**Project Report**

**On**

**A preprocessing approach at intelligent driving model car**

Submitted by

**Rishik Mishra**

**Roll No: 181500569**

Under The Guidance of

**Mr Piyush Vashisht**

**Assistant Professor**

**Department of Computer engineering and application**

**Institute of engineering and technology**

Department of Computer Engineering & Applications

Institute of Engineering & Technology



**GLA University**

**Mathura- 281406, INDIA**

**2020**





Declaration

I hereby declare that the work which is being presented in the Project report on “**An approach at intelligent driving model car”,** in partial fulfillment of the requirements for Mini Project I, is an authentic record of my own work carried under the supervision of  **Mr Piyush Vashisht**

**Signature of Candidate:**

****

**Name of Candidate: Rishik Mishra**

**Roll. No. : 181500569**

**Course: B.Tech (CSE)**

**Year: 3rd**

**Semester: 6th**

Contents

[**Abstract**](#_heading=h.2et92p0) **3**

[**Introduction**](#_heading=h.tyjcwt) **3**

[**Problem Statement**](#_heading=h.3dy6vkm) **4**

[**Data Acquisition**](#_heading=h.4d34og8) **5**

[**Hardware Required**](#_heading=h.uan4vxmqv2w8) **6**

[**Literature Review**](#_heading=h.s1ykhm9gj8iw) **7**

[**Methodology**](#_heading=h.2s8eyo1) **9**

[**Pre Training Phase**](#_heading=h.nx13qkit3kya) **10**

[**Image Segmentation**](#_heading=h.te9hr4obdab) **11**

[**Traffic sign classification**](#_heading=h.3rdcrjn) **12**

[**Sensor Data Phase**](#_heading=h.grrcdu1fovf7) **12**

[**Camera data:**](#_heading=h.28ma5jdcievw) **13**

[**Arduino sensor data :**](#_heading=h.11o9wh9k2an6) **14**

[**Final training stage**](#_heading=h.m6qjrocmxl0) **15**

[**Construction**](#_heading=h.q6b9gc25xsu5) **16**

[**Results**](#_heading=h.2ecp6r3b8zab) **17**

[**Segmentation Stage**](#_heading=h.wy6lz9a6yl1n) **18**

[**Traffic light classification stage**](#_heading=h.4lki3t30qq8k) **19**

[**Code**](#_heading=h.udhtzcm8ju1h) **20**

[**Main Processing Code**](#_heading=h.3dfi623vv65d) **20**

[**Model Preparation Code**](#_heading=h.wgt87hdrk4j1) **22**

[**Drive Code**](#_heading=h.anhgx2jnw0ix) **23**

[**Model Training Code**](#_heading=h.lh8bjawntv53) **25**

# Abstract

Self-driving cars are autonomous vehicles that can drive by themselves without any human interference and have the potential to mark the technological revolution of the next decade. This work presents the development of a low-cost prototype of a miniature self-driving car model using simple and easily available technologies. The objective of the work is to avoid accidents caused due to driver faults. In this prototype, Raspberry Pi controller and H-bridge to arduino drive two DC motors to realize vehicle automation. Sensors like ir sensors and vgi cameras are used for obstacle detection and avoidance, image processing for pedestrian detection, computer vision for processing images and machine learning for intelligent systems have been deployed.

# Introduction

Intelligent driving is not just an avenue for commercial vehicles like cars and trucks but also for industrial,healthcare and academic uses.Intelligent driving refers to the ability of a bot to drive without the aid of humans and to intelligently identify obstacles and course.Multiple companies and academics have been working on “self driving cars” which has faced a lot of criticism from multiple advocacy groups citing the unexplainable nature of these vehicle and touting the self driving feature as a danger to society.,We have thus implemented intelligent driving in a bot ,meant to be used in a variety of scenarios ,from hospitals,schools,emergency situation to vehicular assistance.The intelligent base of our driving system consists of a raspberry pi and an arduino uno,very readily available and inexpensive boards which also add the benefit of the project moving towards open source.The method we intend to implement works well for any situation as instead of processing and learning on raw images we train our model on semantically segmented masks.

# Problem Statement

The objective of intelligent driving cars is to create a fully functional automated car that is able to reduce human effort and better traffic flow. intelligent-driving cars were created to provide benefits to the society we live in, such as providing transportation for those people who are not able to drive because of age or physical impairment or automating the work in dangerous workplaces.

# Data Acquisition

For the first phase of this study.ie. in the semantic segmentation stage we have used the Mapillary Vistas Dataset, a novel, large scale street-level image dataset containing 25,000 high resolution images annotated into 66 object categories with additional, instance-specific labels for 37 classes. Annotation is performed in a dense and fine-grained style by using polygons for delineating individual objects. The dataset is 5× larger than the total amount of fine annotations for Cityscapes and contains images from all around the world, captured at various conditions regarding weather, season and daytime. Images come from different imaging devices (mobile phones, tablets, action cameras, professional capturing rigs) and differently experienced photographers. In such a way, our dataset has been designed and compiled to cover diversity, richness of detail and geographic extent.

For the traffic sign detection stage we have used the TSDD dataset which includes 10000 traffic scene images containing many kinds of signs. The images are collected under different time,weather conditions,lighting conditions as well as moving blurring.

For the final training we have used a custom dataset created on a specifically created scenario with the model car driven manually on 2 separate tracks and the 2 minute videos of each track from the camera mounted on the model car,resulting in 24000 images along with additional data like ir sensor data and steering angles.

# Hardware Required

|  |  |  |
| --- | --- | --- |
| S.No | Item Name | Quantity |
| 1. | 150 RPM dual shaft geared motor | 4 |
| 2. | 5000 mAh lithium polymer power bank | 1 |
| 3. | 5V 0.2A Cooling Fan | 1 |
| 4. | Raspberry Pi 4 b | 1 |
| 5. | Raspberry pi 4 heatsink | 1 |
| 6. | Battery holder case 4x 1.5V AA | 1 |
| 7. | IR Proximity Sensor | 3 |
| 8. | NiMH 3000mAh battery | 4 |
| 9. | Raspberry pi 5MP camera | 1 |
| 10. | L293D Motor Shield | 1 |
| 11. | 16GB MicroSDHC | 1 |
| 12. | Arduino Uno | 1 |
| 13. | Wheels 2.5’’ | 4 |
| 14. | Laptop | 1 |
| 15. | BreadBoard | 1 |
| 16. | Jumper Cables(m-m,f-m,f-f) | 150 |
| 17. | USB A to USB B cable | 1 |
| 18. | LED | 16 |

# Literature Review

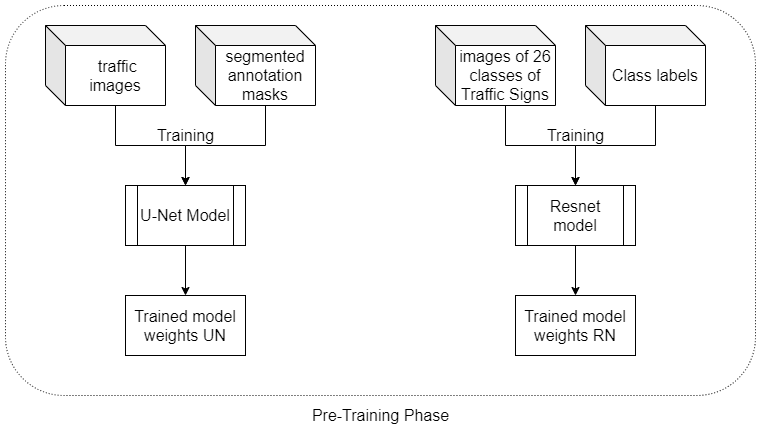
Many technical advances that enable self-driving cars are of course due to software and algorithmic innovation. There have been incredible advances in machine learning that improve the ability to perceive the world, new tracking and planning algorithms allow for safer and smoother driving, and the software infrastructure to simulate and analyze large amounts of data in data centers have all been key contributors towards making self-driving cars. The rapid development of self-driving capabilities, Google’s self-driving car project began in 2009 and transitioned to its own business entity – Waymo – within Google’s parent company (Alphabet) in 2016. Waymo’s self-driving cars contain a broad set of technologies that enable our cars to sense the vehicle surroundings, perceive and understand what is happening in the vehicle vicinity, and determine the safe and efficient actions that the vehicle should take. From a hardware perspective, we can divide Waymo’s self-driving technology into three key areas: sensing, compute, and embedded control. Our sensors capture information about the vehicle surroundings, position, and environment. The sensors send their information to a high-performance computer. The computer fuses, processes, and interprets the sensor data, ultimately generating trajectories that the vehicle must follow. The computer passes these trajectories to embedded control systems, which in turn communicate with the vehicle actuators to manipulate steering, braking, and throttle. Self-Driving Car requires several concepts that needed to be known in order to have it getting implemented they are Computer Vision, Sensor Fusion, Deep Learning, Path Planning, Actuator. Computer vision allows us to understand how computers can be made for gaining information from digital images or videos. From engineering perspective, it is used to automate tasks that the human visualize how system can do it. Sensor fusion combine variety of sensory data or data derived from various sources so that the resulting information has less uncertainty in them rather than how it would be when these sources were used individually. Deep learning is one part of machine learning that is based on learning data representations, such as opposing to task-specific algorithms used. Path-planning is for autonomous mobile robots etc. that lets robots find the shortest or the optimal path between two points

# Methodology

Fig: 1.0 Complete Process Diagram

Our method works by a cooperative operation between an arduino which acts as a drive unit and a raspberry pi which acts as the brain.The arduino is connected to an h-bridge which controls 2 geared dc motors and 1 servo motor, the arduino is further connected to 3 ir sensors which measure the distance between the objects on 3 sides.The arduino sends data to the raspberry pi through usb bus and receives serial instruction from the pi after he raw dat has been processed and a prediction has been made.

## Pre Training Phase



For the training of our final model we need the images to be available in a processed form instead of in the raw form.For this step/ we first semantically segment the images into various objects (from 1 of 128) so the images from any scenario could be generalised.Then from the segmented images the traffic sign segment is extracted and sent to a classifier for the classification between various types of traffic signs.

### Image Segmentation

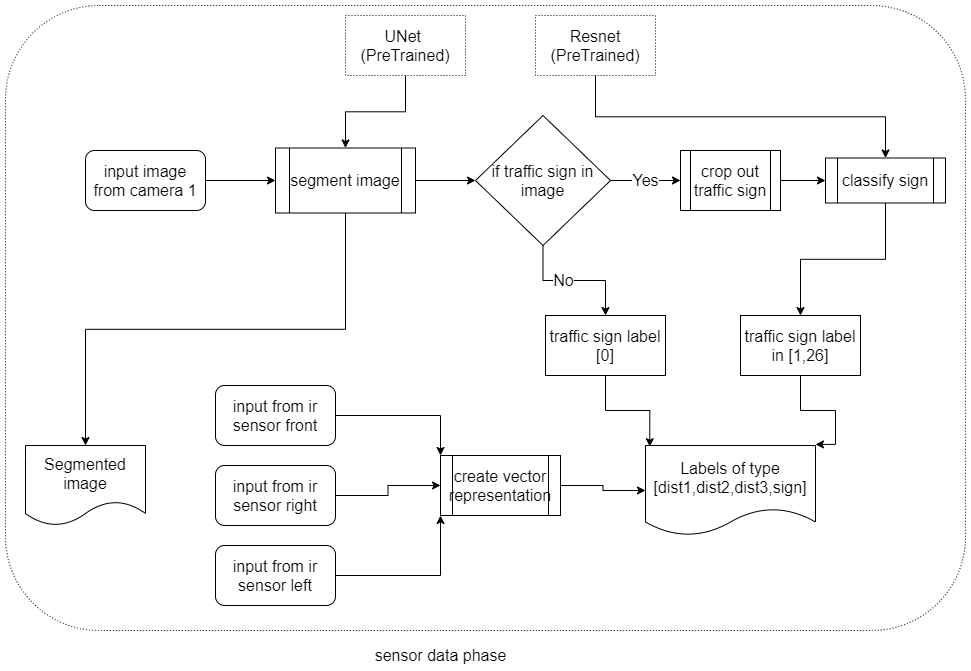
Image segmentation is a critical process in computer vision. It involves dividing a visual input into segments to simplify image analysis. Segments represent objects or parts of objects, and comprise sets of pixels, or “super-pixels”.For image segmentation we have used a modified version of the UNet model with 32 layers and filters ranging from 16 increasing in the magnitude 2\*x to reach 128 and for upsampling we have used convolution transpose.

### Traffic sign classification

Traffic sign classification is the process of automatically recognizing traffic signs along the road, including speed limit signs, yield signs, merge signs, etc.For this task we have used a Resnet architecture pre trained on imagenet dataset and then the top layers are trained with the TSDD dataset.

This stage results in 2 trained models which can be used to predict the segmentation and the traffic signs present in 256\*256\*3 images.

## Sensor Data Phase



In this stage the data is collected from 3 sources which are:

### Camera data:

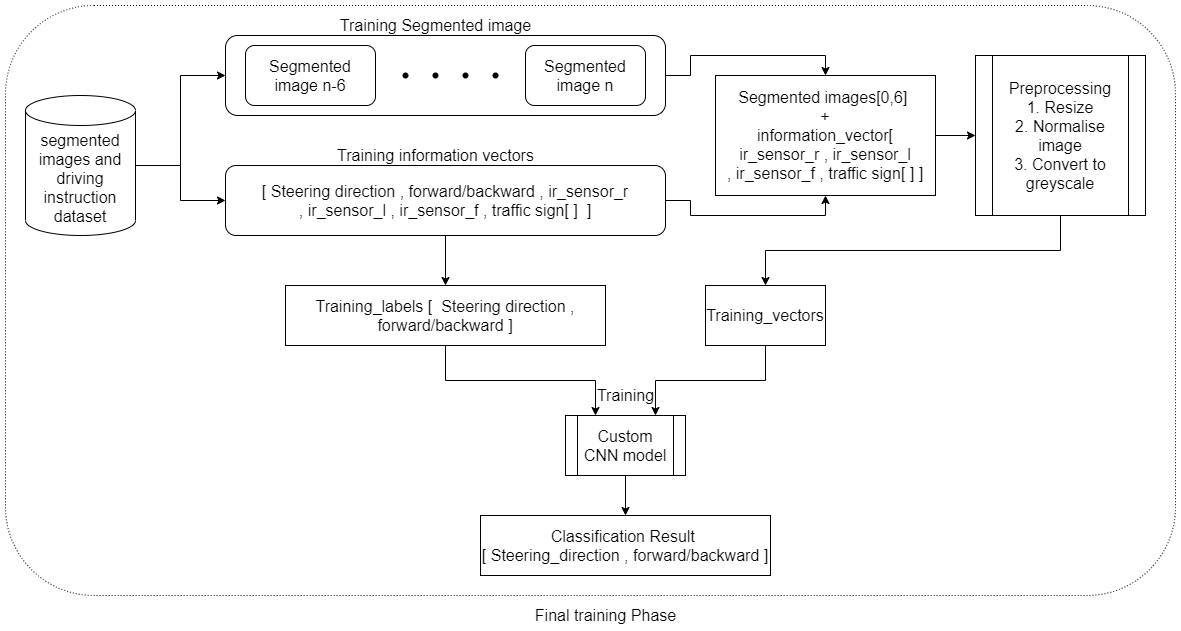
the camera mounted on top of the car provides a constant feed of 256\*256\*3 images of the path which is passed to the image segmentation with weights taken from the previous stage and the traffic sign classification model ,this step results in a processed segmented image and the label array with direction.

### Arduino sensor data :

the arduino is connected to 3 ir proximity sensors which are mounted on the back,right and left sides of the vehicle and relay the 0/1 information of whether the vehicle is in close proximity with another object.

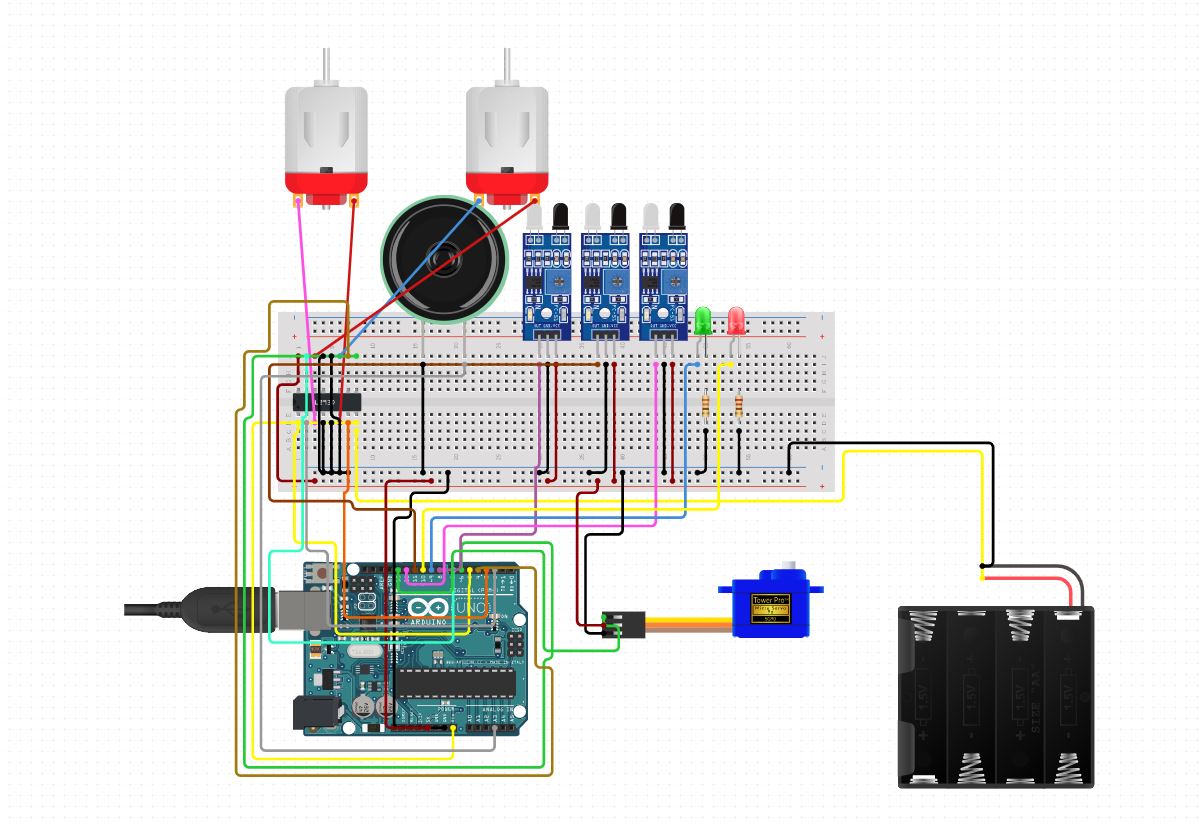
These sensor and camera datas are processed using the pretrained models (UNet for segmentation and CNN for traffic light classification).This information is arranged in a [length,5] numpy array ,where each element contains [ir\_sensor\_right,ir\_sensor\_left,ir\_sensor\_back,traffic\_light,image(segmented)],this data is appended with the direction logged from the user which is one from {f,b,l,r,s}.

## Final training stage



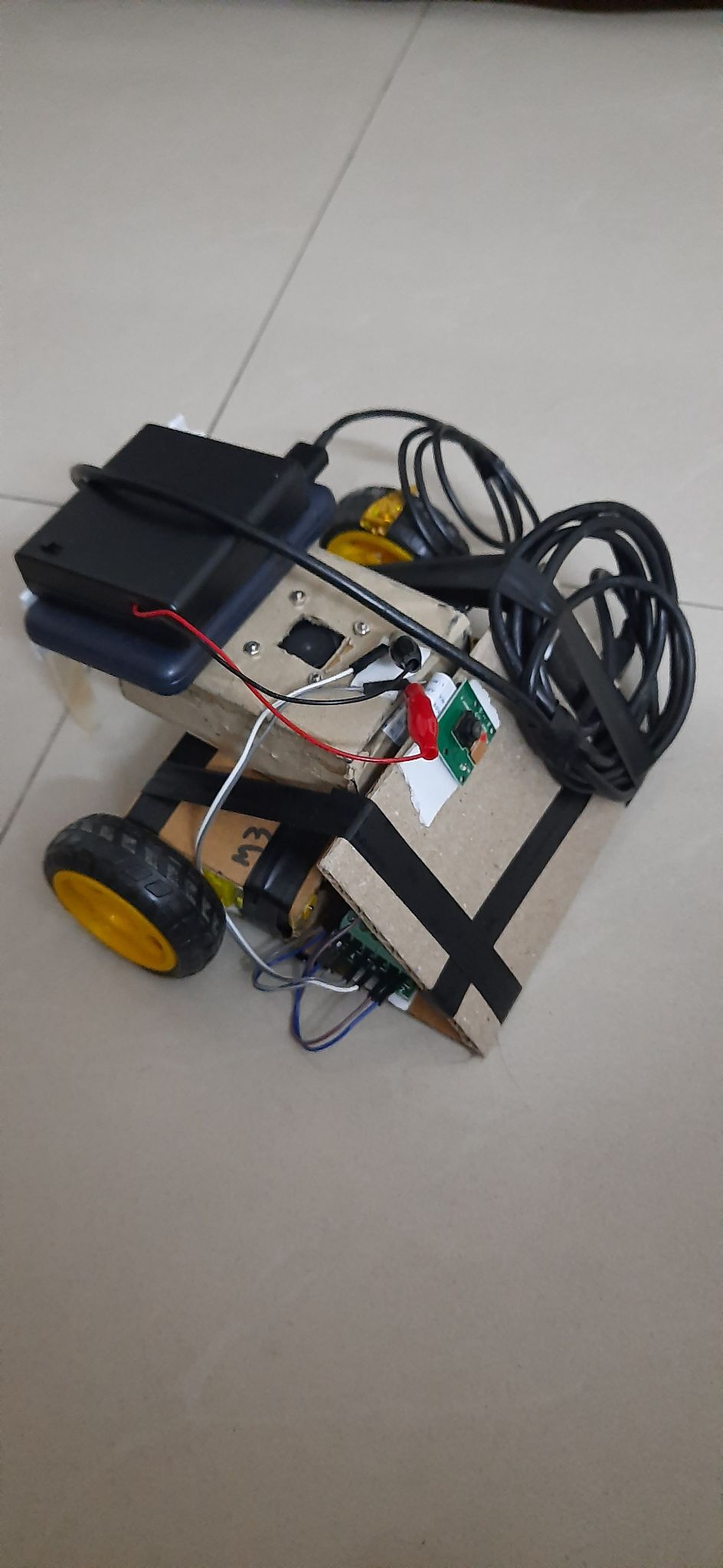
The pre-trained models and the collected sensor data is brought together and the vehicle is driven manually across 2 tracks for the training of the deep learning model which can be used to predict the moves in a testing scenario. This step will be covered in a future version of the project.

# Construction



The figure above explains the circuit connection for the driving unit which consists of an arduino uno with pin 5,6,7,8 connected to the motor a and b input pins on the L293D motor driver and 15,17 to the enable pins ,the motors are power externally by a battery pack of 3000mAh 4.8V capacity,the servo motors activation pin is connected to pin 9 of the arduino board and ir sensors are connected to pins 10 through 12 the speaker would be included in a future version of the build.

The com port of the arduino is connected to a raspberry pi 4 which is also connected to a camera module,the python program in raspberry pi is used to communicated and send driving instructions to the arduino through pyserial library.

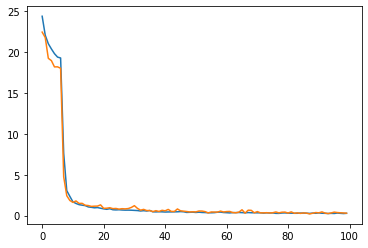
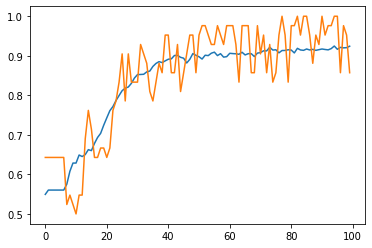


Version 1.0 and Version 2.0 of the intelligent driving vehicle,version 1.0 of the vehicle does not include any ir sensors while version 2.0 includes ir sensors and uv light for night time driving.

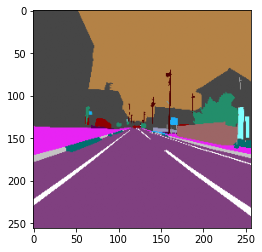
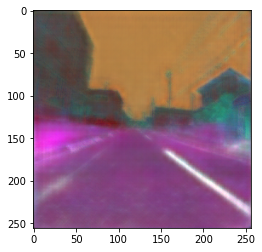
The power bank in the model is used to power the raspberry pi 4 while the battery pack powers the motors and the sensors,the camera module is mounted on the front of the vehicle in inverted position and created motor driving unit is placed at the bottom of the model.

# Results

## Segmentation Stage

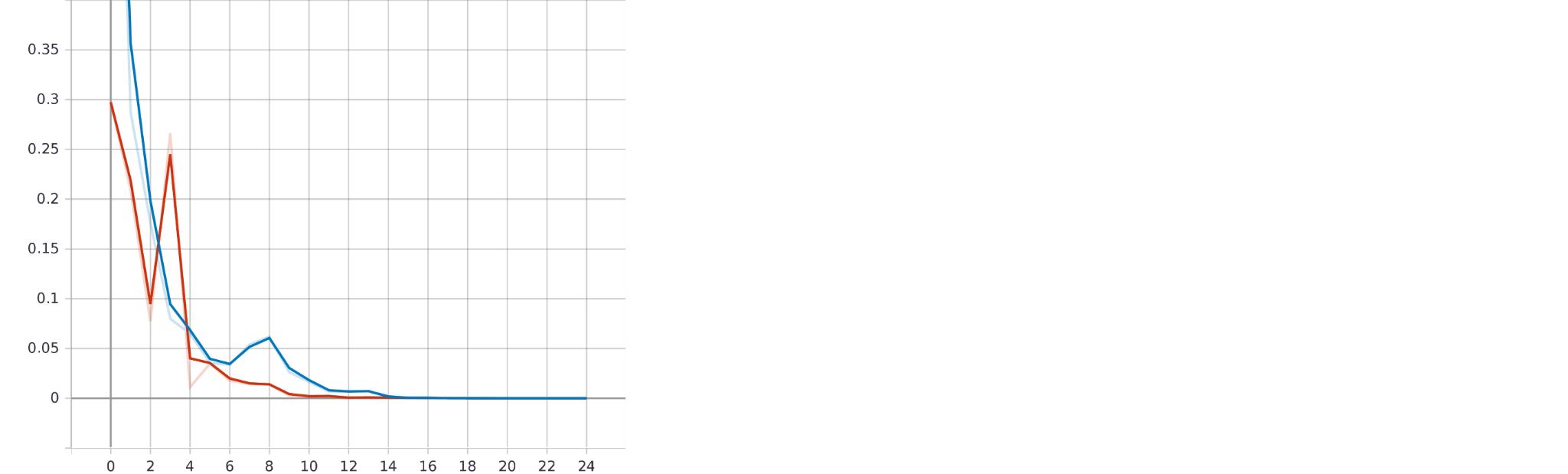
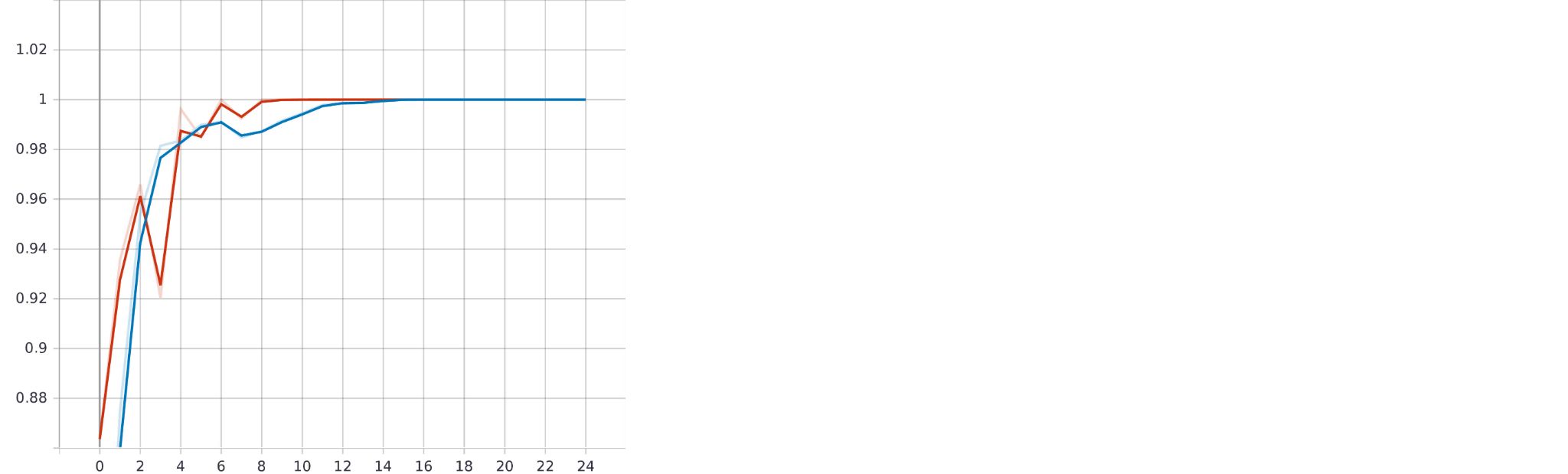


The above graphs are the accuracy and loss graphs for the segmentation training phase with the right graph being the loss graph with y axis representing mse loss and x axis representing the epochs and the left graph being the accuracy graph with y axis being accuracy and x axis being epoch.The orange lines represent validation set and blue being training set.



The above images represent the original image, predict image segments and ground truth segments for an image of the road predicted by UNet model trained by us.

## Traffic light classification stage



The above graphs are the accuracy and loss graphs for the classification stage phase with the right graph being the loss graph with y axis representing mse loss and x axis representing the epochs and the left graph being the accuracy graph with y axis being accuracy and x axis being epoch.The orange lines represent validation set and blue being training set.



000\_0002.png;165;151;23;12;149;138;0;

000\_0003.png;128;122;22;14;116;105;3;

000\_0010.png;80;73;14;8;67;63;5;

000\_0011.png;186;174;36;15;155;157;17;

The results of this stage as mentioned above produce the image size,sign bounding top right and bottom left coordinates and the class of the image

**The last stage of the process will be covered in part 2 of the project**

# Code

## Main Processing Code

import tenso

import tty, sys

import tensorflow as tf

from tensorflow.keras.layers import Input, Reshape, Dropout, Dense

from tensorflow.keras.layers import Flatten, BatchNormalization

from tensorflow.keras.layers import Activation, ZeroPadding2D

from tensorflow.keras.layers import LeakyReLU,Concatenate,concatenate

from tensorflow.keras.layers import UpSampling2D, Conv2D,Conv2DTranspose,MaxPooling2D

from tensorflow.keras.models import Sequential, Model, load\_model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import LeakyReLU,Add

import tensorflow

import cv2

import numpy as np

from PIL import Image

import os

import time

import matplotlib.pyplot as plt

from tensorflow.keras.utils import plot\_model

import serial

driver = serial.Serial('/dev/ttyUSB0',9600,timeout=1)

step = {"f":[b'f',b's'],"b":[b'b',b's'],"l":[b'r',b's'],"r":[b'l',b's'],"s":[b's',b's']}

step\_time = {'f':2,'b':1,'l':0.20,'r':0.20}

st = ['f','b','l','r']

status = True

segnet = tenso.use\_model(tenso.seg\_model((256,256,3)),"/home/pi/Desktop/segnet.h5")

vid\_feed = cv2.VideoCapture(0)

trainset = []

tty.setcbreak(sys.stdin)

while(True):

\_, frame = vid\_feed.read()

frame = cv2.rotate(frame,cv2.ROTATE\_180)

#pred\_frame = tenso.predict(segnet,frame)

print('ready')

inp = [0,0,0,0]

i = x=sys.stdin.read(1)[0]

print(i)

if(i!='q'):

try:

print(step[i][0])

driver.write(step[i][0])

time.sleep(step\_time[i])

#driver.write(step[i][1])

#time.sleep(0.1)

ino = driver.readline().decode('utf-8')[0:2]

if(len(ino)>0):

inp[0] = int(ino[0])

inp[1] = int(ino[1])

inp[2] = st.index(i)

inp[3] = frame

trainset.append(inp)

except:

pass

else:

status = False

break

print(len(trainset))

trainset = np.array(trainset)

print(trainset.shape)

print("GoodBye")

np.save("trainset.npy",trainset)

driver.close()

vid\_feed.release()

cv2.destroyAllWindows()

## Model Preparation Code

import tensorflow as tf

from tensorflow.keras.layers import Input, Reshape, Dropout, Dense

from tensorflow.keras.layers import Flatten, BatchNormalization

from tensorflow.keras.layers import Activation, ZeroPadding2D

from tensorflow.keras.layers import LeakyReLU,Concatenate,concatenate

from tensorflow.keras.layers import UpSampling2D, Conv2D,Conv2DTranspose,MaxPooling2D

from tensorflow.keras.models import Sequential, Model, load\_model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import LeakyReLU,Add

import tensorflow

import cv2

import numpy as np

from PIL import Image

import os

import time

import matplotlib.pyplot as plt

from tensorflow.keras.utils import plot\_model

def seg\_model(shape):

input = Input(shape)

seg1 = Conv2D(16,(4,4),strides=(1,1),padding='same',activation='relu')(input)

m1 = MaxPooling2D((2,2))(seg1)

seg2 = Conv2D(16,(4,4),strides=(1,1),padding='same',activation='relu')(m1)

m2 = MaxPooling2D((2,2))(seg2)

seg3 = Conv2D(32,(4,4),strides=(1,1),padding='same',activation='relu')(m2)

m3 = MaxPooling2D((2,2))(seg3)

seg4 = Conv2D(32,(4,4),strides=(1,1),padding='same',activation='relu')(m3)

m4 = MaxPooling2D((2,2))(seg4)

seg5 = Conv2D(64,(4,4),strides=(1,1),padding='same',activation='relu')(m4)

m5 = MaxPooling2D((2,2))(seg5)

seg6 = Conv2D(64,(4,4),strides=(1,1),padding='same',activation='relu')(m5)

m6 = MaxPooling2D((2,2))(seg6)

seg7 = Conv2D(128,(4,4),strides=(1,1),padding='same',activation='relu')(m6)

m7 = MaxPooling2D((2,2))(seg7)

seg8 = Conv2D(128,(4,4),strides=(1,1),padding='same',activation='relu')(m7)

deg1 = Conv2DTranspose(128,(4,4),strides=(2,2),padding='same',activation='relu')(seg8)

deg1 = concatenate([deg1,seg7],axis=3)

deg1 = Conv2DTranspose(64,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg6],axis=3)

deg1 = Conv2DTranspose(64,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg5],axis=3)

deg1 = Conv2DTranspose(32,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg4],axis=3)

deg1 = Conv2DTranspose(32,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg3],axis=3)

deg1 = Conv2DTranspose(16,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg2],axis=3)

deg1 = Conv2DTranspose(16,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg1],axis=3)

deg1 = Conv2DTranspose(3,(4,4),strides=(1,1),padding='same',activation='relu')(deg1)

model = Model(input,deg1)

return(model)

def use\_model(model,weights):

model.compile(Adam(0.0003),loss='binary\_crossentropy',metrics=['acc'])

model.load\_weights(weights)

return(model)

def predict(model,img):

image = cv2.resize(img,(256,256))/255

pred\_image = model.predict(image.reshape(-1,256,256,3)).reshape(256,256,3)

return(pred\_image)

## Drive Code

int m11 = 2;

int m12 = 3;

int m21 = 4;

int m22 = 5;

int dir = 0;

int ind\_f = 7;

int ind\_b = 8;

boolean go = false;

void setup() {

pinMode(ind\_b,OUTPUT);

pinMode(ind\_f,OUTPUT);

pinMode(m11,OUTPUT);

pinMode(m12,OUTPUT);

pinMode(m21,OUTPUT);

pinMode(m22,OUTPUT);

Serial.begin(9600);

}

void loop() {

if(go==false){

digitalWrite(m11,HIGH);

digitalWrite(m12,LOW);

digitalWrite(m21,LOW);

digitalWrite(m22,HIGH);

}

if(Serial.available()>0){

dir = Serial.read();

Serial.print("received : ");

Serial.println(dir);

if(dir == 50){

go=true;

digitalWrite(ind\_f,HIGH);

digitalWrite(ind\_b,LOW);

digitalWrite(m11,HIGH);

digitalWrite(m12,LOW);

digitalWrite(m21,LOW);

digitalWrite(m22,HIGH);

}

else if(dir == 54){

go=true;

digitalWrite(ind\_f,HIGH);

digitalWrite(ind\_b,HIGH);

digitalWrite(m11,HIGH);

digitalWrite(m12,LOW);

digitalWrite(m21,HIGH);

digitalWrite(m22,LOW);

}

else if(dir == 53){

go=true;

digitalWrite(ind\_f,LOW);

digitalWrite(ind\_b,HIGH);

digitalWrite(m11,LOW);

digitalWrite(m12,LOW);

digitalWrite(m21,LOW);

digitalWrite(m22,LOW);

}

else if(dir == 52){

go=true;

digitalWrite(ind\_f,HIGH);

digitalWrite(ind\_b,HIGH);

digitalWrite(m11,LOW);

digitalWrite(m12,HIGH);

digitalWrite(m21,LOW);

digitalWrite(m22,HIGH);

}

else if(dir == 56){

go=true;

digitalWrite(ind\_f,LOW);

digitalWrite(ind\_b,HIGH);

digitalWrite(m11,LOW);

digitalWrite(m12,HIGH);

digitalWrite(m21,HIGH);

digitalWrite(m22,LOW);

}

}

delay(200);

}

## Model Training Code

# -\*- coding: utf-8 -\*-

"""prets.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/15f7CcB92yySP0yzstXP9SU6cFcbOyfgm

"""

import tensorflow as tf

from tensorflow.keras.layers import Input, Reshape, Dropout, Dense

from tensorflow.keras.layers import Flatten, BatchNormalization

from tensorflow.keras.layers import Activation, ZeroPadding2D

from tensorflow.keras.layers import LeakyReLU,Concatenate,concatenate

from tensorflow.keras.layers import UpSampling2D, Conv2D,Conv2DTranspose,MaxPooling2D

from tensorflow.keras.models import Sequential, Model, load\_model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import LeakyReLU,Add

import tensorflow

import cv2

import numpy as np

from PIL import Image

from tqdm import tqdm

import os

import time

import matplotlib.pyplot as plt

from tensorflow.keras.utils import plot\_model

# Commented out IPython magic to ensure Python compatibility.

try:

from google.colab import drive

drive.mount('/content/drive', force\_remount=True)

COLAB = True

print("Note: using Google CoLab")

# %tensorflow\_version 2.x

except:

print("Note: not using Google CoLab")

COLAB = False

anns = open("/content/drive/MyDrive/TSRD-Train Annotation (1)/TsignRecgTrain4170Annotation.txt")

im,lab = [],[]

count = 0

g = len(os.listdir('/content/drive/MyDrive/tsrd-train (1)'))

for i in anns.readlines():

count += 1

print(count/g \* 100 , end = ' ')

f = i.split(';')

img = cv2.imread(os.path.join('/content/drive/MyDrive/tsrd-train (1)',f[0]))

dro = img.shape[0]/100.0

dri = img.shape[1]/100.0

img = cv2.resize(img,(100,100))

im.append(img)

for i in range(1,len(f)-2,2):

f[i] = int(f[i])/dri

f[i+1] = int(f[i+1])/dro

f[-2] = int(f[-2])

lab.append([int(fg) for fg in f[3:8]])

print(' ',end ='\r')

im = np.array(im)

im.reshape(-1,100,100,3)

lab = np.array(lab).reshape(-1,5)

model = Sequential()

model.add(Input((100,100,3)))

model.add(Conv2D(4,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(8,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(16,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(32,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Flatten())

model.add(Dense(1024,activation='relu'))

model.add(Dense(256,activation='relu'))

model.add(Dense(128,activation='relu'))

model.add(Dense(32,activation='relu'))

model.add(Dense(5,activation='relu'))

model.compile('adam',loss='log\_cosh',metrics=['acc'])

mod\_plot = plot\_model(model,'model.png',show\_dtype=True,dpi = 220,show\_shapes=True)

mod\_sum = model.summary()

model.fit(im,lab,epochs = 100,validation\_split=0.01,shuffle=True)

histo = model.history

plt.plot(histo.history['acc'])

plt.plot(histo.history['val\_acc'])

plt.show()

plt.plot(histo.history['loss'])

plt.plot(histo.history['val\_loss'])

plt.show()

model.save('/content/drive/MyDrive/sdcar/coords.h5')

model = Sequential()

model.add(Input((100,100,3)))

model.add(Conv2D(4,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(8,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(16,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(32,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64,(4,4),strides=(1,1),activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Flatten())

model.add(Dense(1024,activation='relu'))

model.add(Dense(256,activation='relu'))

model.add(Dense(128,activation='relu'))

model.add(Dense(58,activation='sigmoid'))

model.compile('adam',loss='binary\_crossentropy',metrics=['acc'])

laby = []

for i in range(len(lab)):

d = np.zeros(58)

d[lab[i][-1]] = 1

laby.append(d)

laby = np.array(laby).reshape(-1,58)

model.fit(im,laby,epochs = 20,validation\_split=0.01,shuffle=True)

model.save('/content/drive/MyDrive/sdcar/mod\_class.h5')

import tensorflow as tf

from tensorflow.keras.layers import Input, Reshape, Dropout, Dense

from tensorflow.keras.layers import Flatten, BatchNormalization

from tensorflow.keras.layers import Activation, ZeroPadding2D

from tensorflow.keras.layers import LeakyReLU,Concatenate,concatenate

from tensorflow.keras.layers import UpSampling2D, Conv2D,Conv2DTranspose,MaxPooling2D

from tensorflow.keras.models import Sequential, Model, load\_model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import LeakyReLU,Add

import tensorflow

import cv2

import numpy as np

from PIL import Image

from tqdm import tqdm

import os

import time

import matplotlib.pyplot as plt

from tensorflow.keras.utils import plot\_model

input = Input((256,256,3))

seg1 = Conv2D(16,(4,4),strides=(1,1),padding='same',activation='relu')(input)

m1 = MaxPooling2D((2,2))(seg1)

seg2 = Conv2D(16,(4,4),strides=(1,1),padding='same',activation='relu')(m1)

m2 = MaxPooling2D((2,2))(seg2)

seg3 = Conv2D(32,(4,4),strides=(1,1),padding='same',activation='relu')(m2)

m3 = MaxPooling2D((2,2))(seg3)

seg4 = Conv2D(32,(4,4),strides=(1,1),padding='same',activation='relu')(m3)

m4 = MaxPooling2D((2,2))(seg4)

seg5 = Conv2D(64,(4,4),strides=(1,1),padding='same',activation='relu')(m4)

m5 = MaxPooling2D((2,2))(seg5)

seg6 = Conv2D(64,(4,4),strides=(1,1),padding='same',activation='relu')(m5)

m6 = MaxPooling2D((2,2))(seg6)

seg7 = Conv2D(128,(4,4),strides=(1,1),padding='same',activation='relu')(m6)

m7 = MaxPooling2D((2,2))(seg7)

seg8 = Conv2D(128,(4,4),strides=(1,1),padding='same',activation='relu')(m7)

deg1 = Conv2DTranspose(128,(4,4),strides=(2,2),padding='same',activation='relu')(seg8)

deg1 = concatenate([deg1,seg7],axis=3)

deg1 = Conv2DTranspose(64,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg6],axis=3)

deg1 = Conv2DTranspose(64,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg5],axis=3)

deg1 = Conv2DTranspose(32,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg4],axis=3)

deg1 = Conv2DTranspose(32,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg3],axis=3)

deg1 = Conv2DTranspose(16,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg2],axis=3)

deg1 = Conv2DTranspose(16,(4,4),strides=(2,2),padding='same',activation='relu')(deg1)

deg1 = concatenate([deg1,seg1],axis=3)

deg1 = Conv2DTranspose(3,(4,4),strides=(1,1),padding='same',activation='relu')(deg1)

model = Model(input,deg1)

model.compile(Adam(0.0003),loss='binary\_crossentropy',metrics=['acc'])

plot\_model(model,'model.png',show\_dtype=True,dpi = 220,show\_shapes=True)

impath = "/content/drive/MyDrive/sdcar/image"

segpath = "/content/drive/MyDrive/sdcar/segs"

im = []

seg = []

g = len(os.listdir(impath))

count = 0

for i in os.listdir(impath):

count += 1

print(count/g \* 100 , end = ' ')

img = cv2.imread(os.path.join(impath,i))

img = cv2.resize(img,(256,256))

img = img/255

im.append(img)

segu = os.path.join(segpath,i[0:-4]+'.png')

img = cv2.imread(segu)

img = cv2.resize(img,(256,256))

img = img/255

seg.append(img)

print(' ',end ='\r')

if(count>500):

break

model.load\_weights("/content/drive/MyDrive/sdcar/segnet.h5")

im = np.array(im)

seg = np.array(seg)

np.save("/content/drive/MyDrive/im.npy",im)

np.save("/content/drive/MyDrive/seg.npy",seg)

im = np.load("/content/drive/MyDrive/im.npy")

seg = np.load("/content/drive/MyDrive/seg.npy")

# Commented out IPython magic to ensure Python compatibility.

# Load the TensorBoard notebook extension

# %reload\_ext tensorboard

import datetime

logdir = os.path.join("/content/drive/MyDrive/sdcar/logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

tensorboard\_callback = tf.keras.callbacks.TensorBoard(logdir, histogram\_freq=1)

!kill 325

# Commented out IPython magic to ensure Python compatibility.

# %tensorboard --logdir "/content/drive/MyDrive/sdcar/logs"

model.fit(im,seg,epochs=400,validation\_split=0.01,callbacks=[tensorboard\_callback])

plt.imshow(model.predict(im[100].reshape(-1,256,256,3)).reshape(256,256,3))

plt.imshow(model.predict(imf.reshape(-1,256,256,3)).reshape(256,256,3))

imf = cv2.resize(cv2.imread("/content/949586-516e.jpg"),(256,256))/255

plt.imshow(im[100])

plt.imshow(im[100])

plt.show()

plt.imshow(seg[100])

plt.show()

model.save("/content/drive/MyDrive/sdcar/segnet\_binco\_0003.h5")