Configuration Manual

MSc Research Project
MSc. Data Analytics

Aishwarya Ratnakar Shetty
Student ID: x19237740

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Aishwarya Ratnakar Shetty

Student ID: x19237740

Programme: MSc. Data Analytics **Year:** 2021-2022

Module: MSc. Research Project

Lecturer: Dr. Catherine Mulwa

Submission Due Date: 15th August 2022

Project Title: Detection of Driver Distraction Using Deep Learning

Word Count: 919 **Page Count:** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Aishwarya Ratnakar Shetty

Date: 15th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Detection of Driver Distraction Using Deep Learning

Aishwarya Ratnakar Shetty
x19237740

1 Introduction

The main objective of this research is to recognize driver distraction using transfer learning and CNN (Convolutional Neural Network). The system was created using CNN and the pretrained models ResNet50, VGG16, and VGG19. The "Detection of Driver Detection" study's setup, hardware, and software requirements are all included in this configuration file. It also details all of the steps required to complete the study's many steps.

2 System Configuration

The necessary hardware and software will be covered in this section. The sections that follow are explained below.

2.1 Hardware

Below is the hardware configuration that is required.

Table 1: Hardware Config

Hardware	Configuration
Processor	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
Installed RAM	8.00 GB (7.35 GB usable)
System type	64-bit operating system, x64-based processor

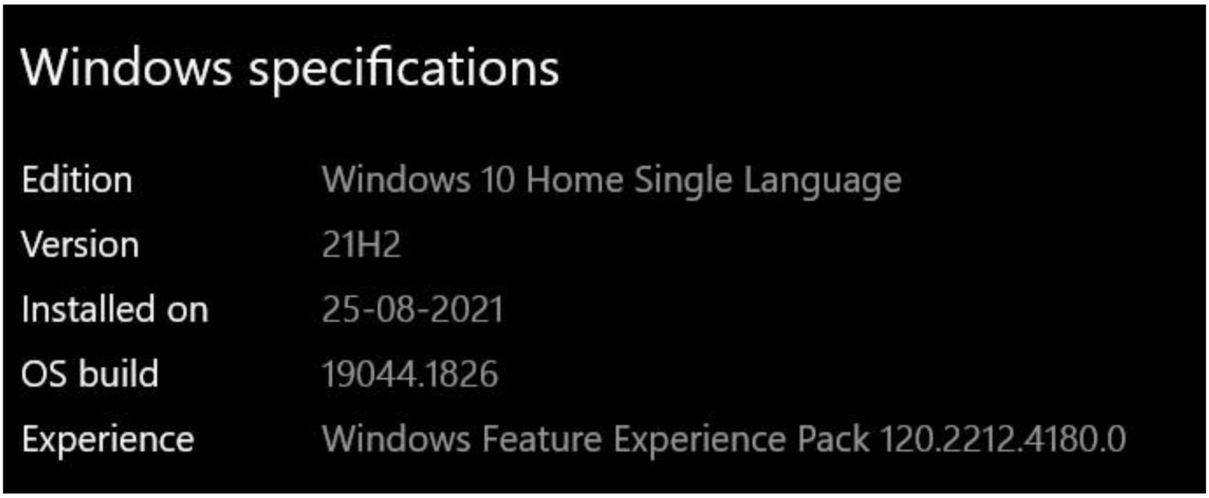


Figure 1: Operating system configuration

2.2 Software Requirements

The tool that is used for this research is Google Colab which used Google drive where the data is stored for the research purpose.

2.2.1 Setup of Google Colab

Steps:

1. Search Google Colab using google search

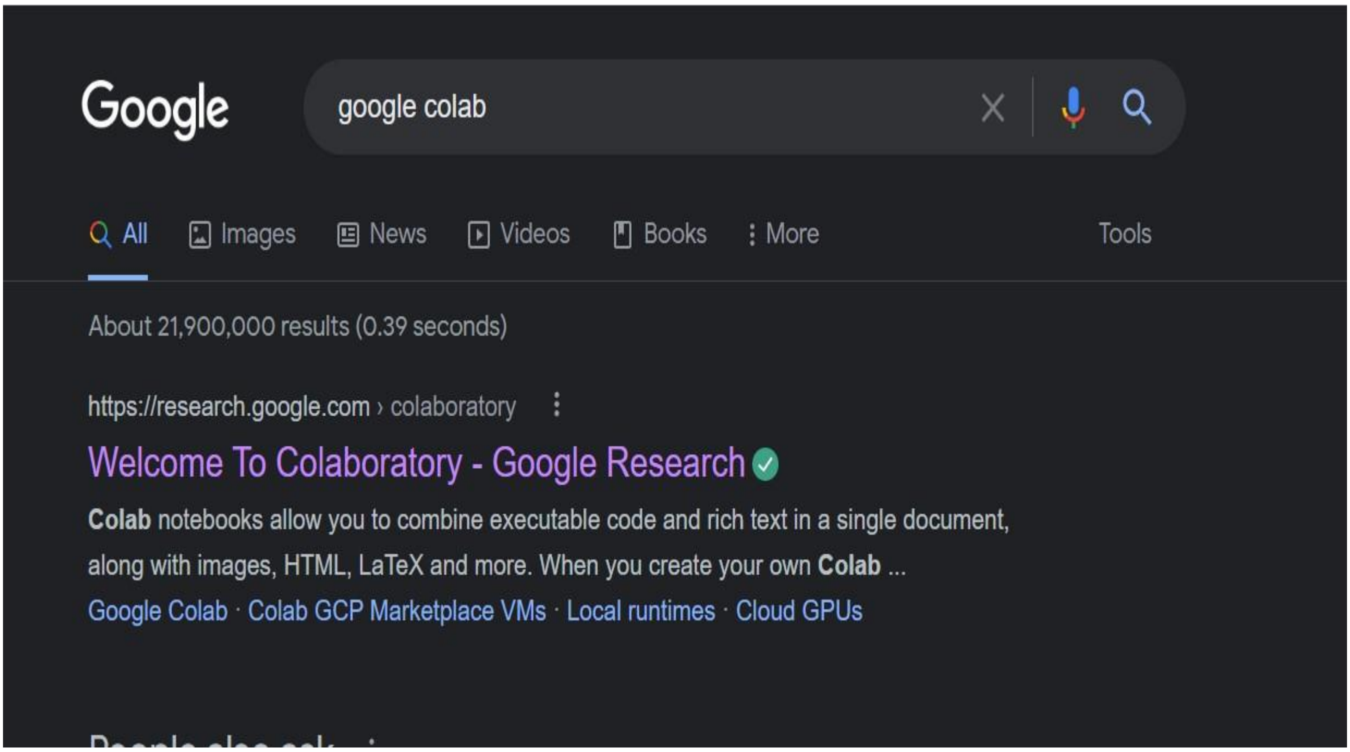


Figure 2: Google Search

2. Click on first result ‘Welcome to Colaboratory’ which will direct you to the page where it will ask to create new file or open an already existing file.

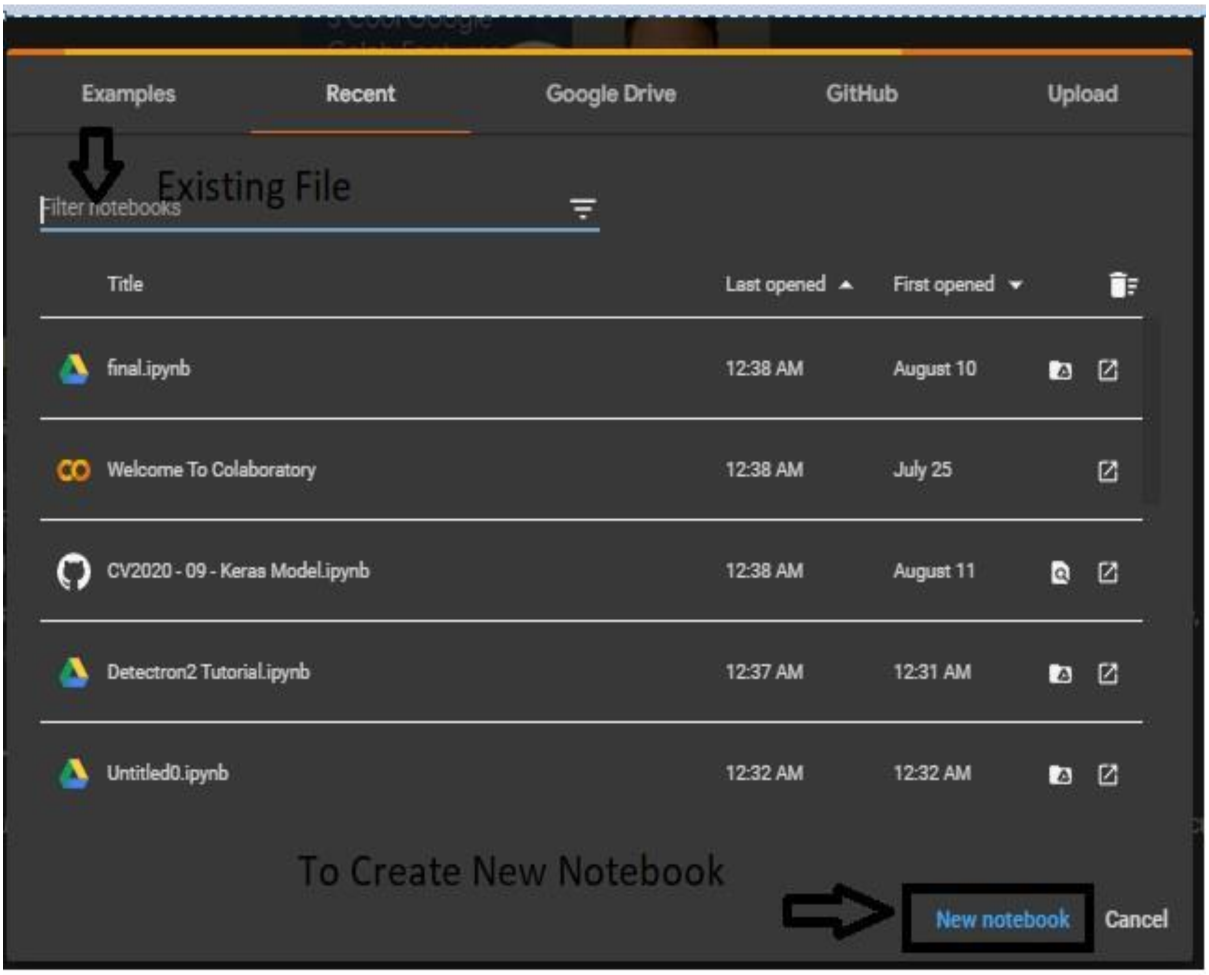


Figure 3: Create a New Notebook

3. Implementation, Evaluation and Results

3.1 **Loading Data in drive**

The dataset which is gathered from Kaggle and after Manual construction of dataset where images are deleted from each section is uploaded to drive

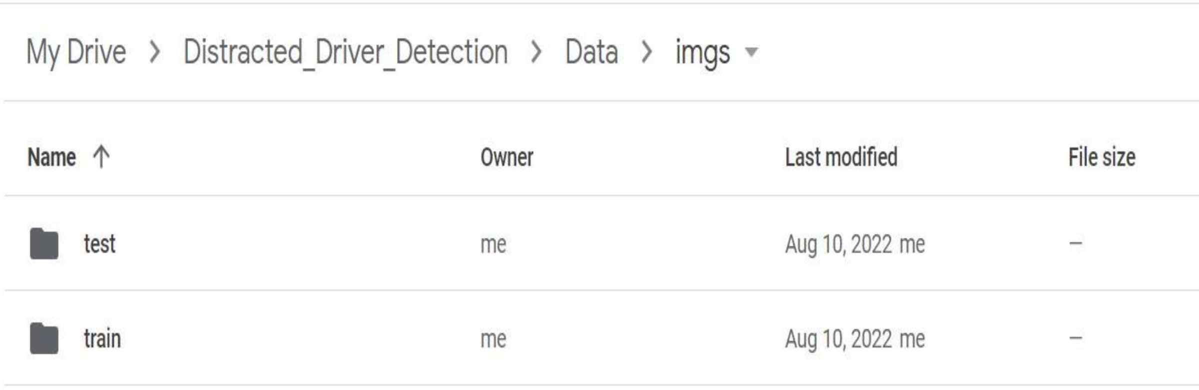


Figure 4: Dataset of Driver Distraction Detection

3.1.1 **Authorization**

The data that is uploaded in the drive is loaded in the Colab. While executing the code, a authentication is needed to access the drive**.**

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

Figure 5: Mounting drive and google Colab

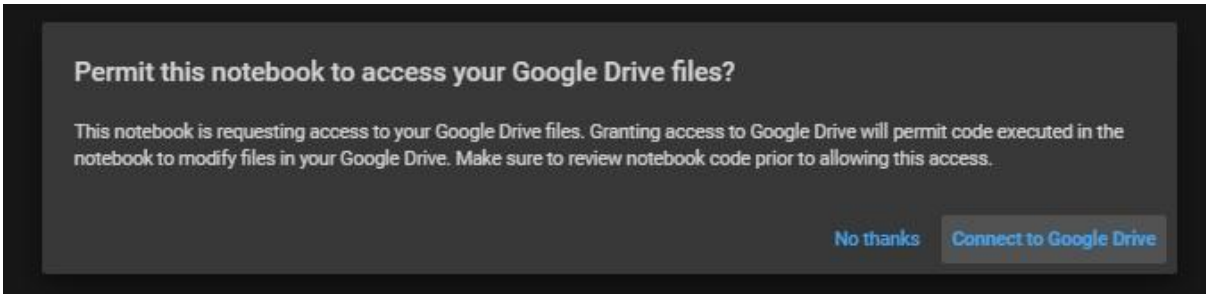


Figure 6: Giving access to connect to drive

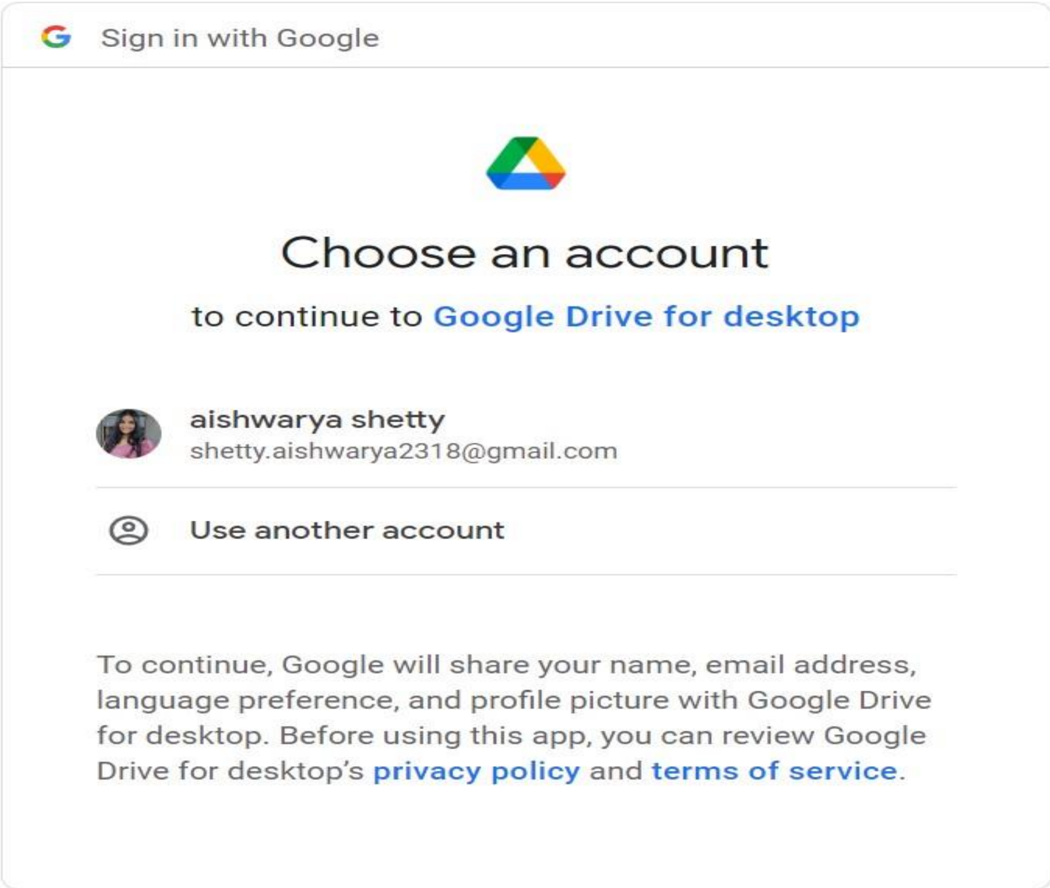


Figure 7: Choose the Account

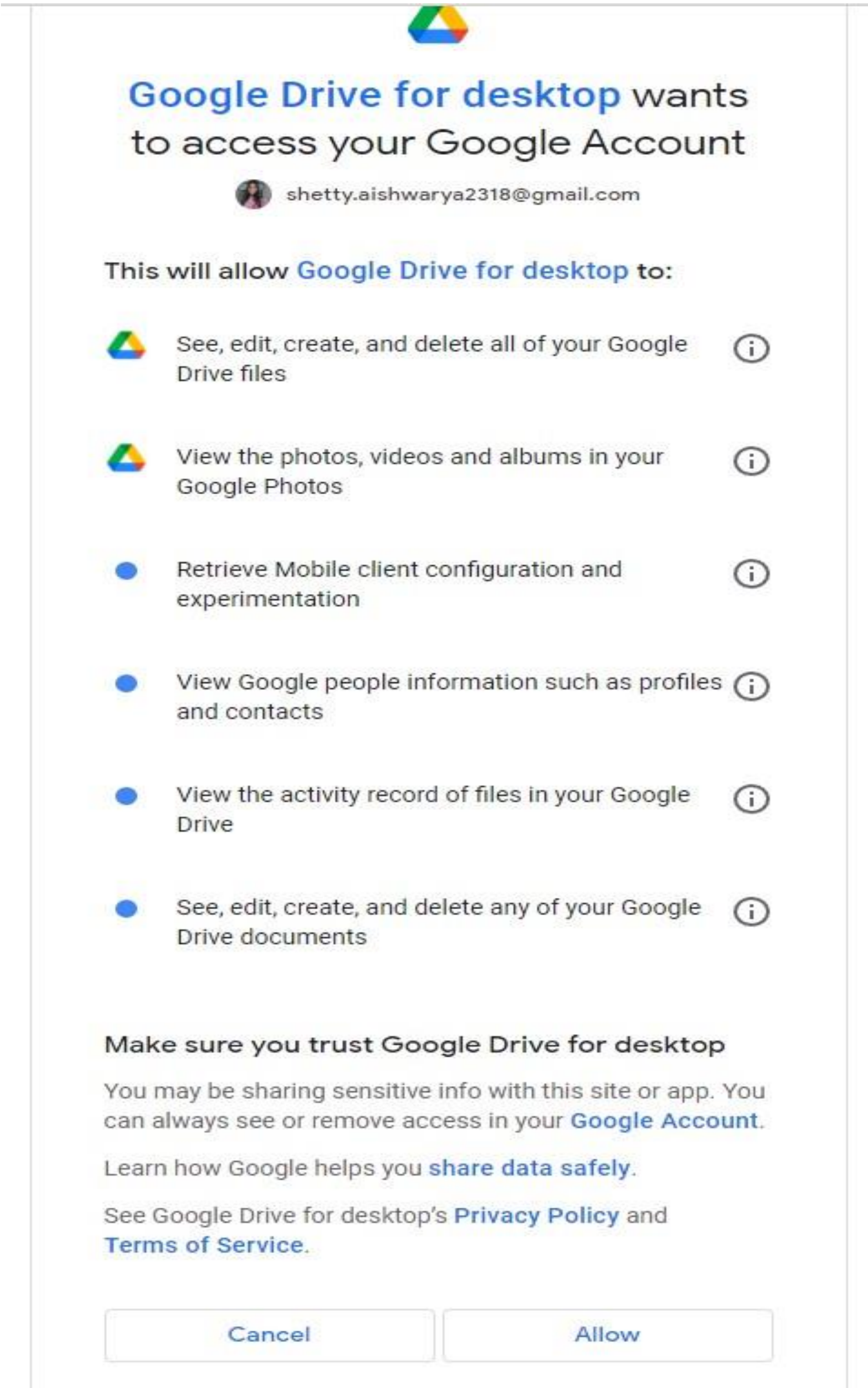


Figure 8: Click on allow to give permission

3.2 **Importing important libraries and packages**.

```
!pip install keras==2.3.0
!pip install tensorflow==2.0.0
!pip install h5py==2.10.0
!pip install opencv-python==3.4.4.19
```

```
[5] import os
from glob import glob
import random
import time
import tensorflow
import datetime
os.environ['KERAS_BACKEND'] = 'tensorflow'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # 3 = INFO, WARNING, and ERROR messages are not printed

from tqdm import tqdm

import numpy as np
import pandas as pd
from IPython.display import FileLink
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
%matplotlib inline
from IPython.display import display, Image
import matplotlib.image as mpimg
import cv2

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_files
from keras.utils import np_utils
from sklearn.utils import shuffle
from sklearn.metrics import log_loss

from keras.models import Sequential, Model
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.applications.vgg16 import VGG16
```

Figure 9: Importing libraries and packages

3.3 Preliminary Data exploratory, Data Pre-processing and Data Augmentation

```
NUMBER_CLASSES = 10
# Color type: 1 - grey, 3 - rgb

def get_cv2_image(path, img_rows, img_cols, color_type=3):
    # Loading as Grayscale image
    if color_type == 1:
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    elif color_type == 3:
        img = cv2.imread(path, cv2.IMREAD_COLOR)
    # Reduce size
    img = cv2.resize(img, (img_rows, img_cols))
    return img
```

Figure 10: Loading as gray scale and colour image and then resizing the image.

An empty list is established where the photos in the array and label classes will be kept before reading the images in each class folder. The looping in train folder will read the picture, store the label class, run over each class, and then attach the image and class.

```python
# Training
def load_train(img_rows, img_cols, color_type=3):
    start_time = time.time()
    train_images = []
    train_labels = []
    # Loop over the training folder
    for classed in tqdm(range(NUMBER_CLASSES)):
        print('Loading directory c{}'.format(classed))
        files = glob(os.path.join('..','/content/drive/MyDrive/Distracted_Driver_Detection/Data/imgs', 'train', 'c' + str(classed), '*.jpg'))
        for file in files:
            img = get_cv2_image(file, img_rows, img_cols, color_type)
            train_images.append(img)
            train_labels.append(classed)
    print("Data Loaded in {} second".format(time.time() - start_time))
    return train_images, train_labels

def read_and_normalize_train_data(img_rows, img_cols, color_type):
    X, labels = load_train(img_rows, img_cols, color_type)
    y = np_utils.to_categorical(labels, 10)
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    x_train = np.array(x_train, dtype=np.uint8).reshape(-1,img_rows,img_cols,color_type)
    x_test = np.array(x_test, dtype=np.uint8).reshape(-1,img_rows,img_cols,color_type)

    return x_train, x_test, y_train, y_test,labels
```

Figure 11: Appending image class and label class

```python
# Validation
def load_test(size=100, img_rows=64, img_cols=64, color_type=3):
    path = os.path.join('..', '/content/drive/MyDrive/Distracted_Driver_Detection/Data/imgs', 'test', '*.jpg')
    files = sorted(glob(path))
    X_test, X_test_id = [], []
    total = 0
    files_size = len(files)
    for file in tqdm(files):
        if total >= size or total >= files_size:
            break
        file_base = os.path.basename(file)
        img = get_cv2_image(file, img_rows, img_cols, color_type)
        X_test.append(img)
        X_test_id.append(file_base)
        total += 1
    return X_test, X_test_id

def read_and_normalize_sampled_test_data(size, img_rows, img_cols, color_type=3):
    test_data, test_ids = load_test(size, img_rows, img_cols, color_type)

    test_data = np.array(test_data, dtype=np.uint8)
    test_data = test_data.reshape(-1,img_rows,img_cols,color_type)

    return test_data, test_ids
```

Figure 12: Appending image class and label class

3.3.1 Resizing the images into 64x64

```python
img_rows = 64
img_cols = 64
color_type = 1
```

Figure 13: Resizing the image

```python
x_train, x_test, y_train, y_test,labels = read_and_normalize_train_data(img_rows, img_cols, color_type)
print('Train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
```

Figure 14: Model Building for train and test data

```python
# Statistics
# Load the list of names
names = [item[17:19] for item in sorted(glob("../content/drive/MyDrive/Distracted_Driver_Detection/Data/imgs/train/*/"))]
test_files_size = len(np.array(glob(os.path.join('..', '/content/drive/MyDrive/Distracted_Driver_Detection/Data/imgs/test', 'test', '*.jpg'))))
x_train_size = len(x_train)
categories_size = len(names)
x_test_size = len(x_test)
print('There are %s total images.\n' % (test_files_size + x_train_size + x_test_size))
print('There are %d training images.' % x_train_size)
print('There are %d total training categories.' % categories_size)
print('There are %d validation images.' % x_test_size)
print('There are %d test images.'% test_files_size)
```

Figure 15: Loading the parameters for train, test and validation

```python
plt.figure(figsize = (10,10))
# Count the number of images per category
sns.countplot(labels)
```

Figure16: plotting Count of images in each class

```python
activity_map = {'c0': 'Safe driving',
                'c1': 'Texting - right',
                'c2': 'Talking on the phone - right',
                'c3': 'Texting - left',
                'c4': 'Talking on the phone - left',
                'c5': 'Operating the radio',
                'c6': 'Drinking',
                'c7': 'Reaching behind',
                'c8': 'Hair and makeup',
                'c9': 'Talking to passenger'}
```

Figure 17: Mapping the images

```python
plt.figure(figsize = (12, 20))
image_count = 1
BASE_URL = '/content/drive/MyDrive/Distracted_Driver_Detection/Data/imgs/train/'
for directory in os.listdir(BASE_URL):
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(5, 2, image_count)
                image_count += 1
                image = mpimg.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(activity_map[directory])
```

Figure 18: Plotting the mapped images

```python
batch_size = 32
train_datagen = ImageDataGenerator( rescale=1 / 255.0,
                                    zoom_range=0.05,
                                    width_shift_range=0.05,
                                    height_shift_range=0.05,
                                    shear_range=0.05,
                                    fill_mode="nearest")
```

Figure 19: Data augmentation using the above-mentioned features

```python
test_datagen = ImageDataGenerator(rescale=1.0/ 255, validation_split = 0.2)
```

```python
nb_train_samples = x_train.shape[0]
nb_validation_samples = x_test.shape[0]
print(nb_train_samples)
print(nb_validation_samples)
training_generator = train_datagen.flow(x_train, y_train, batch_size=batch_size)
validation_generator = test_datagen.flow(x_test, y_test, batch_size=batch_size)
```

Figure 20: Splitting the augmented images into train and validation

```python
models_dir = "saved_models"
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

checkpointer = ModelCheckpoint(filepath='saved_models/weights_best_cnn_model.h5',
                               monitor='val_loss', mode='min',
                               verbose=1, save_best_only=True)
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
callbacks = [checkpointer, es]
```

Figure 22: Saving the file

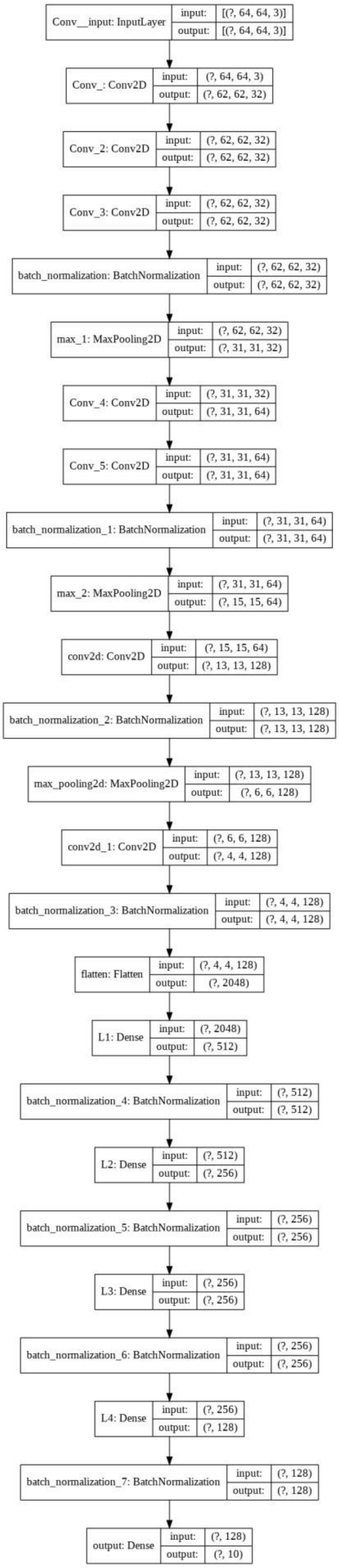
3.4 Implementation, Evaluation and results of Convolutional Neural Network

```python
#CNN Model
cnn_model = models.Sequential()
cnn_model.add(layers.Conv2D(32,(3,3),activation = 'relu',name = 'Conv_',input_shape = (64,64,3)))
cnn_model.add(layers.Conv2D(32,(3,3),activation = 'relu',name = 'Conv_2',padding = 'same'))
cnn_model.add(layers.Conv2D(32,(3,3),activation = 'relu',name = 'Conv_3',padding = 'same'))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.MaxPooling2D((2,2),name = 'max_1'))
cnn_model.add(layers.Conv2D(64,(3,3),activation = 'relu',name = 'Conv_4',padding='same'))
cnn_model.add(layers.Conv2D(64,(3,3),activation = 'relu',name = 'Conv_5',padding='same'))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.MaxPooling2D((2,2),name = 'max_2'))

cnn_model.add(layers.Conv2D(128,(3,3),activation='relu'))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.MaxPooling2D((2,2)))
cnn_model.add(layers.Conv2D(128,(3,3),activation='relu'))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.Flatten())
cnn_model.add(layers.Dense(512,activation = 'relu',name = 'L1',))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.Dense(256,activation = 'relu',name = 'L2'))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.Dense(256,activation = 'relu',name = 'L3'))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.Dense(128,activation = 'relu' ,name ='L4'))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.Dense(10,activation = 'softmax',name = 'output'))
```

Figure 23: Implementation of CNN model

```python
from tensorflow.keras.utils import plot_model
plot_model(cnn_model,"model.png",show_shapes=True,show_layer_names=True)
```

Figure 24: plotting flow of CNN model.



Conv__input: InputLayer | input: [(?, 64, 64, 3)] | output: [(?, 64, 64, 3)]
Conv_: Conv2D | input: (?, 64, 64, 3) | output: (?, 62, 62, 32)
Conv_2: Conv2D | input: (?, 62, 62, 32) | output: (?, 62, 62, 32)
Conv_3: Conv2D | input: (?, 62, 62, 32) | output: (?, 62, 62, 32)
batch_normalization: BatchNormalization | input: (?, 62, 62, 32) | output: (?, 62, 62, 32)
max_1: MaxPooling2D | input: (?, 62, 62, 32) | output: (?, 31, 31, 32)
Conv_4: Conv2D | input: (?, 31, 31, 32) | output: (?, 31, 31, 64)
Conv_5: Conv2D | input: (?, 31, 31, 64) | output: (?, 31, 31, 64)
batch_normalization_1: BatchNormalization | input: (?, 31, 31, 64) | output: (?, 31, 31, 64)
max_2: MaxPooling2D | input: (?, 31, 31, 64) | output: (?, 15, 15, 64)
conv2d: Conv2D | input: (?, 15, 15, 64) | output: (?, 13, 13, 128)
batch_normalization_2: BatchNormalization | input: (?, 13, 13, 128) | output: (?, 13, 13, 128)
max_pooling2d: MaxPooling2D | input: (?, 13, 13, 128) | output: (?, 6, 6, 128)
conv2d_1: Conv2D | input: (?, 6, 6, 128) | output: (?, 4, 4, 128)
batch_normalization_3: BatchNormalization | input: (?, 4, 4, 128) | output: (?, 4, 4, 128)
flatten: Flatten | input: (?, 4, 4, 128) | output: (?, 2048)
L1: Dense | input: (?, 2048) | output: (?, 512)
batch_normalization_4: BatchNormalization | input: (?, 512) | output: (?, 512)
L2: Dense | input: (?, 512) | output: (?, 256)
batch_normalization_5: BatchNormalization | input: (?, 256) | output: (?, 256)
L3: Dense | input: (?, 256) | output: (?, 256)
batch_normalization_6: BatchNormalization | input: (?, 256) | output: (?, 256)
L4: Dense | input: (?, 256) | output: (?, 128)
batch_normalization_7: BatchNormalization | input: (?, 128) | output: (?, 128)
output: Dense | input: (?, 128) | output: (?, 10)

```
cnn_model.compile(optimizer = 'adam' , loss='categorical_crossentropy', metrics=['acc'])

checkpoint = ModelCheckpoint('saved_models/weights_best_cnn_model.h5', monitor='val_acc', verbose=1, save_best_only=True, mode='max')
```

```
cnn_model_history = cnn_model.fit_generator(training_generator,
                          steps_per_epoch = nb_train_samples // batch_size,
                          epochs = 10,
                          callbacks=[es, checkpoint],
                          verbose = 1,
                          validation_data = validation_generator,
                          validation_steps = nb_validation_samples // batch_size)
```

Figure 25: Running epochs

```
acc = cnn_model_history.history['acc']
val_acc = cnn_model_history.history['val_acc']

loss = cnn_model_history.history['loss']
val_loss = cnn_model_history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b' ,label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Figure 26: Plotting validation accuracy and validation loss

```
cnn_model.save('/content/drive/MyDrive/Distracted_Driver_Detection/saved_models/Cnn_model.h5')
```

```
predict_x=cnn_model.predict(xx_test)
classes_x=np.argmax(predict_x,axis=1)
```

```
print("accuracy: ", accuracy_score(ytest_labels,classes_x))
```

Figure 27: Test accuracy

```
cmat = confusion_matrix(ytest_labels,classes_x)
plt.figure(figsize=(16,16))
sns.heatmap(cmat, annot = True, cbar = False, cmap='Paired', fmt="d");
```

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(10):
    prec,recall,_,_ = precision_recall_fscore_support(np.array(ytest_labels)==l,
                                                      np.array(classes_x)==l,
                                                      pos_label=True,average=None)
    res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

Figure 28: Plotting confusion matrix

3.5 **Implementation, Evaluation and results of Residual Network50**

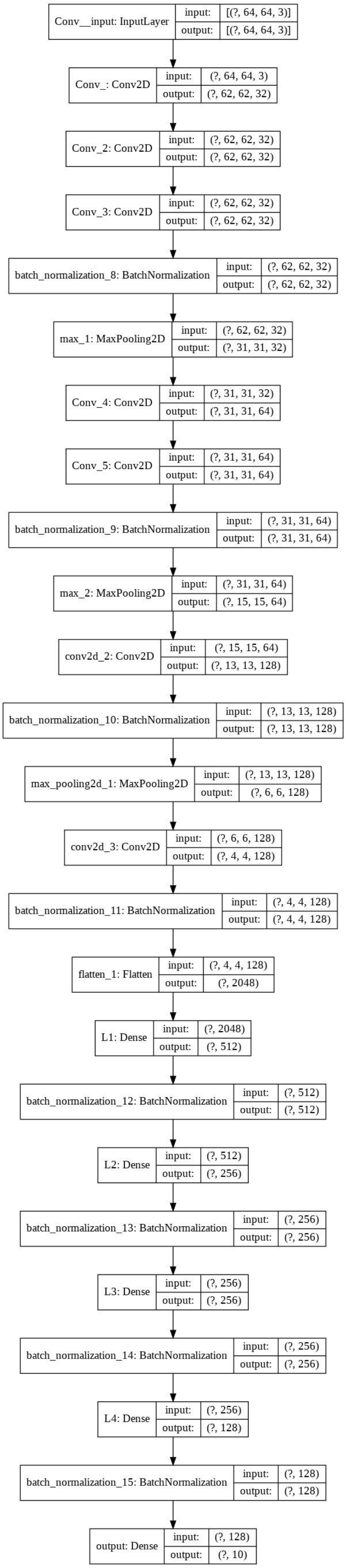
Transfer learning with resnet50

```
conv_model = resnet50.ResNet50(weights='imagenet',include_top=False,input_shape = (64,64,3))

for layer in conv_model.layers[:-3]:
    layer.trainable=False  #The role of the embedding layer is to map a category into a dense space in a way that is useful for the task

resnet_model = models.Sequential()
resnet_model.add(layers.Conv2D(32,(3,3),activation = 'relu',name = 'Conv_',input_shape = (64,64,3)))
resnet_model.add(layers.Conv2D(32,(3,3),activation = 'relu',name = 'Conv_2',padding = 'same'))
resnet_model.add(layers.Conv2D(32,(3,3),activation = 'relu',name = 'Conv_3',padding = 'same'))
resnet_model.add(layers.BatchNormalization())
resnet_model.add(layers.MaxPooling2D((2,2),name = 'max_1'))
resnet_model.add(layers.Conv2D(64,(3,3),activation = 'relu',name = 'Conv_4',padding='same'))
resnet_model.add(layers.Conv2D(64,(3,3),activation = 'relu',name = 'Conv_5',padding='same'))
resnet_model.add(layers.BatchNormalization())
resnet_model.add(layers.MaxPooling2D((2,2),name = 'max_2'))

resnet_model.add(layers.Conv2D(128,(3,3),activation='relu'))
resnet_model.add(layers.BatchNormalization())
resnet_model.add(layers.MaxPooling2D((2,2)))
resnet_model.add(layers.Conv2D(128,(3,3),activation='relu'))
resnet_model.add(layers.BatchNormalization())
resnet_model.add(layers.Flatten())
resnet_model.add(layers.Dense(512,activation = 'relu',name = 'L1',))
resnet_model.add(layers.BatchNormalization())
resnet_model.add(layers.Dense(256,activation = 'relu',name = 'L2'))
resnet_model.add(layers.BatchNormalization())
resnet_model.add(layers.Dense(256,activation = 'relu',name = 'L3'))
resnet_model.add(layers.BatchNormalization())
resnet_model.add(layers.Dense(128,activation = 'relu' ,name ='L4'))
resnet_model.add(layers.BatchNormalization())
resnet_model.add(layers.Dense(10,activation = 'softmax',name = 'output'))
```

Figure 29: Implementation of ResNet50

```
plot_model(resnet_model,"model.png",show_shapes=True,show_layer_names=True)
```

Figure 30: Plotting the flow for ResNet50



Conv__input: InputLayer
input: [(?, 64, 64, 3)]
output: [(?, 64, 64, 3)]
Conv_: Conv2D
input: (?, 64, 64, 3)
output: (?, 62, 62, 32)
Conv_2: Conv2D
input: (?, 62, 62, 32)
output: (?, 62, 62, 32)
Conv_3: Conv2D
input: (?, 62, 62, 32)
output: (?, 62, 62, 32)
batch_normalization_8: BatchNormalization
input: (?, 62, 62, 32)
output: (?, 62, 62, 32)
max_1: MaxPooling2D
input: (?, 62, 62, 32)
output: (?, 31, 31, 32)
Conv_4: Conv2D
input: (?, 31, 31, 32)
output: (?, 31, 31, 64)
Conv_5: Conv2D
input: (?, 31, 31, 64)
output: (?, 31, 31, 64)
batch_normalization_9: BatchNormalization
input: (?, 31, 31, 64)
output: (?, 31, 31, 64)
max_2: MaxPooling2D
input: (?, 31, 31, 64)
output: (?, 15, 15, 64)
conv2d_2: Conv2D
input: (?, 15, 15, 64)
output: (?, 13, 13, 128)
batch_normalization_10: BatchNormalization
input: (?, 13, 13, 128)
output: (?, 13, 13, 128)
max_pooling2d_1: MaxPooling2D
input: (?, 13, 13, 128)
output: (?, 6, 6, 128)
conv2d_3: Conv2D
input: (?, 6, 6, 128)
output: (?, 4, 4, 128)
batch_normalization_11: BatchNormalization
input: (?, 4, 4, 128)
output: (?, 4, 4, 128)
flatten_1: Flatten
input: (?, 4, 4, 128)
output: (?, 2048)
L1: Dense
input: (?, 2048)
output: (?, 512)
batch_normalization_12: BatchNormalization
input: (?, 512)
output: (?, 512)
L2: Dense
input: (?, 512)
output: (?, 256)
batch_normalization_13: BatchNormalization
input: (?, 256)
output: (?, 256)
L3: Dense
input: (?, 256)
output: (?, 256)
batch_normalization_14: BatchNormalization
input: (?, 256)
output: (?, 256)
L4: Dense
input: (?, 256)
output: (?, 128)
batch_normalization_15: BatchNormalization
input: (?, 128)
output: (?, 128)
output: Dense
input: (?, 128)
output: (?, 10)

```
resnet_model.compile(optimizer = optimizers.Adam(learning_rate=.0001) ,
              loss='categorical_crossentropy',
              metrics=['acc'])
```

```
models_dir = "saved_models"
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

checkpointer = ModelCheckpoint(filepath='saved_models/weights_best_resnet50_model.h5',
                               monitor='val_loss', mode='min',
                               verbose=1, save_best_only=True)
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
callbacks = [checkpointer, es]
```

```
stop_criteria = EarlyStopping(monitor='val_loss', mode='min', verbose=1,patience=2)
resnet_model_history = resnet_model.fit(x = training_generator,
          steps_per_epoch=nb_train_samples // batch_size,
          epochs=10,
          validation_data = validation_generator,
          validation_steps= nb_validation_samples // batch_size)
```

Figure 31: Running Epoch

```
acc = resnet_model_history.history['acc']
val_acc = resnet_model_history.history['val_acc']

loss = resnet_model_history.history['loss']
val_loss = resnet_model_history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b' ,label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Figure 32: Training and Validation accuracy and loss plot

```
resnet_model.save('/content/drive/MyDrive/Distracted_Driver_Detection/saved_models/resnet_model.h5')
```

```
predict_x=resnet_model.predict(xx_test)
classes_x=np.argmax(predict_x,axis=1)
```

```
print("accuracy: ", accuracy_score(ytest_labels,classes_x))
```

```
accuracy:  0.9443254817987152
```

```
print(classification_report(ytest_labels,classes_x))
```

Figure 33: Test Accuracy

```
cmat = confusion_matrix(ytest_labels,classes_x)
plt.figure(figsize=(16,16))
sns.heatmap(cmat, annot = True, cbar = False, cmap='Paired', fmt="d");
```

Figure 34: loading Heatmap.

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(10):
    prec,recall,_,_ = precision_recall_fscore_support(np.array(ytest_labels)==l,
                                                      np.array(classes_x)==l,
                                                      pos_label=True,average=None)
    res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

3.6 Implementation, Evaluation and results of Convolutional Visual Geometry Group16

```python
def vgg_std16_model(img_rows, img_cols, color_type=3):
    nb_classes = 10
    # Remove fully connected layer and replace
    # with softmax for classifying 10 classes
    vgg16_model = VGG16(weights="imagenet", include_top=False)

    # Freeze all layers of the pre-trained model
    for layer in vgg16_model.layers:
        layer.trainable = False

    x = vgg16_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(nb_classes, activation = 'softmax')(x)

    model = Model(input = vgg16_model.input, output = predictions)

    return model
```

Figure 34: Implementation VGG16 model

```python
# Load the VGG16 network
print("Loading network...")
model_vgg16 = vgg_std16_model(img_rows, img_cols)

model_vgg16.summary()

model_vgg16.compile(loss='categorical_crossentropy',
                    optimizer='rmsprop',
                    metrics=['accuracy'])
```

```python
checkpoint = ModelCheckpoint('/content/drive/MyDrive/Distracted_Driver_Detection/saved_models/weights_best_vgg16.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='max')
history_v4 = model_vgg16.fit_generator(training_generator,
                    steps_per_epoch = nb_train_samples // batch_size,
                    epochs = 10,
                    callbacks=[es, checkpoint],
                    verbose = 1,
                    class_weight='auto',
                    validation_data = validation_generator,
                    validation_steps = nb_validation_samples // batch_size)
```

Figure 35: Running epochs

```python
acc = history_v4.history['accuracy']
val_acc = history_v4.history['val_accuracy']

loss = history_v4.history['loss']
val_loss = history_v4.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b' ,label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Figure36: Plotting training and validation accuracy and loss

```python
print(classification_report(ytest_labels,classes_x))
```

```python
model_vgg16.save('/content/drive/MyDrive/Distracted_Driver_Detection/saved_models/weights_best_vgg16.h5')
```

```python
predict_x=model_vgg16.predict(xx_test)
classes_x=np.argmax(predict_x,axis=1)
```

```python
print("accuracy: ", accuracy_score(ytest_labels,classes_x))
```

Figure 37: Test Accuracy

```python
cmat = confusion_matrix(ytest_labels,classes_x)
plt.figure(figsize=(16,16))
sns.heatmap(cmat, annot = True, cbar = False, cmap='Paired', fmt="d");
```

Figure 38: Loading Heatmap

```python
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(10):
    prec,recall,_,_ = precision_recall_fscore_support(np.array(ytest_labels)==l,
                                                      np.array(classes_x)==l,
                                                      pos_label=True,average=None)
    res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

3.7 Implementation, Evaluation and results of Convolutional Visual Geometry Group19

```python
def vgg_std19_model(img_rows, img_cols, color_type=3):
    nb_classes = 10
    # Remove fully connected layer and replace
    # with softmax for classifying 10 classes
    vgg19_model = VGG19(weights="imagenet", include_top=False)

    # Freeze all layers of the pre-trained model
    for layer in vgg19_model.layers:
        layer.trainable = False

    x = vgg19_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(nb_classes, activation = 'softmax')(x)

    model = Model(input = vgg19_model.input, output = predictions)

    return model
```

Figure 39: Implementing VGG19 model

```python
# Load the VGG16 network
print("Loading network...")
model_vgg19 = vgg_std19_model(img_rows, img_cols)

model_vgg19.summary()

model_vgg19.compile(loss='categorical_crossentropy',
                    optimizer='rmsprop',
                    metrics=['accuracy'])
```

```python
checkpoint = ModelCheckpoint('/content/drive/MyDrive/Distracted_Driver_Detection/saved_models/weights_best_vgg19.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='max')
history_vgg19 = model_vgg19.fit_generator(training_generator,
                    steps_per_epoch = nb_train_samples // batch_size,
                    epochs = 10,
                    callbacks=[es, checkpoint],
                    verbose = 1,
                    class_weight='auto',
                    validation_data = validation_generator,
                    validation_steps = nb_validation_samples // batch_size)
```

Figure 40: Running epochs

```
model_vgg19.save('/content/drive/MyDrive/Distracted_Driver_Detection/saved_models/weights_best_vgg19.h5')
```

```
acc = history_vgg19.history['accuracy']
val_acc = history_vgg19.history['val_accuracy']

loss = history_vgg19.history['loss']
val_loss = history_vgg19.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b' ,label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Figure 41: Plotting training and validation accuracy and loss

```
print(classification_report(ytest_labels,classes_x))
```

```
predict_x=model_vgg19.predict(xx_test)
classes_x=np.argmax(predict_x,axis=1)
```

```
print("accuracy: ", accuracy_score(ytest_labels,classes_x))
```

Figure 42: Test Accuracy

```
cmat = confusion_matrix(ytest_labels,classes_x)
plt.figure(figsize=(16,16))
sns.heatmap(cmat, annot = True, cbar = False, cmap='Paired', fmt="d");
```

Figure 43: Loading Heatmap

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(10):
    prec,recall,_,_ = precision_recall_fscore_support(np.array(ytest_labels)==l,
                                                      np.array(classes_x)==l,
                                                      pos_label=True,average=None)
    res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

References