-26 20:12:33.885567: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 205520896 exceeds 10% of free system memory.

2022-08-26 20:12:45.798111: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 411041792 exceeds 10% of free system memory.

16/16 [==============================] - 213s 12s/step - loss: 0.6504 - accuracy: 0.6305 - val_loss: 0.6042 - val_accuracy: 0.6500

Epoch 2/40

```
16/16 [==============================] - 178s 11s/step
- loss: 0.5570 - accuracy: 0.7369 - val_loss: 0.7287 -
val_accuracy: 0.5000

Epoch 3/40

16/16 [==============================] - 159s 10s/step
- loss: 0.5123 - accuracy: 0.7631 - val_loss: 0.5175 -
val_accuracy: 0.8000

Epoch 4/40

16/16 [==============================] - 179s 11s/step
- loss: 0.4877 - accuracy: 0.7671 - val_loss: 0.4922 -
val_accuracy: 0.8000

Epoch 5/40

16/16 [==============================] - 165s 10s/step
- loss: 0.4586 - accuracy: 0.7972 - val_loss: 0.6063 -
val_accuracy: 0.6000

Epoch 6/40

16/16 [==============================] - 173s 11s/step
- loss: 0.4486 - accuracy: 0.7871 - val_loss: 0.4545 -
val_accuracy: 0.8500

Epoch 7/40

16/16 [==============================] - 160s 10s/step
- loss: 0.4029 - accuracy: 0.8394 - val_loss: 0.4670 -
val_accuracy: 0.8000

Epoch 8/40
```

```
16/16 [==============================] - 178s 11s/step
- loss: 0.3789 - accuracy: 0.8534 - val_loss: 0.5142 -
val_accuracy: 0.7000

Epoch 9/40

16/16 [==============================] - 163s 10s/step
- loss: 0.3664 - accuracy: 0.8434 - val_loss: 0.4965 -
val_accuracy: 0.7000

Epoch 10/40

16/16 [==============================] - 156s 10s/step
- loss: 0.3851 - accuracy: 0.8193 - val_loss: 0.4390 -
val_accuracy: 0.8000

Epoch 11/40

16/16 [==============================] - 176s 11s/step
- loss: 0.3345 - accuracy: 0.8614 - val_loss: 0.4382 -
val_accuracy: 0.7500

Epoch 12/40

16/16 [==============================] - 186s 12s/step
- loss: 0.3272 - accuracy: 0.8735 - val_loss: 0.5018 -
val_accuracy: 0.7000

Epoch 13/40

16/16 [==============================] - 182s 11s/step
- loss: 0.3093 - accuracy: 0.8675 - val_loss: 0.4575 -
val_accuracy: 0.7500

Epoch 14/40
```

```
16/16 [==============================] - 185s 12s/step
- loss: 0.2902 - accuracy: 0.8795 - val_loss: 0.4380 -
val_accuracy: 0.7500

Epoch 15/40

16/16 [==============================] - 181s 11s/step
- loss: 0.2762 - accuracy: 0.8956 - val_loss: 0.3871 -
val_accuracy: 0.8500

Epoch 16/40

16/16 [==============================] - 187s 12s/step
- loss: 0.2683 - accuracy: 0.8976 - val_loss: 0.3540 -
val_accuracy: 0.8000

Epoch 17/40

16/16 [==============================] - 181s 11s/step
- loss: 0.2584 - accuracy: 0.8956 - val_loss: 0.4145 -
val_accuracy: 0.8000

Epoch 18/40

16/16 [==============================] - 176s 11s/step
- loss: 0.2685 - accuracy: 0.8755 - val_loss: 0.4025 -
val_accuracy: 0.8000

Epoch 19/40

16/16 [==============================] - 175s 11s/step
- loss: 0.3332 - accuracy: 0.8414 - val_loss: 0.3535 -
val_accuracy: 0.8500

Epoch 20/40
```

```
16/16 [==============================] - 202s 13s/step
- loss: 0.2617 - accuracy: 0.8916 - val_loss: 0.5382 -
val_accuracy: 0.7500

Epoch 21/40

16/16 [==============================] - 193s 12s/step
- loss: 0.2552 - accuracy: 0.8876 - val_loss: 0.3917 -
val_accuracy: 0.8000

Epoch 22/40

16/16 [==============================] - 168s 10s/step
- loss: 0.2241 - accuracy: 0.9137 - val_loss: 0.3187 -
val_accuracy: 0.9500

Epoch 23/40

16/16 [==============================] - 165s 10s/step
- loss: 0.2371 - accuracy: 0.8936 - val_loss: 0.5655 -
val_accuracy: 0.7500

Epoch 24/40

16/16 [==============================] - 170s 11s/step
- loss: 0.2075 - accuracy: 0.9237 - val_loss: 0.3478 -
val_accuracy: 0.8000

Epoch 25/40

16/16 [==============================] - 169s 11s/step
- loss: 0.1782 - accuracy: 0.9498 - val_loss: 0.3705 -
val_accuracy: 0.8000

Epoch 26/40
```

16/16 [==========] - 173s 11s/step - loss: 0.1742 - accuracy: 0.9498 - val_loss: 0.3498 - val_accuracy: 0.8000

Epoch 27/40

16/16 [==========] - 183s 11s/step - loss: 0.1617 - accuracy: 0.9598 - val_loss: 0.3086 - val_accuracy: 0.8500

Epoch 28/40

16/16 [==========] - 169s 11s/step - loss: 0.1580 - accuracy: 0.9538 - val_loss: 0.4094 - val_accuracy: 0.7500

Epoch 29/40

16/16 [==========] - 169s 11s/step - loss: 0.1536 - accuracy: 0.9538 - val_loss: 0.3529 - val_accuracy: 0.8000

Epoch 30/40

16/16 [==========] - 189s 12s/step - loss: 0.1562 - accuracy: 0.9498 - val_loss: 0.5833 - val_accuracy: 0.7500

Epoch 31/40

16/16 [==========] - 177s 11s/step - loss: 0.1543 - accuracy: 0.9498 - val_loss: 0.4137 - val_accuracy: 0.8000

Epoch 32/40

16/16 [==========] - 169s 11s/step - loss: 0.1416 - accuracy: 0.9618 - val_loss: 0.2905 - val_accuracy: 0.8000

Epoch 33/40

16/16 [==========] - 170s 11s/step - loss: 0.1331 - accuracy: 0.9618 - val_loss: 0.3253 - val_accuracy: 0.8500

Epoch 34/40

16/16 [==========] - 170s 11s/step - loss: 0.1243 - accuracy: 0.9739 - val_loss: 0.2890 - val_accuracy: 0.8000

Epoch 35/40

16/16 [==========] - 173s 11s/step - loss: 0.1197 - accuracy: 0.9639 - val_loss: 0.4212 - val_accuracy: 0.8000

Epoch 36/40

16/16 [==========] - 189s 12s/step - loss: 0.1194 - accuracy: 0.9679 - val_loss: 0.4173 - val_accuracy: 0.7500

Epoch 37/40

16/16 [==========] - 176s 11s/step - loss: 0.1094 - accuracy: 0.9799 - val_loss: 0.4559 - val_accuracy: 0.7500

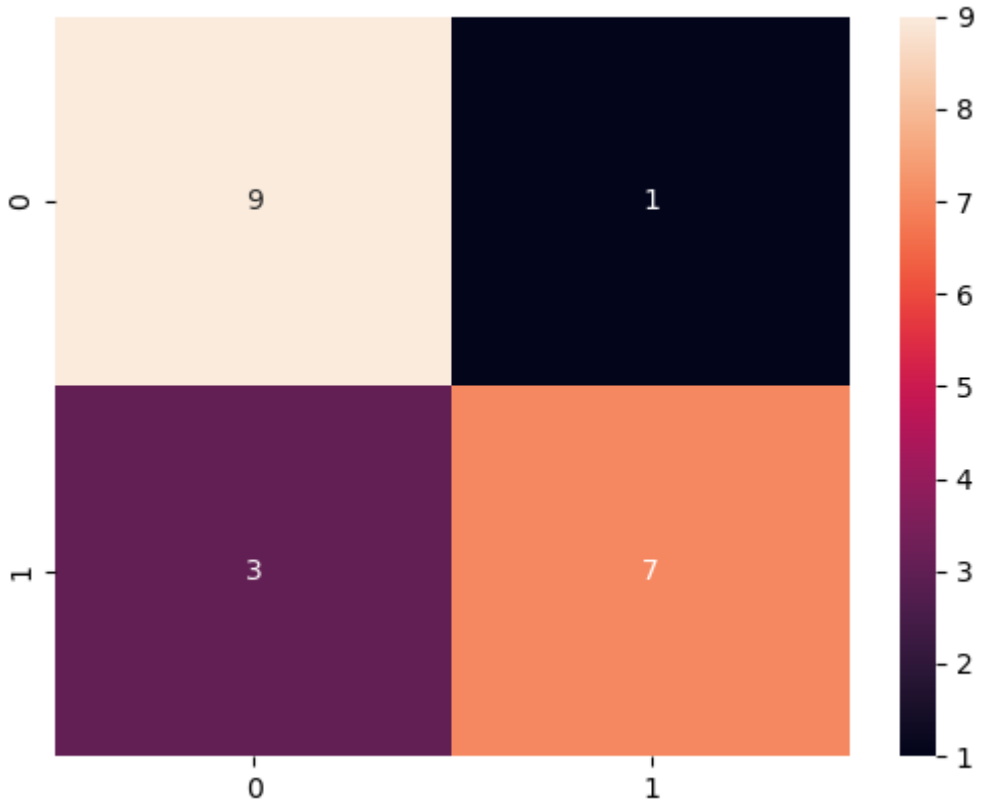
Epoch 38/40

```
16/16 [==============================] - 174s 11s/step - loss: 0.1078 - accuracy: 0.9779 - val_loss: 0.4595 - val_accuracy: 0.7500

Epoch 39/40

16/16 [==============================] - 170s 11s/step - loss: 0.1020 - accuracy: 0.9779 - val_loss: 0.3108 - val_accuracy: 0.8000

Epoch 40/40

16/16 [==============================] - 157s 10s/step - loss: 0.0906 - accuracy: 0.9779 - val_loss: 0.3104 - val_accuracy: 0.8000
```

```
prediction_NN = vgg_model.predict(x_test)

prediction_NN = np.argmax(prediction_NN, axis=-1)

prediction_NN = le.inverse_transform(prediction_NN)


from sklearn import metrics

#from sklearn.metrics import classification_report

#from sklearn.metrics import average_precision_score

print ("Accuracy = ", metrics.accuracy_score(test_labels, prediction_NN))
```

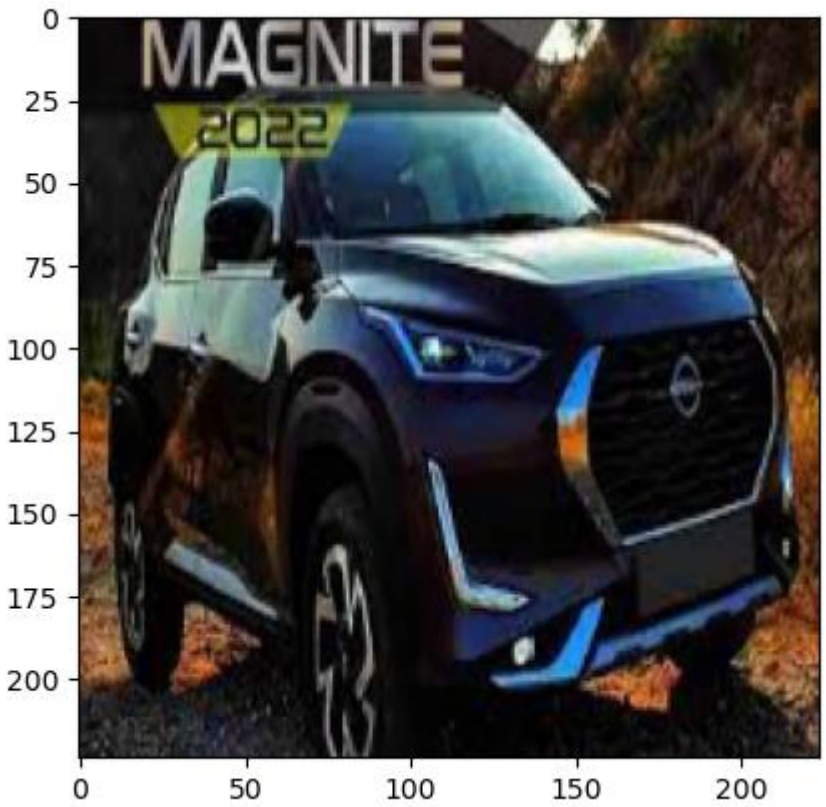
```
Accuracy =  0.8
```

```
#Confusion Matrix - verify accuracy of each class
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_labels, prediction_NN)
print(cm)
sns.heatmap(cm, annot=True)
```

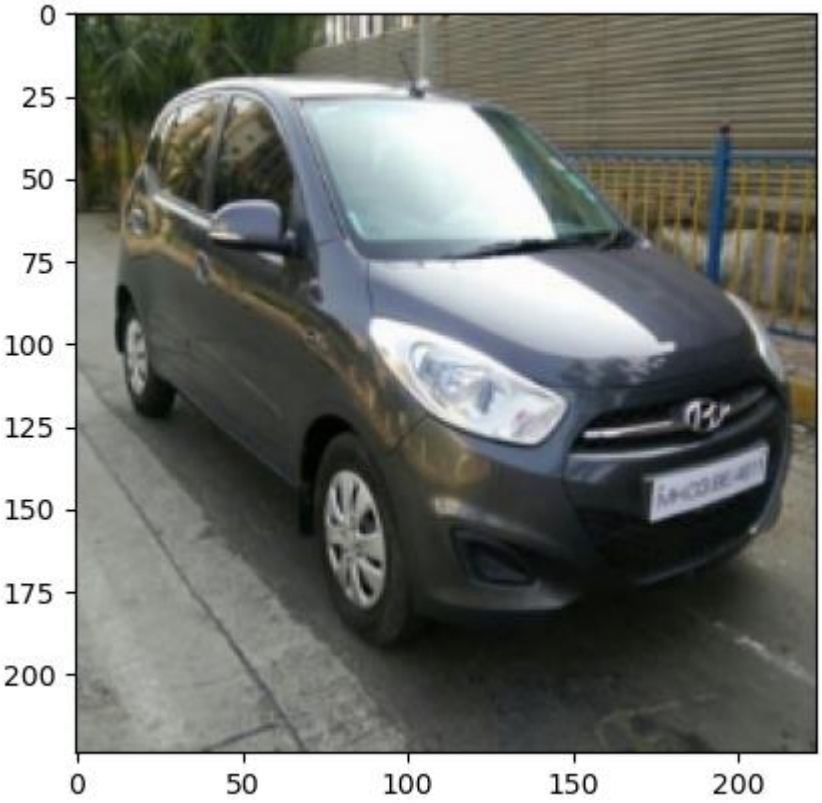



```
#Check results on a few select images


#n=5 dog park. NN not as good as RF.
n=7  #Select the index of image to be loaded for testing
img = x_test[n]
plt.imshow(img)
```

```
input_img = np.expand_dims(img, axis=0) #Expand dims so the input is (num images, x, y, c)

prediction = np.argmax(vgg_model.predict(input_img)) #argmax to convert categorical back to original

prediction = le.inverse_transform([prediction])  #Reverse the label encoder to original name

print("The prediction for this image is: ", prediction)

print("The actual label for this image is: ", test_labels[n])
```




```
The prediction for this image is:  ['Safe']

The actual label for this image is:  Safe
```



The prediction for this image is: ['Unsafe']

The actual label for this image is: Unsafe

AutoKeras

-*- coding: utf-8 -*-

"""

Created on Tue Aug 23 22:00:26 2022

@author: LAXMINARAYANRO

```
"""


import autokeras as ak
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2

import os
import seaborn as sns


print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 128

train_images = []
train_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
```

```
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
```

```
    img = cv2.imread(img_path, cv2.IMREAD_COLOR)
    img = cv2.resize(img, (SIZE, SIZE))
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    test_images.append(img)
    test_labels.append(fruit_label)

test_images = np.array(test_images)
test_labels = np.array(test_labels)

#Encode labels from text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data into test and train datasets (already split but
assigning to meaningful convention)
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded
```

```
##################################################
#################
# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# #One hot encode y values for neural network.
# from tensorflow.keras.utils import to_categorical
# y_train_one_hot = to_categorical(y_train)
# y_test_one_hot = to_categorical(y_test)


#Documentation: https://autokeras.com/image_classifier/
clf = ak.ImageClassifier(max_trials=1) #MaxTrials - max.
number of keras models to try
clf.fit(x_train, y_train, epochs=1)


#Evaluate the classifier on test data
_, acc = clf.evaluate(x_test, y_test)
print("Accuracy = ", (acc * 100.0), "%")
# val_acc = clf.clf['val_accuracy']
# plt.plot(epochs, acc, 'y', label='Training acc')
```

```
# plt.plot(epochs, val_acc, 'r', label='Validation acc')
# plt.title('Training and validation accuracy')
# plt.xlabel('Epochs')
# plt.ylabel('Accuracy')
# plt.legend()
# plt.show()

from sklearn import metrics
#from sklearn.metrics import classification_report
#from sklearn.metrics import average_precision_score
print ("Accuracy = ", metrics.accuracy_score(test_labels, clf))
#Confusion Matrix - verify accuracy of each class
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_labels, clf)
print(cm)
sns.heatmap(cm, annot=True)

# get the final best performing model
model = clf.export_model()
print(model.summary())

#Save the model
```

```
model.save('cifar_model.h5')
```

Ensemble Learning (VGG16, Resnet50, MobileNetV2)

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 22 13:29:57 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
import os
import seaborn as sns

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import (
```

```
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.resnet50 import
ResNet50
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.mobilenet_v2 import
MobileNetV2
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 224
#Channel=3  #Resize images

#lists
train_images = []
train_labels = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists


test_images = []
test_labels = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)
```

```
#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded



# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

##################################################
####
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
##################################################
vgg_model = Sequential()

pretrained_model=
tf.keras.applications.MobileNetV2(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model.layers:
```

```
    layer.trainable=False

vgg_model.add(pretrained_model)

vgg_model.add(Flatten())
vgg_model.add(Dense(512, activation='relu'))
vgg_model.add(Dense(2, activation='softmax'))

vgg_model.summary()

vgg_model.compile(optimizer=Adam(learning_rate=0.001),loss='categorical_crossentropy',metrics=['accuracy'])

#history = model.fit(lung_train, infect_train, epochs = 20, validation_data = (lung_test, infect_test))

#epochs=10
history=vgg_model.fit(x_train, y_train_one_hot, epochs=10, validation_data = (x_test, y_test_one_hot))

vgg_model.save('E:/fffiiles/college pracs and projects/Project/Car/Code/vgg_model.hdf5')
```

```
##################################################
#####################
resnet50_model = Sequential()


pretrained_model1=
tf.keras.applications.ResNet50(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model1.layers:
        layer.trainable=False


resnet50_model.add(pretrained_model1)


resnet50_model.add(Flatten())
resnet50_model.add(Dense(512, activation='relu'))
resnet50_model.add(Dense(2, activation='softmax'))


resnet50_model.summary()


resnet50_model.compile(optimizer=Adam(learning_rate=0.0
01),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
#history = model.fit(lung_train, infect_train, epochs = 20,
validation_data = (lung_test, infect_test))


#epochs=10

history=resnet50_model.fit(x_train, y_train_one_hot,
epochs=10, validation_data = (x_test, y_test_one_hot))
```

```
Epoch 1/10

16/16 [==============================] - 91s 5s/step -
loss: 0.8604 - accuracy: 0.5382 - val_loss: 0.7086 -
val_accuracy: 0.5000

Epoch 2/10

16/16 [==============================] - 74s 5s/step -
loss: 0.6988 - accuracy: 0.5643 - val_loss: 0.7965 -
val_accuracy: 0.5000

Epoch 3/10

16/16 [==============================] - 77s 5s/step -
loss: 0.6395 - accuracy: 0.6365 - val_loss: 0.7438 -
val_accuracy: 0.5000

Epoch 4/10

16/16 [==============================] - 85s 5s/step -
loss: 0.6527 - accuracy: 0.6245 - val_loss: 0.6845 -
val_accuracy: 0.5500

Epoch 5/10
```

```
16/16 [==============================] - 72s 4s/step - loss: 0.6551 - accuracy: 0.6104 - val_loss: 0.7294 - val_accuracy: 0.5000

Epoch 6/10

16/16 [==============================] - 72s 4s/step - loss: 0.6314 - accuracy: 0.6426 - val_loss: 0.8862 - val_accuracy: 0.5000

Epoch 7/10

16/16 [==============================] - 75s 5s/step - loss: 0.6240 - accuracy: 0.6446 - val_loss: 0.6662 - val_accuracy: 0.6500

Epoch 8/10

16/16 [==============================] - 73s 5s/step - loss: 0.6742 - accuracy: 0.6245 - val_loss: 0.9539 - val_accuracy: 0.5000

Epoch 9/10

16/16 [==============================] - 79s 5s/step - loss: 0.7047 - accuracy: 0.5683 - val_loss: 0.6937 - val_accuracy: 0.5000

Epoch 10/10

16/16 [==============================] - 67s 4s/step - loss: 0.6137 - accuracy: 0.6667 - val_loss: 0.6854 - val_accuracy: 0.5000
```

```
resnet50_model.save('E:/fffiiles/college pracs and projects/Project/Car/Code/resnet50_model.hdf5')
```

```
##################################################
##########################
mobilenet_v2_model = Sequential()


pretrained_model3=
tf.keras.applications.MobileNetV2(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model3.layers:
        layer.trainable=False


mobilenet_v2_model.add(pretrained_model3)


mobilenet_v2_model.add(Flatten())
mobilenet_v2_model.add(Dense(512, activation='relu'))
mobilenet_v2_model.add(Dense(2, activation='softmax'))


mobilenet_v2_model.summary()


mobilenet_v2_model.compile(optimizer=Adam(learning_rate
=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
#history = model.fit(lung_train, infect_train, epochs = 20,
validation_data = (lung_test, infect_test))


#epochs=10
history=mobilenet_v2_model.fit(x_train, y_train_one_hot,
epochs=10, validation_data = (x_test, y_test_one_hot))


mobilenet_v2_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/mobilenet_v2_model.hdf5')
##################################################
#######################3
from keras.models import load_model
from sklearn.metrics import accuracy_score


model1 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/vgg_model.hdf5')
model2 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/resnet50_model.hdf5')
model3 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/mobilenet_v2_model.hdf5')


models = [model1, model2, model3]


preds = [model.predict(x_test) for model in models]
```

```
preds=np.array(preds)
summed = np.sum(preds, axis=0)

# argmax across classes
ensemble_prediction = np.argmax(summed, axis=1)



prediction1 = model1.predict(x_test)
prediction2 = model2.predict(x_test)
prediction3 = model3.predict(x_test)

prediction1 = np.argmax(prediction1, axis=-1)
prediction2 = np.argmax(prediction2, axis=-1)
prediction3 = np.argmax(prediction3, axis=-1)

prediction1 = le.inverse_transform(prediction1)
prediction2 = le.inverse_transform(prediction2)
prediction3 = le.inverse_transform(prediction3)
ensemble_prediction =
le.inverse_transform(ensemble_prediction)

accuracy1 = accuracy_score(test_labels, prediction1)
```

```
accuracy2 = accuracy_score(test_labels, prediction2)
accuracy3 = accuracy_score(test_labels, prediction3)
ensemble_accuracy = accuracy_score(test_labels,
ensemble_prediction)

print('Accuracy Score for model1 = ', accuracy1)
print('Accuracy Score for model2 = ', accuracy2)
print('Accuracy Score for model3 = ', accuracy3)
print('Accuracy Score for average ensemble = ',
ensemble_accuracy)
```

```
Accuracy Score for model1 =  0.8
Accuracy Score for model2 =  0.5
Accuracy Score for model3 =  0.85
Accuracy Score for average ensemble =  0.8
```

```
##########################################
#Weighted average ensemble
models = [model1, model2, model3]
preds = [model.predict(x_test) for model in models]
preds=np.array(preds)
weights = [0.4, 0.2, 0.4]

#Use tensordot to sum the products of all elements over
specified axes.
```

```
weighted_preds = np.tensordot(preds, weights,
axes=((0),(0)))

weighted_ensemble_prediction =
np.argmax(weighted_preds, axis=1)

weighted_ensemble_prediction =
le.inverse_transform(weighted_ensemble_prediction)

weighted_accuracy = accuracy_score(test_labels,
weighted_ensemble_prediction)


print('Accuracy Score for model1 = ', accuracy1)

print('Accuracy Score for model2 = ', accuracy2)

print('Accuracy Score for model3 = ', accuracy3)

print('Accuracy Score for average ensemble = ',
ensemble_accuracy)

print('Accuracy Score for weighted average ensemble = ',
weighted_accuracy)
```

```
Accuracy Score for model1 =  0.8

Accuracy Score for model2 =  0.5

Accuracy Score for model3 =  0.85

Accuracy Score for average ensemble =  0.8

Accuracy Score for weighted average ensemble =  0.8
```

```
##########################################

#Grid search for the best combination of w1, w2, w3 that
gives maximum acuracy
```

```
models = [model1, model2, model3]
preds1 = [model.predict(x_test) for model in models]
preds1=np.array(preds1)

import pandas as pd
df = pd.DataFrame([])

for w1 in range(0, 5):
    for w2 in range(0,5):
        for w3 in range(0,5):
            wts = [w1/10.,w2/10.,w3/10.]
            wted_preds1 = np.tensordot(preds1, wts,
axes=((0),(0)))
            wted_ensemble_pred = np.argmax(wted_preds1,
axis=1)
            weighted_accuracy = accuracy_score(y_test,
wted_ensemble_pred)
            df =
df.append(pd.DataFrame({'wt1':wts[0],'wt2':wts[1],
                                    'wt3':wts[2],
'acc':weighted_accuracy*100}, index=[0]),
ignore_index=True)
```

```
max_acc_row = df.iloc[df['acc'].idxmax()]
print("Max accuracy of ", max_acc_row[0], " obained with w1=", max_acc_row[1],
    " w2=", max_acc_row[2], " and w3=", max_acc_row[3])
```

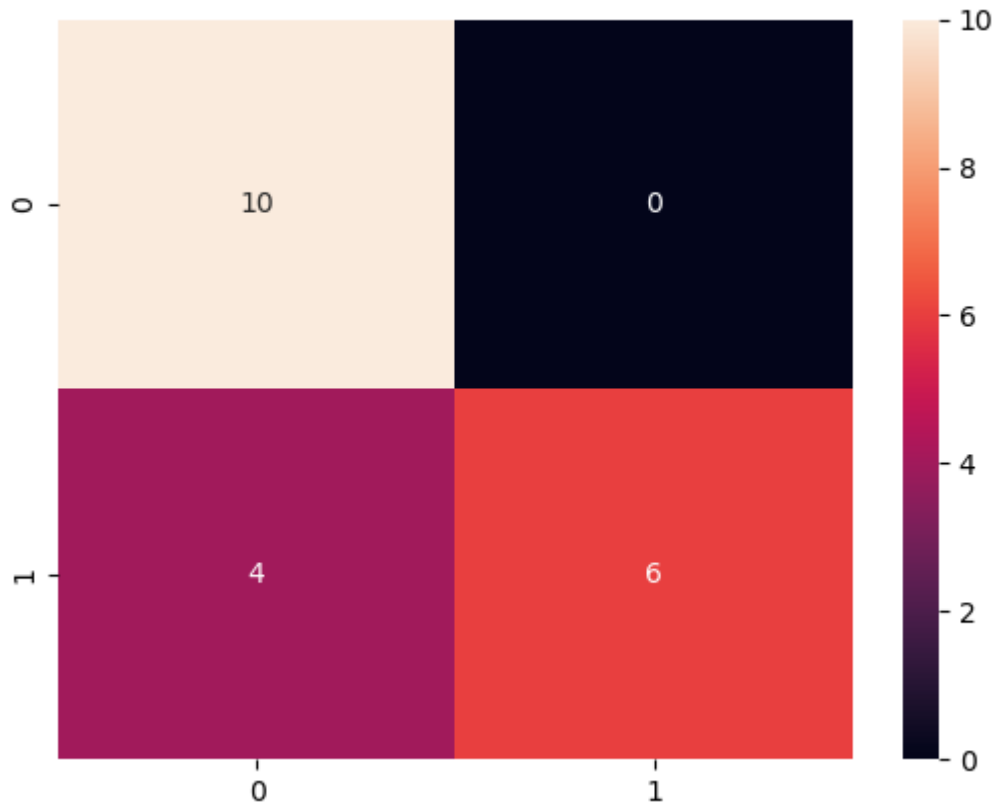
Max accuracy of 0.3 obained with w1= 0.0 w2= 0.1 and w3= 90.0

```
##################################################################################
### Explore metrics for the ideal weighted ensemble model.

models = [model1, model2, model3]
preds = [model.predict(x_test) for model in models]
preds=np.array(preds)
ideal_weights = [0.0, 0.1, 90.0]

#Use tensordot to sum the products of all elements over specified axes.
ideal_weighted_preds = np.tensordot(preds, ideal_weights, axes=((0),(0)))
```

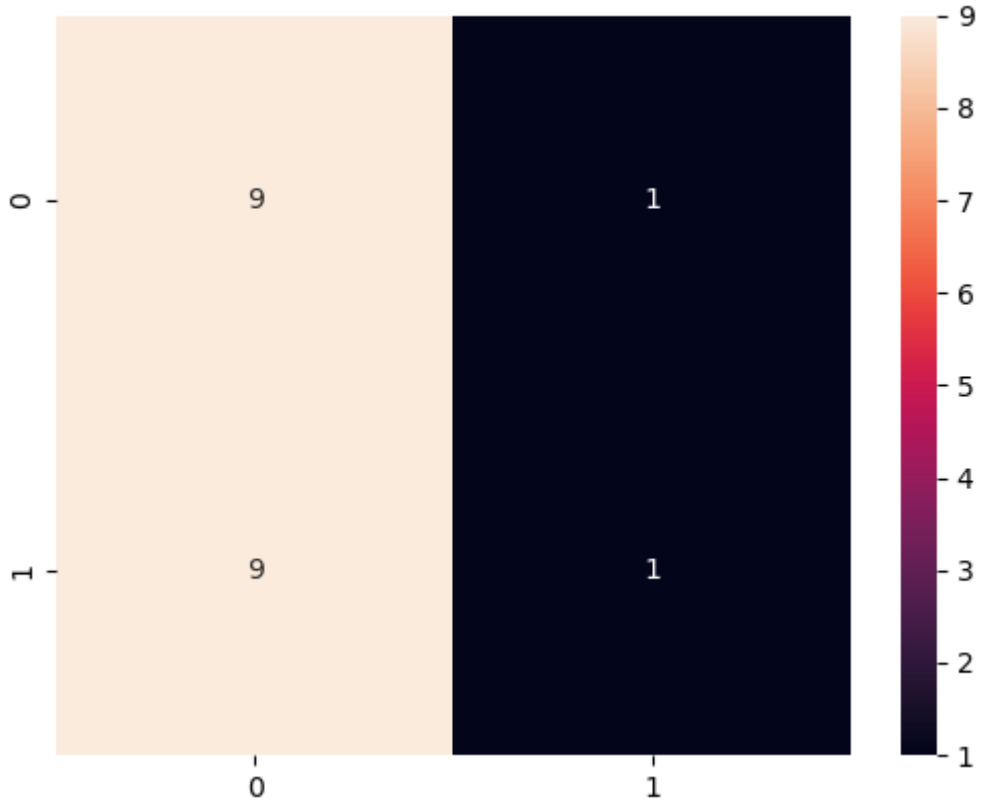
```
ideal_weighted_ensemble_prediction =
np.argmax(ideal_weighted_preds, axis=1)
ideal_weighted_ensemble_prediction =
le.inverse_transform(ideal_weighted_ensemble_prediction)


ideal_weighted_accuracy = accuracy_score(test_labels,
ideal_weighted_ensemble_prediction)



print('Accuracy Score for ideal weighted average ensemble =
', ideal_weighted_accuracy)
```

```
Accuracy Score for ideal weighted average ensemble =  0.85
```

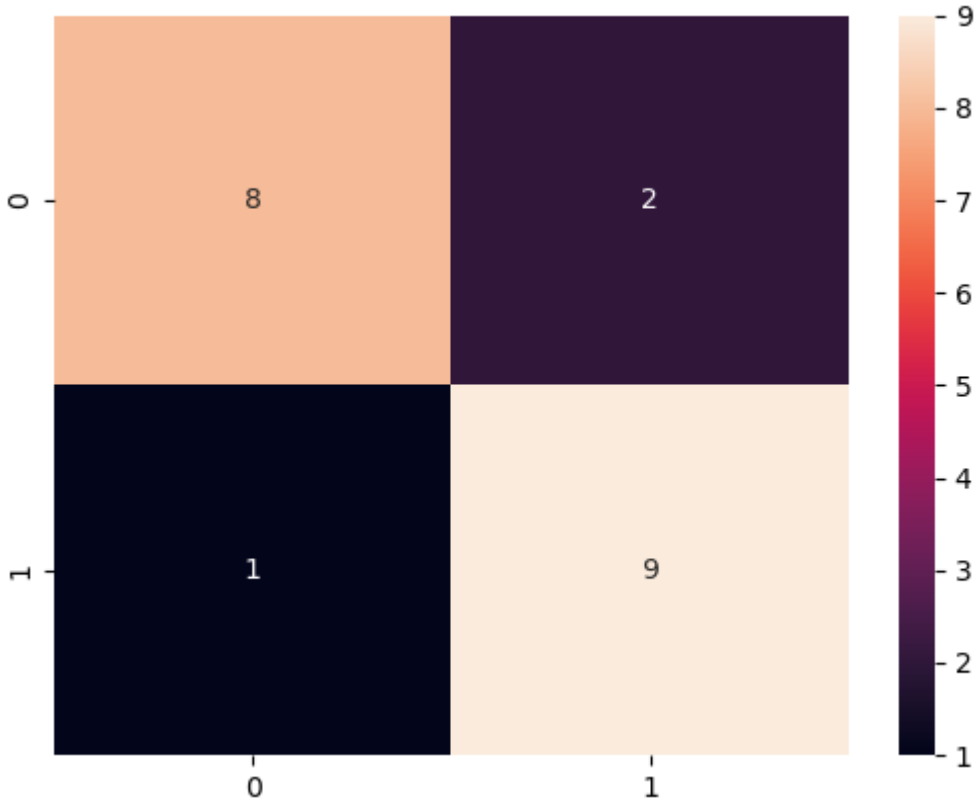
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_labels, prediction1)
sns.heatmap(cm, annot=True)
```



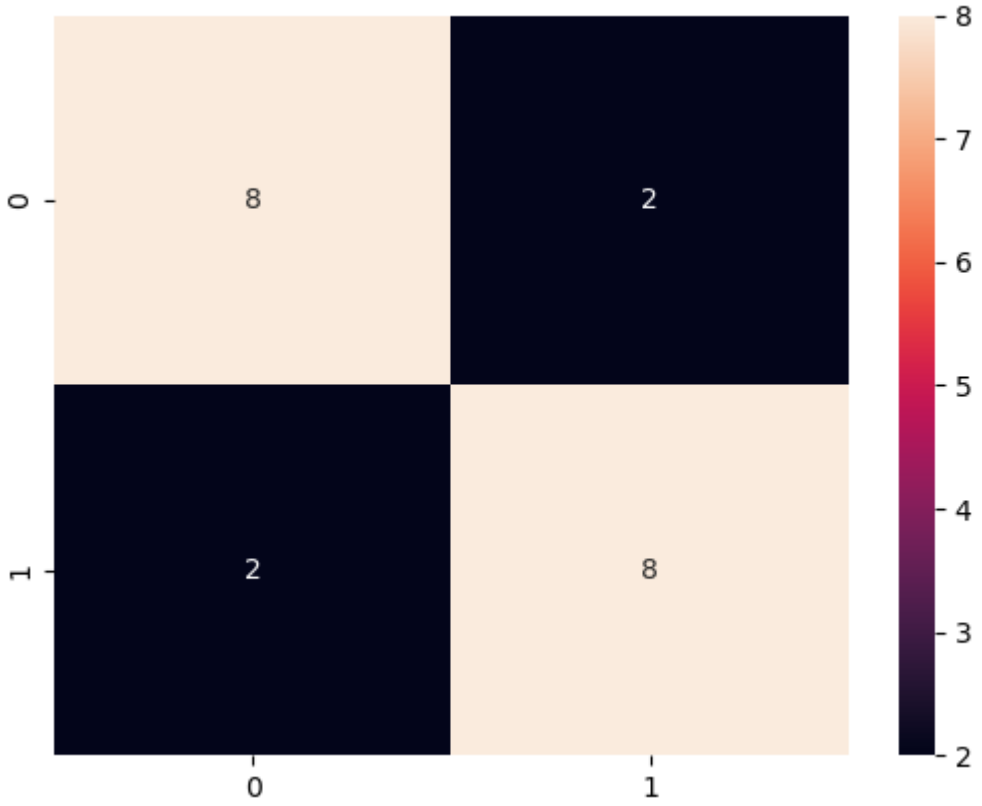

```
cm1 = confusion_matrix(test_labels, prediction2)
sns.heatmap(cm1, annot=True)
```



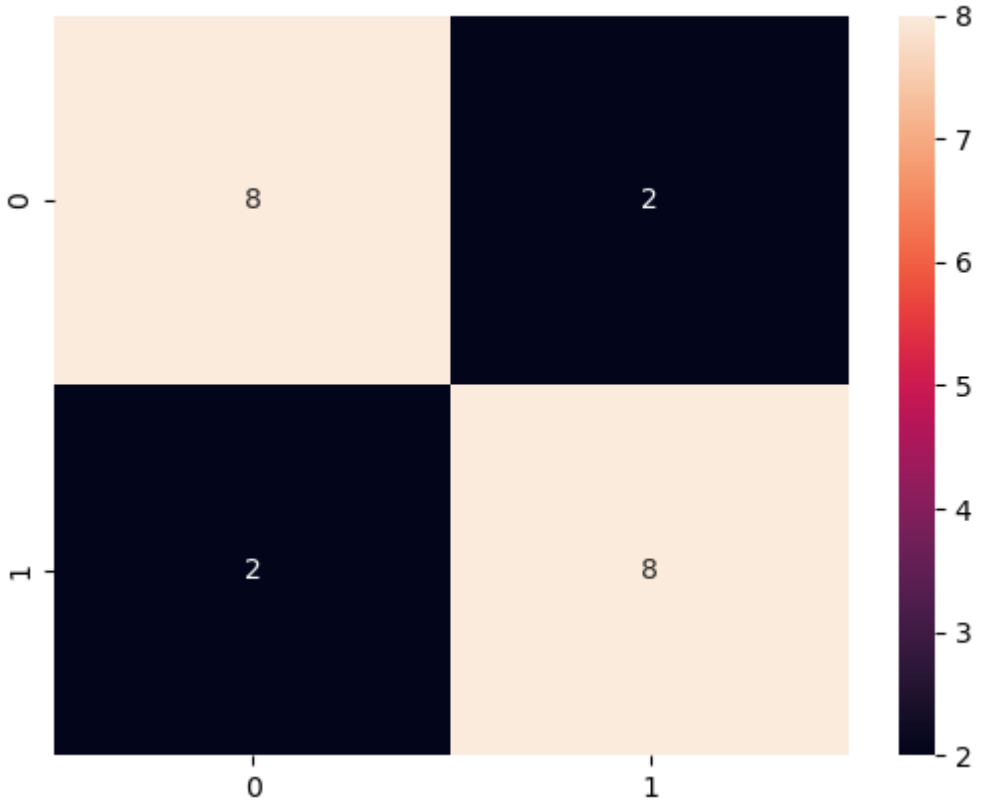

```
cm2 = confusion_matrix(test_labels, prediction3)
sns.heatmap(cm2, annot=True)
```



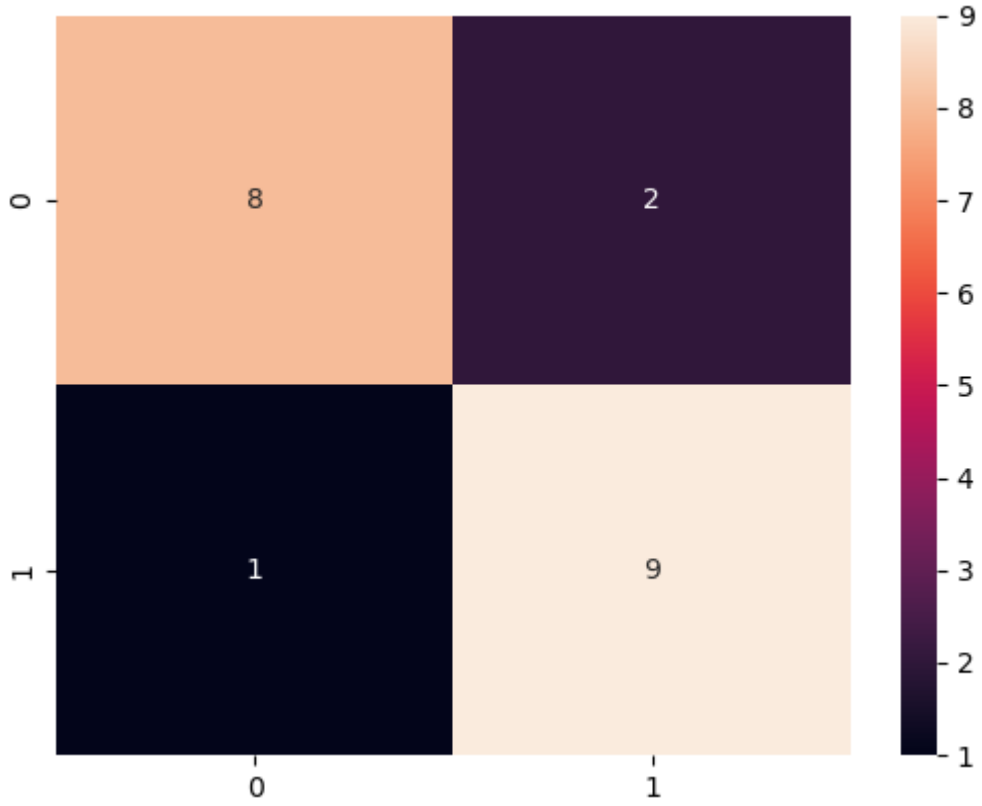

```
cm3 = confusion_matrix(test_labels, ensemble_prediction)
sns.heatmap(cm3, annot=True)
```




```
cm4 = confusion_matrix(test_labels,
weighted_ensemble_prediction)

sns.heatmap(cm4, annot=True)
```




```
cm5 = confusion_matrix(test_labels,
ideal_weighted_ensemble_prediction)

sns.heatmap(cm5, annot=True)
```



Xception

-*- coding: utf-8 -*-

"""

Created on Sat Aug 27 16:31:23 2022

@author: LAXMINARAYANRO

"""

```
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
import os
import seaborn as sns

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.xception import Xception

import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))
```

```
SIZE = 224
#Channel=3  #Resize images

#lists
train_images = []
train_labels = []

for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
```

```
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists


test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)
```

```
# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded



# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

##################################################
####
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
```

```
##################################################
xception_model = Sequential()

pretrained_model=
tf.keras.applications.Xception(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False

xception_model.add(pretrained_model)

xception_model.add(Flatten())
xception_model.add(Dense(512, activation='relu'))
xception_model.add(Dense(2, activation='softmax'))

xception_model.summary()

xception_model.compile(optimizer=Adam(learning_rate=0.0
01),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
#history = model.fit(lung_train, infect_train, epochs = 20,
validation_data = (lung_test, infect_test))


#epochs=10
history=xception_model.fit(x_train, y_train_one_hot,
epochs=10, validation_data = (x_test, y_test_one_hot))


xception_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/xception_model.hdf5')
```

InceptionV3

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 17:05:26 2022

@author: LAXMINARAYANRO
"""


# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 17:01:31 2022
```

```
@author: LAXMINARAYANRO
"""

# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 16:31:23 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
import os
import seaborn as sns

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
```

```
)
from tensorflow.keras.applications.inception_v3 import
InceptionV3

import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 224
#Channel=3  #Resize images

#lists
train_images = []
train_labels = []

for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
```

```
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists


test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
```

```
    img = cv2.imread(img_path, cv2.IMREAD_COLOR)
    img = cv2.resize(img, (SIZE, SIZE))
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    test_images.append(img)
    test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded
```

```
# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

##################################################
####
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
#################################################
inceptionv3_model = Sequential()

pretrained_model=
tf.keras.applications.InceptionV3(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False

inceptionv3_model.add(pretrained_model)
```

```
inceptionv3_model.add(Flatten())
inceptionv3_model.add(Dense(512, activation='relu'))
inceptionv3_model.add(Dense(2, activation='softmax'))

inceptionv3_model.summary()

inceptionv3_model.compile(optimizer=Adam(learning_rate=
0.001),loss='categorical_crossentropy',metrics=['accuracy'])

#history = model.fit(lung_train, infect_train, epochs = 20,
validation_data = (lung_test, infect_test))

#epochs=10
history=inceptionv3_model.fit(x_train, y_train_one_hot,
epochs=10, validation_data = (x_test, y_test_one_hot))

inceptionv3_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/inceptionv3_model.hdf5')
```

Densenet121

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Sat Aug 27 17:10:03 2022

@author: LAXMINARAYANRO
"""

# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 17:01:31 2022

@author: LAXMINARAYANRO
"""

# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 16:31:23 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
```

```
import cv2
import os
import seaborn as sns

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.densenet import
DenseNet121

import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 224
#Channel=3  #Resize images
```

```
#lists
train_images = []
train_labels = []

for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)
```

```
# test/validation into lists


test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)


#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)


# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

```
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded



# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

##################################################
####
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
#################################################
densenet_model = Sequential()

pretrained_model=
tf.keras.applications.DenseNet121(include_top=False,
```

```
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model.layers:
    layer.trainable=False

densenet_model.add(pretrained_model)

densenet_model.add(Flatten())
densenet_model.add(Dense(512, activation='relu'))
densenet_model.add(Dense(2, activation='softmax'))

densenet_model.summary()

densenet_model.compile(optimizer=Adam(learning_rate=0.0
01),loss='categorical_crossentropy',metrics=['accuracy'])

#history = model.fit(lung_train, infect_train, epochs = 20,
validation_data = (lung_test, infect_test))

#epochs=10
history=densenet_model.fit(x_train, y_train_one_hot,
epochs=10, validation_data = (x_test, y_test_one_hot))
```

```
densenet_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/densenet_model.hdf5')
```

EfficientnetB7

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 19:24:07 2022

@author: LAXMINARAYANRO
"""

# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 17:10:03 2022

@author: LAXMINARAYANRO
"""

# -*- coding: utf-8 -*-
"""
```

```
Created on Sat Aug 27 17:01:31 2022

@author: LAXMINARAYANRO
"""

# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 16:31:23 2022

@author: LAXMINARAYANRO
"""

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
import os
import seaborn as sns

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import (
```

```
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.efficientnet import
EfficientNetB7


import tensorflow as tf
from tensorflow.keras.optimizers import Adam


# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))


SIZE = 224
#Channel=3  #Resize images


#lists
train_images = []
train_labels = []


for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
```

```
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists


test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
```

```
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
```

```
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded



# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

##################################################
####
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
##################################################
efficientnet_model = Sequential()

pretrained_model=
tf.keras.applications.EfficientNetB7(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False
```

```
efficientnet_model.add(pretrained_model)

efficientnet_model.add(Flatten())
efficientnet_model.add(Dense(512, activation='relu'))
efficientnet_model.add(Dense(2, activation='softmax'))

efficientnet_model.summary()

efficientnet_model.compile(optimizer=Adam(learning_rate=
0.001),loss='categorical_crossentropy',metrics=['accuracy'])

#history = model.fit(lung_train, infect_train, epochs = 20,
validation_data = (lung_test, infect_test))

#epochs=10
history=efficientnet_model.fit(x_train, y_train_one_hot,
epochs=10, validation_data = (x_test, y_test_one_hot))

efficientnet_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/efficientnet_model.hdf5')
```

VGG19

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 17:01:31 2022

@author: LAXMINARAYANRO
"""


# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 16:31:23 2022

@author: LAXMINARAYANRO
"""


import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
import os
import seaborn as sns

from tensorflow.keras.models import Model, Sequential
```

```
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.vgg19 import VGG19

import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 224
#Channel=3  #Resize images

#lists
train_images = []
train_labels = []

for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
```

```
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists


test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
```

```
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
```

```
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded



# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

##################################################
####
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
##################################################
vgg19_model = Sequential()

pretrained_model=
tf.keras.applications.VGG19(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False
```

```
vgg19_model.add(pretrained_model)

vgg19_model.add(Flatten())
vgg19_model.add(Dense(512, activation='relu'))
vgg19_model.add(Dense(2, activation='softmax'))

vgg19_model.summary()

vgg19_model.compile(optimizer=Adam(learning_rate=0.001)
,loss='categorical_crossentropy',metrics=['accuracy'])

#history = model.fit(lung_train, infect_train, epochs = 20,
validation_data = (lung_test, infect_test))

#epochs=10
history=vgg19_model.fit(x_train, y_train_one_hot,
epochs=10, validation_data = (x_test, y_test_one_hot))

vgg19_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/vgg19_model.hdf5')
```

NasnetMobile

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 20:28:13 2022

@author: LAXMINARAYANRO
"""
```

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 19:59:56 2022

@author: LAXMINARAYANRO
"""
```

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 19:24:07 2022

@author: LAXMINARAYANRO
"""
```

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 17:10:03 2022

@author: LAXMINARAYANRO
"""


# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 17:01:31 2022

@author: LAXMINARAYANRO
"""


# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 16:31:23 2022

@author: LAXMINARAYANRO
"""


import numpy as np
```

```
import matplotlib.pyplot as plt
import glob
import cv2
import os
import seaborn as sns

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.nasnet import
NASNetMobile

import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 224
```

```
#Channel=3  #Resize images


#lists
train_images = []
train_labels = []


for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)


#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)
```

```
# test/validation into lists

test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# text to integers.
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded



# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

##################################################
####
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
#################################################
nasnetmobile_model = Sequential()
```

```
pretrained_model=
tf.keras.applications.NASNetMobile(include_top=False,
                input_shape=(SIZE,SIZE,3),
                pooling='avg',classes=2,
                weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False
nasnetmobile_model.add(pretrained_model)


nasnetmobile_model.add(Flatten())
nasnetmobile_model.add(Dense(512, activation='relu'))
nasnetmobile_model.add(Dense(2, activation='softmax'))


nasnetmobile_model.summary()


nasnetmobile_model.compile(optimizer=Adam(learning_rate
=0.001),loss='categorical_crossentropy',metrics=['accuracy'])


#history = model.fit(lung_train, infect_train, epochs = 20,
validation_data = (lung_test, infect_test))


#epochs=10
```

```
history=nasnetmobile_model.fit(x_train, y_train_one_hot,
epochs=10, validation_data = (x_test, y_test_one_hot))


nasnetmobile_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/nasnetmobile_model.hdf5')
```

Ensemble Learning

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 27 16:46:45 2022


@author: LAXMINARAYANRO
"""
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
import os
import seaborn as sns


from tensorflow.keras.models import Model, Sequential
```

```
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D, Dense
)
from tensorflow.keras.applications.xception import Xception

import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# Read input images
print(os.listdir("E:/fffiiles/college pracs and
projects/Project/Car"))

SIZE = 224
#Channel=3  #Resize images

#lists
train_images = []
train_labels = []

for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
```

```
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists


test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
```

```
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# text to integers.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

#Split data
```

```
x_train, y_train, x_test, y_test = train_images,
train_labels_encoded, test_images, test_labels_encoded



# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

##################################################
####
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

from keras.models import load_model
from sklearn.metrics import accuracy_score

model1 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/vgg_model.hdf5')
model2 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/resnet50_model.hdf5')
model3 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/mobilenet_v2_model.hdf5')
```

```
model4 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/vgg19_model.hdf5')

model5 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/Xception_model.hdf5')

model6 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/densenet_model.hdf5')

model7 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/efficientnet_model.hdf5')

model8 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/inceptionv3_model.hdf5')

model9 = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/nasnetmobile_model.hdf5')


models = [model1, model2, model3, model4, model5,
model6, model7, model8, model9]


preds = [model.predict(x_test) for model in models]

preds=np.array(preds)

summed = np.sum(preds, axis=0)


# argmax across classes

ensemble_prediction = np.argmax(summed, axis=1)
```

```
prediction1 = model1.predict(x_test)
prediction2 = model2.predict(x_test)
prediction3 = model3.predict(x_test)
prediction4 = model4.predict(x_test)
prediction5 = model5.predict(x_test)
prediction6 = model6.predict(x_test)
prediction7 = model7.predict(x_test)
prediction8 = model8.predict(x_test)
prediction9 = model9.predict(x_test)

prediction1 = np.argmax(prediction1, axis=-1)
prediction2 = np.argmax(prediction2, axis=-1)
prediction3 = np.argmax(prediction3, axis=-1)
prediction4 = np.argmax(prediction4, axis=-1)
prediction5 = np.argmax(prediction5, axis=-1)
prediction6 = np.argmax(prediction6, axis=-1)
prediction7 = np.argmax(prediction7, axis=-1)
prediction8 = np.argmax(prediction8, axis=-1)
prediction9 = np.argmax(prediction9, axis=-1)

prediction1 = le.inverse_transform(prediction1)
prediction2 = le.inverse_transform(prediction2)
```

```
prediction3 = le.inverse_transform(prediction3)
prediction4 = le.inverse_transform(prediction4)
prediction5 = le.inverse_transform(prediction5)
prediction6 = le.inverse_transform(prediction6)
prediction7 = le.inverse_transform(prediction7)
prediction8 = le.inverse_transform(prediction8)
prediction9 = le.inverse_transform(prediction9)
ensemble_prediction =
le.inverse_transform(ensemble_prediction)

accuracy1 = accuracy_score(test_labels, prediction1)
accuracy2 = accuracy_score(test_labels, prediction2)
accuracy3 = accuracy_score(test_labels, prediction3)
accuracy4 = accuracy_score(test_labels, prediction4)
accuracy5 = accuracy_score(test_labels, prediction5)
accuracy6 = accuracy_score(test_labels, prediction6)
accuracy7 = accuracy_score(test_labels, prediction7)
accuracy8 = accuracy_score(test_labels, prediction8)
accuracy9 = accuracy_score(test_labels, prediction9)
ensemble_accuracy = accuracy_score(test_labels,
ensemble_prediction)
```

```
print('Accuracy Score for model1 = ', accuracy1)
print('Accuracy Score for model2 = ', accuracy2)
print('Accuracy Score for model3 = ', accuracy3)
print('Accuracy Score for model1 = ', accuracy4)
print('Accuracy Score for model2 = ', accuracy5)
print('Accuracy Score for model3 = ', accuracy6)
print('Accuracy Score for model1 = ', accuracy7)
print('Accuracy Score for model2 = ', accuracy8)
print('Accuracy Score for model3 = ', accuracy9)
print('Accuracy Score for average ensemble = ',
ensemble_accuracy)
```

```
Accuracy Score for VGG16 =  0.8
Accuracy Score for ResNet50 =  0.5
Accuracy Score for MobileNetV2 =  0.85
Accuracy Score for VGG19 =  0.85
Accuracy Score for Xception =  0.7
Accuracy Score for DenseNet121 =  0.9
Accuracy Score for EfficientNetB7 =  0.5
Accuracy Score for InceptionV3 =  0.6
Accuracy Score for NasNetMobile =  0.85
Accuracy Score for average ensemble =  0.95
```

```
##########################################
```

```
#Weighted average ensemble

models = [model1, model2, model3, model4, model5, model6, model7, model8, model9]

preds = [model.predict(x_test) for model in models]

preds=np.array(preds)

weights = [0.1, 0.1, 0.1, 0.1, 0.1, 0.2, 0.1, 0.1, 0.1]


#Use tensordot to sum the products of all elements over specified axes.

weighted_preds = np.tensordot(preds, weights, axes=((0),(0)))

weighted_ensemble_prediction = np.argmax(weighted_preds, axis=1)

weighted_ensemble_prediction = le.inverse_transform(weighted_ensemble_prediction)

weighted_accuracy = accuracy_score(test_labels, weighted_ensemble_prediction)


print('Accuracy Score for weighted average ensemble = ', weighted_accuracy)
```

```
Accuracy Score for weighted average ensemble =  0.9
```

```
##########################################

#Grid search for the best combination of w1, w2, w3 that gives maximum acuracy
```

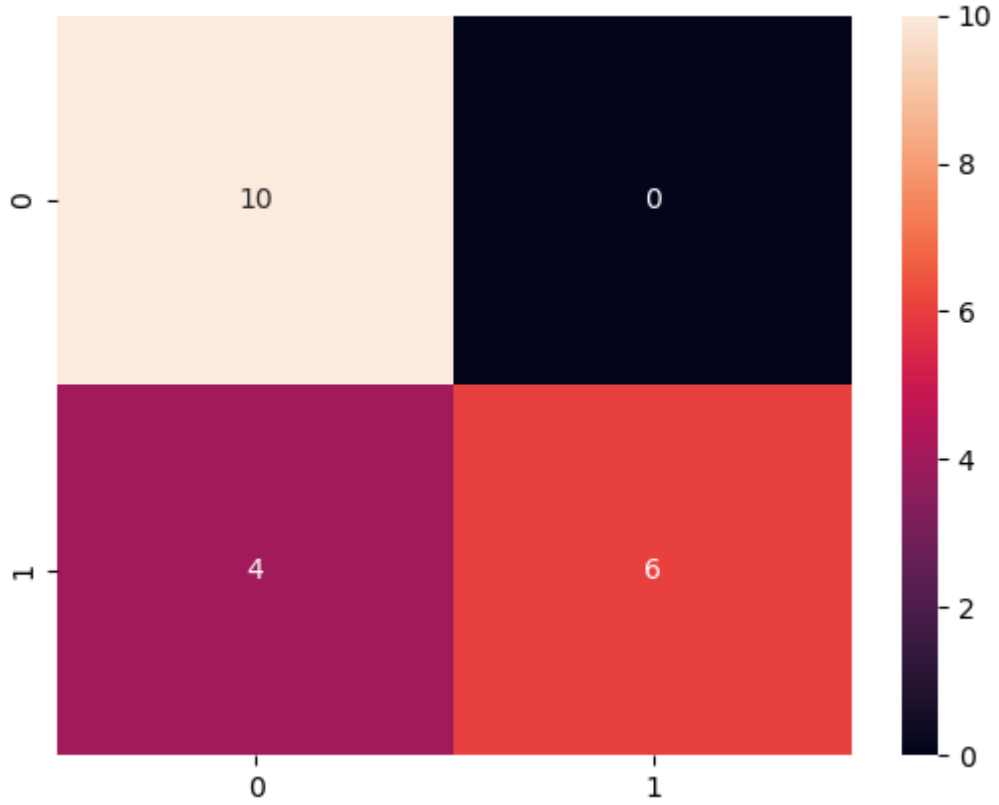
```
models = [model1, model2, model3, model4, model5,
model6, model7, model8, model9]
preds1 = [model.predict(x_test) for model in models]
preds1=np.array(preds1)

import pandas as pd
df = pd.DataFrame([])

for w1 in range(0,5):
    for w2 in range(0,5):
        for w3 in range(0,5):
            for w4 in range(0,5):
                for w5 in range(0,5):
                    for w6 in range(0,5):
                        for w7 in range(0,5):
                            for w8 in range(0,5):
                                for w9 in range(0,5):
                                    wts =
[w1/10.,w2/10.,w3/10.,w4/10.,w5/10.,w6/10.,w7/10.,w8/10.
,w9/10.]
                                    wted_preds1 = np.tensordot(preds1, wts,
axes=((0),(0)))
```

```
                    wted_ensemble_pred =
np.argmax(wted_preds1, axis=1)
                    weighted_accuracy =
accuracy_score(y_test, wted_ensemble_pred)
                    df =
df.append(pd.DataFrame({'wt1':wts[0],'wt2':wts[1],
                                        'wt3':wts[2],
'wt4':wts[3],'wt5':wts[4],'wt6':wts[5],'wt7':wts[6],'wt8':wts[7
], 'wt9':wts[8], 'acc':weighted_accuracy*100}, index=[0]),
ignore_index=True)


max_acc_row = df.iloc[df['acc'].idxmax()]
print("Max accuracy of ", max_acc_row[0], " obained with
w1=", max_acc_row[1],
    " w2=", max_acc_row[2], " and w3=", max_acc_row[3], "
and w4=", max_acc_row[4], " and w5=", max_acc_row[5], "
and w6=", max_acc_row[6] ," and w7=", max_acc_row[7], "
and w8=", max_acc_row[8] ," and w9=", max_acc_row[9])
```

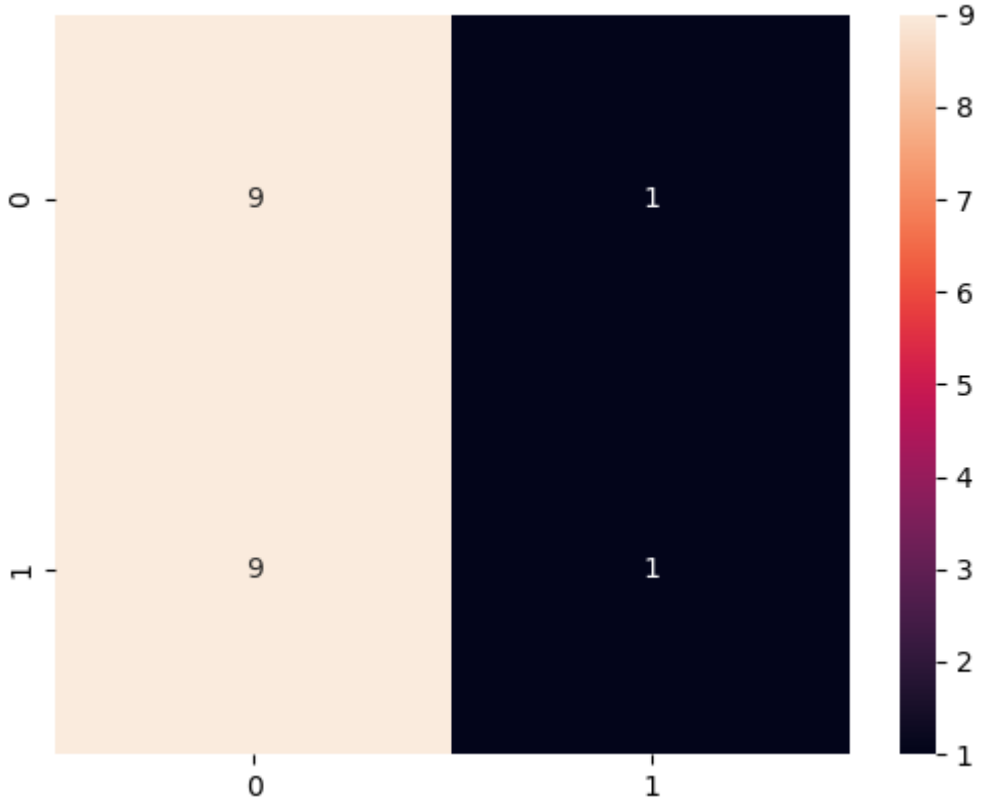
```
##################################################
#########################
### Explore metrics for the ideal weighted ensemble model.
```

```
models = [model1, model2, model3]
preds = [model.predict(x_test) for model in models]
preds=np.array(preds)
ideal_weights = [0.0, 0.1, 90.0]

#Use tensordot to sum the products of all elements over
specified axes.
ideal_weighted_preds = np.tensordot(preds, ideal_weights,
axes=((0),(0)))
ideal_weighted_ensemble_prediction =
np.argmax(ideal_weighted_preds, axis=1)
ideal_weighted_ensemble_prediction =
le.inverse_transform(ideal_weighted_ensemble_prediction)

ideal_weighted_accuracy = accuracy_score(test_labels,
ideal_weighted_ensemble_prediction)


print('Accuracy Score for ideal weighted average ensemble =
', ideal_weighted_accuracy)

from sklearn.metrics import confusion_matrix
```

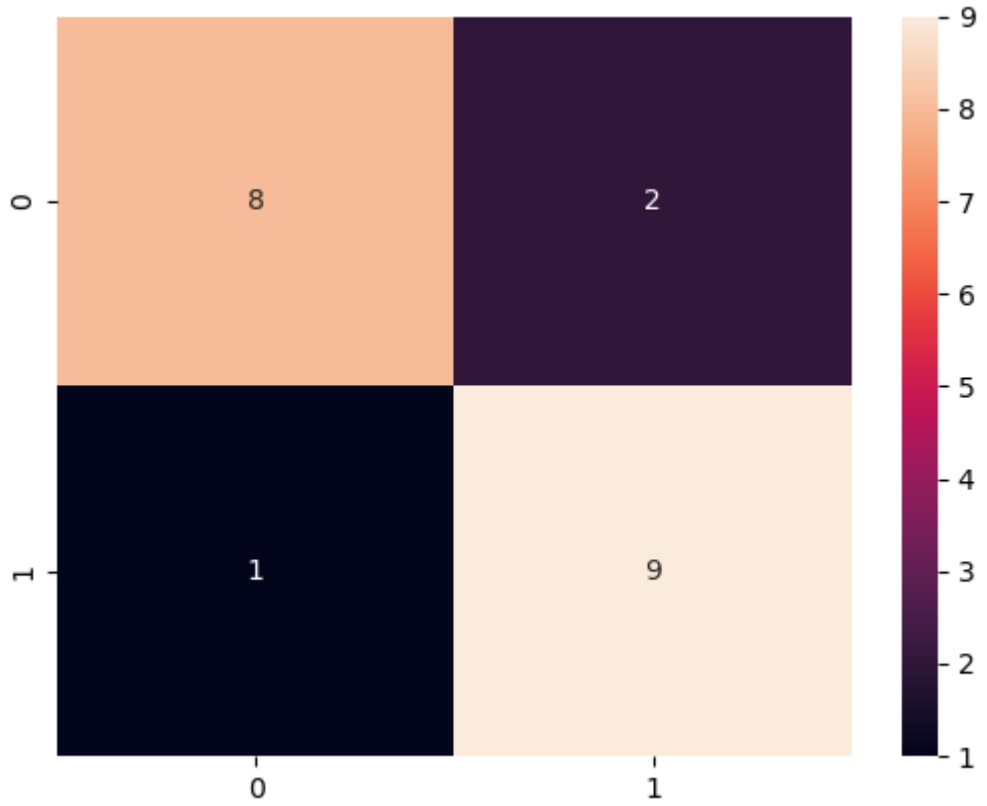
```
cm = confusion_matrix(test_labels, prediction1)
sns.heatmap(cm, annot=True)
```



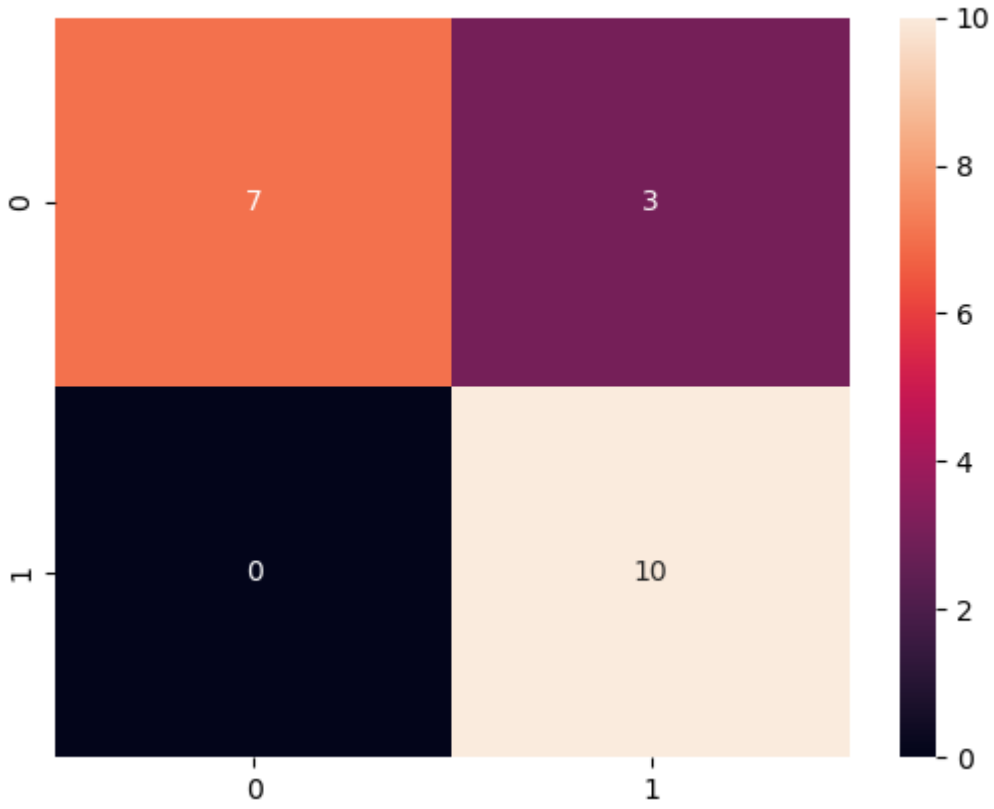

```
cm1 = confusion_matrix(test_labels, prediction2)
sns.heatmap(cm1, annot=True)
```



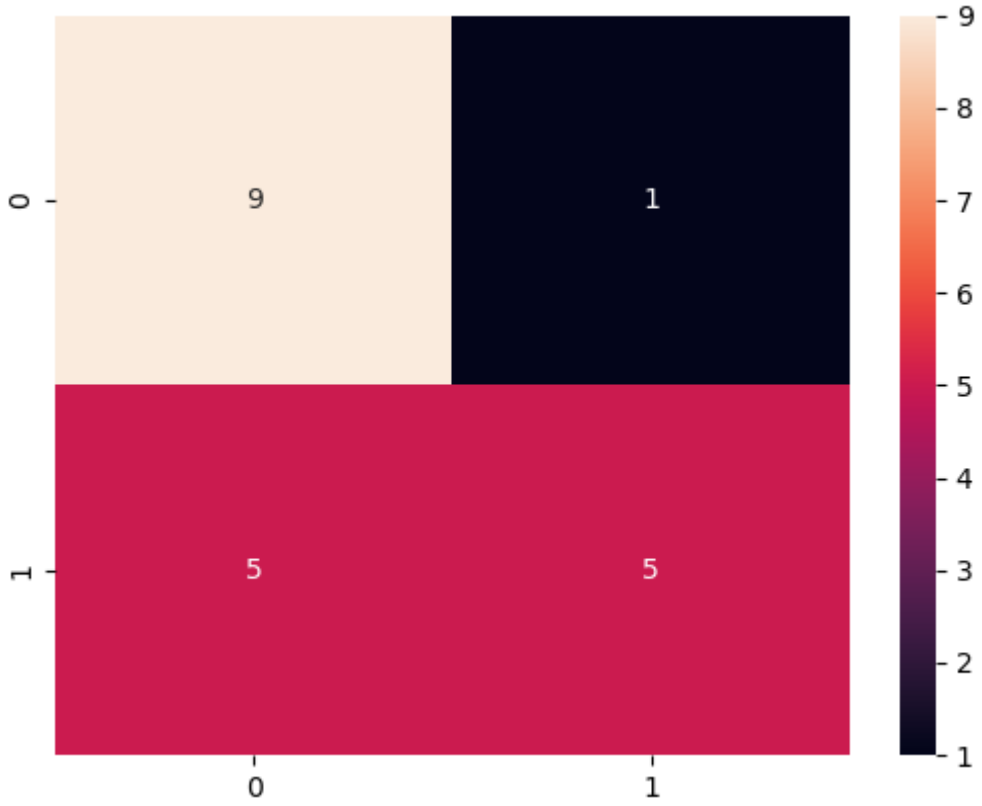

```
cm2 = confusion_matrix(test_labels, prediction3)
sns.heatmap(cm2, annot=True)
```



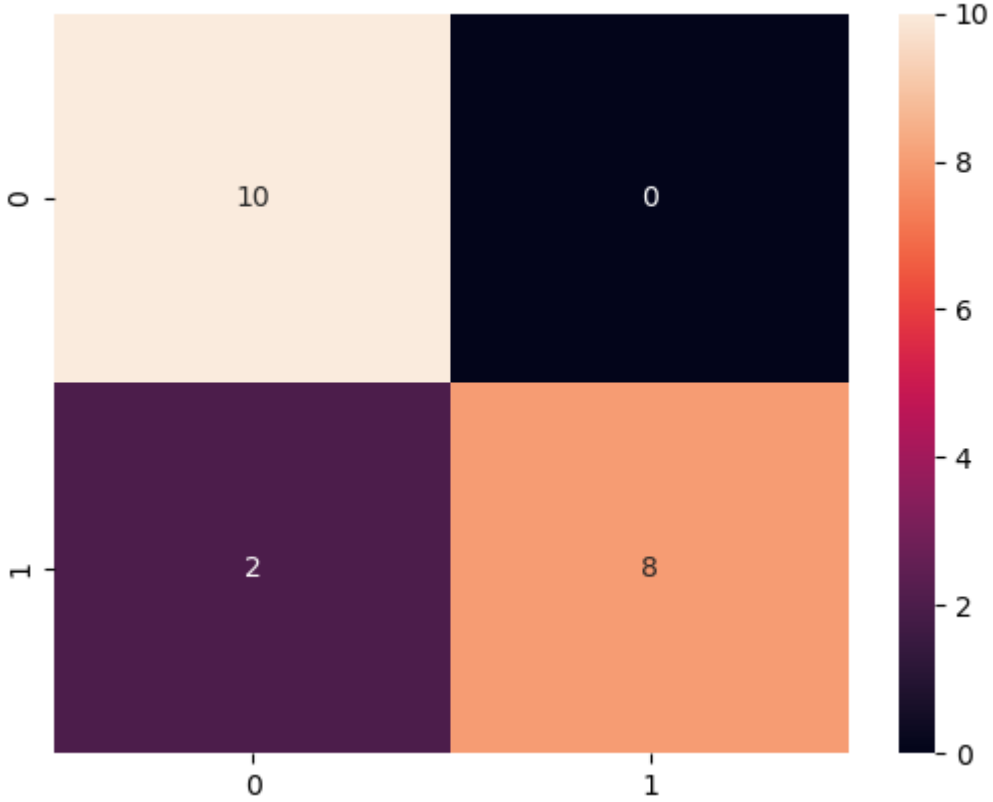

```
cm3 = confusion_matrix(test_labels, prediction4)
sns.heatmap(cm3, annot=True)
```



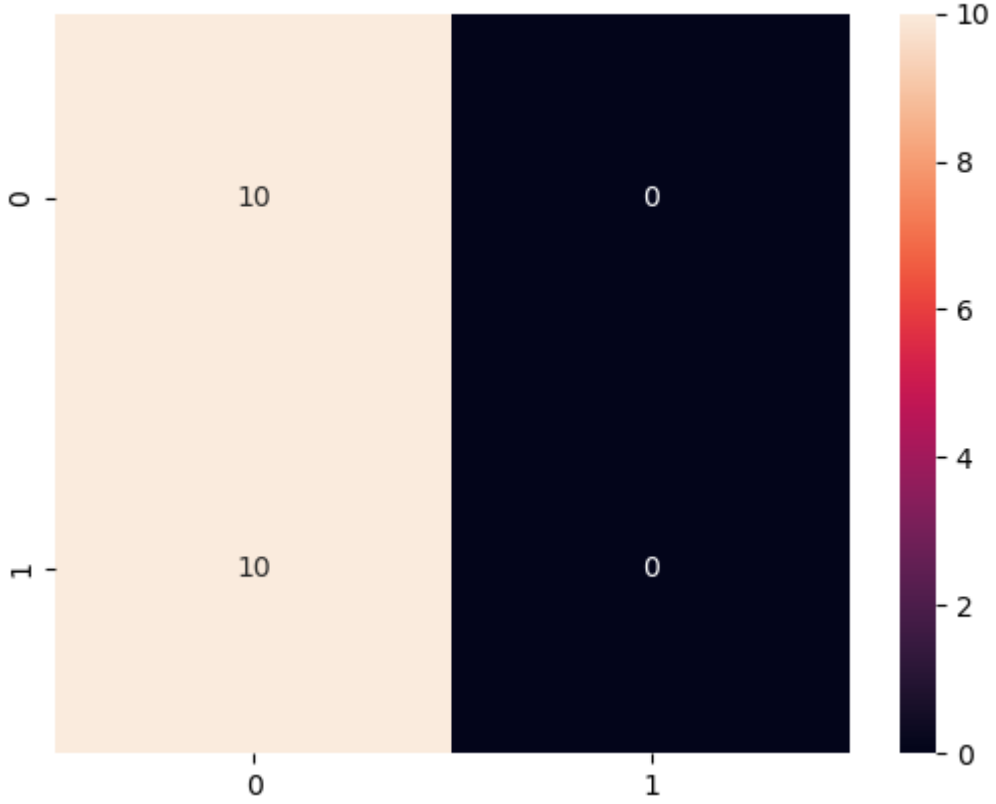

```
cm4 = confusion_matrix(test_labels, prediction5)
sns.heatmap(cm4, annot=True)
```



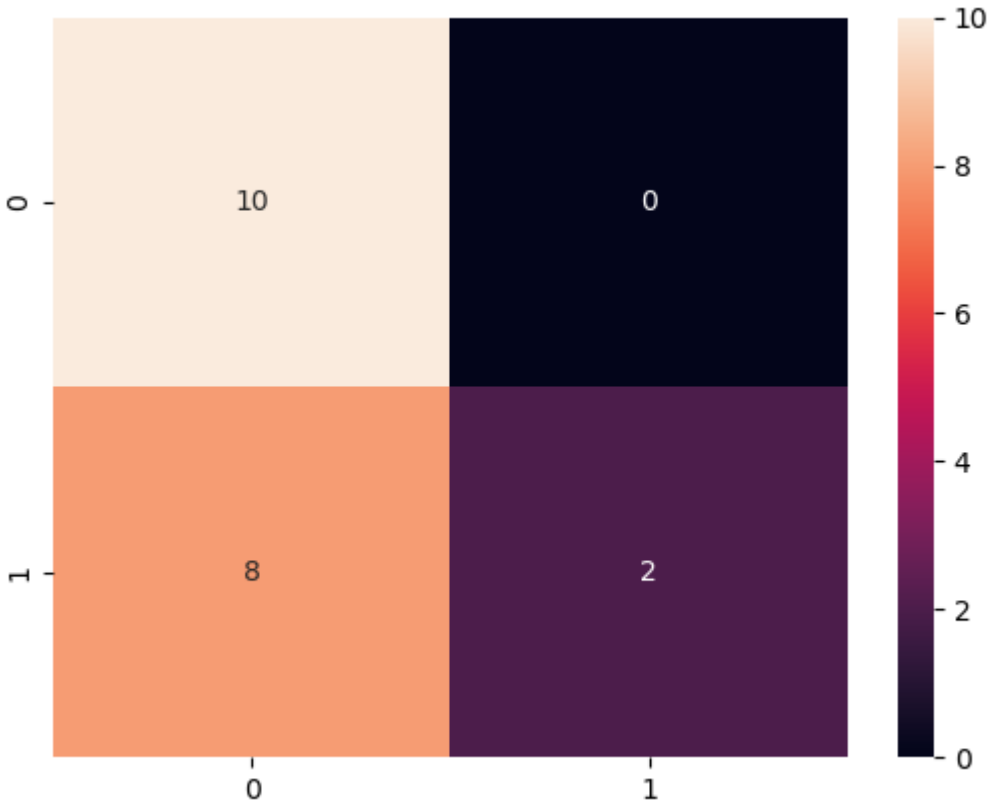

```
cm5 = confusion_matrix(test_labels, prediction6)
sns.heatmap(cm5, annot=True)
```



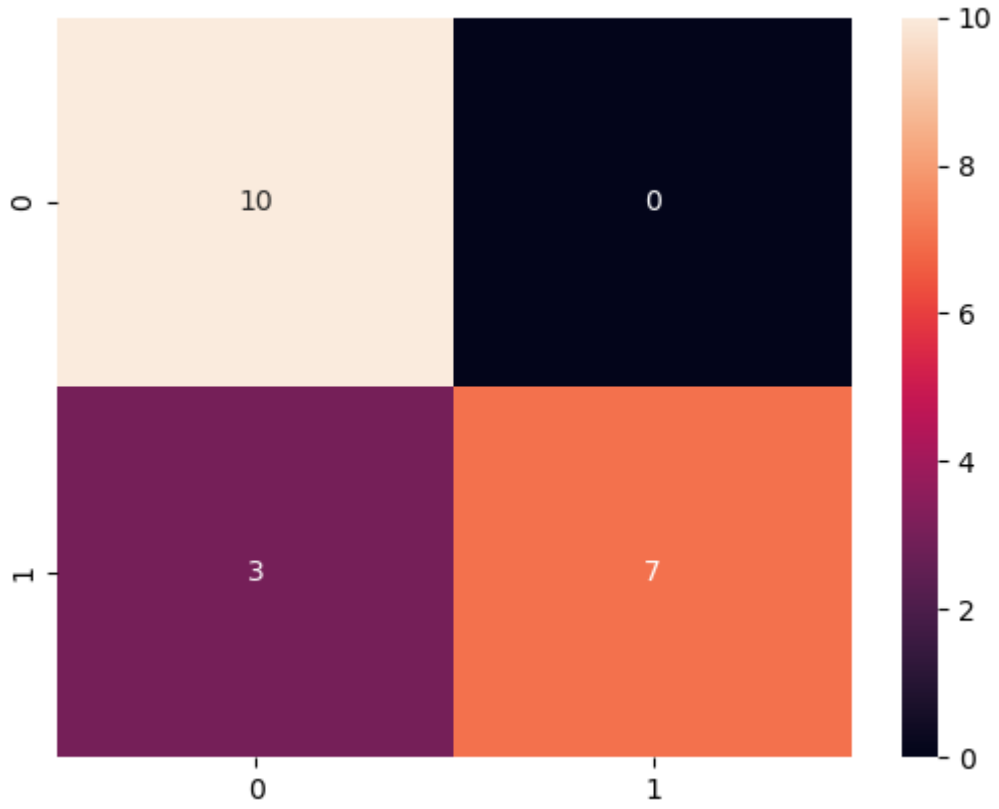

```
cm6 = confusion_matrix(test_labels, prediction7)
sns.heatmap(cm6, annot=True)
```



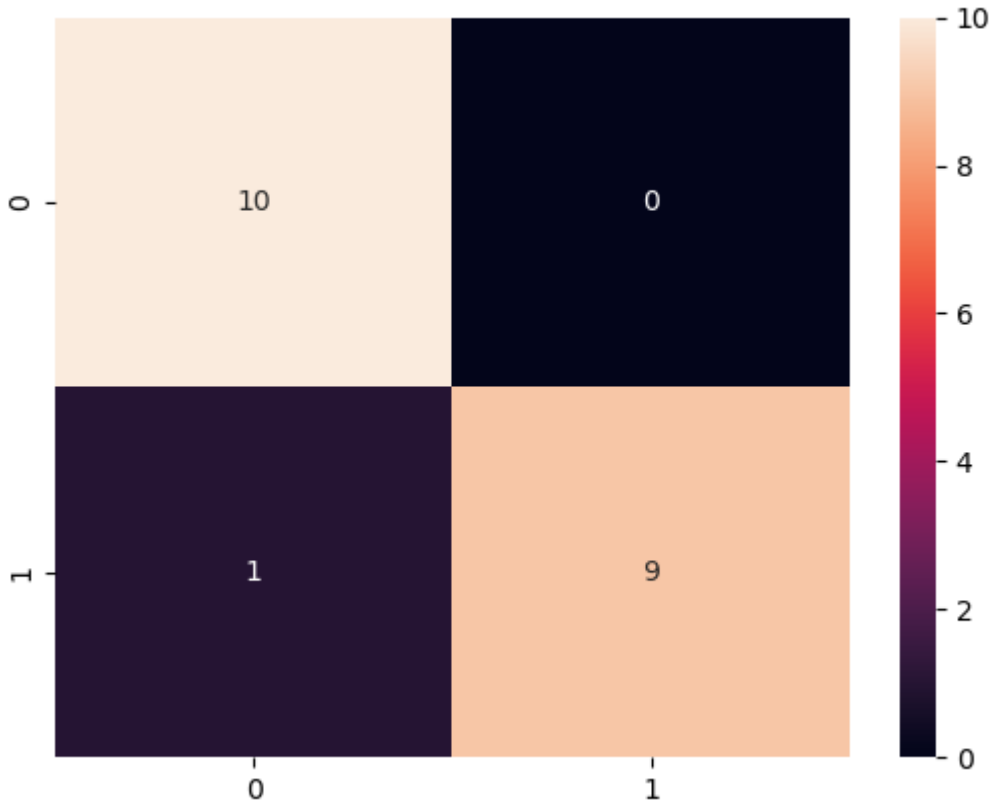

```
cm7 = confusion_matrix(test_labels, prediction8)
sns.heatmap(cm7, annot=True)
```



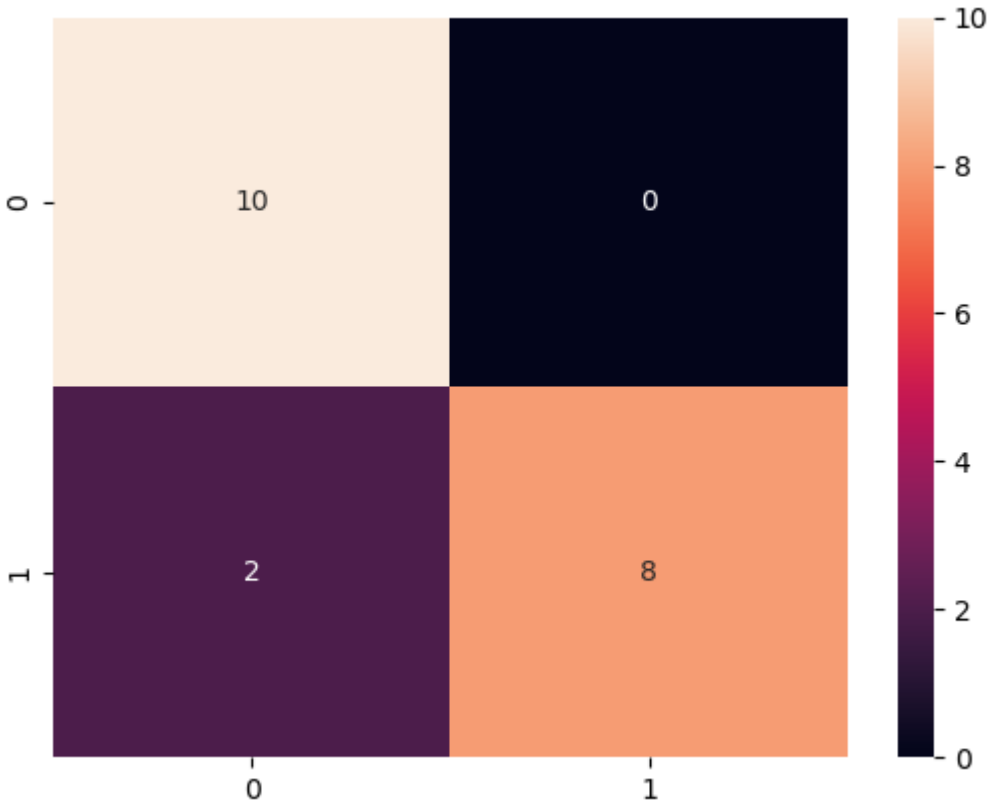

```
cm8 = confusion_matrix(test_labels, prediction9)
sns.heatmap(cm8, annot=True)
```




```
cm9 = confusion_matrix(test_labels, ensemble_prediction)
sns.heatmap(cm9, annot=True)
```




```
cm10 = confusion_matrix(test_labels,
weighted_ensemble_prediction)

sns.heatmap(cm10, annot=True)
```



------Supervised Gan

-*- coding: utf-8 -*-
"""

Created on Sun Sep 11 18:53:17 2022

@author: LAXMINARAYANRO
"""

from numpy import expand_dims, zeros, ones, asarray

from numpy.random import randn, randint

```
#from keras.datasets.mnist import load_data
from keras.optimizers import Adam
from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import (
    LeakyReLU, SeparableConv2D, MaxPooling2D, Activation,
Flatten, Dropout, Conv2D, Dense,Conv2DTranspose,Input,
Dense, Lambda, Reshape
)
#from keras.utils import to_categorical

from matplotlib import pyplot as plt
from keras import backend as K
import numpy as np

import glob
import cv2
import os

SIZE = 28
#Channel=3  #Resize images
```

```
#lists
train_images = []
train_labels = []
#E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Createtry/Gantrain/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)
```

```
# test/validation into lists
#E:/fffiiles/college pracs and projects/Project/Car/Model_images/test/*
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Createtry/Gantest/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

```
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

trainX, trainy, testX, testy = train_images,
train_labels_encoded, test_images, test_labels_encoded

# testy = np.asarray(testy).reshape((-1,1))
# trainy = np.asarray(trainy).reshape((-1,1))
# testy = testy.astype(np.uint8)
# trainy = trainy.astype(np.uint8)
# define the standalone generator model
def define_generator(latent_dim):

    in_lat = Input(shape=(latent_dim,))
    #Start with enough dense nodes to be reshaped and
ConvTransposed to 28x28x1
    n_nodes = 256 * 7 * 7
    X = Dense(n_nodes)(in_lat)
    X = LeakyReLU(alpha=0.2)(X)
    X = Reshape((7, 7, 256))(X)
```

```
    X = Conv2DTranspose(128, (3,3), strides=(2,2),
padding='same')(X) #14x14x128
    X = LeakyReLU(alpha=0.2)(X)


    X = Conv2DTranspose(64, (3,3), strides=(1,1),
padding='same')(X) #14x14x64
    X = LeakyReLU(alpha=0.2)(X)
    # output
    out_layer = Conv2DTranspose(3, (3,3), strides=(2,2),
activation='tanh',
                        padding='same')(X) #28x28x1
    # define model
    model = Model(in_lat, out_layer)
    return model

gen_model=define_generator(100)
#print(gen_model.summary())


# define the base discriminator model for sup and unsup
discriminators
#This is the base discriminator that supervised and
unsupervised are going to share weights from.
```

```
#(I know that the above ssentence is bad english!!!)
def define_discriminator(in_shape=(28,28,3), n_classes=2):
    in_image = Input(shape=in_shape)
    X = Conv2D(32, (3,3), strides=(2,2),
padding='same')(in_image)
    X = LeakyReLU(alpha=0.2)(X)

    X = Conv2D(64, (3,3), strides=(2,2), padding='same')(X)
    X = LeakyReLU(alpha=0.2)(X)

    X = Conv2D(128, (3,3), strides=(2,2), padding='same')(X)
    X = LeakyReLU(alpha=0.2)(X)

    X = Flatten()(X)
    X = Dropout(0.4)(X) #Consider adding more dropout layers
to minimize overfitting - remember we work with limited
labeled data.
    X = Dense(n_classes)(X)

    model = Model(inputs=in_image, outputs=X)

    return model
```

```
#Define the supervised discriminator.

#Multiclass classification, so we will use softmax activation.

#To avoid converting our labels to categorical, we will work
with sparse categorical crossentropy loss.

def define_sup_discriminator(disc):

    model=Sequential()

    model.add(disc)

    model.add(Activation('softmax'))

    #Let us use sparse categorical loss so we dont have to
convert our Y to categorical

    model.compile(optimizer=Adam(learning_rate=0.0002,
beta_1=0.5),

loss="sparse_categorical_crossentropy",metrics=['accuracy'])

    return model



#Define the unsupervised discriminator

#Takes the output of the supervised, just before the softmax
activation.

#Then, adds a layer with calculation of sum of exponential
outputs. (defined below as custom_activation)

# Reference: https://arxiv.org/abs/1606.03498
```

```
#This custom activation layer gives a value close to 0 for smaller activations
#in the prior discriminator layer. It gives values close to 1 for large activations.
#This way it gives low activation for fake images. No need for sigmoid anymore.


# custom activation function for the unsupervised discriminator
#D(x) = Z(x) / (Z(x) + 1) where Z(x) = sum(exp(l(x))). l(x) is the output from sup discr. prior to softmax
def custom_activation(x):
    Z_x = K.sum(K.exp(x), axis=-1, keepdims=True)
    D_x = Z_x /(Z_x+1)

    return D_x

#You can also try the built in tensorflow function: tf.reduce_logsumexp(class_logits, 1)


def define_unsup_discriminator(disc):
    model=Sequential()
```

```
    model.add(disc)
    model.add(Lambda(custom_activation))
    model.compile(loss='binary_crossentropy',
optimizer=Adam(learning_rate=0.0002, beta_1=0.5))
    return model



disc=define_discriminator()
disc_sup=define_sup_discriminator(disc)
disc_unsup=define_unsup_discriminator(disc)
#print(disc_unsup.summary())



# define the combined generator and discriminator model,
for updating the generator
def define_gan(gen_model, disc_unsup):

        disc_unsup.trainable = False # make unsup.
discriminator not trainable
        gan_output = disc_unsup(gen_model.output) #Gen.
output is the input to disc.
        model = Model(gen_model.input, gan_output)
```

```
        model.compile(loss='binary_crossentropy',
optimizer=Adam(learning_rate=0.0002, beta_1=0.5))
        return model

gan_model = define_gan(gen_model, disc_unsup)
# print(gan_model.summary())

# load the images
def load_real_samples(n_classes=2):
    #(trainX, trainy), (_, _) = load_data()
    X = expand_dims(trainX, axis=-1)
    X = X.astype('float32')
    X = (X - 127.5) / 127.5  # scale from [0,255] to [-1,1] as we
will be using tanh activation.
    print(X.shape, trainy.shape)
    return [X, trainy]

#data = load_real_samples()

#select subset of the dataset for supervised training
#Let us pick only 100 samples to be used in supervised
training.
```

```
#Also, we need to ensure we pick 10 samples per class to ensure a good balance
#of data between classes.
def select_supervised_samples(dataset, n_samples=100, n_classes=2):
    X, y = dataset
    X_list, y_list = list(), list()
    n_per_class = int(n_samples / n_classes) #Number of amples per class.
    for i in range(n_classes):
    X_with_class = X[y == i] # get all images for this class
    ix = randint(0, len(X_with_class), n_per_class) # choose random images for each class
    [X_list.append(X_with_class[j]) for j in ix] # add to list
    [y_list.append(i) for j in ix]
    return asarray(X_list), asarray(y_list) #Returns a list of 2 numpy arrays corresponding to X and Y


# Pick real samples from the dataset.
#Return both images and corresponding labels in addition to y=1 indicating that the images are real.
#Remember that we will not use the labels for unsupervised, only used for supervised.
```

```
def generate_real_samples(dataset, n_samples):

    images, labels = dataset
    ix = randint(0, images.shape[0], n_samples)
    X, labels = images[ix], labels[ix] #Select random images and corresponding labels
    y = ones((n_samples, 1)) #Label all images as 1 as these are real images. (for the discriminator training)
    return [X, labels], y

# generate latent points, to be used as inputs to the generator.
def generate_latent_points(latent_dim, n_samples):
    z_input = randn(latent_dim * n_samples)
    z_input = z_input.reshape(n_samples, latent_dim) # reshape for input to the network
    return z_input

# Generate fake images using the generator and above latent points as input to it.
#We do not care about labeles so the generator will not know anything about the labels.
def generate_fake_samples(generator, latent_dim, n_samples):
```

```
    z_input = generate_latent_points(latent_dim,
n_samples)
    fake_images = generator.predict(z_input)
    # create class labels
    y = zeros((n_samples, 1)) #Label all images as 0 as these
are fake images. (for the discriminator training)
    return fake_images, y

# report accuracy and save plots & the model periodically.
def summarize_performance(step, gen_model, disc_sup,
latent_dim, dataset, n_samples=100):
    # Generate fake images
    X, _ = generate_fake_samples(gen_model, latent_dim,
n_samples)

    X = (X + 1) / 2.0 # scale to [0,1] for plotting
    # plot images
    for i in range(100):
        plt.subplot(10, 10, 1 + i)
        plt.axis('off')
        plt.imshow(X[i, :, :, 0], cmap='gray_r')
    # save plot to drive
```

```
        filename1 = 'generated_plot_%04d.png' % (step+1)
        plt.savefig(filename1)
        plt.close()

        # evaluate the discriminator
        X, y = dataset
        _, acc = disc_sup.evaluate(X, y, verbose=0)
        print('Discriminator Accuracy: %.3f%%' % (acc * 100))
        # save the generator model
        filename2 = 'gen_model_%04d.h5' % (step+1)
        gen_model.save(filename2)
        # save the Discriminator (classifier) model
        filename3 = 'disc_sup_%04d.h5' % (step+1)
        disc_sup.save(filename3)
        print('>Saved: %s, %s, and %s' % (filename1, filename2,
filename3))

# train the generator and discriminator
def train(gen_model, disc_unsup, disc_sup, gan_model,
dataset, latent_dim, n_epochs=20, n_batch=100):

    # select supervised dataset for training.
```

```
    #Remember that we are not using all 60k images, just a
subset (100 images, 10 per class. )
        X_sup, y_sup = select_supervised_samples(dataset)
        #print(X_sup.shape, y_sup.shape)

        bat_per_epo = int(dataset[0].shape[0] / n_batch)
        # iterations
        n_steps = bat_per_epo * n_epochs

        half_batch = int(n_batch / 2)
        print('n_epochs=%d, n_batch=%d, 1/2=%d, b/e=%d,
steps=%d' % (n_epochs,
                                        n_batch, half_batch,
                                        bat_per_epo, n_steps))

    #  enumerate epochs
        for i in range(n_steps):
                # update supervised discriminator (disc_sup) on
real samples.
        #Remember that we use real labels to train as this is
supervised.
        #This is the discriminator we really care about at the
end.
```

```
#Also, this is a multiclass classifier, not binary. Therefore, our y values
#will be the real class labels for MNIST. (NOT 1 or 0 indicating real or fake.)
        [Xsup_real, ysup_real], _ = generate_real_samples([X_sup, y_sup], half_batch)
        sup_loss, sup_acc = disc_sup.train_on_batch(Xsup_real, ysup_real)

        # update unsupervised discriminator (disc_unsup) - just like in our regular GAN.
#Remember that we will not train on labels as this is unsupervised, just binary as in our regular GAN.
#The y_real below indicates 1s telling the discriminator that these images are real.
#do not confuse this with class labels.
#We will discard this discriminator at the end.
        [X_real, _], y_real = generate_real_samples(dataset, half_batch) #
        d_loss_real = disc_unsup.train_on_batch(X_real, y_real)
#Now train on fake.
        X_fake, y_fake = generate_fake_samples(gen_model, latent_dim, half_batch)
```

```
            d_loss_fake = disc_unsup.train_on_batch(X_fake,
y_fake)


            # update generator (gen) - like we do in regular
GAN.
      #We can discard this model at the end as our primary
goal is to train a multiclass classifier (sup. disc.)
            X_gan, y_gan = generate_latent_points(latent_dim,
n_batch), ones((n_batch, 1))
            gan_loss = gan_model.train_on_batch(X_gan,
y_gan)


            # summarize loss on this batch
            print('>%d, c[%.3f,%.0f], d[%.3f,%.3f], g[%.3f]' %
(i+1, sup_loss, sup_acc*100, d_loss_real, d_loss_fake,
gan_loss))
            # evaluate the model performance periodically
            if (i+1) % (bat_per_epo * 1) == 0:
                  summarize_performance(i, gen_model,
disc_sup, latent_dim, dataset)


##################################################
################################
# TRAIN
```

```
##################################

latent_dim = 100

# create the discriminator models
disc=define_discriminator() #Bare discriminator model...
disc_sup=define_sup_discriminator(disc) #Supervised discriminator model
disc_unsup=define_unsup_discriminator(disc) #Unsupervised discriminator model.

gen_model = define_generator(latent_dim) #Generator
gan_model = define_gan(gen_model, disc_unsup) #GAN
dataset = load_real_samples() #Define the dataset by loading real samples. (This will be a list of 2 numpy arrays, X and y)

# train the model
# NOTE: 1 epoch = 600 steps in this example.
train(gen_model, disc_unsup, disc_sup, gan_model, dataset, latent_dim, n_epochs=10, n_batch=100)
n_batch=100)
n_epochs=10, n_batch=100, 1/2=50, b/e=17, steps=170
2/2 [==============================] - 1s 59ms/step
```

```
>1, c[0.687,54], d[0.402,1.101], g[0.405]
2/2 [==============================] - 0s 35ms/step
>2, c[0.691,50], d[0.381,1.102], g[0.404]
2/2 [==============================] - 0s 32ms/step
>3, c[0.684,54], d[0.363,1.104], g[0.403]
2/2 [==============================] - 0s 31ms/step


2022-09-11 20:12:36.037754: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found
2022-09-11 20:12:36.189543: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2022-09-11 20:12:36.575562: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX AVX2
```

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-11 20:24:59.669138: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

>4, c[0.675,54], d[0.350,1.108], g[0.401]

2/2 [==============================] - 0s 31ms/step

>5, c[0.657,62], d[0.342,1.115], g[0.398]

2/2 [==============================] - 0s 33ms/step

>6, c[0.697,40], d[0.319,1.123], g[0.394]

2/2 [==============================] - 0s 33ms/step

>7, c[0.662,60], d[0.305,1.135], g[0.390]

2/2 [==============================] - 0s 32ms/step

>8, c[0.671,64], d[0.292,1.144], g[0.387]

2/2 [==============================] - 0s 34ms/step

>9, c[0.654,68], d[0.278,1.152], g[0.386]

2/2 [==============================] - 0s 35ms/step

>10, c[0.626,76], d[0.261,1.154], g[0.386]

2/2 [==============================] - 0s 35ms/step

>11, c[0.665,58], d[0.258,1.157], g[0.389]

2/2 [==============================] - 0s 35ms/step

>12, c[0.655,66], d[0.231,1.146], g[0.394]

2/2 [==============================] - 0s 39ms/step

```
>13, c[0.626,80], d[0.213,1.134], g[0.402]
2/2 [==============================] - 0s 32ms/step
>14, c[0.610,78], d[0.216,1.116], g[0.413]
2/2 [==============================] - 0s 30ms/step
>15, c[0.618,78], d[0.203,1.099], g[0.424]
2/2 [==============================] - 0s 32ms/step
>16, c[0.628,64], d[0.194,1.087], g[0.433]
2/2 [==============================] - 0s 33ms/step
>17, c[0.625,68], d[0.183,1.081], g[0.440]
4/4 [==============================] - 0s 39ms/step
Discriminator Accuracy: 54.954%
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
>Saved: generated_plot_0017.png, gen_model_0017.h5, and disc_sup_0017.h5
2/2 [==============================] - 0s 31ms/step
>18, c[0.638,70], d[0.175,1.084], g[0.439]
2/2 [==============================] - 0s 31ms/step
>19, c[0.600,70], d[0.188,1.101], g[0.435]
2/2 [==============================] - 0s 32ms/step
>20, c[0.625,64], d[0.159,1.097], g[0.436]
```

```
2/2 [==============================] - 0s 34ms/step
>21, c[0.590,70], d[0.188,1.103], g[0.430]
2/2 [==============================] - 0s 31ms/step
>22, c[0.541,80], d[0.168,1.113], g[0.429]
2/2 [==============================] - 0s 31ms/step
>23, c[0.583,72], d[0.184,1.116], g[0.429]
2/2 [==============================] - 0s 31ms/step
>24, c[0.563,76], d[0.136,1.118], g[0.433]
2/2 [==============================] - 0s 31ms/step
>25, c[0.654,68], d[0.130,1.125], g[0.435]
2/2 [==============================] - 0s 30ms/step
>26, c[0.590,72], d[0.113,1.153], g[0.432]
2/2 [==============================] - 0s 29ms/step
>27, c[0.609,70], d[0.144,1.185], g[0.429]
2/2 [==============================] - 0s 32ms/step
>28, c[0.572,72], d[0.129,1.196], g[0.429]
2/2 [==============================] - 0s 33ms/step
>29, c[0.588,68], d[0.151,1.196], g[0.444]
2/2 [==============================] - 0s 35ms/step
>30, c[0.563,80], d[0.134,1.145], g[0.471]
2/2 [==============================] - 0s 33ms/step
>31, c[0.589,70], d[0.172,1.072], g[0.514]
```

```
2/2 [==============================] - 0s 32ms/step
>32, c[0.590,74], d[0.160,0.990], g[0.583]
2/2 [==============================] - 0s 30ms/step
>33, c[0.558,68], d[0.148,0.873], g[0.645]
2/2 [==============================] - 0s 37ms/step
>34, c[0.510,78], d[0.144,0.823], g[0.685]
4/4 [==============================] - 0s 36ms/step
Discriminator Accuracy: 53.111%
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
>Saved: generated_plot_0034.png, gen_model_0034.h5, and disc_sup_0034.h5
2/2 [==============================] - 0s 32ms/step
>35, c[0.553,74], d[0.161,0.783], g[0.702]
2/2 [==============================] - 0s 29ms/step
>36, c[0.499,80], d[0.131,0.790], g[0.687]
2/2 [==============================] - 0s 29ms/step
>37, c[0.595,64], d[0.144,0.865], g[0.628]
2/2 [==============================] - 0s 32ms/step
>38, c[0.491,86], d[0.122,0.971], g[0.548]
2/2 [==============================] - 0s 33ms/step
```

```
>39, c[0.508,82], d[0.123,1.054], g[0.494]
2/2 [==============================] - 0s 29ms/step
>40, c[0.545,72], d[0.137,1.091], g[0.470]
2/2 [==============================] - 0s 32ms/step
>41, c[0.524,82], d[0.125,1.101], g[0.474]
2/2 [==============================] - 0s 32ms/step
>42, c[0.445,82], d[0.113,1.091], g[0.485]
2/2 [==============================] - 0s 30ms/step
>43, c[0.490,80], d[0.104,1.073], g[0.503]
2/2 [==============================] - 0s 40ms/step
>44, c[0.545,66], d[0.118,1.011], g[0.534]
2/2 [==============================] - 0s 30ms/step
>45, c[0.479,84], d[0.146,0.948], g[0.573]
2/2 [==============================] - 0s 39ms/step
>46, c[0.456,84], d[0.136,0.867], g[0.618]
2/2 [==============================] - 0s 29ms/step
>47, c[0.503,78], d[0.162,0.820], g[0.650]
2/2 [==============================] - 0s 32ms/step
>48, c[0.594,68], d[0.139,0.808], g[0.657]
2/2 [==============================] - 0s 33ms/step
>49, c[0.420,94], d[0.133,0.825], g[0.636]
2/2 [==============================] - 0s 39ms/step
```

```
>50, c[0.447,82], d[0.135,0.887], g[0.590]
2/2 [==============================] - 0s 32ms/step
>51, c[0.511,80], d[0.127,0.976], g[0.536]
4/4 [==============================] - 0s 42ms/step
Discriminator Accuracy: 53.456%
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
>Saved: generated_plot_0051.png, gen_model_0051.h5, and disc_sup_0051.h5
2/2 [==============================] - 0s 31ms/step
>52, c[0.387,84], d[0.143,1.043], g[0.493]
2/2 [==============================] - 0s 40ms/step
>53, c[0.429,86], d[0.181,1.055], g[0.492]
2/2 [==============================] - 0s 34ms/step
>54, c[0.480,82], d[0.127,1.013], g[0.529]
2/2 [==============================] - 0s 39ms/step
>55, c[0.494,78], d[0.101,0.914], g[0.596]
2/2 [==============================] - 0s 32ms/step
>56, c[0.450,78], d[0.151,0.831], g[0.651]
2/2 [==============================] - 0s 30ms/step
>57, c[0.442,78], d[0.094,0.752], g[0.725]
```

```
2/2 [==============================] - 0s 31ms/step
>58, c[0.426,82], d[0.103,0.692], g[0.774]
2/2 [==============================] - 0s 31ms/step
>59, c[0.404,86], d[0.090,0.647], g[0.826]
2/2 [==============================] - 0s 29ms/step
>60, c[0.546,68], d[0.104,0.619], g[0.854]
2/2 [==============================] - 0s 32ms/step
>61, c[0.421,82], d[0.069,0.607], g[0.857]
2/2 [==============================] - 0s 31ms/step
>62, c[0.378,88], d[0.083,0.625], g[0.838]
2/2 [==============================] - 0s 38ms/step
>63, c[0.372,90], d[0.079,0.627], g[0.822]
2/2 [==============================] - 0s 40ms/step
>64, c[0.291,96], d[0.056,0.636], g[0.821]
2/2 [==============================] - 0s 31ms/step
>65, c[0.456,82], d[0.070,0.625], g[0.843]
2/2 [==============================] - 0s 30ms/step
>66, c[0.368,86], d[0.070,0.595], g[0.871]
2/2 [==============================] - 0s 29ms/step
>67, c[0.371,90], d[0.067,0.556], g[0.935]
2/2 [==============================] - 0s 33ms/step
>68, c[0.349,86], d[0.064,0.510], g[1.015]
```

```
4/4 [==============================] - 0s 45ms/step

Discriminator Accuracy: 54.090%

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

>Saved: generated_plot_0068.png, gen_model_0068.h5, and disc_sup_0068.h5

2/2 [==============================] - 0s 36ms/step


2022-09-11 20:12:36.037754: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-11 20:12:36.189543: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-11 20:12:36.575562: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural
```

Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-11 20:24:59.669138: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:26:04.608708: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

>69, c[0.371,92], d[0.071,0.459], g[1.089]

2/2 [==============================] - 0s 34ms/step

>70, c[0.416,84], d[0.079,0.418], g[1.184]

2/2 [==============================] - 0s 30ms/step

>71, c[0.388,84], d[0.057,0.382], g[1.232]

2/2 [==============================] - 0s 29ms/step

>72, c[0.348,86], d[0.037,0.368], g[1.291]

2/2 [==============================] - 0s 29ms/step

>73, c[0.360,88], d[0.040,0.345], g[1.321]

2/2 [==============================] - 0s 40ms/step

>74, c[0.367,86], d[0.040,0.325], g[1.361]

2/2 [==============================] - 0s 31ms/step

>75, c[0.452,76], d[0.049,0.325], g[1.371]

```
2/2 [==============================] - 0s 33ms/step
>76, c[0.325,90], d[0.037,0.322], g[1.406]
2/2 [==============================] - 0s 39ms/step
>77, c[0.339,92], d[0.030,0.324], g[1.446]
2/2 [==============================] - 0s 30ms/step
>78, c[0.399,82], d[0.044,0.301], g[1.458]
2/2 [==============================] - 0s 33ms/step
>79, c[0.293,92], d[0.053,0.319], g[1.501]
2/2 [==============================] - 0s 31ms/step
>80, c[0.298,94], d[0.060,0.308], g[1.512]
2/2 [==============================] - 0s 31ms/step
>81, c[0.268,98], d[0.042,0.305], g[1.624]
2/2 [==============================] - 0s 31ms/step
>82, c[0.424,86], d[0.068,0.293], g[1.735]
2/2 [==============================] - 0s 31ms/step
>83, c[0.325,94], d[0.073,0.283], g[1.834]
2/2 [==============================] - 0s 30ms/step
>84, c[0.326,86], d[0.076,0.280], g[1.985]
2/2 [==============================] - 0s 30ms/step
>85, c[0.268,92], d[0.098,0.271], g[1.929]
4/4 [==============================] - 0s 41ms/step
Discriminator Accuracy: 53.859%
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

>Saved: generated_plot_0085.png, gen_model_0085.h5, and disc_sup_0085.h5

2/2 [==============================] - 0s 33ms/step

>86, c[0.312,94], d[0.097,0.369], g[2.011]

2/2 [==============================] - 0s 32ms/step

>87, c[0.362,82], d[0.159,0.367], g[1.973]

2/2 [==============================] - 0s 30ms/step

>88, c[0.374,92], d[0.106,0.271], g[2.435]

2/2 [==============================] - 0s 32ms/step

>89, c[0.282,94], d[0.176,0.318], g[2.393]

2/2 [==============================] - 0s 33ms/step

>90, c[0.296,94], d[0.171,0.275], g[2.665]

2/2 [==============================] - 0s 35ms/step

>91, c[0.303,92], d[0.194,0.202], g[2.755]

2/2 [==============================] - 0s 30ms/step

>92, c[0.278,98], d[0.213,0.363], g[3.099]

2/2 [==============================] - 0s 30ms/step

>93, c[0.255,96], d[0.309,0.256], g[3.435]

2/2 [==============================] - 0s 33ms/step
```

```
>94, c[0.380,82], d[0.206,0.287], g[4.016]
2/2 [==============================] - 0s 31ms/step
>95, c[0.354,86], d[0.188,0.105], g[3.431]
2/2 [==============================] - 0s 33ms/step
>96, c[0.267,94], d[0.231,0.480], g[4.681]
2/2 [==============================] - 0s 30ms/step
>97, c[0.335,88], d[0.493,0.071], g[3.186]
2/2 [==============================] - 0s 30ms/step
>98, c[0.378,84], d[0.240,0.463], g[3.973]
2/2 [==============================] - 0s 30ms/step
>99, c[0.309,90], d[0.323,0.121], g[3.336]
2/2 [==============================] - 0s 29ms/step
>100, c[0.337,86], d[0.330,0.587], g[3.612]
2/2 [==============================] - 0s 33ms/step
>101, c[0.288,94], d[0.267,0.075], g[3.582]
2/2 [==============================] - 0s 30ms/step
>102, c[0.305,96], d[0.202,0.191], g[3.119]
4/4 [==============================] - 0s 39ms/step
Discriminator Accuracy: 53.687%
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

```
>Saved: generated_plot_0102.png, gen_model_0102.h5, and disc_sup_0102.h5
2/2 [==============================] - 0s 36ms/step
>103, c[0.280,88], d[0.366,0.441], g[3.882]
2/2 [==============================] - 0s 29ms/step
>104, c[0.267,96], d[0.492,0.090], g[3.392]
2/2 [==============================] - 0s 49ms/step
>105, c[0.504,74], d[0.873,0.853], g[3.249]
2/2 [==============================] - 0s 36ms/step
>106, c[0.326,86], d[0.449,0.096], g[3.222]
2/2 [==============================] - 0s 33ms/step
>107, c[0.241,96], d[0.550,0.329], g[2.932]
2/2 [==============================] - 0s 31ms/step
>108, c[0.250,94], d[0.319,0.158], g[3.109]
2/2 [==============================] - 0s 31ms/step
>109, c[0.357,86], d[0.488,0.244], g[3.049]
2/2 [==============================] - 0s 31ms/step
>110, c[0.323,90], d[0.319,0.189], g[2.933]
2/2 [==============================] - 0s 33ms/step
>111, c[0.313,92], d[0.402,0.188], g[2.921]
2/2 [==============================] - 0s 34ms/step
>112, c[0.287,90], d[0.363,0.260], g[2.779]
```

```
2/2 [==============================] - 0s 32ms/step
>113, c[0.280,92], d[0.266,0.115], g[3.271]
2/2 [==============================] - 0s 31ms/step
>114, c[0.228,92], d[0.380,0.215], g[2.624]
2/2 [==============================] - 0s 30ms/step
>115, c[0.375,88], d[0.320,0.420], g[2.813]
2/2 [==============================] - 0s 30ms/step
>116, c[0.265,90], d[0.424,0.317], g[2.949]
2/2 [==============================] - 0s 32ms/step
>117, c[0.235,94], d[0.548,0.492], g[2.662]
2/2 [==============================] - 0s 30ms/step
>118, c[0.222,96], d[0.408,0.564], g[2.882]
2/2 [==============================] - 0s 31ms/step
>119, c[0.321,86], d[0.273,0.433], g[3.018]
4/4 [==============================] - 0s 38ms/step
Discriminator Accuracy: 53.341%
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
>Saved: generated_plot_0119.png, gen_model_0119.h5, and disc_sup_0119.h5
2/2 [==============================] - 0s 33ms/step
```

```
>120, c[0.218,98], d[0.414,0.656], g[2.666]
2/2 [==============================] - 0s 36ms/step
>121, c[0.273,92], d[0.238,0.474], g[3.285]
2/2 [==============================] - 0s 29ms/step
>122, c[0.273,92], d[0.337,0.738], g[3.184]
2/2 [==============================] - 0s 30ms/step
>123, c[0.355,88], d[0.465,0.589], g[2.825]
2/2 [==============================] - 0s 30ms/step
>124, c[0.229,92], d[0.426,0.483], g[2.401]
2/2 [==============================] - 0s 35ms/step
>125, c[0.294,92], d[0.444,0.686], g[2.570]
2/2 [==============================] - 0s 30ms/step
>126, c[0.278,92], d[0.345,0.977], g[2.827]
2/2 [==============================] - 0s 31ms/step
>127, c[0.245,96], d[0.690,0.813], g[2.484]
2/2 [==============================] - 0s 30ms/step
>128, c[0.233,96], d[0.678,0.717], g[2.752]
2/2 [==============================] - 0s 32ms/step
>129, c[0.241,96], d[0.626,0.492], g[2.372]
2/2 [==============================] - 0s 33ms/step
>130, c[0.182,100], d[0.555,0.363], g[2.217]
2/2 [==============================] - 0s 32ms/step
```

```
>131, c[0.262,94], d[0.578,0.998], g[2.568]
2/2 [==============================] - 0s 32ms/step
>132, c[0.191,96], d[0.729,0.489], g[2.338]
2/2 [==============================] - 0s 32ms/step
>133, c[0.268,92], d[0.494,0.521], g[2.541]
2/2 [==============================] - 0s 34ms/step
>134, c[0.278,94], d[0.981,0.757], g[2.406]
2/2 [==============================] - 0s 31ms/step
>135, c[0.203,94], d[0.655,0.413], g[2.315]
2/2 [==============================] - 0s 36ms/step
>136, c[0.260,92], d[0.799,0.716], g[2.270]
4/4 [==============================] - 0s 39ms/step
Discriminator Accuracy: 52.304%
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
>Saved: generated_plot_0136.png, gen_model_0136.h5, and disc_sup_0136.h5
2/2 [==============================] - 0s 38ms/step


2022-09-11 20:12:36.037754: W tensorflow/stream_executor/platform/default/dso_loader.cc
```

:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-11 20:12:36.189543: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-11 20:12:36.575562: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-11 20:24:59.669138: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:26:04.608708: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:02.445514: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:02.488046: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

>137, c[0.332,92], d[0.933,0.639], g[2.256]

2/2 [==========] - 0s 36ms/step

>138, c[0.270,92], d[0.818,0.548], g[1.924]

2/2 [==========] - 0s 37ms/step

2022-09-11 20:12:36.037754: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-11 20:12:36.189543: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-11 20:12:36.575562: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-11 20:24:59.669138: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:26:04.608708: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:02.445514: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:02.488046: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:04.756422: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:04.795108: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

>139, c[0.227,94], d[0.800,1.002], g[1.724]

2/2 [==========] - 0s 38ms/step

>140, c[0.233,96], d[0.789,0.708], g[1.711]

2/2 [==========] - 0s 33ms/step

>141, c[0.234,96], d[0.862,1.012], g[1.757]

```
2/2 [==============================] - 0s 40ms/step
>142, c[0.234,96], d[0.812,0.660], g[1.614]
2/2 [==============================] - 0s 33ms/step
>143, c[0.234,96], d[1.150,0.838], g[1.265]
2/2 [==============================] - 0s 32ms/step
>144, c[0.194,98], d[0.975,1.029], g[1.337]
2/2 [==============================] - 0s 32ms/step
>145, c[0.242,92], d[0.896,1.032], g[1.468]
2/2 [==============================] - 0s 34ms/step
>146, c[0.198,98], d[1.092,0.871], g[1.395]
2/2 [==============================] - 0s 35ms/step
>147, c[0.261,94], d[0.898,0.824], g[1.273]
2/2 [==============================] - 0s 29ms/step
>148, c[0.204,100], d[1.120,0.813], g[1.129]
2/2 [==============================] - 0s 41ms/step
>149, c[0.213,98], d[0.980,1.112], g[1.160]
2/2 [==============================] - 0s 40ms/step
>150, c[0.200,100], d[0.878,0.910], g[1.274]
2/2 [==============================] - 0s 30ms/step
>151, c[0.299,88], d[1.073,1.022], g[1.229]
2/2 [==============================] - 0s 31ms/step
>152, c[0.240,94], d[0.895,1.045], g[1.126]
```

```
2/2 [==============================] - 0s 30ms/step

>153, c[0.233,98], d[0.754,1.026], g[1.091]

4/4 [==============================] - 0s 40ms/step

Discriminator Accuracy: 52.938%

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

>Saved: generated_plot_0153.png, gen_model_0153.h5, and disc_sup_0153.h5

2/2 [==============================] - 0s 33ms/step

>154, c[0.256,96], d[0.826,0.760], g[1.225]

2/2 [==============================] - 0s 32ms/step

>155, c[0.193,100], d[0.720,0.778], g[1.253]

2/2 [==============================] - 0s 37ms/step


2022-09-11 20:12:36.037754: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-11 20:12:36.189543: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at
```

https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-11 20:12:36.575562: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-11 20:24:59.669138: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:26:04.608708: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:02.445514: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:02.488046: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:04.756422: W tensorflow/core/data/root_dataset.cc:247] Optimization loop failed: CANCELLED: Operation was cancelled

```
2022-09-11 20:27:04.795108: W
tensorflow/core/data/root_dataset.cc:247] Optimization
loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:19.389924: W
tensorflow/core/data/root_dataset.cc:247] Optimization
loop failed: CANCELLED: Operation was cancelled

2022-09-11 20:27:19.740192: W
tensorflow/core/data/root_dataset.cc:247] Optimization
loop failed: CANCELLED: Operation was cancelled

>156, c[0.196,100], d[0.665,0.661], g[1.349]

2/2 [==============================] - 0s 36ms/step

>157, c[0.208,96], d[0.696,0.608], g[1.100]

2/2 [==============================] - 0s 35ms/step

>158, c[0.202,98], d[0.633,0.844], g[1.151]

2/2 [==============================] - 0s 37ms/step

>159, c[0.231,96], d[0.620,0.743], g[1.192]

2/2 [==============================] - 0s 37ms/step

>160, c[0.207,96], d[0.622,0.767], g[1.054]

2/2 [==============================] - 0s 36ms/step

>161, c[0.228,98], d[0.647,0.927], g[0.948]

2/2 [==============================] - 0s 31ms/step

>162, c[0.159,100], d[0.784,0.914], g[0.833]

2/2 [==============================] - 0s 40ms/step
```

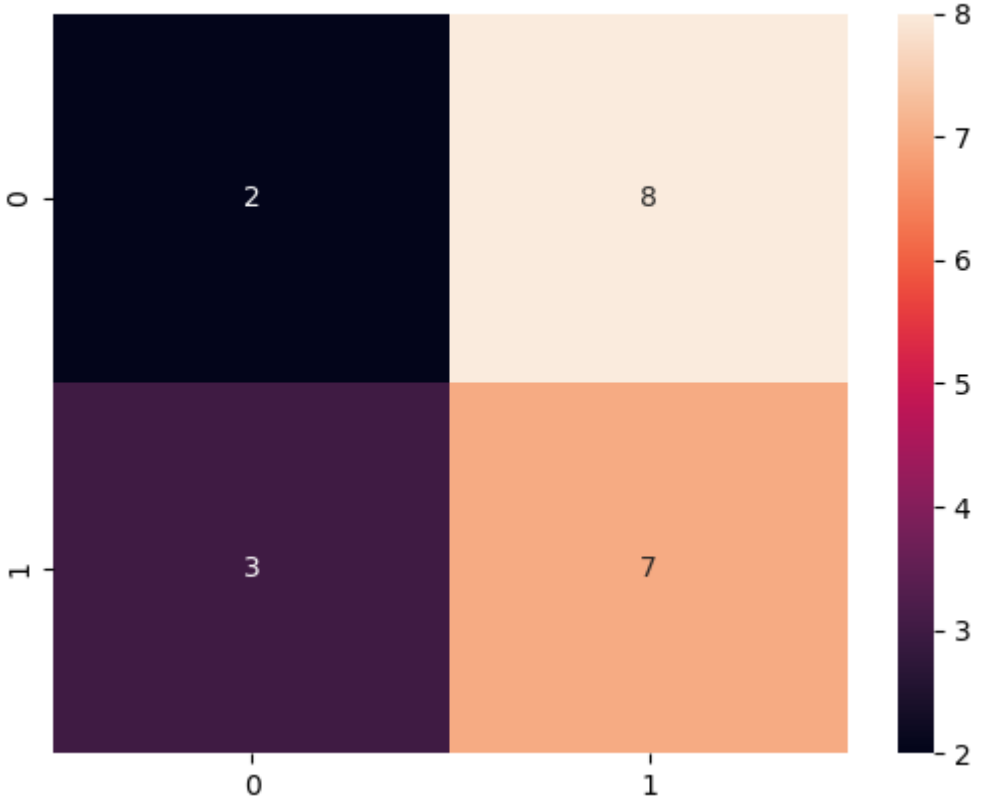
```
>163, c[0.183,98], d[0.919,1.118], g[0.841]
2/2 [==============================] - 0s 34ms/step
>164, c[0.167,100], d[0.839,1.161], g[0.827]
2/2 [==============================] - 0s 34ms/step
>165, c[0.181,100], d[0.888,1.069], g[0.819]
2/2 [==============================] - 0s 31ms/step
>166, c[0.189,96], d[1.019,0.960], g[0.854]
2/2 [==============================] - 0s 36ms/step
>167, c[0.180,100], d[1.098,0.873], g[0.844]
2/2 [==============================] - 0s 33ms/step
>168, c[0.210,98], d[0.832,0.892], g[0.966]
2/2 [==============================] - 0s 32ms/step
>169, c[0.191,96], d[0.706,0.605], g[1.042]
2/2 [==============================] - 0s 30ms/step
>170, c[0.209,92], d[0.780,0.672], g[1.012]
4/4 [==============================] - 0s 38ms/step
Discriminator Accuracy: 53.283%
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
>Saved: generated_plot_0170.png, gen_model_0170.h5, and disc_sup_0170.h5
```

```
##################################################################################
#EVALUATE THE SUPERVISED DISCRIMINATOR ON TEST DATA
# This is the model we want as a classifier.
##################################################################################
from keras.models import load_model
# load the model
#disc_sup_trained_model = load_model('disc_sup_6000.h5')
disc_sup_trained_model=disc_sup
# load the dataset
#(_, _), (testX, testy) = load_data()

# expand to 3d, e.g. add channels
testX = expand_dims(testX, axis=-1)

# convert from ints to floats
testX = testX.astype('float32')

# scale from [0,255] to [-1,1]
testX = (testX - 127.5) / 127.5
```

```
# evaluate the model
_, test_acc = disc_sup_trained_model.evaluate(testX, testy, verbose=0)
print('Test Accuracy: %.3f%%' % (test_acc * 100))
```

Test Accuracy: 45.000%

```
# Predicting the Test set results
y_pred_test = disc_sup_trained_model.predict(testX)
prediction_test = np.argmax(y_pred_test, axis=1)

# Confusion Matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(testy, prediction_test)
sns.heatmap(cm, annot=True)
```



Autoencoder for anomaly detection

-*- coding: utf-8 -*-

"""

Created on Sun Aug 28 17:52:37 2022

@author: LAXMINARAYANRO

"""

```
import pandas as pd
df = pd.read_csv('E:/fffiiles/college pracs and projects/Project/Car/CSV/Only_Global_NCAPhh.csv')


print(df.groupby('Bodyshell_Integrity')['Bodyshell_Integrity'].count())
```

```
Bodyshell_Integrity
Stable      19
Unstable    36
```

```
df.drop(['Year'], axis=1, inplace=True)
df.drop(['Name'], axis=1, inplace=True)
df.drop(['Body_Type'], axis=1, inplace=True)
df.drop(['Airbags'], axis=1, inplace=True)
df.drop(['Brand'], axis=1, inplace=True)
df.drop(['Testing_Speed (km/hr)'], axis=1, inplace=True)
print(df.head())
```

```
Name: Bodyshell_Integrity, dtype: int64
   Adult_Occupant_(Stars)  ...  Bodyshell_Integrity
0                       4  ...               Stable
1                       0  ...             Unstable
```

```
2                 0  ...          Unstable
3                 0  ...            Stable
4                 0  ...            Stable

[5 rows x 8 columns]
```

#If there are missing entries, drop them.

df.dropna(inplace=True,axis=1)

#COnvert non-numeric to numeric

df.Bodyshell_Integrity[df.Bodyshell_Integrity == 'Stable'] = 1

df.Bodyshell_Integrity[df.Bodyshell_Integrity == 'Unstable'] = 2

```
C:\Users\LAXMIN~1\AppData\Local\Temp/ipykernel_11864/2547431975.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.Bodyshell_Integrity[df.Bodyshell_Integrity == 'Stable'] = 1
```

C:\Users\LAXMIN~1\AppData\Local\Temp/ipykernel_11864/2547431975.py:6: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
df.Bodyshell_Integrity[df.Bodyshell_Integrity == 'Unstable'] = 2
```

```
Stable_mask = df['Bodyshell_Integrity']== 1 #All good to be True for good data points

Unstable_mask = df['Bodyshell_Integrity']== 2 #All values False for good data points


df.drop('Bodyshell_Integrity',axis=1,inplace=True)


df_stable = df[Stable_mask]

df_unstable = df[Unstable_mask]


print(f"Stable count: {len(df_stable)}")

print(f"Unstable count: {len(df_unstable)}")
```

```
Stable count: 19
Unstable count: 36
# This is the feature vector that goes to the neural net
x_stable = df_stable.values
x_unstable = df_unstable.values

from sklearn.model_selection import train_test_split

x_stable_train, x_stable_test = train_test_split(
    x_stable, test_size=0.25, random_state=42)

print(f"Stable train count: {len(x_stable_train)}")
print(f"Stable test count: {len(x_stable_test)}")
Stable train count: 14
Stable test count: 5
##########
#Define the autoencoder model
#Since we're dealing with numeric values we can use only
Dense layers.

from sklearn import metrics
import numpy as np
```

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(10, input_dim=x_stable.shape[1],
activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(x_stable.shape[1]))
model.compile(loss='mean_squared_error',
optimizer='adam')
model.summary()

model.fit(x_stable_train,x_stable_train,verbose=1,epochs=10
0)
```

```
Output from spyder call 'get_namespace_view':
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense (Dense)              (None, 10)              80

 dense_1 (Dense)            (None, 3)               33

 dense_2 (Dense)            (None, 10)              40

 dense_3 (Dense)            (None, 7)               77

=================================================================
Total params: 230
Trainable params: 230
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x_stable_train,x_stable_train,verbose=1,epochs=100)
Epoch 1/100
1/1 [==============================] - 10s 10s/step - loss: 353190.7500
Epoch 2/100
```

```
1/1 [==============================] - 0s 35ms/step - loss: 351423.0312

Epoch 3/100

1/1 [==============================] - 0s 8ms/step - loss: 349689.5000

Epoch 4/100

1/1 [==============================] - 0s 9ms/step - loss: 347990.5625

Epoch 5/100

1/1 [==============================] - 0s 5ms/step - loss: 346326.4375

Epoch 6/100

1/1 [==============================] - 0s 6ms/step - loss: 344697.2188

Epoch 7/100

1/1 [==============================] - 0s 6ms/step - loss: 343103.0000

Epoch 8/100

1/1 [==============================] - 0s 7ms/step - loss: 341543.7812

Epoch 9/100

1/1 [==============================] - 0s 5ms/step - loss: 340019.4688

Epoch 10/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 338529.9688

Epoch 11/100

1/1 [==============================] - 0s 5ms/step - loss: 337075.8125

Epoch 12/100

1/1 [==============================] - 0s 7ms/step - loss: 335656.0625

Epoch 13/100

1/1 [==============================] - 0s 5ms/step - loss: 334270.3125

Epoch 14/100

1/1 [==============================] - 0s 6ms/step - loss: 332917.6875

Epoch 15/100

1/1 [==============================] - 0s 6ms/step - loss: 331597.8125

Epoch 16/100

1/1 [==============================] - 0s 7ms/step - loss: 330310.2812

Epoch 17/100

1/1 [==============================] - 0s 7ms/step - loss: 329055.0938

Epoch 18/100
```

```
1/1 [==============================] - 0s 9ms/step - loss: 327831.2812

Epoch 19/100

1/1 [==============================] - 0s 8ms/step - loss: 326638.0625

Epoch 20/100

1/1 [==============================] - 0s 9ms/step - loss: 325475.2500

Epoch 21/100

1/1 [==============================] - 0s 8ms/step - loss: 324341.5625

Epoch 22/100

1/1 [==============================] - 0s 7ms/step - loss: 323236.3438

Epoch 23/100

1/1 [==============================] - 0s 8ms/step - loss: 322158.8125

Epoch 24/100

1/1 [==============================] - 0s 8ms/step - loss: 321108.0938

Epoch 25/100

1/1 [==============================] - 0s 7ms/step - loss: 320083.8125

Epoch 26/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 319084.7500

Epoch 27/100

1/1 [==============================] - 0s 9ms/step - loss: 318109.9062

Epoch 28/100

1/1 [==============================] - 0s 10ms/step - loss: 317158.5625

Epoch 29/100

1/1 [==============================] - 0s 14ms/step - loss: 316229.9375

Epoch 30/100

1/1 [==============================] - 0s 8ms/step - loss: 315323.4375

Epoch 31/100

1/1 [==============================] - 0s 8ms/step - loss: 314437.9375

Epoch 32/100

1/1 [==============================] - 0s 7ms/step - loss: 313572.6562

Epoch 33/100

1/1 [==============================] - 0s 7ms/step - loss: 312726.7188

Epoch 34/100
```

```
1/1 [==============================] - 0s 76ms/step - loss: 311899.5312

Epoch 35/100

1/1 [==============================] - 0s 7ms/step - loss: 311090.0625

Epoch 36/100

1/1 [==============================] - 0s 4ms/step - loss: 310386.3438

Epoch 37/100

1/1 [==============================] - 0s 6ms/step - loss: 310007.1562

Epoch 38/100

1/1 [==============================] - 0s 5ms/step - loss: 309649.4375

Epoch 39/100

1/1 [==============================] - 0s 6ms/step - loss: 309293.4375

Epoch 40/100

1/1 [==============================] - 0s 5ms/step - loss: 308938.8125

Epoch 41/100

1/1 [==============================] - 0s 6ms/step - loss: 308585.3125

Epoch 42/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 308232.7812

Epoch 43/100

1/1 [==============================] - 0s 7ms/step - loss: 307881.0625

Epoch 44/100

1/1 [==============================] - 0s 66ms/step - loss: 307530.0625

Epoch 45/100

1/1 [==============================] - 0s 5ms/step - loss: 307179.7188

Epoch 46/100

1/1 [==============================] - 0s 6ms/step - loss: 306830.0000

Epoch 47/100

1/1 [==============================] - 0s 6ms/step - loss: 306480.7812

Epoch 48/100

1/1 [==============================] - 0s 5ms/step - loss: 306132.1562

Epoch 49/100

1/1 [==============================] - 0s 7ms/step - loss: 305784.0312

Epoch 50/100
```

```
1/1 [==============================] - 0s 7ms/step - loss: 305436.4062

Epoch 51/100

1/1 [==============================] - 0s 7ms/step - loss: 305089.2188

Epoch 52/100

1/1 [==============================] - 0s 6ms/step - loss: 304742.4688

Epoch 53/100

1/1 [==============================] - 0s 6ms/step - loss: 304396.0625

Epoch 54/100

1/1 [==============================] - 0s 6ms/step - loss: 304050.0625

Epoch 55/100

1/1 [==============================] - 0s 5ms/step - loss: 303704.2500

Epoch 56/100

1/1 [==============================] - 0s 5ms/step - loss: 303358.7188

Epoch 57/100

1/1 [==============================] - 0s 7ms/step - loss: 303013.2812

Epoch 58/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 302667.9375

Epoch 59/100

1/1 [==============================] - 0s 7ms/step - loss: 302322.5000

Epoch 60/100

1/1 [==============================] - 0s 5ms/step - loss: 301976.9062

Epoch 61/100

1/1 [==============================] - 0s 7ms/step - loss: 301631.0312

Epoch 62/100

1/1 [==============================] - 0s 8ms/step - loss: 301284.8125

Epoch 63/100

1/1 [==============================] - 0s 5ms/step - loss: 300938.0625

Epoch 64/100

1/1 [==============================] - 0s 7ms/step - loss: 300590.7188

Epoch 65/100

1/1 [==============================] - 0s 6ms/step - loss: 300242.5312

Epoch 66/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 299893.5000

Epoch 67/100

1/1 [==============================] - 0s 6ms/step - loss: 299543.3750

Epoch 68/100

1/1 [==============================] - 0s 7ms/step - loss: 299192.0312

Epoch 69/100

1/1 [==============================] - 0s 6ms/step - loss: 298839.3438

Epoch 70/100

1/1 [==============================] - 0s 5ms/step - loss: 298485.1562

Epoch 71/100

1/1 [==============================] - 0s 5ms/step - loss: 298129.3438

Epoch 72/100

1/1 [==============================] - 0s 5ms/step - loss: 297771.7188

Epoch 73/100

1/1 [==============================] - 0s 6ms/step - loss: 297412.0938

Epoch 74/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 297050.3438

Epoch 75/100

1/1 [==============================] - 0s 7ms/step - loss: 296686.3125

Epoch 76/100

1/1 [==============================] - 0s 6ms/step - loss: 296319.8438

Epoch 77/100

1/1 [==============================] - 0s 7ms/step - loss: 295950.7812

Epoch 78/100

1/1 [==============================] - 0s 6ms/step - loss: 295579.0000

Epoch 79/100

1/1 [==============================] - 0s 6ms/step - loss: 295204.3125

Epoch 80/100

1/1 [==============================] - 0s 6ms/step - loss: 294826.5625

Epoch 81/100

1/1 [==============================] - 0s 6ms/step - loss: 294445.5938

Epoch 82/100
```

```
1/1 [==============================] - 0s 7ms/step - loss: 294061.2812

Epoch 83/100

1/1 [==============================] - 0s 5ms/step - loss: 293673.4375

Epoch 84/100

1/1 [==============================] - 0s 7ms/step - loss: 293281.9375

Epoch 85/100

1/1 [==============================] - 0s 6ms/step - loss: 292886.6562

Epoch 86/100

1/1 [==============================] - 0s 6ms/step - loss: 292487.4062

Epoch 87/100

1/1 [==============================] - 0s 6ms/step - loss: 292084.0625

Epoch 88/100

1/1 [==============================] - 0s 5ms/step - loss: 291676.4375

Epoch 89/100

1/1 [==============================] - 0s 8ms/step - loss: 291264.4375

Epoch 90/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 290847.9062

Epoch 91/100

1/1 [==============================] - 0s 7ms/step - loss: 290426.6875

Epoch 92/100

1/1 [==============================] - 0s 6ms/step - loss: 290000.6562

Epoch 93/100

1/1 [==============================] - 0s 7ms/step - loss: 289569.6250

Epoch 94/100

1/1 [==============================] - 0s 6ms/step - loss: 289133.4688

Epoch 95/100

1/1 [==============================] - 0s 7ms/step - loss: 288692.0625

Epoch 96/100

1/1 [==============================] - 0s 6ms/step - loss: 288245.2812

Epoch 97/100

1/1 [==============================] - 0s 6ms/step - loss: 287792.9375

Epoch 98/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 287334.9375
Epoch 99/100
1/1 [==============================] - 0s 5ms/step - loss: 286871.0625
Epoch 100/100
1/1 [==============================] - 0s 5ms/step - loss: 286401.2188
```

```
pred = model.predict(x_stable_test)
score1 = np.sqrt(metrics.mean_squared_error(pred,x_stable_test))

pred = model.predict(x_stable)
score2 = np.sqrt(metrics.mean_squared_error(pred,x_stable))

pred = model.predict(x_unstable)
score3 = np.sqrt(metrics.mean_squared_error(pred,x_unstable))

#Reconstruction error
print(f"Insample Stable Score (RMSE): {score1}".format(score1))
print(f"Out of Sample Stable Score (RMSE): {score2}")
```

```
print(f"Unstable sample Score (RMSE): {score3}")
```

Insample Stable Score (RMSE): 488.558667003364

Out of Sample Stable Score (RMSE): 522.9673236916708

Unstable sample Score (RMSE): 442.96239795904023

AutoEncoder for image with CNN

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 29 20:48:34 2022

@author: LAXMINARAYANRO
"""

from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.models import Sequential

import numpy as np
import matplotlib.pyplot as plt

import os
```

```
import cv2
from tensorflow.keras.utils import img_to_array

# x is noisy data and y is clean data
SIZE = 256

from tqdm import tqdm
noisy_data=[]
path1 = 'E:/fffiiles/college pracs and
projects/Project/Car/Createtry/Noisy/'
files=os.listdir(path1)
for i in tqdm(files):
    img=cv2.imread(path1+'/'+i,cv2.IMREAD_COLOR)
    #img=cv2.imread(path1+'/'+i,0)
      #Change 0 to 1 for color images
    img=cv2.resize(img,(SIZE, SIZE))
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    noisy_data.append(img_to_array(img))


clean_data=[]
```

```
path2 = 'E:/fffiiles/college pracs and
projects/Project/Car/Createtry/Clean/'

files=os.listdir(path2)

for i in tqdm(files):

    # img=cv2.imread(path2+'/'+i,0)  #Change 0 to 1 for color
images

    # img=cv2.resize(img,(SIZE, SIZE))

    img=cv2.imread(path2+'/'+i,cv2.IMREAD_COLOR)#Change
0 to 1 for color images

    img=cv2.resize(img,(SIZE, SIZE))

    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

    clean_data.append(img_to_array(img))


noisy_train = np.reshape(noisy_data, (len(noisy_data), SIZE,
SIZE, 3))#3

noisy_train = noisy_train.astype('float32') / 255.


clean_train = np.reshape(clean_data, (len(clean_data), SIZE,
SIZE, 3))

clean_train = clean_train.astype('float32') / 255.



#Displaying images with noise
```

```
plt.figure(figsize=(10, 2))
for i in range(1,4):
    ax = plt.subplot(1, 4, i)
    plt.imshow(noisy_train[i].reshape(SIZE, SIZE,3),
cmap="binary")#SIZE, SIZE,3
plt.show()

#Displaying clean images
plt.figure(figsize=(10, 2))
for i in range(1,4):
    ax = plt.subplot(1, 4, i)
    plt.imshow(clean_train[i].reshape(SIZE, SIZE,3),
cmap="binary")
plt.show()



model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',
padding='same', input_shape=(SIZE, SIZE, 3)))#3
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))
```

```
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))


model.add(MaxPooling2D((2, 2), padding='same'))

model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(1, (3, 3), activation='relu',
padding='same'))

model.compile(optimizer='adam',
loss='mean_squared_error', metrics=['accuracy'])

model.summary()
```

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(noisy_train, clean_train,
                                                    test_size = 0.20, random_state = 0)



model.fit(x_train, y_train, epochs=100, batch_size=8, shuffle=True, verbose = 1,
          validation_split = 0.1)
```

```
Epoch 1/100

14/14 [==============================] - 15s 1s/step - loss: 0.0993 - accuracy: 0.0612 - val_loss: 0.1027 - val_accuracy: 0.0792

Epoch 2/100

14/14 [==============================] - 14s 986ms/step - loss: 0.1016 - accuracy: 0.0549 - val_loss: 0.1024 - val_accuracy: 0.0340

Epoch 3/100

14/14 [==============================] - 13s 962ms/step - loss: 0.0999 - accuracy: 0.0604 - val_loss: 0.1015 - val_accuracy: 0.0477

Epoch 4/100
```

14/14 [==========] - 13s 953ms/step - loss: 0.0982 - accuracy: 0.0565 - val_loss: 0.1019 - val_accuracy: 0.0838

Epoch 5/100

14/14 [==========] - 14s 1s/step - loss: 0.1012 - accuracy: 0.0477 - val_loss: 0.1136 - val_accuracy: 0.1087

Epoch 6/100

14/14 [==========] - 13s 920ms/step - loss: 0.1013 - accuracy: 0.0616 - val_loss: 0.1020 - val_accuracy: 0.0294

Epoch 7/100

14/14 [==========] - 13s 936ms/step - loss: 0.1000 - accuracy: 0.0586 - val_loss: 0.1017 - val_accuracy: 0.0596

Epoch 8/100

14/14 [==========] - 14s 977ms/step - loss: 0.0987 - accuracy: 0.0508 - val_loss: 0.1015 - val_accuracy: 0.0914

Epoch 9/100

14/14 [==========] - 14s 975ms/step - loss: 0.0976 - accuracy: 0.0691 - val_loss: 0.1007 - val_accuracy: 0.0755

Epoch 10/100

```
14/14 [==============================] - 14s 992ms/step - loss: 0.0973 - accuracy: 0.0709 - val_loss: 0.1011 - val_accuracy: 0.0751

Epoch 11/100

14/14 [==============================] - 13s 928ms/step - loss: 0.0968 - accuracy: 0.0681 - val_loss: 0.1005 - val_accuracy: 0.0667

Epoch 12/100

14/14 [==============================] - 13s 929ms/step - loss: 0.0969 - accuracy: 0.0665 - val_loss: 0.1007 - val_accuracy: 0.0735

Epoch 13/100

14/14 [==============================] - 13s 940ms/step - loss: 0.0978 - accuracy: 0.0674 - val_loss: 0.1004 - val_accuracy: 0.0685

Epoch 14/100

14/14 [==============================] - 14s 1s/step - loss: 0.0979 - accuracy: 0.0681 - val_loss: 0.1035 - val_accuracy: 0.0356

Epoch 15/100

14/14 [==============================] - 13s 946ms/step - loss: 0.0996 - accuracy: 0.0616 - val_loss: 0.1005 - val_accuracy: 0.0872

Epoch 16/100
```

```
14/14 [==============================] - 13s
948ms/step - loss: 0.0973 - accuracy: 0.0612 - val_loss:
0.1024 - val_accuracy: 0.0896

Epoch 17/100

14/14 [==============================] - 13s
944ms/step - loss: 0.0978 - accuracy: 0.0680 - val_loss:
0.1010 - val_accuracy: 0.0841

Epoch 18/100

14/14 [==============================] - 13s
956ms/step - loss: 0.0964 - accuracy: 0.0821 - val_loss:
0.1022 - val_accuracy: 0.0586

Epoch 19/100

14/14 [==============================] - 14s
979ms/step - loss: 0.0967 - accuracy: 0.0719 - val_loss:
0.0998 - val_accuracy: 0.0716

Epoch 20/100

14/14 [==============================] - 13s
950ms/step - loss: 0.0960 - accuracy: 0.0715 - val_loss:
0.1005 - val_accuracy: 0.0724

Epoch 21/100

14/14 [==============================] - 13s
943ms/step - loss: 0.0967 - accuracy: 0.0696 - val_loss:
0.1024 - val_accuracy: 0.0408

Epoch 22/100
```

```
14/14 [==============================] - 13s 940ms/step - loss: 0.0963 - accuracy: 0.0653 - val_loss: 0.1012 - val_accuracy: 0.0837

Epoch 23/100

14/14 [==============================] - 14s 1s/step - loss: 0.0953 - accuracy: 0.0819 - val_loss: 0.1022 - val_accuracy: 0.0613

Epoch 24/100

14/14 [==============================] - 13s 936ms/step - loss: 0.0966 - accuracy: 0.0718 - val_loss: 0.0999 - val_accuracy: 0.0793

Epoch 25/100

14/14 [==============================] - 13s 930ms/step - loss: 0.0964 - accuracy: 0.0772 - val_loss: 0.1010 - val_accuracy: 0.0700

Epoch 26/100

14/14 [==============================] - 13s 954ms/step - loss: 0.0953 - accuracy: 0.0675 - val_loss: 0.1006 - val_accuracy: 0.0929

Epoch 27/100

14/14 [==============================] - 14s 986ms/step - loss: 0.0961 - accuracy: 0.0808 - val_loss: 0.1016 - val_accuracy: 0.0604

Epoch 28/100
```

```
14/14 [==============================] - 14s 963ms/step - loss: 0.0975 - accuracy: 0.0720 - val_loss: 0.1023 - val_accuracy: 0.0433

Epoch 29/100

14/14 [==============================] - 13s 944ms/step - loss: 0.0984 - accuracy: 0.0569 - val_loss: 0.1036 - val_accuracy: 0.0987

Epoch 30/100

14/14 [==============================] - 13s 950ms/step - loss: 0.0981 - accuracy: 0.0602 - val_loss: 0.1009 - val_accuracy: 0.0878

Epoch 31/100

14/14 [==============================] - 13s 928ms/step - loss: 0.0999 - accuracy: 0.0534 - val_loss: 0.0993 - val_accuracy: 0.0777

Epoch 32/100

14/14 [==============================] - 14s 1s/step - loss: 0.0958 - accuracy: 0.0814 - val_loss: 0.1009 - val_accuracy: 0.0695

Epoch 33/100

14/14 [==============================] - 13s 938ms/step - loss: 0.0957 - accuracy: 0.0767 - val_loss: 0.1000 - val_accuracy: 0.0772

Epoch 34/100
```

14/14 [==========] - 13s 937ms/step - loss: 0.0963 - accuracy: 0.0570 - val_loss: 0.1017 - val_accuracy: 0.0973

Epoch 35/100

14/14 [==========] - 14s 998ms/step - loss: 0.0957 - accuracy: 0.0797 - val_loss: 0.1004 - val_accuracy: 0.0784

Epoch 36/100

14/14 [==========] - 14s 999ms/step - loss: 0.0948 - accuracy: 0.0822 - val_loss: 0.0999 - val_accuracy: 0.0805

Epoch 37/100

14/14 [==========] - 15s 1s/step - loss: 0.0947 - accuracy: 0.0725 - val_loss: 0.1011 - val_accuracy: 0.0901

Epoch 38/100

14/14 [==========] - 14s 1s/step - loss: 0.0962 - accuracy: 0.0796 - val_loss: 0.1005 - val_accuracy: 0.0739

Epoch 39/100

14/14 [==========] - 14s 995ms/step - loss: 0.0942 - accuracy: 0.0789 - val_loss: 0.1008 - val_accuracy: 0.0868

Epoch 40/100

```
14/14 [==============================] - 14s 1s/step - loss: 0.0961 - accuracy: 0.0695 - val_loss: 0.1002 - val_accuracy: 0.0910

Epoch 41/100

14/14 [==============================] - 14s 1s/step - loss: 0.0947 - accuracy: 0.0774 - val_loss: 0.1003 - val_accuracy: 0.0855

Epoch 42/100

14/14 [==============================] - 14s 971ms/step - loss: 0.0956 - accuracy: 0.0776 - val_loss: 0.0992 - val_accuracy: 0.0760

Epoch 43/100

14/14 [==============================] - 14s 975ms/step - loss: 0.0941 - accuracy: 0.0806 - val_loss: 0.1020 - val_accuracy: 0.0884

Epoch 44/100

14/14 [==============================] - 13s 963ms/step - loss: 0.0951 - accuracy: 0.0727 - val_loss: 0.1007 - val_accuracy: 0.0975

Epoch 45/100

14/14 [==============================] - 14s 1s/step - loss: 0.0983 - accuracy: 0.0708 - val_loss: 0.1020 - val_accuracy: 0.0572

Epoch 46/100
```

```
14/14 [==============================] - 14s 976ms/step - loss: 0.0984 - accuracy: 0.0753 - val_loss: 0.1011 - val_accuracy: 0.0453

Epoch 47/100

14/14 [==============================] - 14s 971ms/step - loss: 0.0980 - accuracy: 0.0615 - val_loss: 0.1006 - val_accuracy: 0.0799

Epoch 48/100

14/14 [==============================] - 14s 970ms/step - loss: 0.0949 - accuracy: 0.0761 - val_loss: 0.1004 - val_accuracy: 0.0920

Epoch 49/100

14/14 [==============================] - 15s 1s/step - loss: 0.0951 - accuracy: 0.0681 - val_loss: 0.0996 - val_accuracy: 0.0873

Epoch 50/100

14/14 [==============================] - 15s 1s/step - loss: 0.0976 - accuracy: 0.0776 - val_loss: 0.1008 - val_accuracy: 0.0696

Epoch 51/100

14/14 [==============================] - 14s 1s/step - loss: 0.0936 - accuracy: 0.0775 - val_loss: 0.1011 - val_accuracy: 0.0967

Epoch 52/100
```

14/14 [==========] - 14s 1s/step - loss: 0.0954 - accuracy: 0.0817 - val_loss: 0.1028 - val_accuracy: 0.0577

Epoch 53/100

14/14 [==========] - 16s 1s/step - loss: 0.0951 - accuracy: 0.0639 - val_loss: 0.1035 - val_accuracy: 0.1040

Epoch 54/100

14/14 [==========] - 15s 1s/step - loss: 0.0956 - accuracy: 0.0815 - val_loss: 0.1014 - val_accuracy: 0.0721

Epoch 55/100

14/14 [==========] - 15s 1s/step - loss: 0.0938 - accuracy: 0.0783 - val_loss: 0.1008 - val_accuracy: 0.0879

Epoch 56/100

14/14 [==========] - 16s 1s/step - loss: 0.0938 - accuracy: 0.0783 - val_loss: 0.1001 - val_accuracy: 0.0847

Epoch 57/100

14/14 [==========] - 16s 1s/step - loss: 0.0943 - accuracy: 0.0780 - val_loss: 0.1009 - val_accuracy: 0.0708

Epoch 58/100

14/14 [==========] - 15s 1s/step - loss: 0.0951 - accuracy: 0.0815 - val_loss: 0.1007 - val_accuracy: 0.0648

Epoch 59/100

14/14 [==========] - 15s 1s/step - loss: 0.0951 - accuracy: 0.0697 - val_loss: 0.1010 - val_accuracy: 0.0908

Epoch 60/100

14/14 [==========] - 15s 1s/step - loss: 0.0945 - accuracy: 0.0803 - val_loss: 0.1004 - val_accuracy: 0.0765

Epoch 61/100

14/14 [==========] - 14s 1s/step - loss: 0.0946 - accuracy: 0.0769 - val_loss: 0.1009 - val_accuracy: 0.0694

Epoch 62/100

14/14 [==========] - 13s 955ms/step - loss: 0.0940 - accuracy: 0.0743 - val_loss: 0.1036 - val_accuracy: 0.0976

Epoch 63/100

14/14 [==========] - 13s 949ms/step - loss: 0.0934 - accuracy: 0.0780 - val_loss: 0.0989 - val_accuracy: 0.0773

Epoch 64/100

14/14 [==========] - 14s 983ms/step - loss: 0.0947 - accuracy: 0.0817 - val_loss: 0.1015 - val_accuracy: 0.0738

Epoch 65/100

14/14 [==========] - 14s 1s/step - loss: 0.0947 - accuracy: 0.0751 - val_loss: 0.1001 - val_accuracy: 0.0803

Epoch 66/100

14/14 [==========] - 14s 1s/step - loss: 0.0936 - accuracy: 0.0794 - val_loss: 0.0999 - val_accuracy: 0.0793

Epoch 67/100

14/14 [==========] - 14s 974ms/step - loss: 0.0935 - accuracy: 0.0785 - val_loss: 0.1014 - val_accuracy: 0.0942

Epoch 68/100

14/14 [==========] - 14s 979ms/step - loss: 0.0935 - accuracy: 0.0799 - val_loss: 0.0993 - val_accuracy: 0.0877

Epoch 69/100

14/14 [==========] - 14s 994ms/step - loss: 0.0932 - accuracy: 0.0787 - val_loss: 0.0997 - val_accuracy: 0.0872

Epoch 70/100

```
14/14 [==============================] - 15s 1s/step - loss: 0.0940 - accuracy: 0.0782 - val_loss: 0.1000 - val_accuracy: 0.0930

Epoch 71/100

14/14 [==============================] - 13s 962ms/step - loss: 0.0930 - accuracy: 0.0825 - val_loss: 0.1006 - val_accuracy: 0.0840

Epoch 72/100

14/14 [==============================] - 14s 1s/step - loss: 0.0930 - accuracy: 0.0803 - val_loss: 0.1010 - val_accuracy: 0.0766

Epoch 73/100

14/14 [==============================] - 14s 1s/step - loss: 0.0936 - accuracy: 0.0803 - val_loss: 0.1006 - val_accuracy: 0.0702

Epoch 74/100

14/14 [==============================] - 14s 1s/step - loss: 0.0931 - accuracy: 0.0740 - val_loss: 0.1023 - val_accuracy: 0.0960

Epoch 75/100

14/14 [==============================] - 13s 939ms/step - loss: 0.0943 - accuracy: 0.0782 - val_loss: 0.0998 - val_accuracy: 0.0947

Epoch 76/100
```

14/14 [==========] - 14s 987ms/step - loss: 0.0951 - accuracy: 0.0724 - val_loss: 0.1037 - val_accuracy: 0.0958

Epoch 77/100

14/14 [==========] - 14s 990ms/step - loss: 0.0957 - accuracy: 0.0663 - val_loss: 0.1004 - val_accuracy: 0.0904

Epoch 78/100

14/14 [==========] - 14s 1s/step - loss: 0.0945 - accuracy: 0.0820 - val_loss: 0.0989 - val_accuracy: 0.0819

Epoch 79/100

14/14 [==========] - 15s 1s/step - loss: 0.0933 - accuracy: 0.0802 - val_loss: 0.1019 - val_accuracy: 0.0726

Epoch 80/100

14/14 [==========] - 17s 1s/step - loss: 0.0933 - accuracy: 0.0782 - val_loss: 0.0994 - val_accuracy: 0.0846

Epoch 81/100

14/14 [==========] - 14s 972ms/step - loss: 0.0927 - accuracy: 0.0818 - val_loss: 0.1002 - val_accuracy: 0.0883

Epoch 82/100

14/14 [==========] - 14s 1s/step - loss: 0.0932 - accuracy: 0.0794 - val_loss: 0.1002 - val_accuracy: 0.0917

Epoch 83/100

14/14 [==========] - 14s 996ms/step - loss: 0.0937 - accuracy: 0.0698 - val_loss: 0.1023 - val_accuracy: 0.1024

Epoch 84/100

14/14 [==========] - 14s 980ms/step - loss: 0.0951 - accuracy: 0.0791 - val_loss: 0.0999 - val_accuracy: 0.0826

Epoch 85/100

14/14 [==========] - 14s 980ms/step - loss: 0.0940 - accuracy: 0.0834 - val_loss: 0.1008 - val_accuracy: 0.0751

Epoch 86/100

14/14 [==========] - 14s 1s/step - loss: 0.0937 - accuracy: 0.0792 - val_loss: 0.1000 - val_accuracy: 0.0767

Epoch 87/100

14/14 [==========] - 16s 1s/step - loss: 0.0949 - accuracy: 0.0776 - val_loss: 0.1012 - val_accuracy: 0.0636

Epoch 88/100

```
14/14 [==============================] - 14s 994ms/step - loss: 0.0966 - accuracy: 0.0653 - val_loss: 0.1061 - val_accuracy: 0.0324

Epoch 89/100

14/14 [==============================] - 14s 981ms/step - loss: 0.0974 - accuracy: 0.0668 - val_loss: 0.1012 - val_accuracy: 0.0785

Epoch 90/100

14/14 [==============================] - 15s 1s/step - loss: 0.0953 - accuracy: 0.0729 - val_loss: 0.1005 - val_accuracy: 0.0909

Epoch 91/100

14/14 [==============================] - 17s 1s/step - loss: 0.0954 - accuracy: 0.0820 - val_loss: 0.1010 - val_accuracy: 0.0743

Epoch 92/100

14/14 [==============================] - 14s 1s/step - loss: 0.0948 - accuracy: 0.0781 - val_loss: 0.0996 - val_accuracy: 0.0872

Epoch 93/100

14/14 [==============================] - 14s 1s/step - loss: 0.0940 - accuracy: 0.0749 - val_loss: 0.0997 - val_accuracy: 0.0942

Epoch 94/100
```

```
14/14 [==============================] - 14s 982ms/step - loss: 0.0961 - accuracy: 0.0862 - val_loss: 0.1034 - val_accuracy: 0.0629

Epoch 95/100

14/14 [==============================] - 14s 1s/step - loss: 0.0957 - accuracy: 0.0737 - val_loss: 0.0992 - val_accuracy: 0.0866

Epoch 96/100

14/14 [==============================] - 13s 932ms/step - loss: 0.0946 - accuracy: 0.0775 - val_loss: 0.1002 - val_accuracy: 0.0859

Epoch 97/100

14/14 [==============================] - 14s 996ms/step - loss: 0.0930 - accuracy: 0.0835 - val_loss: 0.1000 - val_accuracy: 0.0800

Epoch 98/100

14/14 [==============================] - 13s 963ms/step - loss: 0.0926 - accuracy: 0.0826 - val_loss: 0.1010 - val_accuracy: 0.0782

Epoch 99/100

14/14 [==============================] - 14s 1s/step - loss: 0.0928 - accuracy: 0.0792 - val_loss: 0.1000 - val_accuracy: 0.0855

Epoch 100/100
```

```
14/14 [==============================] - 15s 1s/step - loss: 0.0927 - accuracy: 0.0877 - val_loss: 0.1046 - val_accuracy: 0.0649
```

```
print("Test_Accuracy: {:.2f}%".format(model.evaluate(np.array(x_test), np.array(y_test))[1]*100))
```

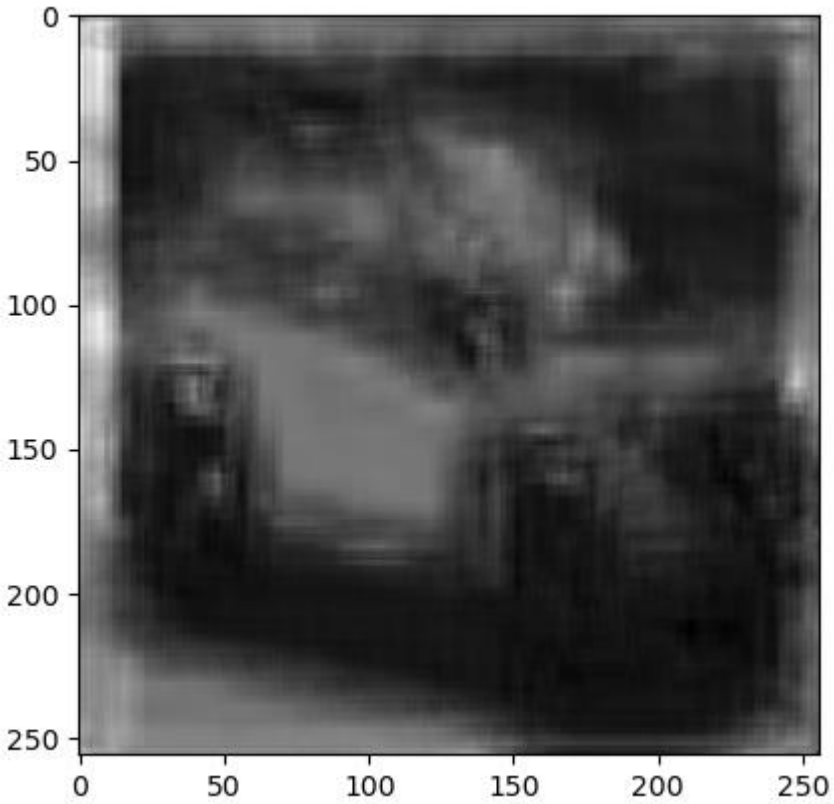
```
1/1 [==============================] - 1s 1s/step - loss: 0.0991 - accuracy: 0.0421
Test_Accuracy: 4.21%
```

Test_Accuracy: 6.09% after augmentation

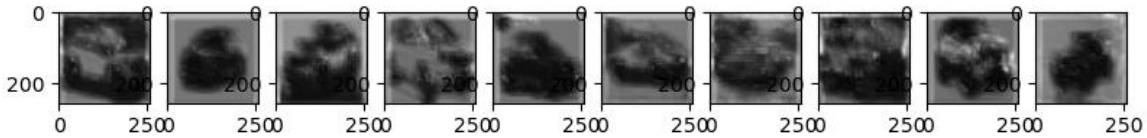
```
model.save('denoising_autoencoder.model')
```

```
no_noise_img = model.predict(x_test)
```

```
plt.imshow(no_noise_img[i].reshape(SIZE,SIZE),cmap="gray")
#plt.imsave('sandstone/denoised_images/denoised_image.tif', no_noise_img[3].reshape(SIZE,SIZE))
```




```
plt.figure(figsize=(40, 4))
for i in range(10):
    # display original
    ax = plt.subplot(3, 20, i + 1)
    plt.imshow(y_test[i].reshape(SIZE,SIZE,3), cmap="gray")

    # display reconstructed (after noise removed) image
    ax = plt.subplot(3, 20, 40 +i+ 1)
    plt.imshow(no_noise_img[i].reshape(SIZE,SIZE),
cmap="gray")
plt.show()
```



Autokeras Domain Adaptation

-*- coding: utf-8 -*-

"""

Created on Thu Sep 1 08:20:25 2022

@author: LAXMINARAYANRO

```
"""

from matplotlib.pyplot import imshow
import numpy as np
import cv2
from tensorflow.keras.utils import img_to_array
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.models import Sequential
import os

SIZE=256



from tqdm import tqdm
img_data=[]
path1 = 'E:/fffiiles/college pracs and projects/Project/Car/Createtry/AugNoisy/'
files=os.listdir(path1)
for i in tqdm(files):
    img=cv2.imread(path1+'/'+i,1)   #Change 0 to 1 for color images
    img=cv2.resize(img,(SIZE, SIZE))
```

```
    img_data.append(img_to_array(img))



img2_data=[]
path2 = 'E:/fffiiles/college pracs and
projects/Project/Car/Createtry/AugClean/'
files=os.listdir(path2)
for i in tqdm(files):
    img=cv2.imread(path2+'/'+i,1)  #Change 0 to 1 for color
images
    img=cv2.resize(img,(SIZE, SIZE))
    img2_data.append(img_to_array(img))


img_array = np.reshape(img_data, (len(img_data), SIZE, SIZE,
3))
img_array = img_array.astype('float32') / 255.


img_array2 = np.reshape(img2_data, (len(img2_data), SIZE,
SIZE, 3))
img_array2 = img_array2.astype('float32') / 255.
```

```
#Original einstein image for prediction as monalisa
img_data3=[]

img3=cv2.imread('E:/fffiiles/college pracs and projects/Project/Car/Createtry/Test.png', 1)   #Change 0 to 1 for color images
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)#Changing BGR to RGB to show images in true colors
img3=cv2.resize(img3,(SIZE, SIZE))
img_data3.append(img_to_array(img3))

img_array3 = np.reshape(img_data3, (len(img_data3), SIZE, SIZE, 3))
img_array3 = img_array3.astype('float32') / 255.

import time
start=time.time()



model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(SIZE, SIZE, 3)))
model.add(MaxPooling2D((2, 2), padding='same'))
```

```
model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))


model.add(MaxPooling2D((2, 2), padding='same'))

model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(3, (3, 3), activation='relu',
padding='same'))

model.compile(optimizer='adam',
loss='mean_squared_error', metrics=['accuracy'])
model.summary()
```

```
import tensorflow as tf

#callbacks =
[tf.keras.callbacks.TensorBoard(log_dir='einstein_logs')]



model.fit(img_array, img_array2,

        epochs=20,

        shuffle=True,

        )


finish=time.time()

print('total_time = ', finish-start)
```

```
Epoch 1/20


2022-09-01 22:32:47.891251: W
tensorflow/stream_executor/platform/default/dso_loader.cc
:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror:
cudnn64_8.dll not found

2022-09-01 22:32:48.084218: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:1850]
Cannot dlopen some GPU libraries. Please make sure the
missing libraries mentioned above are installed properly if
you would like to use GPU. Follow the guide at
```

https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-01 22:32:49.318940: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-01 22:34:32.803970: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:32:47.891251: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-01 22:32:48.084218: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-01 22:32:49.318940: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-01 22:34:32.803970: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:34.311471: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:32:47.891251: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-01 22:32:48.084218: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at

https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-01 22:32:49.318940: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-01 22:34:32.803970: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:34.311471: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:36.505401: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:32:47.891251: W tensorflow/stream_executor/platform/default/dso_loader.cc

:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-01 22:32:48.084218: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-01 22:32:49.318940: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-01 22:34:32.803970: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:34.311471: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:36.505401: W tensorflow/core/framework/cpu_allocator_impl.cc:82]

Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:39.498983: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

1/28 [>.....] - ETA: 6:12 - loss: 0.3023 - accuracy: 0.2973

2022-09-01 22:32:47.891251: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-01 22:32:48.084218: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-01 22:32:49.318940: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-09-01 22:34:32.803970: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:34.311471: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:36.505401: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:39.498983: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

2022-09-01 22:34:40.349576: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 268435456 exceeds 10% of free system memory.

28/28 [==============================] - 134s 4s/step - loss: 0.1660 - accuracy: 0.4040

Epoch 2/20

28/28 [==============================] - 142s 5s/step - loss: 0.1081 - accuracy: 0.3191

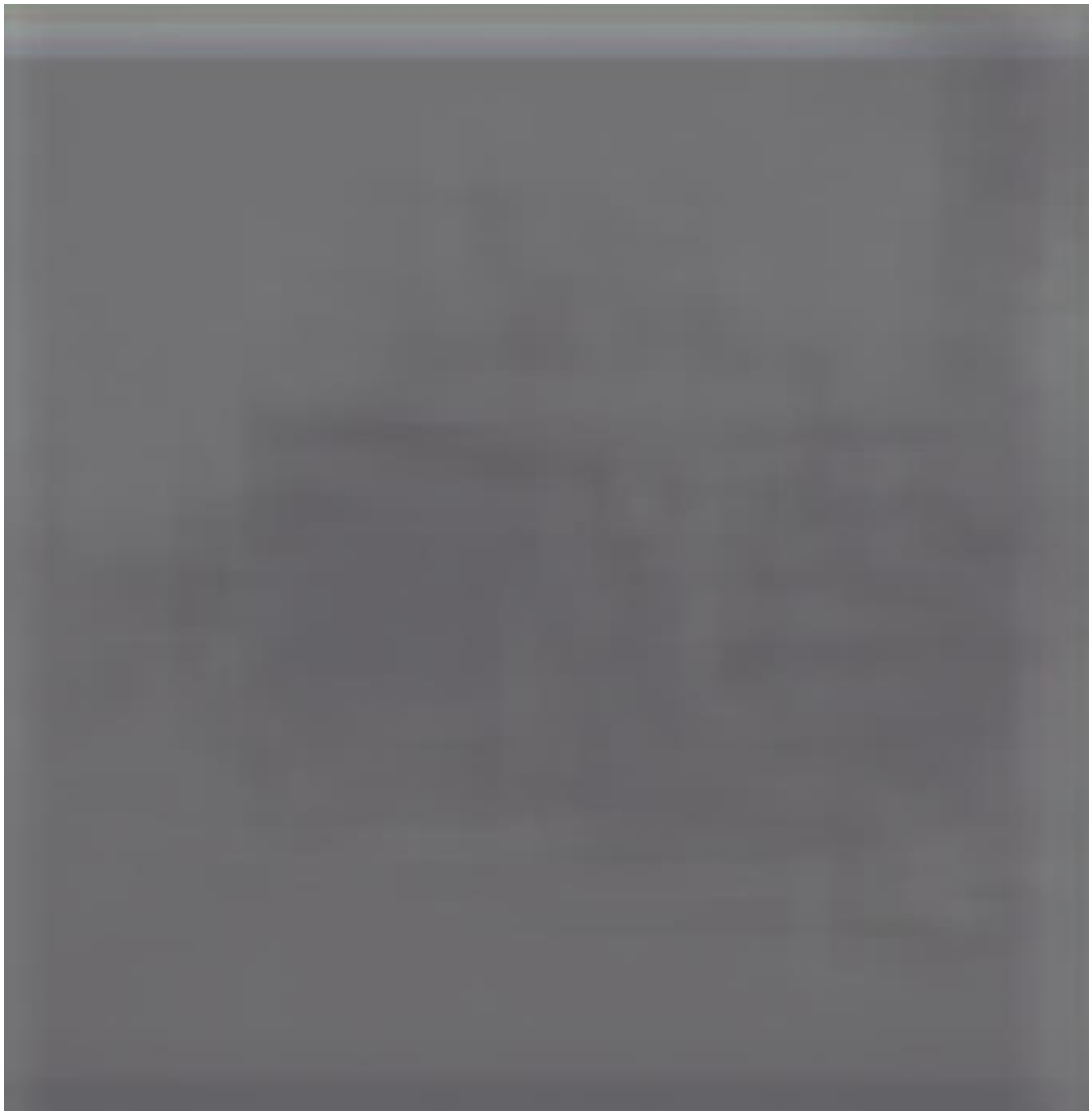
Epoch 3/20

```
28/28 [==============================] - 144s 5s/step - loss: 0.1036 - accuracy: 0.3158
Epoch 4/20
28/28 [==============================] - 148s 5s/step - loss: 0.1026 - accuracy: 0.3108
Epoch 5/20
28/28 [==============================] - 138s 5s/step - loss: 0.1024 - accuracy: 0.3056
Epoch 6/20
28/28 [==============================] - 141s 5s/step - loss: 0.1014 - accuracy: 0.3047
Epoch 7/20
28/28 [==============================] - 132s 5s/step - loss: 0.1011 - accuracy: 0.3053
Epoch 8/20
28/28 [==============================] - 137s 5s/step - loss: 0.1022 - accuracy: 0.3047
Epoch 9/20
28/28 [==============================] - 148s 5s/step - loss: 0.1011 - accuracy: 0.3077
Epoch 10/20
28/28 [==============================] - 151s 5s/step - loss: 0.1011 - accuracy: 0.3041
Epoch 11/20
```

```
28/28 [==============================] - 137s 5s/step - loss: 0.1011 - accuracy: 0.3091
Epoch 12/20
28/28 [==============================] - 134s 5s/step - loss: 0.1008 - accuracy: 0.3041
Epoch 13/20
28/28 [==============================] - 137s 5s/step - loss: 0.1006 - accuracy: 0.3161
Epoch 14/20
28/28 [==============================] - 140s 5s/step - loss: 0.1005 - accuracy: 0.2982
Epoch 15/20
28/28 [==============================] - 127s 4s/step - loss: 0.1004 - accuracy: 0.3063
Epoch 16/20
28/28 [==============================] - 123s 4s/step - loss: 0.1007 - accuracy: 0.3069
Epoch 17/20
28/28 [==============================] - 125s 4s/step - loss: 0.1010 - accuracy: 0.3066
Epoch 18/20
28/28 [==============================] - 123s 4s/step - loss: 0.1014 - accuracy: 0.3056
Epoch 19/20
```

```
28/28 [==============================] - 125s 4s/step - loss: 0.1003 - accuracy: 0.3025

Epoch 20/20

28/28 [==============================] - 122s 4s/step - loss: 0.1024 - accuracy: 0.3048

total_time =  3083.7756032943726
```

```
model.save('E:/fffiiles/college pracs and projects/Project/Car/Createtry/domain_adaptation.model')


print("Output")

pred = model.predict(img_array3)




imshow(pred[0].reshape(SIZE,SIZE,3))
```



Anomoly Detection

-*- coding: utf-8 -*-

"""

Created on Sun Aug 28 18:57:02 2022

```
@author: LAXMINARAYANRO
"""

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import random
# import os
# print(os.listdir("E:/fffiiles/college pracs and projects/Project/Car"))

#Size of our input images
SIZE = 256

##################################################################################
```

```
#Define generators for training, validation and also anomaly
data.

batch_size = 7
datagen = ImageDataGenerator(rescale=1./255)



train_generator = datagen.flow_from_directory(
    'E:/fffiiles/college pracs and
projects/Project/Car/Auto/Safe_Train/',
    target_size=(SIZE, SIZE),
    batch_size=batch_size,
    class_mode='input'
    )

validation_generator = datagen.flow_from_directory(
    'E:/fffiiles/college pracs and
projects/Project/Car/Auto/Safe_Test/',
    target_size=(SIZE, SIZE),
    batch_size=batch_size,
    class_mode='input'
    )
```

```
anomaly_generator = datagen.flow_from_directory(
    'E:/fffiiles/college pracs and projects/Project/Car/Auto/Unsafe_Train/',
    target_size=(SIZE, SIZE),
    batch_size=batch_size,
    class_mode='input'
    )

#Define the autoencoder.
#Try to make the bottleneck layer size as small as possible to make it easy for
#density calculations and also picking appropriate thresholds.

#Encoder
model = Sequential()
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(SIZE, SIZE, 3)))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
```

```
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))

#Decoder
model.add(Conv2D(8, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(16, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu',
padding='same'))
model.add(UpSampling2D((2, 2)))

model.add(Conv2D(3, (3, 3), activation='sigmoid',
padding='same'))

model.compile(optimizer='adam',
loss='mean_squared_error', metrics=['mse'])
```

model.summary()

2022-09-02 07:52:45.282543: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found

2022-09-02 07:52:45.307961: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-09-02 07:52:45.337810: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

Model: "sequential"

_____

 Layer (type) Output Shape Param #

==========

conv2d (Conv2D)	(None, 256, 256, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	18464
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 16)	4624
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_3 (Conv2D)	(None, 32, 32, 8)	1160
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 8)	0
conv2d_4 (Conv2D)	(None, 16, 16, 8)	584

```
up_sampling2d (UpSampling2D  (None, 32, 32, 8)        0
)

conv2d_5 (Conv2D)            (None, 32, 32, 16)       1168

up_sampling2d_1 (UpSampling  (None, 64, 64, 16)       0
2D)

conv2d_6 (Conv2D)            (None, 64, 64, 32)       4640

up_sampling2d_2 (UpSampling  (None, 128, 128, 32)     0
2D)

conv2d_7 (Conv2D)            (None, 128, 128, 64)     18496

up_sampling2d_3 (UpSampling  (None, 256, 256, 64)     0
2D)

conv2d_8 (Conv2D)            (None, 256, 256, 3)      1731

=================================================================
================
```

Total params: 52,659

Trainable params: 52,659

Non-trainable params: 0

_____

```
steps_per_epoch = len(train_generator)//batch_size


#validation_steps = len(validation_generator)//batch_size


#steps_per_epoch= 100 // batch_size,  validation_steps=75
// batch_size,


#Fit the model.
history = model.fit(
        train_generator,


        epochs=120,
        validation_data=validation_generator,
        validation_steps=75 // batch_size,
        shuffle = True)
```

Epoch 1/120

43/43 [==========] - ETA: 0s - loss: 0.0795 - mse: 0.0795 WARNING:tensorflow:Your input ran

```
out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 10 batches). You may need to use the repeat() function when building your dataset.

43/43 [==============================] - 105s 2s/step - loss: 0.0795 - mse: 0.0795 - val_loss: 0.0445 - val_mse: 0.0445

Epoch 2/120

43/43 [==============================] - 100s 2s/step - loss: 0.0358 - mse: 0.0358

Epoch 3/120

43/43 [==============================] - 93s 2s/step - loss: 0.0297 - mse: 0.0297

Epoch 4/120

43/43 [==============================] - 92s 2s/step - loss: 0.0271 - mse: 0.0271

Epoch 5/120

43/43 [==============================] - 94s 2s/step - loss: 0.0254 - mse: 0.0254

Epoch 6/120

43/43 [==============================] - 101s 2s/step - loss: 0.0235 - mse: 0.0235

Epoch 7/120

43/43 [==============================] - 99s 2s/step - loss: 0.0219 - mse: 0.0219
```

```
Epoch 8/120
43/43 [==============================] - 96s 2s/step - loss: 0.0208 - mse: 0.0208
Epoch 9/120
43/43 [==============================] - 101s 2s/step - loss: 0.0206 - mse: 0.0206
Epoch 10/120
43/43 [==============================] - 95s 2s/step - loss: 0.0200 - mse: 0.0200
Epoch 11/120
43/43 [==============================] - 95s 2s/step - loss: 0.0196 - mse: 0.0196
Epoch 12/120
43/43 [==============================] - 95s 2s/step - loss: 0.0191 - mse: 0.0191
Epoch 13/120
43/43 [==============================] - 96s 2s/step - loss: 0.0189 - mse: 0.0189
Epoch 14/120
43/43 [==============================] - 96s 2s/step - loss: 0.0189 - mse: 0.0189
Epoch 15/120
43/43 [==============================] - 92s 2s/step - loss: 0.0183 - mse: 0.0183
```

```
Epoch 16/120
43/43 [==============================] - 88s 2s/step - loss: 0.0183 - mse: 0.0183
Epoch 17/120
43/43 [==============================] - 94s 2s/step - loss: 0.0183 - mse: 0.0183
Epoch 18/120
43/43 [==============================] - 91s 2s/step - loss: 0.0181 - mse: 0.0181
Epoch 19/120
43/43 [==============================] - 92s 2s/step - loss: 0.0180 - mse: 0.0180
Epoch 20/120
43/43 [==============================] - 93s 2s/step - loss: 0.0177 - mse: 0.0177
Epoch 21/120
43/43 [==============================] - 93s 2s/step - loss: 0.0179 - mse: 0.0179
Epoch 22/120
43/43 [==============================] - 97s 2s/step - loss: 0.0176 - mse: 0.0176
Epoch 23/120
43/43 [==============================] - 100s 2s/step - loss: 0.0174 - mse: 0.0174
```

```
Epoch 24/120
43/43 [==============================] - 94s 2s/step - loss: 0.0173 - mse: 0.0173
Epoch 25/120
43/43 [==============================] - 92s 2s/step - loss: 0.0170 - mse: 0.0170
Epoch 26/120
43/43 [==============================] - 96s 2s/step - loss: 0.0171 - mse: 0.0171
Epoch 27/120
43/43 [==============================] - 95s 2s/step - loss: 0.0169 - mse: 0.0169
Epoch 28/120
43/43 [==============================] - 109s 3s/step - loss: 0.0171 - mse: 0.0171
Epoch 29/120
43/43 [==============================] - 100s 2s/step - loss: 0.0167 - mse: 0.0167
Epoch 30/120
43/43 [==============================] - 99s 2s/step - loss: 0.0165 - mse: 0.0165
Epoch 31/120
43/43 [==============================] - 103s 2s/step - loss: 0.0166 - mse: 0.0166
```

```
Epoch 32/120
43/43 [==============================] - 96s 2s/step - loss: 0.0167 - mse: 0.0167
Epoch 33/120
43/43 [==============================] - 104s 2s/step - loss: 0.0166 - mse: 0.0166
Epoch 34/120
43/43 [==============================] - 103s 2s/step - loss: 0.0164 - mse: 0.0164
Epoch 35/120
43/43 [==============================] - 100s 2s/step - loss: 0.0164 - mse: 0.0164
Epoch 36/120
43/43 [==============================] - 102s 2s/step - loss: 0.0162 - mse: 0.0162
Epoch 37/120
43/43 [==============================] - 93s 2s/step - loss: 0.0161 - mse: 0.0161
Epoch 38/120
43/43 [==============================] - 92s 2s/step - loss: 0.0163 - mse: 0.0163
Epoch 39/120
43/43 [==============================] - 97s 2s/step - loss: 0.0160 - mse: 0.0160
```

```
Epoch 40/120
43/43 [==============================] - 95s 2s/step - loss: 0.0159 - mse: 0.0159
Epoch 41/120
43/43 [==============================] - 97s 2s/step - loss: 0.0159 - mse: 0.0159
Epoch 42/120
43/43 [==============================] - 103s 2s/step - loss: 0.0161 - mse: 0.0161
Epoch 43/120
43/43 [==============================] - 105s 2s/step - loss: 0.0165 - mse: 0.0165
Epoch 44/120
43/43 [==============================] - 89s 2s/step - loss: 0.0158 - mse: 0.0158
Epoch 45/120
43/43 [==============================] - 85s 2s/step - loss: 0.0157 - mse: 0.0157
Epoch 46/120
43/43 [==============================] - 85s 2s/step - loss: 0.0156 - mse: 0.0156
Epoch 47/120
43/43 [==============================] - 103s 2s/step - loss: 0.0158 - mse: 0.0158
```

```
Epoch 48/120
43/43 [==============================] - 85s 2s/step - loss: 0.0156 - mse: 0.0156
Epoch 49/120
43/43 [==============================] - 90s 2s/step - loss: 0.0159 - mse: 0.0159
Epoch 50/120
43/43 [==============================] - 100s 2s/step - loss: 0.0157 - mse: 0.0157
Epoch 51/120
43/43 [==============================] - 102s 2s/step - loss: 0.0154 - mse: 0.0154
Epoch 52/120
43/43 [==============================] - 102s 2s/step - loss: 0.0154 - mse: 0.0154
Epoch 53/120
43/43 [==============================] - 99s 2s/step - loss: 0.0159 - mse: 0.0159
Epoch 54/120
43/43 [==============================] - 100s 2s/step - loss: 0.0158 - mse: 0.0158
Epoch 55/120
43/43 [==============================] - 112s 3s/step - loss: 0.0156 - mse: 0.0156
```

```
Epoch 56/120
43/43 [==============================] - 108s 3s/step - loss: 0.0156 - mse: 0.0156
Epoch 57/120
43/43 [==============================] - 107s 2s/step - loss: 0.0154 - mse: 0.0154
Epoch 58/120
43/43 [==============================] - 113s 3s/step - loss: 0.0152 - mse: 0.0152
Epoch 59/120
43/43 [==============================] - 108s 2s/step - loss: 0.0156 - mse: 0.0156
Epoch 60/120
43/43 [==============================] - 104s 2s/step - loss: 0.0172 - mse: 0.0172
Epoch 61/120
43/43 [==============================] - 98s 2s/step - loss: 0.0156 - mse: 0.0156
Epoch 62/120
43/43 [==============================] - 87s 2s/step - loss: 0.0154 - mse: 0.0154
Epoch 63/120
43/43 [==============================] - 114s 3s/step - loss: 0.0152 - mse: 0.0152
```

```
Epoch 64/120
43/43 [==============================] - 106s 2s/step - loss: 0.0151 - mse: 0.0151
Epoch 65/120
43/43 [==============================] - 92s 2s/step - loss: 0.0152 - mse: 0.0152
Epoch 66/120
43/43 [==============================] - 103s 2s/step - loss: 0.0151 - mse: 0.0151
Epoch 67/120
43/43 [==============================] - 103s 2s/step - loss: 0.0150 - mse: 0.0150
Epoch 68/120
43/43 [==============================] - 103s 2s/step - loss: 0.0150 - mse: 0.0150
Epoch 69/120
43/43 [==============================] - 102s 2s/step - loss: 0.0152 - mse: 0.0152
Epoch 70/120
43/43 [==============================] - 101s 2s/step - loss: 0.0150 - mse: 0.0150
Epoch 71/120
43/43 [==============================] - 100s 2s/step - loss: 0.0150 - mse: 0.0150
```

```
Epoch 72/120
43/43 [==============================] - 100s 2s/step - loss: 0.0154 - mse: 0.0154
Epoch 73/120
43/43 [==============================] - 101s 2s/step - loss: 0.0149 - mse: 0.0149
Epoch 74/120
43/43 [==============================] - 93s 2s/step - loss: 0.0148 - mse: 0.0148
Epoch 75/120
43/43 [==============================] - 86s 2s/step - loss: 0.0151 - mse: 0.0151
Epoch 76/120
43/43 [==============================] - 112s 3s/step - loss: 0.0150 - mse: 0.0150
Epoch 77/120
43/43 [==============================] - 103s 2s/step - loss: 0.0148 - mse: 0.0148
Epoch 78/120
43/43 [==============================] - 96s 2s/step - loss: 0.0149 - mse: 0.0149
Epoch 79/120
43/43 [==============================] - 93s 2s/step - loss: 0.0150 - mse: 0.0150
```

```
Epoch 80/120
43/43 [==============================] - 94s 2s/step - loss: 0.0147 - mse: 0.0147
Epoch 81/120
43/43 [==============================] - 97s 2s/step - loss: 0.0147 - mse: 0.0147
Epoch 82/120
43/43 [==============================] - 95s 2s/step - loss: 0.0147 - mse: 0.0147
Epoch 83/120
43/43 [==============================] - 99s 2s/step - loss: 0.0148 - mse: 0.0148
Epoch 84/120
43/43 [==============================] - 100s 2s/step - loss: 0.0147 - mse: 0.0147
Epoch 85/120
43/43 [==============================] - 94s 2s/step - loss: 0.0150 - mse: 0.0150
Epoch 86/120
43/43 [==============================] - 94s 2s/step - loss: 0.0153 - mse: 0.0153
Epoch 87/120
43/43 [==============================] - 96s 2s/step - loss: 0.0147 - mse: 0.0147
```

```
Epoch 88/120
43/43 [==============================] - 99s 2s/step - loss: 0.0149 - mse: 0.0149
Epoch 89/120
43/43 [==============================] - 94s 2s/step - loss: 0.0146 - mse: 0.0146
Epoch 90/120
43/43 [==============================] - 94s 2s/step - loss: 0.0145 - mse: 0.0145
Epoch 91/120
43/43 [==============================] - 93s 2s/step - loss: 0.0145 - mse: 0.0145
Epoch 92/120
43/43 [==============================] - 94s 2s/step - loss: 0.0148 - mse: 0.0148
Epoch 93/120
43/43 [==============================] - 92s 2s/step - loss: 0.0145 - mse: 0.0145
Epoch 94/120
43/43 [==============================] - 94s 2s/step - loss: 0.0149 - mse: 0.0149
Epoch 95/120
43/43 [==============================] - 92s 2s/step - loss: 0.0146 - mse: 0.0146
```

```
Epoch 96/120
43/43 [==============================] - 95s 2s/step - loss: 0.0145 - mse: 0.0145
Epoch 97/120
43/43 [==============================] - 100s 2s/step - loss: 0.0147 - mse: 0.0147
Epoch 98/120
43/43 [==============================] - 93s 2s/step - loss: 0.0144 - mse: 0.0144
Epoch 99/120
43/43 [==============================] - 100s 2s/step - loss: 0.0144 - mse: 0.0144
Epoch 100/120
43/43 [==============================] - 96s 2s/step - loss: 0.0146 - mse: 0.0146
Epoch 101/120
43/43 [==============================] - 94s 2s/step - loss: 0.0145 - mse: 0.0145
Epoch 102/120
43/43 [==============================] - 95s 2s/step - loss: 0.0146 - mse: 0.0146
Epoch 103/120
43/43 [==============================] - 95s 2s/step - loss: 0.0146 - mse: 0.0146
```

```
Epoch 104/120
43/43 [==============================] - 94s 2s/step - loss: 0.0143 - mse: 0.0143
Epoch 105/120
43/43 [==============================] - 94s 2s/step - loss: 0.0144 - mse: 0.0144
Epoch 106/120
43/43 [==============================] - 92s 2s/step - loss: 0.0143 - mse: 0.0143
Epoch 107/120
43/43 [==============================] - 94s 2s/step - loss: 0.0145 - mse: 0.0145
Epoch 108/120
43/43 [==============================] - 93s 2s/step - loss: 0.0143 - mse: 0.0143
Epoch 109/120
43/43 [==============================] - 94s 2s/step - loss: 0.0144 - mse: 0.0144
Epoch 110/120
43/43 [==============================] - 92s 2s/step - loss: 0.0149 - mse: 0.0149
Epoch 111/120
43/43 [==============================] - 93s 2s/step - loss: 0.0143 - mse: 0.0143
```

```
Epoch 112/120
43/43 [==============================] - 92s 2s/step - loss: 0.0142 - mse: 0.0142
Epoch 113/120
43/43 [==============================] - 95s 2s/step - loss: 0.0143 - mse: 0.0143
Epoch 114/120
43/43 [==============================] - 98s 2s/step - loss: 0.0142 - mse: 0.0142
Epoch 115/120
43/43 [==============================] - 88s 2s/step - loss: 0.0141 - mse: 0.0141
Epoch 116/120
43/43 [==============================] - 88s 2s/step - loss: 0.0141 - mse: 0.0141
Epoch 117/120
43/43 [==============================] - 82s 2s/step - loss: 0.0142 - mse: 0.0142
Epoch 118/120
43/43 [==============================] - 81s 2s/step - loss: 0.0141 - mse: 0.0141
Epoch 119/120
43/43 [==============================] - 86s 2s/step - loss: 0.0142 - mse: 0.0142
```

```
Epoch 120/120
43/43 [==============================] - 83s 2s/step - loss: 0.0142 - mse: 0.0142
```

```
model.save('E:/fffiiles/college pracs and projects/Project/Car/Code/model12.hdf5')
```

```
#plot the training and validation accuracy and loss at each epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Get all batches generated by the datagen and pick a batch for prediction
```

```
#Just to test the model.
data_batch = []  #Capture all training batches as a numpy array
img_num = 0
while img_num <= train_generator.batch_index:   #gets each generated batch of size batch_size
    data = train_generator.next()
    data_batch.append(data[0])
    img_num = img_num + 1

predicted = model.predict(data_batch[0])  #Predict on the first batch of images


#Sanity check, view few images and corresponding reconstructions
image_number = random.randint(0, predicted.shape[0])
plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.imshow(data_batch[0][image_number])
plt.subplot(122)
plt.imshow(predicted[image_number])
plt.show()
```

#Let us examine the reconstruction error between our validation data (good/normal images)

and the anomaly images

validation_error = model.evaluate_generator(validation_generator)

anomaly_error = model.evaluate_generator(anomaly_generator)

print("Recon. error for the validation (normal) data is: ", validation_error)

print("Recon. error for the anomaly data is: ", anomaly_error)

Recon. error for the validation (normal) data is: [0.015407955273985863, 0.015407955273985863]

Recon. error for the anomaly data is: [0.013864816166460514, 0.013864816166460514]

#Let us extract (or build) the encoder network, with trained weights.

#This is used to get the compressed output (latent space) of the input image.

#The compressed output is then used to calculate the KDE

```
encoder_model = Sequential()

encoder_model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(SIZE, SIZE, 3), weights=model.layers[0].get_weights()) )

encoder_model.add(MaxPooling2D((2, 2), padding='same'))

encoder_model.add(Conv2D(32, (3, 3), activation='relu', padding='same', weights=model.layers[2].get_weights()))

encoder_model.add(MaxPooling2D((2, 2), padding='same'))

encoder_model.add(Conv2D(16, (3, 3), activation='relu', padding='same', weights=model.layers[4].get_weights()))

encoder_model.add(MaxPooling2D((2, 2), padding='same'))

encoder_model.add(Conv2D(8, (3, 3), activation='relu', padding='same', weights=model.layers[6].get_weights()))

encoder_model.add(MaxPooling2D((2, 2), padding='same'))

encoder_model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 256, 256, 64)      1792

max_pooling2d_4 (MaxPooling  (None, 128, 128, 64)      0
2D)

conv2d_10 (Conv2D)           (None, 128, 128, 32)      18464

max_pooling2d_5 (MaxPooling  (None, 64, 64, 32)        0
2D)

conv2d_11 (Conv2D)           (None, 64, 64, 16)        4624

max_pooling2d_6 (MaxPooling  (None, 32, 32, 16)        0
2D)

conv2d_12 (Conv2D)           (None, 32, 32, 8)         1160

max_pooling2d_7 (MaxPooling  (None, 16, 16, 8)         0
2D)
```

```
=================================================================
Total params: 26,040
Trainable params: 26,040
Non-trainable params: 0
_________________________________________________________________
##################################################################
# Calculate KDE using sklearn
from sklearn.neighbors import KernelDensity

#Get encoded output of input images = Latent space
encoded_images = encoder_model.predict_generator(train_generator)

# Flatten the encoder output because KDE from sklearn takes 1D vectors as input
encoder_output_shape = encoder_model.output_shape #Here, we have 16x16x16
out_vector_shape = encoder_output_shape[1]*encoder_output_shape[2]*encoder_output_shape[3]
```

```
encoded_images_vector = [np.reshape(img,
(out_vector_shape)) for img in encoded_images]


#Fit KDE to the image latent data

kde = KernelDensity(kernel='gaussian',
bandwidth=0.2).fit(encoded_images_vector)


#Calculate density and reconstruction error to find their
means values for

#good and anomaly images.

#We use these mean and sigma to set thresholds.

def calc_density_and_recon_error(batch_images):


    density_list=[]

    recon_error_list=[]

    for im in range(0, batch_images.shape[0]-1):


        img  = batch_images[im]

        img = img[np.newaxis, :,:,:]

        encoded_img = encoder_model.predict([[img]]) # Create
a compressed version of the image using the encoder
```

```
        encoded_img = [np.reshape(img, (out_vector_shape))
for img in encoded_img] # Flatten the compressed image
        density = kde.score_samples(encoded_img)[0] # get a
density score for the new image
        reconstruction = model.predict([[img]])
        reconstruction_error =
model.evaluate([reconstruction],[[img]], batch_size = 1)[0]
        density_list.append(density)
        recon_error_list.append(reconstruction_error)

    average_density = np.mean(np.array(density_list))
    stdev_density = np.std(np.array(density_list))

    average_recon_error =
np.mean(np.array(recon_error_list))
    stdev_recon_error = np.std(np.array(recon_error_list))

    return average_density, stdev_density,
average_recon_error, stdev_recon_error

#Get average and std dev. of density and recon. error for
uninfected and anomaly (parasited) images.
#For this let us generate a batch of images for each.
```

```
train_batch = train_generator.next()[0]
anomaly_batch = anomaly_generator.next()[0]

uninfected_values =
calc_density_and_recon_error(train_batch)
anomaly_values =
calc_density_and_recon_error(anomaly_batch)
```

```
1/1 [==============================] - 0s 470ms/step
1/1 [==============================] - 1s 563ms/step -
loss: 0.0105 - mse: 0.0105
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 109ms/step -
loss: 0.0183 - mse: 0.0183
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 94ms/step -
loss: 0.0181 - mse: 0.0181
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 109ms/step
```

```
1/1 [==============================] - 0s 109ms/step - loss: 0.0176 - mse: 0.0176
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 109ms/step - loss: 0.0197 - mse: 0.0197
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 109ms/step - loss: 0.0154 - mse: 0.0154
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 109ms/step - loss: 0.0117 - mse: 0.0117
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 94ms/step - loss: 0.0126 - mse: 0.0126
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 109ms/step - loss: 0.0138 - mse: 0.0138
1/1 [==============================] - 0s 47ms/step
```

```
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 94ms/step - loss: 0.0133 - mse: 0.0133
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 94ms/step - loss: 0.0193 - mse: 0.0193
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 94ms/step - loss: 0.0182 - mse: 0.0182
```

```
#Now, input unknown images and sort as Good or Anomaly
def check_anomaly(img_path):
    density_threshold = 2500 #Set this value based on the above exercise
    reconstruction_error_threshold = 0.004 # Set this value based on the above exercise
    img  = Image.open(img_path)
    img = np.array(img.resize((256,256), Image.ANTIALIAS))
    plt.imshow(img)
    img = img / 255.
    img = img[np.newaxis, :,:,:]
    encoded_img = encoder_model.predict([[img]])
```

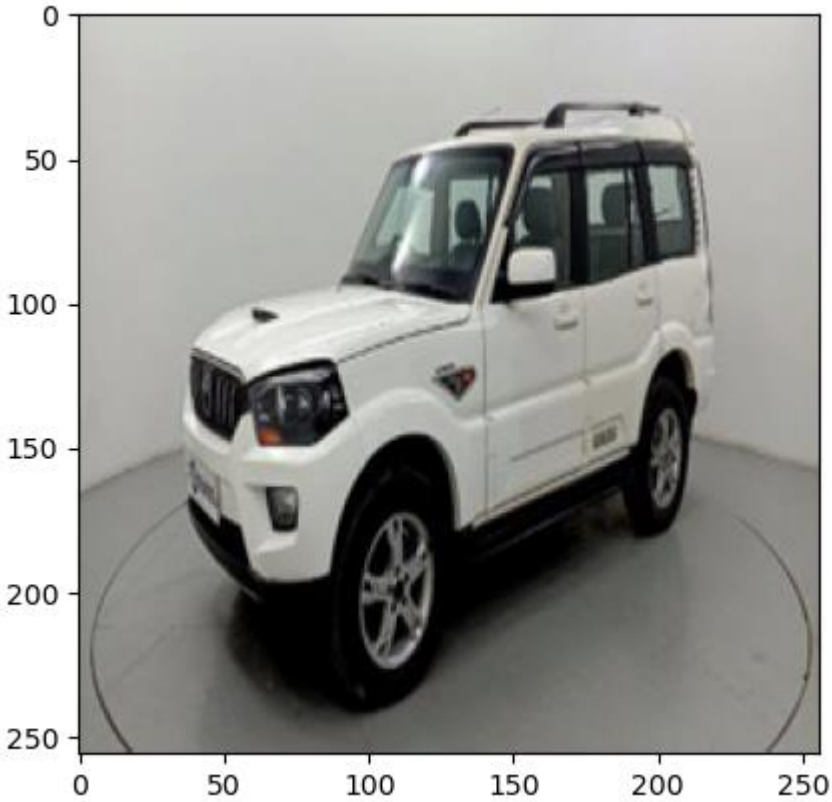
```
    encoded_img = [np.reshape(img, (out_vector_shape)) for
img in encoded_img]
    density = kde.score_samples(encoded_img)[0]


    reconstruction = model.predict([[img]])
    reconstruction_error =
model.evaluate([reconstruction],[[img]], batch_size = 1)[0]


    if density < density_threshold or reconstruction_error >
reconstruction_error_threshold:
        print("Unsafe Car")


    else:
        print("Safe Car")



#Load a couple of test images and verify whether they are
reported as anomalies.
import glob
para_file_paths = glob.glob('E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/Unsafe/*')
```

uninfected_file_paths = glob.glob('E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/Safe/*')

#Anomaly image verification

num=random.randint(0,len(para_file_paths)-1)

check_anomaly(para_file_paths[num])

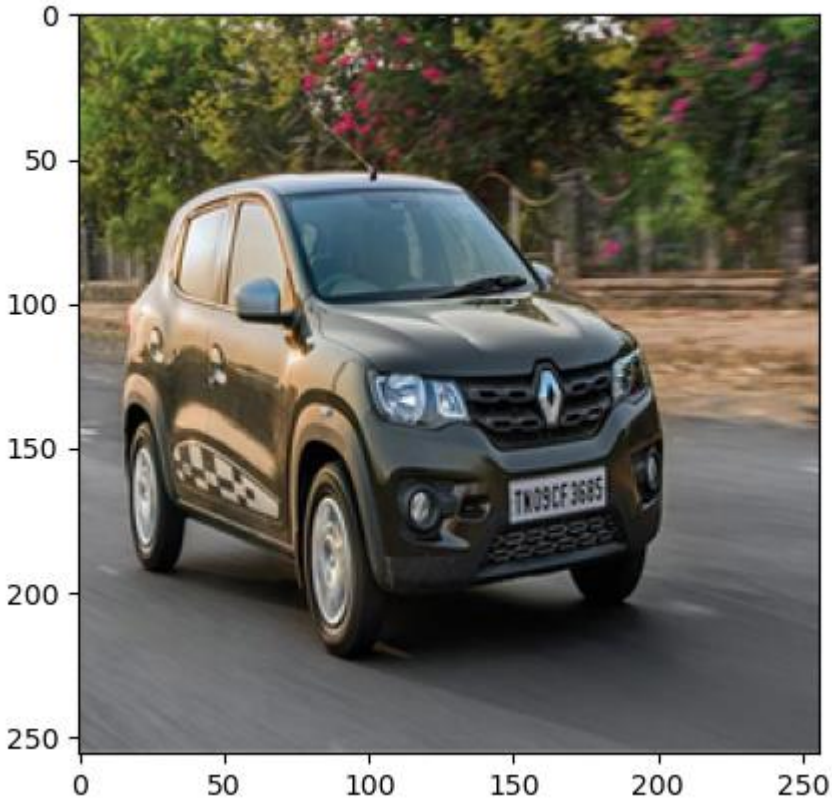


1/1 [==========] - 0s 78ms/step

1/1 [==========] - 0s 109ms/step

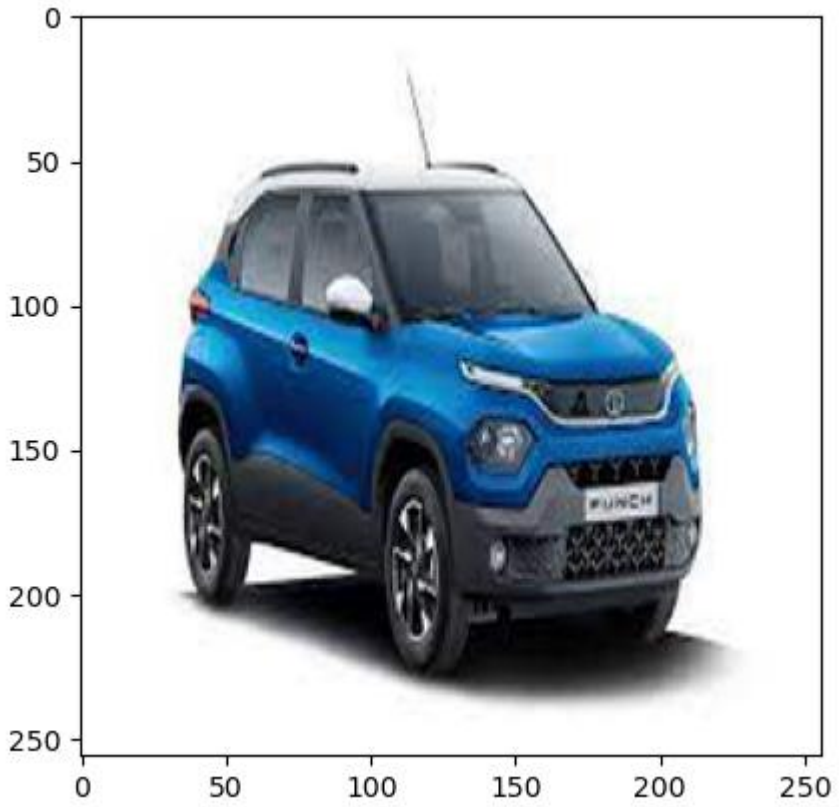
1/1 [==========] - 0s 109ms/step - loss: 0.0095 - mse: 0.0095

Unsafe Car




```
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 94ms/step - loss: 0.0126 - mse: 0.0126
Unsafe Car
```

```
#Good/normal image verification
num=random.randint(0,len(para_file_paths)-1)
check_anomaly(uninfected_file_paths[num])
```



1/1 [==========] - 0s 63ms/step

1/1 [==========] - 0s 109ms/step

1/1 [==========] - 0s 94ms/step - loss: 0.0101 - mse: 0.0101

Unsafe Car

Gan

-*- coding: utf-8 -*-

"""

Created on Sat Sep 3 09:26:20 2022

@author: LAXMINARAYANRO
"""

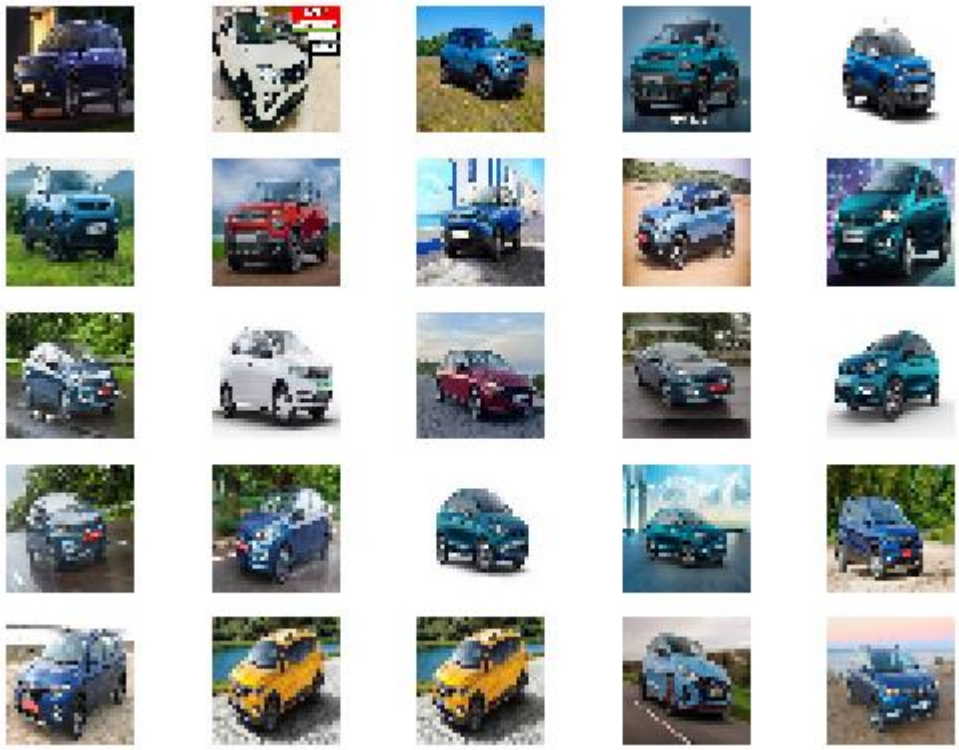
```
from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
#from keras.datasets.cifar10 import load_data
from tensorflow.keras.optimizers import Adam
#from keras.models import Sequential
from tensorflow.keras.models import Model, Sequential
# from keras.layers import Dense
# from keras.layers import Reshape
# from keras.layers import Flatten
# from keras.layers import Conv2D
# from keras.layers import Conv2DTranspose
# from keras.layers import LeakyReLU
# from keras.layers import Dropout
from tensorflow.keras.layers import (
```

```
    LeakyReLU, SeparableConv2D, MaxPooling2D, Activation,
Flatten, Dropout, Conv2D, Dense,Conv2DTranspose,Input,
Dense, Lambda, Reshape
)
import numpy as np
import glob
import cv2
import os

from matplotlib import pyplot as plt
##################################################
#######################

#Load data and plot to get a quick understanding
#CIFAR10 classes are: airplane, automobile, bird, cat, deer,
dog, frog, horse,
# ship, truck
SIZE = 32
#Channel=3  #Resize images

#lists
train_images = []
train_labels = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists


test_images = []
```

```
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)
```

```
trainX, trainy, testX, testy = train_images,
train_labels_encoded, test_images, test_labels_encoded

testy = np.asarray(testy).reshape((-1,1))

trainy = np.asarray(trainy).reshape((-1,1))

testy = testy.astype(np.uint8)

trainy = trainy.astype(np.uint8)


#(trainX, trainy), (testX, testy) = load_data()


# plot 25 images

for i in range(25):

        plt.subplot(5, 5, 1 + i)

        plt.axis('off')

        plt.imshow(trainX[i])

plt.show()
```



##
##############################

#Define generator, discriminator, gan and other helper functions

#Using Sequential method from Keras as it makes the definition of models easy.

#We will use functional way of defining the model for the conditional gan

#but sequential for descriminator and generator as they are straightforward.

```
##################################################
#######################

# define the standalone discriminator model
#Given an input image, the Discriminator outputs the
likelihood of the image being real.
#Binary classification - true or false (1 or 0). So using sigmoid
activation.
def define_discriminator(in_shape=(32,32,3)):
    model = Sequential()

    model.add(Conv2D(128, (3,3), strides=(2,2),
padding='same', input_shape=in_shape)) #16x16x128
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2D(128, (3,3), strides=(2,2),
padding='same')) #8x8x128
    model.add(LeakyReLU(alpha=0.2))

    model.add(Flatten()) #shape of 8192
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid')) #shape of 1
    # compile model
```

```
    opt = Adam(learning_rate=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy',
optimizer=opt, metrics=['accuracy'])
    return model

test_discr = define_discriminator()
print(test_discr.summary())
```

```
Model: "sequential"
_____________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 16, 16, 128)       3584

 leaky_re_lu (LeakyReLU)     (None, 16, 16, 128)       0

 conv2d_1 (Conv2D)           (None, 8, 8, 128)         147584

 leaky_re_lu_1 (LeakyReLU)   (None, 8, 8, 128)         0

 flatten (Flatten)           (None, 8192)              0
```

dropout (Dropout) (None, 8192) 0

dense (Dense) (None, 1) 8193

==========

Total params: 159,361

Trainable params: 159,361

Non-trainable params: 0

_____

define the standalone generator model

#Given input of latent vector, the Generator produces an image.(here: 32x32)

#latent_dim, for example, can be 100, 1D array of size 100

#Here we are only using Dense and conv2dlayers. But network can be complicated based

#on the application. For example, you can use VGG for super res. GAN.

def define_generator(latent_dim): #latent_dim is the dimension of the latent vector (e.g., 100)

```
model = Sequential()
# We will reshape input latent vector into 8x8 image as a starting point.
#So n_nodes for the Dense layer can be 128x8x8 so when we reshape the output
#it would be 8x8x128 and that can be slowly upscaled to 32x32 image for output.
n_nodes = 128 * 8 * 8  #8192 nodes
model.add(Dense(n_nodes, input_dim=latent_dim)) #Dense layer so we can work with 1D latent vector
model.add(LeakyReLU(alpha=0.2))
model.add(Reshape((8, 8, 128)))  #8x8x128 dataset from the latent vector.
# upsample to 16x16
model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')) #16x16x128
model.add(LeakyReLU(alpha=0.2))
# upsample to 32x32
model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')) #32x32x128
model.add(LeakyReLU(alpha=0.2))
# generate
model.add(Conv2D(3, (8,8), activation='tanh', padding='same')) #32x32x3
```

```
    return model  #Model not compiled as it is not directly trained like the discriminator.
                #Generator is trained via GAN combined model.

test_gen = define_generator(100)
print(test_gen.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 8192)              827392

 leaky_re_lu_2 (LeakyReLU)   (None, 8192)              0

 reshape (Reshape)           (None, 8, 8, 128)         0

 conv2d_transpose (Conv2DTra  (None, 16, 16, 128)
262272
 nspose)
```

```
 leaky_re_lu_3 (LeakyReLU)   (None, 16, 16, 128)      0

 conv2d_transpose_1 (Conv2DT  (None, 32, 32, 128)
262272
 ranspose)

 leaky_re_lu_4 (LeakyReLU)   (None, 32, 32, 128)      0

 conv2d_2 (Conv2D)           (None, 32, 32, 3)        24579

=================================================================
Total params: 1,376,515
Trainable params: 1,376,515
Non-trainable params: 0
_________________________________________________________________
```

define the combined generator and discriminator model, for updating the generator

#Discriminator is trained separately so here only generator will be trained by keeping

#the discriminator constant.

```
def define_gan(generator, discriminator):
	discriminator.trainable = False  #Discriminator is trained separately. So set to not trainable.
	# connect generator and discriminator
	model = Sequential()
	model.add(generator)
	model.add(discriminator)
	# compile model
	opt = Adam(learning_rate=0.0002, beta_1=0.5)
	model.compile(loss='binary_crossentropy', optimizer=opt)
	return model



# load cifar training images
def load_real_samples():

	# cConvert to float and scale.
	X = trainX.astype('float32')
	# scale from [0,255] to [-1,1]
	X = (X - 127.5) / 127.5  #Generator uses tanh activation so rescale
```

```
                    #original images to -1 to 1 to match the
output of generator.
    return X


# pick a batch of random real samples to train the GAN
#In fact, we will train the GAN on a half batch of real images
and another
#half batch of fake images.
#For each real image we assign a label 1 and for fake we
assign label 0.
def generate_real_samples(dataset, n_samples):
    # choose random images
    ix = randint(0, dataset.shape[0], n_samples)
    # select the random images and assign it to X
    X = dataset[ix]
    # generate class labels and assign to y
    y = ones((n_samples, 1)) ##Label=1 indicating they are
real
    return X, y


# generate n_samples number of latent vectors as input for
the generator
def generate_latent_points(latent_dim, n_samples):
```

```
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input


# use the generator to generate n fake examples, with class labels
#Supply the generator, latent_dim and number of samples as input.
#Use the above latent point generator to generate latent points.
def generate_fake_samples(generator, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict using generator to generate fake samples.
    X = generator.predict(x_input)
    # Class labels will be 0 as these samples are fake.
    y = zeros((n_samples, 1))  #Label=0 indicating they are fake
    return X, y
```

train the generator and discriminator

#We loop through a number of epochs to train our Discriminator by first selecting

#a random batch of images from our true/real dataset.

#Then, generating a set of images using the generator.

#Feed both set of images into the Discriminator.

#Finally, set the loss parameters for both the real and fake images, as well as the combined loss.

```
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=100, n_batch=128):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)  #the discriminator model is updated for a half batch of real samples
                        #and a half batch of fake samples, combined a single batch.
    # manually enumerate epochs and bacthes.
    for i in range(n_epochs):
        # enumerate batches over the training set
        for j in range(bat_per_epo):
```

```
        # Train the discriminator on real and fake images, separately (half batch each)
      #Research showed that separate training is more effective.
                # get randomly selected 'real' samples
                X_real, y_real = generate_real_samples(dataset, half_batch)
                # update discriminator model weights
        ##train_on_batch allows you to update weights based on a collection
        #of samples you provide
        #Let us just capture loss and ignore accuracy value (2nd output below)
                d_loss_real, _ = d_model.train_on_batch(X_real, y_real)

        # generate 'fake' examples
                X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
                # update discriminator model weights
                d_loss_fake, _ = d_model.train_on_batch(X_fake, y_fake)
```

#d_loss = 0.5 * np.add(d_loss_real, d_loss_fake) #Average loss if you want to report single..

prepare points in latent space as input for the generator

X_gan = generate_latent_points(latent_dim, n_batch)

The generator wants the discriminator to label the generated samples

as valid (ones)

#This is where the generator is trying to trick discriminator into believing

#the generated image is true (hence value of 1 for y)

y_gan = ones((n_batch, 1))

Generator is part of combined model where it got directly linked with the discriminator

Train the generator with latent_dim as x and 1 as y.

Again, 1 as the output as it is adversarial and if generator did a great

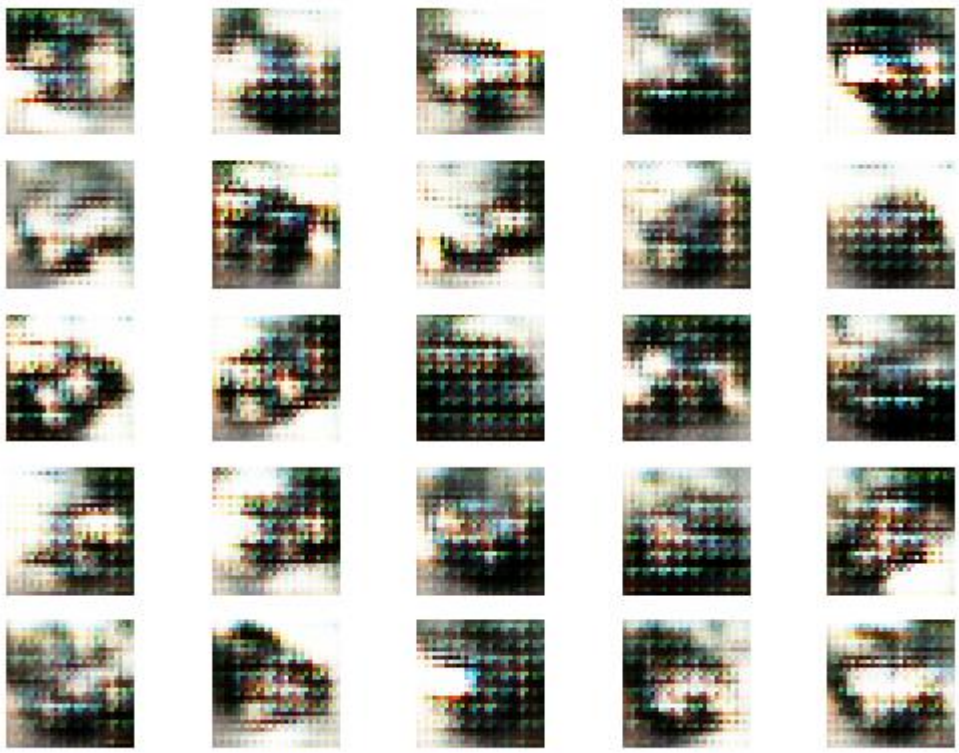
#job of folling the discriminator then the output would be 1 (true)

```
                    # update the generator via the discriminator's
error
                    g_loss = gan_model.train_on_batch(X_gan,
y_gan)


        # Print losses on this batch
                    print('Epoch>%d, Batch %d/%d, d1=%.3f,
d2=%.3f g=%.3f' %
                          (i+1, j+1, bat_per_epo, d_loss_real,
d_loss_fake, g_loss))
      # save the generator model
      g_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/gancar.h5')


##################################################################
#################
#Train the GAN


# size of the latent space
latent_dim = 100
# create the discriminator
discriminator = define_discriminator()
# create the generator
```

```
generator = define_generator(latent_dim)
# create the gan
gan_model = define_gan(generator, discriminator)
# load image data
dataset = load_real_samples()
# train model



train(generator, discriminator, gan_model, dataset,
latent_dim, n_epochs=300)

##################################################
################################

# Now, let us load the generator model and generate images

from keras.models import load_model
from numpy.random import randn



# Plot generated images
def show_plot(examples, n):
```

```
    for i in range(n * n):
        plt.subplot(n, n, 1 + i)
        plt.axis('off')
        plt.imshow(examples[i, :, :, :])
    plt.show()

# load model
model = load_model('E:/fffiiles/college pracs and projects/Project/Car/Code/gancar.h5') #Model trained for 100 epochs

#model.compile(optimizer=Adam(learning_rate=0.001),loss='categorical_crossentropy',metrics=['accuracy'])

#model = load_model('E:/fffiiles/college pracs and projects/Project/Car/Code/gancar.h5') #Model trained for 100 epochs

# generate images
latent_points = generate_latent_points(100, 25)  #Latent dim and n_samples
# generate images
X = model.predict(latent_points)
```

```
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0

import numpy as np
X = (X*255).astype(np.uint8)

# plot the result
show_plot(X, 5)
```



Conditional Gan

-*- coding: utf-8 -*-

"""

Created on Sun Sep 4 18:42:09 2022

@author: LAXMINARAYANRO

"""

-*- coding: utf-8 -*-

```
"""
Created on Sat Sep  3 09:26:20 2022

@author: LAXMINARAYANRO
"""



from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
#from keras.datasets.cifar10 import load_data
from tensorflow.keras.optimizers import Adam
#from keras.models import Sequential
from tensorflow.keras.models import Model, Sequential
# from keras.layers import Dense
# from keras.layers import Reshape
# from keras.layers import Flatten
# from keras.layers import Conv2D
# from keras.layers import Conv2DTranspose
# from keras.layers import LeakyReLU
# from keras.layers import Dropout
```

```
from tensorflow.keras.layers import (
    LeakyReLU, SeparableConv2D, MaxPooling2D, Activation,
Flatten, Dropout, Conv2D, Dense,Conv2DTranspose,Input,
Dense, Lambda, Reshape
)
import numpy as np
import glob
import cv2
import os

from matplotlib import pyplot as plt
##################################################
#######################

#Load data and plot to get a quick understanding
#CIFAR10 classes are: airplane, automobile, bird, cat, deer,
dog, frog, horse,
# ship, truck
SIZE = 256
#Channel=3  #Resize images

#lists
train_images = []
```

```
train_labels = []
#E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*
for directory_path in glob.glob("E:/fffiiles/college pracs and projects/Project/Car/Model_images/train/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists
```

```
#E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        test_images.append(img)
        test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
```

```
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

trainX, trainy, testX, testy = train_images,
train_labels_encoded, test_images, test_labels_encoded
testy = np.asarray(testy).reshape((-1,1))
trainy = np.asarray(trainy).reshape((-1,1))
testy = testy.astype(np.uint8)
trainy = trainy.astype(np.uint8)

#(trainX, trainy), (testX, testy) = load_data()

# plot 25 images
for i in range(4):
    plt.subplot(2, 2, 1 + i)
    plt.axis('off')
    plt.imshow(trainX[i])
plt.show()
```

```
##################################################################################################
#Define generator, discriminator, gan and other helper functions
#Using Sequential method from Keras as it makes the definition of models easy.
#We will use functional way of defining the model for the conditional gan
#but sequential for descriminator and generator as they are straightforward.
##################################################################################################

# define the standalone discriminator model
#Given an input image, the Discriminator outputs the likelihood of the image being real.
#Binary classification - true or false (1 or 0). So using sigmoid activation.
def define_discriminator(in_shape=(256,256,3)):
    model = Sequential()
    model.add(Conv2D(128, (3,3), strides=(2,2), padding='same', input_shape=in_shape)) #16x16x128
    model.add(LeakyReLU(alpha=0.2))
```

```
    model.add(Conv2D(128, (3,3), strides=(2,2),
padding='same')) #8x8x128
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2D(128, (3,3), strides=(2,2),
padding='same')) #8x8x128
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2D(128, (3,3), strides=(2,2),
padding='same')) #8x8x128
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2D(128, (3,3), strides=(2,2),
padding='same')) #8x8x128
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2D(128, (3,3), strides=(2,2),
padding='same')) #8x8x128
    model.add(LeakyReLU(alpha=0.2))
    model.add(Flatten()) #shape of 8192
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid')) #shape of 1
        # compile model
    opt = Adam(learning_rate=0.0002, beta_1=0.5)
```

```
    model.compile(loss='binary_crossentropy', optimizer=opt,
metrics=['accuracy'])
    return model

test_discr = define_discriminator()
print(test_discr.summary())

# define the standalone generator model
# #Given input of latent vector, the Generator produces an
image.(here: 32x32)
#latent_dim, for example, can be 100, 1D array of size 100

#Here we are only using Dense and conv2dlayers. But
network can be complicated based
#on the application. For example, you can use VGG for super
res. GAN.

def define_generator(latent_dim):
    model = Sequential()
    n_nodes = 128 * 8 * 8
    model.add(Dense(n_nodes, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((8, 8, 128)))
```

```
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2),
padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2),
padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2),
padding='same')) #32x32x128
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2DTranspose(128, (4,4), strides=(2,2),
padding='same'))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2DTranspose(128, (4,4), strides=(2,2),
padding='same'))
    model.add(LeakyReLU(alpha=0.2))
        # generate
    model.add(Conv2D(3, (8,8), activation='tanh',
padding='same')) #32x32x3
    return model  #Model not compiled as it is not directly
trained like the discriminator.
                #Generator is trained via GAN combined model.
```

```
test_gen = define_generator(100)
print(test_gen.summary())



# define the combined generator and discriminator model, for updating the generator
#Discriminator is trained separately so here only generator will be trained by keeping
#the discriminator constant.
def define_gan(generator, discriminator):
    discriminator.trainable = False  #Discriminator is trained separately. So set to not trainable.
    # connect generator and discriminator
    model = Sequential()
    model.add(generator)
    model.add(discriminator)
    # compile model
    opt = Adam(learning_rate=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model
```

```
# load cifar training images
def load_real_samples():

    # cConvert to float and scale.
    X = trainX.astype('float32')
    # scale from [0,255] to [-1,1]
    X = (X - 127.5) / 127.5  #Generator uses tanh activation so rescale
                             #original images to -1 to 1 to match the output of generator.
    return X

# pick a batch of random real samples to train the GAN
#In fact, we will train the GAN on a half batch of real images and another
#half batch of fake images.
#For each real image we assign a label 1 and for fake we assign label 0.
def generate_real_samples(dataset, n_samples):
    # choose random images
    ix = randint(0, dataset.shape[0], n_samples)
    # select the random images and assign it to X
    X = dataset[ix]
```

```
    # generate class labels and assign to y
    y = ones((n_samples, 1)) ##Label=1 indicating they are real
    return X, y

# generate n_samples number of latent vectors as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input

# use the generator to generate n fake examples, with class labels
#Supply the generator, latent_dim and number of samples as input.
#Use the above latent point generator to generate latent points.
def generate_fake_samples(generator, latent_dim, n_samples):
    # generate points in latent space
```

```
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict using generator to generate fake samples.
    X = generator.predict(x_input)
    # Class labels will be 0 as these samples are fake.
    y = zeros((n_samples, 1))  #Label=0 indicating they are fake
    return X, y

# train the generator and discriminator
#We loop through a number of epochs to train our Discriminator by first selecting
#a random batch of images from our true/real dataset.
#Then, generating a set of images using the generator.
#Feed both set of images into the Discriminator.
#Finally, set the loss parameters for both the real and fake images, as well as the combined loss.

def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=300, n_batch=128):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)  #the discriminator model is updated for a half batch of real samples
```

#and a half batch of fake samples, combined a single batch.

manually enumerate epochs and bacthes.

for i in range(n_epochs):

enumerate batches over the training set

for j in range(bat_per_epo):

Train the discriminator on real and fake images, separately (half batch each)

#Research showed that separate training is more effective.

get randomly selected 'real' samples

X_real, y_real = generate_real_samples(dataset, half_batch)

update discriminator model weights

##train_on_batch allows you to update weights based on a collection

#of samples you provide

#Let us just capture loss and ignore accuracy value (2nd output below)

d_loss_real, _ = d_model.train_on_batch(X_real, y_real)

generate 'fake' examples

```
                X_fake, y_fake =
generate_fake_samples(g_model, latent_dim, half_batch)
                # update discriminator model weights
                d_loss_fake, _ =
d_model.train_on_batch(X_fake, y_fake)


        #d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
#Average loss if you want to report single..


                # prepare points in latent space as input for
the generator
                X_gan = generate_latent_points(latent_dim,
n_batch)


        # The generator wants the discriminator to label the
generated samples
    # as valid (ones)
    #This is where the generator is trying to trick
discriminator into believing
    #the generated image is true (hence value of 1 for y)

                y_gan = ones((n_batch, 1))
```

```
        # Generator is part of combined model where it got
directly linked with the discriminator
    # Train the generator with latent_dim as x and 1 as y.
    # Again, 1 as the output as it is adversarial and if
generator did a great
    #job of folling the discriminator then the output would
be 1 (true)
                # update the generator via the discriminator's
error
                g_loss = gan_model.train_on_batch(X_gan,
y_gan)


        # Print losses on this batch
                print('Epoch>%d, Batch %d/%d, d1=%.3f,
d2=%.3f g=%.3f' %
                        (i+1, j+1, bat_per_epo, d_loss_real,
d_loss_fake, g_loss))
    # save the generator model
    g_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/gancar256.h5')


##################################################################
#################
#Train the GAN
```

```
# size of the latent space
latent_dim = 100
# create the discriminator
discriminator = define_discriminator()
# create the generator
generator = define_generator(latent_dim)
# create the gan
gan_model = define_gan(generator, discriminator)
# load image data
dataset = load_real_samples()
# train model



train(generator, discriminator, gan_model, dataset,
latent_dim, n_epochs=300)

##################################################
################################

# Now, let us load the generator model and generate images
```

```
from keras.models import load_model
from numpy.random import randn



# Plot generated images
def show_plot(examples, n):
	for i in range(n * n):
		plt.subplot(n, n, 1 + i)
		plt.axis('off')
		plt.imshow(examples[i, :, :, :])
	plt.show()


# load model
model = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/gancar2.h5') #Model trained for
100 epochs


#model.compile(optimizer=Adam(learning_rate=0.001),loss='
categorical_crossentropy',metrics=['accuracy'])


#model = load_model('E:/fffiiles/college pracs and
projects/Project/Car/Code/gancar.h5') #Model trained for
100 epochs
```

```
# generate images
latent_points = generate_latent_points(100, 6)  #Latent dim and n_samples
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0

import numpy as np
X = (X*255).astype(np.uint8)

# plot the result
show_plot(X, 5)
```

Pix2Pix

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep  6 20:52:53 2022

@author: LAXMINARAYANRO
"""

from keras.utils import normalize
import os
import glob
import cv2
import numpy as np
from matplotlib import pyplot


#Resizing images, if needed
SIZE_X = 256
SIZE_Y = 256
#n_classes=4 #Number of classes for segmentation

#Capture training image info as a list
tar_images = []
```

```
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Createtry/Gantrain/Set1"):
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, 1)
        img = cv2.resize(img, (SIZE_Y, SIZE_X))
        tar_images.append(img)

#Convert list to array for machine learning processing
tar_images = np.array(tar_images)

#Capture mask/label info as a list
src_images = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Createtry/Gantrain/Set2"):
    for mask_path in glob.glob(os.path.join(directory_path,
"*.png")):
        mask = cv2.imread(mask_path, 1)
        mask = cv2.resize(mask, (SIZE_Y, SIZE_X), interpolation =
cv2.INTER_NEAREST)  #Otherwise ground truth changes due
to interpolation
        src_images.append(mask)
```

```
#Convert list to array for machine learning processing
src_images = np.array(src_images)

print(np.unique(src_images))



n_samples = 3
for i in range(n_samples):
	pyplot.subplot(2, n_samples, 1 + i)
	pyplot.axis('off')
	pyplot.imshow(src_images[i])
# plot target image
for i in range(n_samples):
	pyplot.subplot(2, n_samples, 1 + n_samples + i)
	pyplot.axis('off')
	pyplot.imshow(tar_images[i])
pyplot.show()

##################################################
#####
```

```
import os
print(os.listdir("E:/fffiiles/college pracs and projects/Project/Car/Code"))
import Pix2PixModel
os.system('Pix2PixModel.py')

from numpy import zeros
from numpy import ones
from numpy.random import randint
from keras.optimizers import Adam
from keras.initializers import RandomNormal
# from keras.models import Model
# from keras.models import Input
# from keras.layers import Conv2D
# from keras.layers import Conv2DTranspose
# from keras.layers import LeakyReLU
# from keras.layers import Activation
# from keras.layers import Concatenate
# from keras.layers import Dropout
# from keras.layers import BatchNormalization
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (
```

```
    BatchNormalization,LeakyReLU,
SeparableConv2D,Embedding,Concatenate, MaxPooling2D,
Activation, Flatten, Dropout, Conv2D,
Dense,Conv2DTranspose,Input, Dense, Lambda, Reshape
)
from matplotlib import pyplot as plt
from tensorflow.keras.utils import plot_model

def define_discriminator(image_shape):

        # weight initialization
        init = RandomNormal(stddev=0.02) #As described in the
original paper

        # source image input
        in_src_image = Input(shape=image_shape)  #Image we
want to convert to another image
        # target image input
        in_target_image = Input(shape=image_shape)  #Image
we want to generate after training.

        # concatenate images, channel-wise
        merged = Concatenate()([in_src_image,
in_target_image])
```

```
        # C64: 4x4 kernel Stride 2x2
        d = Conv2D(64, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(merged)
        d = LeakyReLU(alpha=0.2)(d)
        # C128: 4x4 kernel Stride 2x2
        d = Conv2D(128, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(d)
        d = BatchNormalization()(d)
        d = LeakyReLU(alpha=0.2)(d)
        # C256: 4x4 kernel Stride 2x2
        d = Conv2D(256, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(d)
        d = BatchNormalization()(d)
        d = LeakyReLU(alpha=0.2)(d)
        # C512: 4x4 kernel Stride 2x2
    # Not in the original paper. Comment this block if you
want.
        d = Conv2D(512, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(d)
        d = BatchNormalization()(d)
        d = LeakyReLU(alpha=0.2)(d)
        # second last output layer : 4x4 kernel but Stride 1x1
```

```
    d = Conv2D(512, (4,4), padding='same',
kernel_initializer=init)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU(alpha=0.2)(d)
    # patch output
    d = Conv2D(1, (4,4), padding='same',
kernel_initializer=init)(d)
    patch_out = Activation('sigmoid')(d)
    # define model
    model = Model([in_src_image, in_target_image],
patch_out)
    # compile model
  #The model is trained with a batch size of one image and
Adam opt.
  #with a small learning rate and 0.5 beta.
  #The loss for the discriminator is weighted by 50% for each
model update.

    opt = Adam(learning_rate=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy',
optimizer=opt, loss_weights=[0.5])
    return model
```

```
# disc_model = define_discriminator((256,256,3))
# plot_model(disc_model, to_file='disc_model.png',
show_shapes=True)


################################
#Now define the generator - in our case we will define a U-
net
# define an encoder block to be used in generator
def define_encoder_block(layer_in, n_filters,
batchnorm=True):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # add downsampling layer
    g = Conv2D(n_filters, (4,4), strides=(2,2),
padding='same', kernel_initializer=init)(layer_in)
    # conditionally add batch normalization
    if batchnorm:
        g = BatchNormalization()(g, training=True)
    # leaky relu activation
    g = LeakyReLU(alpha=0.2)(g)
    return g


# define a decoder block to be used in generator
```

```
def decoder_block(layer_in, skip_in, n_filters, dropout=True):
	# weight initialization
	init = RandomNormal(stddev=0.02)
	# add upsampling layer
	g = Conv2DTranspose(n_filters, (4,4), strides=(2,2),
padding='same', kernel_initializer=init)(layer_in)
	# add batch normalization
	g = BatchNormalization()(g, training=True)
	# conditionally add dropout
	if dropout:
		g = Dropout(0.5)(g, training=True)
	# merge with skip connection
	g = Concatenate()([g, skip_in])
	# relu activation
	g = Activation('relu')(g)
	return g

# define the standalone generator model - U-net
def define_generator(image_shape=(256,256,3)):
	# weight initialization
	init = RandomNormal(stddev=0.02)
	# image input
```

```
in_image = Input(shape=image_shape)
# encoder model: C64-C128-C256-C512-C512-C512-C512-C512
e1 = define_encoder_block(in_image, 64, batchnorm=False)
e2 = define_encoder_block(e1, 128)
e3 = define_encoder_block(e2, 256)
e4 = define_encoder_block(e3, 512)
e5 = define_encoder_block(e4, 512)
e6 = define_encoder_block(e5, 512)
e7 = define_encoder_block(e6, 512)
# bottleneck, no batch norm and relu
b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
b = Activation('relu')(b)
# decoder model: CD512-CD512-CD512-C512-C256-C128-C64
d1 = decoder_block(b, e7, 512)
d2 = decoder_block(d1, e6, 512)
d3 = decoder_block(d2, e5, 512)
d4 = decoder_block(d3, e4, 512, dropout=False)
d5 = decoder_block(d4, e3, 256, dropout=False)
d6 = decoder_block(d5, e2, 128, dropout=False)
```

```
    d7 = decoder_block(d6, e1, 64, dropout=False)
    # output
    g = Conv2DTranspose(image_shape[2], (4,4),
strides=(2,2), padding='same', kernel_initializer=init)(d7)
#Modified
    out_image = Activation('tanh')(g)  #Generates images in
the range -1 to 1. So change inputs also to -1 to 1
    # define model
    model = Model(in_image, out_image)
    return model


# gen_model = define_generator((256,256,3))
# plot_model(gen_model, to_file='gen_model.png',
show_shapes=True)



# define the combined generator and discriminator model,
for updating the generator
def define_gan(g_model, d_model, image_shape):
    # make weights in the discriminator not trainable
    for layer in d_model.layers:
        if not isinstance(layer, BatchNormalization):
```

```
                    layer.trainable = False      #Descriminator
layers set to untrainable in the combined GAN but
                                #standalone descriminator will be
trainable.

    # define the source image
    in_src = Input(shape=image_shape)
    # suppy the image as input to the generator
    gen_out = g_model(in_src)
    # supply the input image and generated image as inputs
to the discriminator
    dis_out = d_model([in_src, gen_out])
    # src image as input, generated image and disc. output
as outputs
    model = Model(in_src, [dis_out, gen_out])
    # compile model
    opt = Adam(learning_rate=0.0002, beta_1=0.5)

  #Total loss is the weighted sum of adversarial loss (BCE)
and L1 loss (MAE)
  #Authors suggested weighting BCE vs L1 as 1:100.
    model.compile(loss=['binary_crossentropy', 'mae'],
        optimizer=opt, loss_weights=[1,100])
```

```
    return model

# select a batch of random samples, returns images and target
def generate_real_samples(dataset, n_samples, patch_shape):
    # unpack dataset
    trainA, trainB = dataset
    # choose random instances
    ix = randint(0, trainA.shape[0], n_samples)
    # retrieve selected images
    X1, X2 = trainA[ix], trainB[ix]
    # generate 'real' class labels (1)
    y = ones((n_samples, patch_shape, patch_shape, 1))
    return [X1, X2], y

# generate a batch of images, returns images and targets
def generate_fake_samples(g_model, samples, patch_shape):
    # generate fake instance
    X = g_model.predict(samples)
    # create 'fake' class labels (0)
    y = zeros((len(X), patch_shape, patch_shape, 1))
```

```
    return X, y

# generate samples and save as a plot and save the model
#GAN models do not converge, we just want to find a good balance between
#the generator and the discriminator. Therefore, it makes sense to periodically
#save the generator model and check how good the generated image looks.
def summarize_performance(step, g_model, dataset, n_samples=3):
    # select a sample of input images
    [X_realA, X_realB], _ = generate_real_samples(dataset, n_samples, 1)
    # generate a batch of fake samples
    X_fakeB, _ = generate_fake_samples(g_model, X_realA, 1)
    # scale all pixels from [-1,1] to [0,1]
    X_realA = (X_realA + 1) / 2.0
    X_realB = (X_realB + 1) / 2.0
    X_fakeB = (X_fakeB + 1) / 2.0
    # plot real source images
    for i in range(n_samples):
```

```
        plt.subplot(3, n_samples, 1 + i)
        plt.axis('off')
        plt.imshow(X_realA[i])
    # plot generated target image
    for i in range(n_samples):
        plt.subplot(3, n_samples, 1 + n_samples + i)
        plt.axis('off')
        plt.imshow(X_fakeB[i])
    # plot real target image
    for i in range(n_samples):
        plt.subplot(3, n_samples, 1 + n_samples*2 + i)
        plt.axis('off')
        plt.imshow(X_realB[i])
    # save plot to file
    filename1 = 'E:/fffiiles/college pracs and
projects/Project/Car/pix2piximage_%06d.png' % (step+1)
    plt.savefig(filename1)
    plt.close()
    # save the generator model
    filename2 = 'E:/fffiiles/college pracs and
projects/Project/Car/modelpix2pix_%06d.h5' % (step+1)
    g_model.save(filename2)
```

```
    print('>Saved: %s and %s' % (filename1, filename2))


# train pix2pix models
def train(d_model, g_model, gan_model, dataset, n_epochs=100, n_batch=1):
    # determine the output square shape of the discriminator
    n_patch = d_model.output_shape[1]
    # unpack dataset
    trainA, trainB = dataset
    # calculate the number of batches per training epoch
    bat_per_epo = int(len(trainA) / n_batch)
    # calculate the number of training iterations
    n_steps = bat_per_epo * n_epochs
    # manually enumerate epochs
    for i in range(n_steps):
        # select a batch of real samples
        [X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)
        # generate a batch of fake samples
        X_fakeB, y_fake = generate_fake_samples(g_model, X_realA, n_patch)
        # update discriminator for real samples
```

```
            d_loss1 = d_model.train_on_batch([X_realA,
X_realB], y_real)
            # update discriminator for generated samples
            d_loss2 = d_model.train_on_batch([X_realA,
X_fakeB], y_fake)
            # update the generator
            g_loss, _, _ = gan_model.train_on_batch(X_realA,
[y_real, X_realB])
            # summarize performance
            print('>%d, d1[%.3f] d2[%.3f] g[%.3f]' % (i+1,
d_loss1, d_loss2, g_loss))
            # summarize model performance
            if (i+1) % (bat_per_epo * 10) == 0:
                summarize_performance(i, g_model, dataset)


from Pix2PixModel import define_discriminator,
define_generator, define_gan, train
# define input shape based on the loaded dataset
image_shape = src_images.shape[1:]
# define the models
d_model = define_discriminator(image_shape)
g_model = define_generator(image_shape)
# define the composite model
```

```
gan_model = define_gan(g_model, d_model, image_shape)

#Define data
# load and prepare training images
data = [src_images, tar_images]

#Preprocess data to change input range to values between -1 and 1
# This is because the generator uses tanh activation in the output layer
#And tanh ranges between -1 and 1
def preprocess_data(data):
    # load compressed arrays
    # unpack arrays
    X1, X2 = data[0], data[1]
    # scale from [0,255] to [-1,1]
    X1 = (X1 - 127.5) / 127.5
    X2 = (X2 - 127.5) / 127.5
    return [X1, X2]

dataset = preprocess_data(data)
```

```
from datetime import datetime
start1 = datetime.now()

train(d_model, g_model, gan_model, dataset, n_epochs=2, n_batch=1)
#Reports parameters for each batch (total 1600) for each epoch.
#For 10 epochs we should see 16000

stop1 = datetime.now()
#Execution time of the model
execution_time = stop1-start1
print("Execution time is: ", execution_time)
```

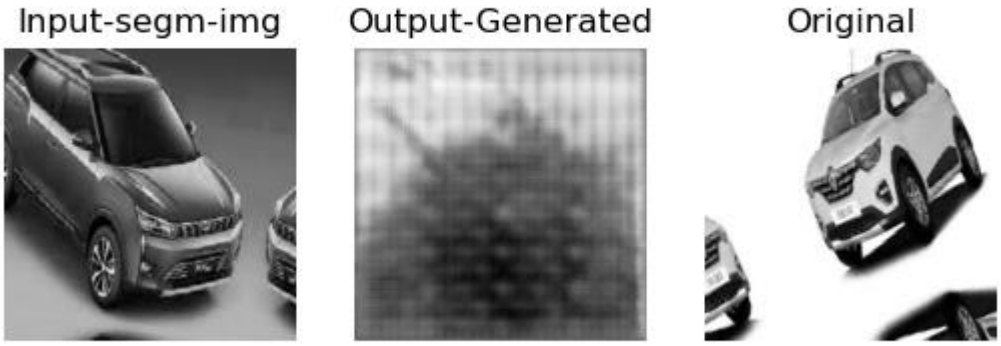
```
>1733, d1[0.000] d2[0.011] g[61.088]
1/1 [==============================] - 0s 307ms/step
>1734, d1[0.001] d2[0.005] g[46.674]
1/1 [==============================] - 0s 315ms/step
>1735, d1[0.001] d2[0.007] g[54.392]
1/1 [==============================] - 0s 354ms/step
>1736, d1[0.002] d2[0.008] g[46.062]
Execution time is:  1:15:04.397521
```

```
#Reports parameters for each batch (total 1096) for each epoch.
```

```
#For 10 epochs we should see 10960
g_model.save('E:/fffiiles/college pracs and projects/Project/Car/savedpix2pixmodel.h5')
##########################################
#Test trained model on a few images...



from keras.models import load_model
from numpy.random import randint
from numpy import vstack
model = load_model('E:/fffiiles/college pracs and projects/Project/Car/savedpix2pixmodel.h5')

# plot source, generated and target images
def plot_images(src_img, gen_img, tar_img):
    images = vstack((src_img, gen_img, tar_img))
    # scale from [-1,1] to [0,1]
    images = (images + 1) / 2.0
    titles = ['Input-segm-img', 'Output-Generated', 'Original']
    # plot images row by row
    for i in range(len(images)):
        # define subplot
```

```
        pyplot.subplot(1, 3, 1 + i)
        # turn off axis
        pyplot.axis('off')
        # plot raw pixel data
        pyplot.imshow(images[i,:,:,0], cmap='gray')
        # show title
        pyplot.title(titles[i])
    pyplot.show()
```

```
[X1, X2] = dataset
# select random example
ix = randint(0, len(X1), 1)
src_image, tar_image = X1[ix], X2[ix]
# generate image from source
gen_image = model.predict(src_image)
# plot all three images
plot_images(src_image, gen_image, tar_image)
```



Conditional Gan

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep  6 13:03:36 2022

@author: LAXMINARAYANRO
"""

from numpy import zeros
from numpy import ones
from numpy.random import randn
```

```
from numpy.random import randint
from tensorflow.keras.optimizers import Adam
#from keras.models import Sequential
from tensorflow.keras.models import Model, Sequential
# from keras.layers import Dense
# from keras.layers import Reshape
# from keras.layers import Flatten
# from keras.layers import Conv2D
# from keras.layers import Conv2DTranspose
# from keras.layers import LeakyReLU
# from keras.layers import Dropout
from tensorflow.keras.layers import (
    LeakyReLU, SeparableConv2D,Embedding,Concatenate,
MaxPooling2D, Activation, Flatten, Dropout, Conv2D,
Dense,Conv2DTranspose,Input, Dense, Lambda, Reshape
)
import numpy as np
import glob
import cv2
import os

from matplotlib import pyplot as plt
```

```
##################################################
#######################

#Load data and plot to get a quick understanding
#CIFAR10 classes are: airplane, automobile, bird, cat, deer,
dog, frog, horse,
# ship, truck

SIZE = 32
#Channel=3  #Resize images

#lists
train_images = []
train_labels = []
#E:/fffiiles/college pracs and
projects/Project/Car/Model_images/train/*
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Createtry/Gantrain/*"):
    label = directory_path.split("\\")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        print(img_path)
```

```
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)

#lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)



# test/validation into lists
#E:/fffiiles/college pracs and
projects/Project/Car/Model_images/test/*
test_images = []
test_labels = []
for directory_path in glob.glob("E:/fffiiles/college pracs and
projects/Project/Car/Createtry/Gantest/*"):
    fruit_label = directory_path.split("\\")[-1]
    for img_path in glob.glob(os.path.join(directory_path,
"*.png")):
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
```

```
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    test_images.append(img)
    test_labels.append(fruit_label)

#lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)

trainX, trainy, testX, testy = train_images,
train_labels_encoded, test_images, test_labels_encoded
testy = np.asarray(testy).reshape((-1,1))
trainy = np.asarray(trainy).reshape((-1,1))
testy = testy.astype(np.uint8)
trainy = trainy.astype(np.uint8)
```

```
# plot 25 images
for i in range(25):
    plt.subplot(5, 5, 1 + i)
    plt.axis('off')
    plt.imshow(trainX[i])
plt.show()
```

```
##################################################################################
#Define generator, discriminator, gan and other helper functions
#We will use functional way of defining model as we have multiple inputs;
#both images and corresponding labels.
##################################################################################

# define the standalone discriminator model
#Given an input image, the Discriminator outputs the likelihood of the image being real.
#Binary classification - true or false (1 or 0). So using sigmoid activation.
```

```
#Unlike regular GAN here we are also providing number of classes as input.
#Input to the model will be both images and labels.
def define_discriminator(in_shape=(32,32,3), n_classes=2):

    # label input
        in_label = Input(shape=(1,))  #Shape 1
        # embedding for categorical input
    #each label (total 10 classes for cifar), will be represented by a vector of size 50.
    #This vector of size 50 will be learnt by the discriminator
        li = Embedding(n_classes, 50)(in_label) #Shape 1,50
        # scale up to image dimensions with linear activation
        n_nodes = in_shape[0] * in_shape[1]  #32x32 = 1024.
        li = Dense(n_nodes)(li)  #Shape = 1, 1024
        # reshape to additional channel
        li = Reshape((in_shape[0], in_shape[1], 1))(li)  #32x32x1



        # image input
        in_image = Input(shape=in_shape) #32x32x3
```

```
    # concat label as a channel
    merge = Concatenate()([in_image, li]) #32x32x4 (4 channels, 3 for image and the other for labels)


    # downsample: This part is same as unconditional GAN upto the output layer.
   #We will combine input label with input image and supply as inputs to the model.
    fe = Conv2D(128, (3,3), strides=(2,2), padding='same')(merge) #16x16x128
    fe = LeakyReLU(alpha=0.2)(fe)
    # downsample
    fe = Conv2D(128, (3,3), strides=(2,2), padding='same')(fe) #8x8x128
    fe = LeakyReLU(alpha=0.2)(fe)
    # flatten feature maps
    fe = Flatten()(fe)  #8192  (8*8*128=8192)
    # dropout
    fe = Dropout(0.4)(fe)
    # output
    out_layer = Dense(1, activation='sigmoid')(fe)  #Shape=1


    # define model
```

```
    ##Combine input label with input image and supply as
inputs to the model.
        model = Model([in_image, in_label], out_layer)
        # compile model
        opt = Adam(learning_rate=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy',
optimizer=opt, metrics=['accuracy'])
        return model

test_discr = define_discriminator()
print(test_discr.summary())



# define the standalone generator model
#latent vector and label as inputs

def define_generator(latent_dim, n_classes=2):

        # label input
        in_label = Input(shape=(1,))  #Input of dimension 1
        # embedding for categorical input
    #each label (total 10 classes for cifar), will be represented
by a vector of size 50.
```

```
li = Embedding(n_classes, 50)(in_label) #Shape 1,50

# linear multiplication
n_nodes = 8 * 8  # To match the dimensions for concatenation later in this step.
li = Dense(n_nodes)(li) #1,64
# reshape to additional channel
li = Reshape((8, 8, 1))(li)


# image generator input
in_lat = Input(shape=(latent_dim,))  #Input of dimension 100

# foundation for 8x8 image
# We will reshape input latent vector into 8x8 image as a starting point.
#So n_nodes for the Dense layer can be 128x8x8 so when we reshape the output
#it would be 8x8x128 and that can be slowly upscaled to 32x32 image for output.
#Note that this part is same as unconditional GAN until the output layer.
```

```
#While defining model inputs we will combine input label and the latent input.
n_nodes = 128 * 8 * 8
gen = Dense(n_nodes)(in_lat)  #shape=8192
gen = LeakyReLU(alpha=0.2)(gen)
gen = Reshape((8, 8, 128))(gen) #Shape=8x8x128
# merge image gen and label input
merge = Concatenate()([gen, li])  #Shape=8x8x129 (Extra channel corresponds to the label)
# upsample to 16x16
gen = Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')(merge) #16x16x128
gen = LeakyReLU(alpha=0.2)(gen)
# upsample to 32x32
gen = Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')(gen) #32x32x128
gen = LeakyReLU(alpha=0.2)(gen)
# output
out_layer = Conv2D(3, (8,8), activation='tanh', padding='same')(gen) #32x32x3
# define model
model = Model([in_lat, in_label], out_layer)
```

```
    return model   #Model not compiled as it is not directly trained like the discriminator.


test_gen = define_generator(100, n_classes=2)
print(test_gen.summary())



# #Generator is trained via GAN combined model.
# define the combined generator and discriminator model, for updating the generator
#Discriminator is trained separately so here only generator will be trained by keeping
#the discriminator constant.
def define_gan(g_model, d_model):
    d_model.trainable = False  #Discriminator is trained separately. So set to not trainable.


  ## connect generator and discriminator...
    # first, get noise and label inputs from generator model
    gen_noise, gen_label = g_model.input  #Latent vector size and label size
    # get image output from the generator model
    gen_output = g_model.output  #32x32x3
```

```
    # generator image output and corresponding input label
are inputs to discriminator
    gan_output = d_model([gen_output, gen_label])
    # define gan model as taking noise and label and
outputting a classification
    model = Model([gen_noise, gen_label], gan_output)
    # compile model
    opt = Adam(learning_rate=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy',
optimizer=opt)
    return model

# load cifar images
def load_real_samples():
    # load dataset
      #cifar
    # convert to floats and scale
    X = trainX.astype('float32')
    # scale from [0,255] to [-1,1]
    X = (X - 127.5) / 127.5   #Generator uses tanh activation
so rescale
```

```
                    #original images to -1 to 1 to match the output of generator.
    return [X, trainy]


# # select real samples
# pick a batch of random real samples to train the GAN
#In fact, we will train the GAN on a half batch of real images and another
#half batch of fake images.
#For each real image we assign a label 1 and for fake we assign label 0.
def generate_real_samples(dataset, n_samples):
    # split into images and labels
    images, labels = dataset
    # choose random instances
    ix = randint(0, images.shape[0], n_samples)
    # select images and labels
    X, labels = images[ix], labels[ix]
    # generate class labels and assign to y (don't confuse this with the above labels that correspond to cifar labels)
    y = ones((n_samples, 1))  #Label=1 indicating they are real
    return [X, labels], y
```

```
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples, n_classes=2):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    z_input = x_input.reshape(n_samples, latent_dim)
    # generate labels
    labels = randint(0, n_classes, n_samples)
    return [z_input, labels]

# use the generator to generate n fake examples, with class labels
#Supply the generator, latent_dim and number of samples as input.
#Use the above latent point generator to generate latent points.
def generate_fake_samples(generator, latent_dim, n_samples):
    # generate points in latent space
    z_input, labels_input = generate_latent_points(latent_dim, n_samples)
```

```
    # predict outputs
    images = generator.predict([z_input, labels_input])
    # create class labels
    y = zeros((n_samples, 1))  #Label=0 indicating they are fake
    return [images, labels_input], y


# train the generator and discriminator
#We loop through a number of epochs to train our Discriminator by first selecting
#a random batch of images from our true/real dataset.
#Then, generating a set of images using the generator.
#Feed both set of images into the Discriminator.
#Finally, set the loss parameters for both the real and fake images, as well as the combined loss.
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=100, n_batch=128):
    bat_per_epo = int(dataset[0].shape[0] / n_batch)
    half_batch = int(n_batch / 2)  #the discriminator model is updated for a half batch of real samples
                        #and a half batch of fake samples, combined a single batch.
    # manually enumerate epochs
```

```
for i in range(n_epochs):
        # enumerate batches over the training set
        for j in range(bat_per_epo):


         # Train the discriminator on real and fake images, separately (half batch each)
      #Research showed that separate training is more effective.
                # get randomly selected 'real' samples
        # get randomly selected 'real' samples
                [X_real, labels_real], y_real = generate_real_samples(dataset, half_batch)


        # update discriminator model weights
        ##train_on_batch allows you to update weights based on a collection
        #of samples you provide
                d_loss_real, _ = d_model.train_on_batch([X_real, labels_real], y_real)


                # generate 'fake' examples
                [X_fake, labels], y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
```

```
# update discriminator model weights
d_loss_fake, _ = d_model.train_on_batch([X_fake, labels], y_fake)

#d_loss = 0.5 * np.add(d_loss_real, d_loss_fake) #Average loss if you want to report single..

# prepare points in latent space as input for the generator
[z_input, labels_input] = generate_latent_points(latent_dim, n_batch)

# The generator wants the discriminator to label the generated samples
# as valid (ones)
#This is where the generator is trying to trick discriminator into believing
#the generated image is true (hence value of 1 for y)
# create inverted labels for the fake samples
y_gan = ones((n_batch, 1))
# Generator is part of combined model where it got directly linked with the discriminator
# Train the generator with latent_dim as x and 1 as y.
```

```
        # Again, 1 as the output as it is adversarial and if
generator did a great
        #job of folling the discriminator then the output would
be 1 (true)
                        # update the generator via the discriminator's
error
                        g_loss = gan_model.train_on_batch([z_input,
labels_input], y_gan)
                        # Print losses on this batch
                        print('Epoch>%d, Batch%d/%d, d1=%.3f,
d2=%.3f g=%.3f' %
                                (i+1, j+1, bat_per_epo, d_loss_real,
d_loss_fake, g_loss))
        # save the generator model
        g_model.save('E:/fffiiles/college pracs and
projects/Project/Car/Code/condgancar256.h5')


#Train the GAN


# size of the latent space
latent_dim = 100
# create the discriminator
d_model = define_discriminator()
# create the generator
```

```
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
# train model
train(g_model, d_model, gan_model, dataset, latent_dim,
n_epochs=50)

##################################################
########
# Now, let us load the generator model and generate images
# Lod the trained model and generate a few images
from numpy import asarray
from numpy.random import randn
from numpy.random import randint
from keras.models import load_model
import numpy as np
#

#Note: CIFAR10 classes are: airplane, automobile, bird, cat,
deer, dog, frog, horse,
# ship, truck
```

```
# load model
model = load_model('E:/fffiiles/college pracs and projects/Project/Car/Code/condgancar256.h5')


# generate multiple images


latent_points, labels = generate_latent_points(100, 4)
# specify labels - generate 10 sets of labels each gping from 0 to 9
labels = asarray([x for _ in range(2) for x in range(2)])
# generate images
X  = model.predict([latent_points, labels])
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
X = (X*255).astype(np.uint8)
# plot the result (10 sets of images, all images in a column should be of same class in the plot)
# Plot generated images
def show_plot(examples, n):
    for i in range(n * n):
        plt.subplot(n, n, 1 + i)
        plt.axis('off')
```

```
            plt.imshow(examples[i, :, :, :])
        plt.show()
```

---

show_plot(X, 4)

SRGAN

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep  9 17:19:54 2022

@author: LAXMINARAYANRO
"""

import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

from keras import layers, Model
from sklearn.model_selection import train_test_split
```

```
import numpy as np
from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import (
    LeakyReLU,PReLU,BatchNormalization,add,UpSampling2D,
SeparableConv2D, MaxPooling2D, Activation, Flatten,
Dropout, Conv2D, Dense,Conv2DTranspose,Input, Dense,
Lambda, Reshape
)
from tqdm import tqdm

##################################################
#######################

#Define blocks to build the generator
def res_block(ip):

    res_model = Conv2D(64, (3,3), padding = "same")(ip)
    res_model = BatchNormalization(momentum =
0.5)(res_model)
    res_model = PReLU(shared_axes = [1,2])(res_model)
```

```
    res_model = Conv2D(64, (3,3), padding =
"same")(res_model)
    res_model = BatchNormalization(momentum =
0.5)(res_model)

    return add([ip,res_model])

def upscale_block(ip):

    up_model = Conv2D(256, (3,3), padding="same")(ip)
    up_model = UpSampling2D( size = 2 )(up_model)
    up_model = PReLU(shared_axes=[1,2])(up_model)

    return up_model

#Generator model
def create_gen(gen_ip, num_res_block):
    layers = Conv2D(64, (9,9), padding="same")(gen_ip)
    layers = PReLU(shared_axes=[1,2])(layers)

    temp = layers
```

```
    for i in range(num_res_block):
        layers = res_block(layers)

    layers = Conv2D(64, (3,3), padding="same")(layers)
    layers = BatchNormalization(momentum=0.5)(layers)
    layers = add([layers,temp])

    layers = upscale_block(layers)
    layers = upscale_block(layers)

    op = Conv2D(3, (9,9), padding="same")(layers)

    return Model(inputs=gen_ip, outputs=op)

#Descriminator block that will be used to construct the
discriminator
def discriminator_block(ip, filters, strides=1, bn=True):

    disc_model = Conv2D(filters, (3,3), strides = strides,
padding="same")(ip)

    if bn:
```

```
        disc_model = BatchNormalization( momentum=0.8
)(disc_model)

    disc_model = LeakyReLU( alpha=0.2 )(disc_model)

    return disc_model



#Descriminartor, as described in the original paper
def create_disc(disc_ip):

    df = 64

    d1 = discriminator_block(disc_ip, df, bn=False)
    d2 = discriminator_block(d1, df, strides=2)
    d3 = discriminator_block(d2, df*2)
    d4 = discriminator_block(d3, df*2, strides=2)
    d5 = discriminator_block(d4, df*4)
    d6 = discriminator_block(d5, df*4, strides=2)
    d7 = discriminator_block(d6, df*8)
    d8 = discriminator_block(d7, df*8, strides=2)
```

```
    d8_5 = Flatten()(d8)
    d9 = Dense(df*16)(d8_5)
    d10 = LeakyReLU(alpha=0.2)(d9)
    validity = Dense(1, activation='sigmoid')(d10)

    return Model(disc_ip, validity)


#VGG19
#We need VGG19 for the feature map obtained by the j-th convolution (after activation)
#before the i-th maxpooling layer within the VGG19 network.(as described in the paper)
#Let us pick the 3rd block, last conv layer.
#Build a pre-trained VGG19 model that outputs image features extracted at the
# third block of the model
# VGG architecture: https://github.com/keras-team/keras/blob/master/keras/applications/vgg19.py
from tensorflow.keras.applications.vgg19 import VGG19

def build_vgg(hr_shape):
```

```
    vgg = VGG19(weights="imagenet",include_top=False,
input_shape=hr_shape)

    return Model(inputs=vgg.inputs,
outputs=vgg.layers[10].output)

#Combined model
def create_comb(gen_model, disc_model, vgg, lr_ip, hr_ip):
    gen_img = gen_model(lr_ip)

    gen_features = vgg(gen_img)

    disc_model.trainable = False
    validity = disc_model(gen_img)

    return Model(inputs=[lr_ip, hr_ip], outputs=[validity,
gen_features])

# 2 losses... adversarial loss and content (VGG) loss
#AdversariaL: is defined based on the probabilities of the
discriminator over all training samples
# use binary_crossentropy
```

```
#Content: feature map obtained by the j-th convolution
(after activation)

#before the i-th maxpooling layer within the VGG19 network.

# MSE between the feature representations of a
reconstructed image

# and the reference image.


##################################################
##################################


#Load first n number of images (to train on a subset of all
images)

#For demo purposes, let us use 5000 images

n=150

lr_list = os.listdir("E:/fffiiles/college pracs and
projects/Project/Car/Createtry/lr_image")[:n]


lr_images = []

for img in lr_list:

    img_lr = cv2.imread("E:/fffiiles/college pracs and
projects/Project/Car/Createtry/lr_image/" + img)

    img_lr = cv2.cvtColor(img_lr, cv2.COLOR_BGR2RGB)

    lr_images.append(img_lr)
```

```
hr_list = os.listdir("E:/fffiiles/college pracs and
projects/Project/Car/Createtry/hr_image")[:n]

hr_images = []
for img in hr_list:
    img_hr = cv2.imread("E:/fffiiles/college pracs and
projects/Project/Car/Createtry/hr_image/" + img)
    img_hr = cv2.cvtColor(img_hr, cv2.COLOR_BGR2RGB)
    hr_images.append(img_hr)

lr_images = np.array(lr_images)
hr_images = np.array(hr_images)

#Sanity check, view few mages
import random
import numpy as np
image_number = random.randint(0, len(lr_images)-1)
plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.imshow(np.reshape(lr_images[image_number], (32, 32,
3)))
```

```
plt.subplot(122)
plt.imshow(np.reshape(hr_images[image_number], (128,
128, 3)))
plt.show()

#Scale values
lr_images = lr_images / 255.
hr_images = hr_images / 255.

#Split to train and test
lr_train, lr_test, hr_train, hr_test = train_test_split(lr_images,
hr_images,
                                        test_size=0.33,
random_state=42)



hr_shape = (hr_train.shape[1], hr_train.shape[2],
hr_train.shape[3])
lr_shape = (lr_train.shape[1], lr_train.shape[2],
lr_train.shape[3])

lr_ip = Input(shape=lr_shape)
```

```
hr_ip = Input(shape=hr_shape)

generator = create_gen(lr_ip, num_res_block = 16)
generator.summary()

discriminator = create_disc(hr_ip)
discriminator.compile(loss="binary_crossentropy",
optimizer="adam", metrics=['accuracy'])
discriminator.summary()

vgg = build_vgg((128,128,3))
print(vgg.summary())
vgg.trainable = False

gan_model = create_comb(generator, discriminator, vgg,
lr_ip, hr_ip)

# 2 losses... adversarial loss and content (VGG) loss
#AdversariaL: is defined based on the probabilities of the
discriminator over all training samples
# use binary_crossentropy
```

```
#Content: feature map obtained by the j-th convolution
(after activation)

#before the i-th maxpooling layer within the VGG19 network.

# MSE between the feature representations of a
reconstructed image

# and the reference image.

gan_model.compile(loss=["binary_crossentropy", "mse"],
loss_weights=[1e-3, 1], optimizer="adam")

gan_model.summary()


#Create a list of images for LR and HR in batches from which
a batch of images

#would be fetched during training.

batch_size = 1

train_lr_batches = []

train_hr_batches = []

for it in range(int(hr_train.shape[0] / batch_size)):

    start_idx = it * batch_size

    end_idx = start_idx + batch_size

    train_hr_batches.append(hr_train[start_idx:end_idx])

    train_lr_batches.append(lr_train[start_idx:end_idx])
```

```
epochs = 5
#Enumerate training over epochs
for e in range(epochs):

    fake_label = np.zeros((batch_size, 1)) # Assign a label of 0 to all fake (generated images)
    real_label = np.ones((batch_size,1)) # Assign a label of 1 to all real images.

    #Create empty lists to populate gen and disc losses.
    g_losses = []
    d_losses = []

    #Enumerate training over batches.
    for b in tqdm(range(len(train_hr_batches))):
        lr_imgs = train_lr_batches[b] #Fetch a batch of LR images for training
        hr_imgs = train_hr_batches[b] #Fetch a batch of HR images for training

        fake_imgs = generator.predict_on_batch(lr_imgs) #Fake images
```

```
#First, train the discriminator on fake and real HR images.
discriminator.trainable = True
d_loss_gen = discriminator.train_on_batch(fake_imgs, fake_label)
d_loss_real = discriminator.train_on_batch(hr_imgs, real_label)

#Now, train the generator by fixing discriminator as non-trainable
discriminator.trainable = False

#Average the discriminator loss, just for reporting purposes.
d_loss = 0.5 * np.add(d_loss_gen, d_loss_real)

#Extract VGG features, to be used towards calculating loss
image_features = vgg.predict(hr_imgs)

#Train the generator via GAN.
#Remember that we have 2 losses, adversarial loss and content (VGG) loss
```

```
        g_loss, _, _ = gan_model.train_on_batch([lr_imgs, hr_imgs], [real_label, image_features])

        #Save losses to a list so we can average and report.
        d_losses.append(d_loss)
        g_losses.append(g_loss)

    #Convert the list of losses to an array to make it easy to average
    g_losses = np.array(g_losses)
    d_losses = np.array(d_losses)

    #Calculate the average losses for generator and discriminator
    g_loss = np.sum(g_losses, axis=0) / len(g_losses)
    d_loss = np.sum(d_losses, axis=0) / len(d_losses)

    #Report the progress during training.
    print("epoch:", e+1 ,"g_loss:", g_loss, "d_loss:", d_loss)

    if (e+1) % 5 == 0: #Change the frequency for model saving, if needed
```

```
#Save the generator after every n epochs (Usually 10 epochs)
generator.save("srgantry1.h5")
```

```
epoch: 4 g_loss: 70.81277332305908 d_loss: [1.16245927 0.94 ]
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 123ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 114ms/step
```

```
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 108ms/step
```

```
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 128ms/step
```

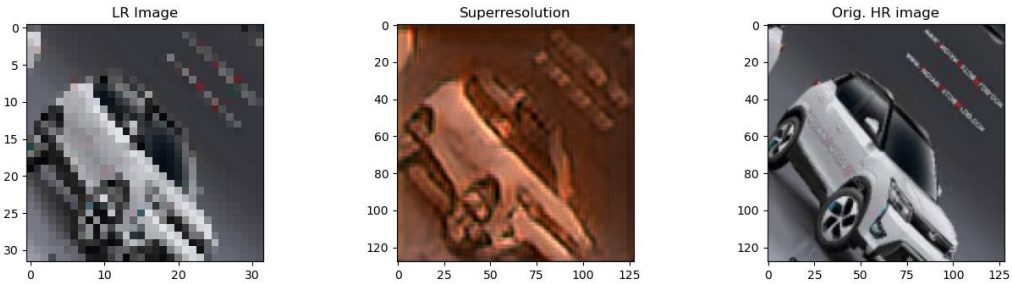
```
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 114ms/step
```

```
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 110ms/step
100%|██████████| 100/100 [06:40<00:00,  4.01s/it]
epoch: 5 g_loss: 64.3456605911255 d_loss: [1.53706785
0.915
```

```
##################################################################################
```

```
#Test - perform super resolution using saved generator model
from keras.models import load_model
from numpy.random import randint

generator = load_model('srgantry1.h5', compile=False)



[X1, X2] = [lr_test, hr_test]
# select random example
ix = randint(0, len(X1), 1)
src_image, tar_image = X1[ix], X2[ix]

# generate image from source
gen_image = generator.predict(src_image)



# plot all three images

plt.figure(figsize=(16, 8))
plt.subplot(231)
plt.title('LR Image')
```

```
plt.imshow(src_image[0,:,:,:])
plt.subplot(232)
plt.title('Superresolution')
plt.imshow(gen_image[0,:,:,:])
plt.subplot(233)
plt.title('Orig. HR image')
plt.imshow(tar_image[0,:,:,:])

plt.show()
```



------dsd

----------sd

##