RC Autonomous Vehicle Project

Joint Institute for Computational Sciences
University of Tennessee

Mentor: Dr Kwai Wong

Julian Halloy – University of Tennessee
Henry Lam – City University of Hong Kong
Beverly Chow – City University of Hong Kong

27 August 2021

Content

Abstract

Research on autonomous vehicles has been conducted for 96 years. It is possible to notice some autonomous vehicles running on the public road in the testing phase, but fatal traffic accidents happening raised safety concerns. To minimize the risks caused by an autonomous vehicle, car-following and edge networking are the long-term objectives of this project. The essential requirement is running the vehicle autonomously, so sign recognition is the short-term objective.

This project is an extensive research project in 2019. Sign recognition is the major focus of this report. The data used were the images collected in the building of the College of Education, University of Tennessee by the robot car. When comparing to the accuracy of the model using ResNet50 network with ImageAI, the accuracy of the model using ResNet18 network with TensorRT was higher. As for the edge networking, the socket module in python was utilized for sending files from server to client. After the completion of sign recognition, car-following would be started and edge networking would be continued.

Background

The autonomous vehicle is a vehicle equipped with an environment sensing system for reducing the reliance on human control unless an accident is likely to happen. However, the safety issue of autonomous vehicles has been raised. Tragic road accidents related to autonomous vehicles took place. The first fatal autonomous vehicle crash was caused by an Uber safety driver in Arizona on 18 March 2018. A woman was killed. Later, different kinds of autonomous vehicle accidents continued occurring. It is believed that the autonomous vehicle helped reduce traffic accidents, but an autonomous vehicle has a higher probability to be involved in accidents than a conventional vehicle (Petrović et al., 2020). Car-following helps autonomous vehicles learn the driving habit of humans (Masmoudi et al., 2019). Edge networking supports sharing information. Therefore, car-following and edge networking are designed to minimize the risk.

Before car-following and edge networking, sign recognition is the main task to run the autonomous vehicle. Classification, one of the functions of a convolution neural network, would be utilized in this task. The neural network is a subfield of the machine learning. The structure of neural networking applies the concept of the human brain. Neural network learns through backpropagation. The difference between the output that the network produce and the output that the network would like to produce modifies the weights of the connections between the units. The network would learn from it once a correct weight is obtained (Wasserman & Wasserman, 2019). Convolution neural network is one of the classes of neural network. Different slices of an image are applied to multiple filters in convolutional networks. It is followed by mapping them one by one and identifying different features of an input image.

As for model training, TensorRT which facilitates high-performance inference on NVIDIA GPUs is the chosen C++ library working with training frameworks. It would select the best algorithms for computation based on the target GPU and would remove useless tensor and memory during training to optimize memory (2021). ResNet18 is the default model for classification. This neural network is pre-trained on the ImageNet data set. It is 18 layers deep including 1 max pool layer, 16 convolution layers and 1 average pool layer. Residual Network (ResNet) solves a grave problem that is vanishing gradient. The gradient will be equal to zero if the network is too deep. This would stop

the update of the parameters of the network. To prevent vanishing gradient, ResNet provides a shortcut to skip connections. Therefore, the model using the ResNet18 network with TensorRT was used.

Objective

There are three objectives of this project. The short-term and main objective is sign recognition so that the vehicle can run autonomously. On the other hand, the long-term objectives are car-following and edge networking. Edge networking has been started but has not yet been completed. Car-following would be our future work.

Equipment

To construct an autonomous vehicle, ELEGOO Smart Robot Car V4.0 was brought. Apart from the robot car, there were 3D printed parts and parts added to the car. The equipment lists are shown below.

3D Painting

Quantity	Part	Note
1	Car Frame	With Camera Mount
1	Jetson Nano Case	
1	Camera Case	
1	Camera Arm	

Additional Part

Quantity	Part	Note
1	NVIDIA Jetson Nano Developer Kit	
1	Monitor	
1	Mouse	
1	Keyboard	
1	SanDisk 64GB Micro SD card	
1	Panda PAUO5 Wi-Fi Dongle	
1	Raspberry Pi Camera Module V2	
1	DROK Buck Converter Module	Adjust to 5V 4A output
1	Zeee 11.1V 1300mAh 120C LiPo Battery with XT60 Plug	
1	DC Male Power Cable Connector Plug Adapter	Solder to the output side of VRM
1	XT60 Male Connector	Solder to the input side of VRM
1	HDMI Cable	For the Monitor
2	M3x20mm Screw	For the Camera Arm
6	M3 Nuts	For the Car Frame
6	M3x40mm Single-pass Copper Cylinder	For the Car Frame

Tool

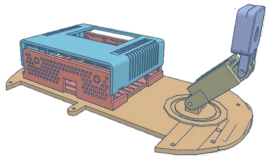
Quantity	Part	Note
3	Screwdrivers	Included in the Robot Car Kit
1	Multimeter	For the Converter Module
1	Soldering Iron and Solder	For soldering
1	Plier	For soldering
1	Insulating Tape	For soldering

Procedure

Construct a Robot Car

ELEGOO Smart Robot Car V4.0 was selected to be modified so that it can run autonomously. A two-layer robot car would be built by following the guidelines of the Assembly Tutorial included in the car kit. To equip the robot car, extra components were added. Jetson Nano, a small single-board computer with high processing power, was used to build an artificial intelligence application. The camera helped capture images. The battery and the buck converter were considered as a power source after soldering the male power cable connecter plug adapter and the male connector to the VRM and connecting the battery to the buck converter.

To place all the parts in the robot car, 3D parts were printed. We made use of the car frame to support Jetson Nano and the camera. The Jetson Nano case protected the Jetson Nano from damage and fixed the Jetson Nano on the car frame. The camera case protected and helped adjust the position of the camera. The camera arm was used for extension. Consequently, DC motors and wheels were included on the bottom layer. ELEGOO UNO R3 and battery were placed on the upper layer. Jetson and camera were on the top layer.



(Fig.1: The 3D Parts)

3D Printer Setting

Quality

Layer Height	0.15 mm

Shell

Wall Thickness	0.8 mm
Top/Bottom Thickness	0.6 mm

Infill

Infill Density	20%
Infill Pattern	Grid

Speed

Print Speed	60 mm/s
Wall Speed	30 mm/s
Top/Bottom Speed	30 mm/s
Travel Speed	120 mm/s
Initial Layer Speed	30 mm/s

Cooling

Fan Speed	100%
Minimum Layer Time	10 s
Minimum Speed	10 mm/s

Support

Support Overhand Angle	50 degree
Support Horizonal Expansion	0 mm

Build Plate Adhesion

Build Plate Adhesion Type	Skirt
Brim Width	5 mm

Set up Jetson Nano

NVIDIA Jetson Nano Developer Kit required a microSD card for storage. Before the first boot, we have to write the unzipped Jetson Nano Developer Kit SD Card Image which was downloaded from the website of NVIDIA to the microSD card. Inserting the microSD card to the Jetson Nano, we created a Linux account.

Power

Jetson Nano supported two power modes. Mode 0 was 5W power consumption while mode 1 was 10W power consumption. In this project, mode 1 was used for better working performance. Jetson Nano supported three power ports as well.

Port	Max Voltage and Max Current
Micro-USB Connector	5V 2.5A
Barrel Jack Connector	5V 4A
GPIO Header	5V 6A

Barrel jack connector would be port selected in this project. A two-pin header jumper shunt shorting block for pin J48 was required to enable the port. 5V 2A was enough to maintain 10W power consumption originally. However, peripherals were connected to the Jetson Nano. Thus, the voltage and current will be adjusted to 5V 4A by VRM if the LiPo battery is the power source.

Install jetson-inference

We installed jetson-inference to classify images with ImageNet. Packages were installed to create bindings for Python 3.6 after ensuring that git and CMake were installed. Then, we cloned the jetson-inference project. When configuring the project with CMake, the model downloader tool and PyTorch installer would run. We downloaded the default model and installed PyTorch. The installation would be ended by compiling the project.

```
sudo apt-get update
sudo apt-get install git cmake libpython3-dev python3-numpy
git clone --recursive https://github.com/dusty-nv/jetson-inference
cd jetson-inference
mkdir build
cd build
cmake ../
make -j$(nproc)
sudo make install
sudo ldconfig
```

Install PySerial and gdown

We installed PySerial to control the motors and run the vehicle manually. We enabled the pip3 command and install PySerial. gdown was installed for downloading files from Google Drive, such as “communicate-with-arduino.py” and “collect-img.py”.

```
apt install python3-pip
pip3 install pyserial
pip3 install gdown
```

Communicate with Arduino

"communicate-with-arduino.py" is a python script that gives instructions to the UNO R3. We downloaded it to Jetson Nano. "Arduino-code.ino" modified for the new version of the robot car is used to control the motors. It would be uploaded to UNO R3. Therefore, the vehicle can be controlled by running the python script and inputting the character commands.

Command	Action
w	Forward
s	Backwards
a	Left
d	Right
q	Stop
t	Return (Turn 180°)
r	Left 90
y	Right 90
z	Slow Down
x	Speed Up

Collect Data

We ran "communicate-with-arduino.py", which controlled the vehicle to move, and "collect-img.py", which controlled the camera to capture images, at the same time. Thus, the screens captured when performing real-time classification would be simulated. The images were collected in the building of the College of Education, University of Tennessee in 2019. 16540 photos were gathered in total.

These photos can be classified into "W", "L", "R", "TL", "TR" and "RE". When the classification result was class W, the vehicle would go forward. When the result was class L, the vehicle would turn left. When the result was class R, the vehicle would turn right. On the other hand, the photos that captured the "Turn Left Sign" printed on green paper belonged to "TL". The photos that captured the "Turn Right Sign" printed on yellow paper belonged to "TR". The photos that captured the "Return Sign" printed on red paper belonged to "RE".

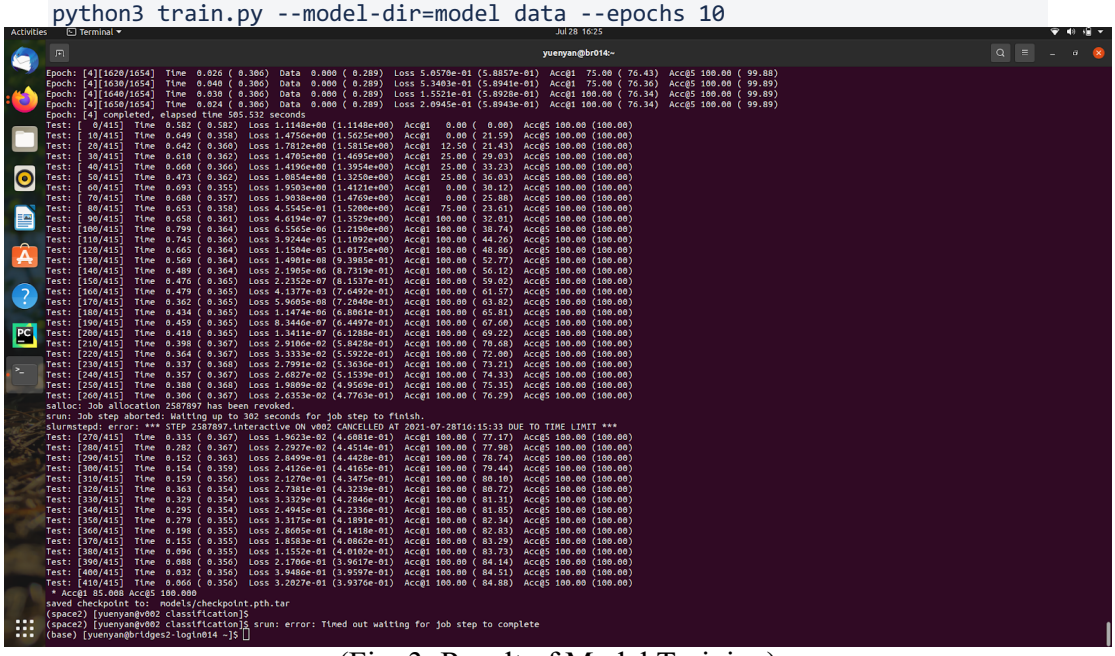
Then, 80% of the photos in each file would be put into the "train" file while the remaining photos would be put into the "test" file. After organizing the files, "train" and "test" had their own "W", "L", "R", "TL", "TR" and "RE" files. We also utilized the "test" file to create the "val" file. Lastly, these files would be transferred to Jetson Nano.

(Fig. 2: The Photos Collected)

Model Training

To perform classification, a model was required as an input. We can customize the training after providing the directory that the model would be saved. In this project, we used the default model, the ResNet18 network. The number of epochs was set to be 10. The batch size was 8.

```
python3 train.py --model-dir=model data --epochs 10
```



(Fig. 3: Result of Model Training)

We received Top-1 accuracy and Top-5 accuracy during the training. Top-1 accuracy was the accuracy that the model predicted correctly while Top-5 accuracy was the accuracy that the correct class was in the top five predictions. We could obtain the accuracy from the Top-1. In the end, it would show the Top-1 average accuracy as well.

To load the model with TensorRT, the model should be converted from PyTorch to Open Neural Network Exchange (ONNX). The model can be used in other frameworks for transfer learning.

```
python3 onnx_export.py --model-dir=models
```

Classification

Apart from the models, the class label was another input for classification. A txt file listing all the classes in alphabetical order was created. Originally, the "imagenet-camera.py" would capture and classify the image. We adjusted the python script by including the commands of controlling the motor so as to achieve self-driving.

After entering the arguments and running the python script, classification would be carried out. The vehicle would move depending on the prediction. For example, the vehicle would turn 180° autonomously when placing the "Return Sign" in front of the camera. If the confidence of the prediction was lowered than 0.3, it would wait for the next prediction that had higher confidence. To stop the program, we can just press "q" or close the pop-out window.

Devices' communication (Socket Module)

In the concept of Edge mesh, edge devices are not only executors but also computing units. In the Edge network, edge devices will do major decisions, which is different from the traditional network that use a centralized cloud to compute (Sahni Yuvraj, Jiannong Cao, Shigeng Zhang, & Lei Yang, 2017).

For developing the communication between edge devices, we used the socket module in python language, which is a module for communicating through defining server and client machines by socket endpoints.

We first set one vehicle as the host of the server and other vehicles will connect to it as clients. By inserting the socket module, we can set up the pipeline for different message types including sentences and files and archiving communication between devices.

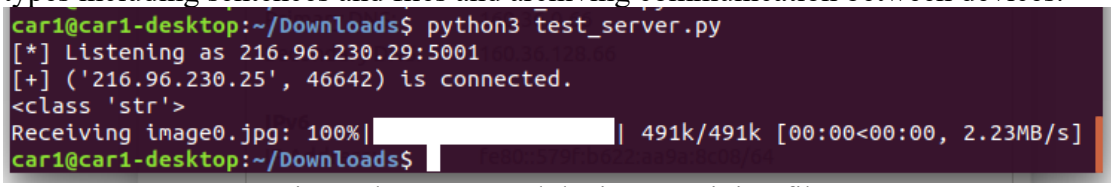


Fig.4a the server end devices receiving files

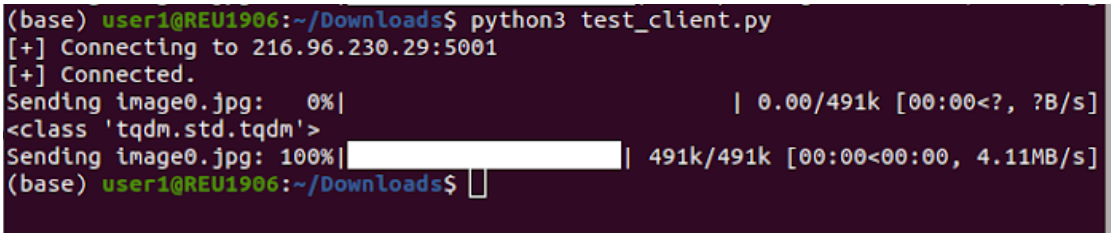
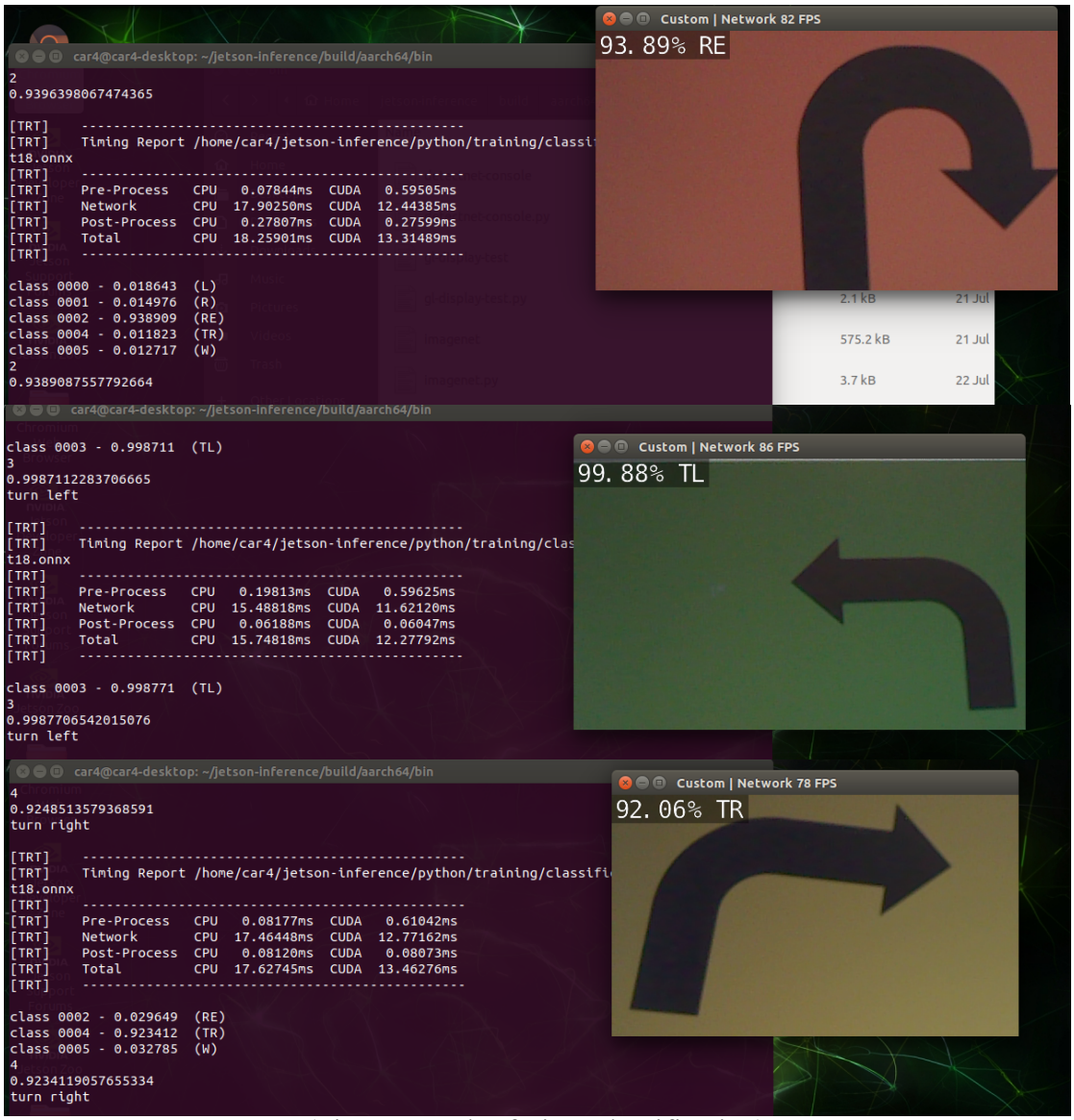


Fig.4b the client end devices transferring files

Result

We first made use of the ResNet network from the ImageAI library to train several models. However, the accuracy of these models was not satisfactory. They gave the wrong solutions when inputting the testing set to the models. For example, they would say that picture showing the return sign belonged to class W. Thus, we turned to TensorRT. The accuracy of the new model trained by the ResNet18 network was much higher. From Fig. 4, the probability could be higher than 90%. The new model was more reliable.

However, there were numerous problems that we needed to tackle. If we run the python script via SSH, the program will not be stopped properly. Lots of memories would be consumed. As a result, we have to reboot the Jetson Nano after running the script twice. Besides, the speed of sign recognition prediction was too high. It generated an immense number of commands to the vehicle. Consequently, the vehicle would run in the wrong direction and get out of control.



(Fig. 5: Result of Sign Classification)

Analysis

According to the result section, the newly trained model with TensorRT with high accuracy reached 90% in most circumstances. Considering that the data sets used for training are the same as the previous research team, some factors give explanations for the great difference between the results. First, the difference can be due to the replacement of ImageAI to TensorRT. We used Imagenet in TensorRT. In fact, Imagenet is the largest of its kind in terms of database size, they have 100,000 images across 200 classes, which makes the Resnet more accurate and validated than ImageAI with more reference.

Moreover, the previous team mentioned that when they were doing the training, they did not have enough images for all output classes after they decided to introduce new output classes, even they collected enough photos after that due to the short of time. However, because we use the old datasets which have been already sorted out into respective classes by the previous team, this probably is a part of the reasons why our result is better.

Besides, TensorRT is designed and optimized for Nvidia's product including Jetson-Nano, compare to Tensorflow and ImageAI, the size will be smaller for installing TensorRT, as we do not require to do swap file before we install it, which give greater space for Jetson-Nano to process the prediction with lower latency.

However, about the SSH problem with remote activating the autonomous function on the Jetson-Nano car, we think it might relate to the display window of the camera view. We had a similar problem when we were working on the ImageAI, it seems like because the display window may sometimes be opened on the computer doing SSH or have a delay, bugs will happen in the program and make it out of control.

Future Works

In the future, there are some goals we want to continue working on. In short term, we want to improve the accuracy of the image capturing function. The previous research team mentioned that random features of the environment of the training images taken like diverse light and objects as recycle bins will affect the accuracy of A.I. training, so we would like to train a new model with newly taken images in a relatively distraction free location in CityU.

In long term, we would like to work on the car following features after training the new model. We will try to use another vehicle to follow the leading autonomous vehicle, which is similar to the image capturing autonomous feature, but it takes the leading vehicle as the signal source instead and it will be comparably harder to achieve since the A.I. cannot rely on colour to predict anymore. Moreover, we will continue to work on edge devices mesh, which we currently built up a communication method between edge devices by socket module, and next we will work on having a not specified host by any devices within the mesh and parallel computing on multiple devices and merge the result by edge mesh.

Conclusion

For this research period, the conclusion is the new platform TensorRT resolved the major problem of Jetson-Nano's memory limitation which also improve the feature's performance compare with the old set of software. Jetson-Nano with TensorRT may have a greater possibility to access some advanced research work which provides greater space for future research on this project.

Acknowledgements

This material is based upon work performed using computational resources supported by the University of Tennessee and Oak Ridge National Laboratory Joint Institute for Computational Sciences (http://www.jics.tennessee.edu). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the University of Tennessee, Oak Ridge National Laboratory, or the Joint Institute for Computational Sciences.

This project was mentored by and Dr Kwai Wong (The University of Tennessee). This project used allocations from the Extreme Science and Engineering Discovery Environment (XSEDE) supported by the National Science Foundation

Appendix

Appendix I: Arduino-code.ino

```
/*define logic control output pin*/
//For Car 3.0
int inR=8;
int inL=7;
/*define channel enable output pins*/
int ENA=5;
int ENB=6;


/*Slow Speed of Car*/
int Sspd_L = 60;
int Sspd_R = 60;

/*Average Speed of Car (0-255) */
int Aspd_L = 100;
int Aspd_R = 100;

/*Fast Speed of Car*/
int Fspd_L = 135;
int Fspd_R = 135;

/*Fast Fast Speed of Car*/
int FFspd_L = 255;
int FFspd_R = 255;
void setup() {

 /*Open the serial port and set the baud rate to 9600 */
 Serial.begin(9600);

/*Set the defined pins to the output*/
  pinMode(inR,OUTPUT);
  pinMode(inL,OUTPUT);
  pinMode(ENA,OUTPUT);
  pinMode(ENB,OUTPUT);

  digitalWrite(ENA, LOW);
  digitalWrite(ENB, LOW);

/* wait for serial port to connect. */
  while (!Serial) {
    ;

}
}
/*define forward function*/
void _mForward_N()
{
  analogWrite(ENA, Aspd_L);
  analogWrite(ENB, Aspd_R);
  digitalWrite(inL,HIGH);//digital output
  digitalWrite(inR,LOW);

  /*Serial.println("Forward");*/
```

```
}

/*define back function*/
void _mBack()
{
 analogWrite(ENA, Aspd_R);
 analogWrite(ENB, Aspd_L);
 digitalWrite(inL,LOW);
 digitalWrite(inR,HIGH);
 /*Serial.println("Back");*/
}

/*define left function*/
void _mleft()
{
 analogWrite(ENA, Fspd_R);
 analogWrite(ENB, Aspd_L);
 digitalWrite(inL,HIGH);
 digitalWrite(inR,LOW);

 /*Serial.println("Left"); */
}

/*define right function*/
void _mright()
{
 analogWrite(ENA, Aspd_R);
 analogWrite(ENB, Fspd_L);
 digitalWrite(inL,HIGH);
 digitalWrite(inR,LOW);
 /*Serial.println("Right"); */
}

void _mTR()
{
 analogWrite(ENA, Fspd_R);
 analogWrite(ENB, FFspd_L);
 digitalWrite(inL,HIGH);
 digitalWrite(inR,LOW);
}

void _mTL()
{
 analogWrite(ENA, FFspd_R);
 analogWrite(ENB, Fspd_L);
 digitalWrite(inL,HIGH);
 digitalWrite(inR,LOW);
}

void _mReturn()
{
 analogWrite(ENA, Fspd_R);
 analogWrite(ENB, FFspd_L);
 digitalWrite(inL,LOW);
 digitalWrite(inR,LOW);
```

```
  }

void _mStop()
{
  digitalWrite(ENA, LOW);
  digitalWrite(ENB, LOW);

  /*Serial.println("Stop"); */
}

void _mAccel()
{
  analogWrite(ENA, Fspd_R);
  analogWrite(ENB, Fspd_L);
  digitalWrite(inL,HIGH);
  digitalWrite(inR,LOW);
}

void _mDeccel()
{
  analogWrite(ENA, Sspd_R);
  analogWrite(ENB, Sspd_L);
  digitalWrite(inL,HIGH);
  digitalWrite(inR,LOW);
}

void loop() {
  char buffer[16];

  /*if we get a command */
  if (Serial.available() > 0) {
    int size = Serial.readBytesUntil('\n', buffer, 12);
    if (buffer[0] == 'W') {
      _mForward_N();
    }

    if (buffer[0] == 'A') {
      _mleft();
    }

    if (buffer[0] == 'S') {
      _mBack();
    }

    if (buffer[0] == 'D') {
      _mright();
    }

    if (buffer[0] == 'Q'){
      _mStop();
    }

    if (buffer[0] == 'X'){
      _mAccel();
    }
```

```
    if (buffer[0] == 'Z'){
      _mDeccel();
    }

    if (buffer[0] == 'T'){
     _mReturn();
    }

    if (buffer[0] == 'R'){
     _mTL();

   }

    if (buffer[0] == 'Y'){
     _mTR();


   }
}

}
```

Appendix II: communicate-with-arduino.py

```
import serial

with serial.Serial('/dev/ttyACM0', 9600, timeout=10) as ser:
   while True:
      move = input('')[0]
      if move in 'wW':
         ser.write(bytes('W\n','utf-8'))
         print('forward')
      if move in 'sS':
         ser.write(bytes('S\n','utf-8'))
         print('backward')
      if move in 'aA':
         ser.write(bytes('A\n','utf-8'))
         print('left')
      if move in 'dD':
         ser.write(bytes('D\n','utf-8'))
         print('right')
      if move in 'qQ':
         ser.write(bytes('Q\n','utf-8'))
         print('stop')
      if move in 'xX':
         ser.write(bytes('X\n','utf-8'))
         print('speed up')
      if move in 'zZ':
         ser.write(bytes('Z\n','utf-8'))
         print('slow down')

      if move in 'tT':
         ser.write(bytes('T\n','utf-8'))
         print('Return')
      if move in 'rR':
         ser.write(bytes('R\n','utf-8'))
         print('Left 90')
      if move in 'yY':
         ser.write(bytes('Y\n','utf-8'))
         print('Right 90')
```

Appendix III: collect-img.py

```
import numpy as np
import cv2
import os


#====================
# Define the camera
#====================
#GStreamer is used to interface with camera on the Jetson Nano.
#The code below requests GStreamer to create a camera stream with 1920 x 1080 at
15 frames per second.
#"nvvidconv flip-method" is useful when the mounting of the camera is of a
different orientation than the default.
# 0 = Default, no rotation
# 1 = Rotate 90 degrees counter-clockwise
# 2 = Rotate 180 degrees
# 3 = Rotate 90 degrees clockwise
# 4 = Flip horizontally
# 5 = Flip across upper right and lower left diagonal
# 6 = Flip vertically
# 7 = Flip across upper left and lower right diagonal
def gstreamer_pipeline (capture_width=1920, capture_height=1080,
display_width=1920, display_height=1080, framerate=15, flip_method=2) :
    return ('nvarguscamerasrc ! '
    'video/x-raw(memory:NVMM), '
    'width=(int)%d, height=(int)%d, '
    'format=(string)NV12, framerate=(fraction)%d/1 ! '
    'nvvidconv flip-method=%d ! '
    'video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! '
    'videoconvert ! '
    'video/x-raw, format=(string)BGR ! appsink'  %
(capture_width,capture_height,framerate,flip_method,display_width,display_height
))


cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=2),
cv2.CAP_GSTREAMER)

#============================
#Make directory for images
#============================
num = 0
while os.path.exists('images{}'.format(num)):
    num += 1

# Create file path for images
dirName = 'images{}'.format(num)
os.mkdir(dirName)
print("Directory " , dirName ,  " Created ")
```

```
img_num = 0

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        cv2.imwrite('%s/image%d.jpg' % (dirName,img_num),frame)
        print('frame captured%d' % (img_num))
        img_num += 1
        # Specifies display size
        display = cv2.resize(frame,(640,480))
        cv2.imshow('display',display)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            print('Pressed Q')
            break
    else:
        break



#=========================================
# Release everything if job is finished
#=========================================
cap.release()
cv2.destroyAllWindows()
```

Appendix IV: imagenet-camera.py

```
# Copyright (c) 2020, NVIDIA CORPORATION. All rights reserved.
#
# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN
NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE
USE OR OTHER
# DEALINGS IN THE SOFTWARE.
#

import jetson.inference
import jetson.utils

import argparse
import sys

import serial

turn=0

# parse the command line
parser = argparse.ArgumentParser(description="Classify a live camera stream using
an image recognition DNN.",
                                 formatter_class=argparse.RawTextHelpFormatter,
epilog=jetson.inference.imageNet.Usage() +
                                 jetson.utils.videoSource.Usage() +
jetson.utils.videoOutput.Usage() + jetson.utils.logUsage())

parser.add_argument("input_URI", type=str, default="", nargs='?', help="URI of
the input stream")
parser.add_argument("output_URI", type=str, default="", nargs='?', help="URI of
the output stream")
```

```
parser.add_argument("--network", type=str, default="googlenet", help="pre-trained model to load (see below for options)")
parser.add_argument("--camera", type=str, default="0", help="index of the MIPI CSI camera to use (e.g. CSI camera 0)\nor for VL42 cameras, the /dev/video device to use.\nby default, MIPI CSI camera 0 will be used.")
parser.add_argument("--width", type=int, default=1280, help="desired width of camera stream (default is 1280 pixels)")
parser.add_argument("--height", type=int, default=720, help="desired height of camera stream (default is 720 pixels)")
parser.add_argument('--headless', action='store_true', default=(), help="run without display")

is_headless = ["--headless"] if sys.argv[0].find('console.py') != -1 else [""]

try:
	opt = parser.parse_known_args()[0]
except:
	print("")
	parser.print_help()
	sys.exit(0)


# load the recognition network
net = jetson.inference.imageNet(opt.network, sys.argv)

# create video sources & outputs
input = jetson.utils.videoSource(opt.input_URI, argv=sys.argv)
output = jetson.utils.videoOutput(opt.output_URI, argv=sys.argv+is_headless)
font = jetson.utils.cudaFont()

with serial.Serial('/dev/ttyACM0', 9600, timeout=10) as ser:
	# process frames until the user exits
	while True:
		# capture the next image
		img = input.Capture()

		# classify the image
		class_id, confidence = net.Classify(img)
		print(class_id)
		print(confidence)

		if (confidence > 0.3):
			if  (class_id == 5):
				ser.write(bytes('W\n','utf-8'))
				turn = 0
				print('forward')

			if  (class_id == 0):
				ser.write(bytes('A\n','utf-8'))
				turn = 0
```

```
                                print('left')

                        if  (class_id == 1):
                                ser.write(bytes('D\n','utf-8'))
                                turn = 0
                                print('right')

                        if  (class_id == 3):
                                ser.write(bytes('R\n', 'utf-8'))
                                print('turn left')
                                #ser.write(bytes('W\n','utf-8'))
                                #time.sleep(0.5)

                        if  (class_id == 4):
                                ser.write(bytes('Y\n', 'utf-8'))
                                print('turn right')
                                turn = 0

                        if  (class_id == 2):
                                if turn == 0:
                                        turn = 1
                                        ser.write(bytes('T\n','utf-8'))
                                        print('Return')

                                else:
                                        ser.write(bytes('W\n','utf-8'))

                # find the object description
                class_desc = net.GetClassDesc(class_id)

                # overlay the result on the image
                font.OverlayText(img, img.width, img.height, "{:05.2f}%
{:s}".format(confidence * 100, class_desc), 5, 5, font.White, font.Gray40)

                # render the image
                output.Render(img)

                # update the title bar
                output.SetStatus("{:s} | Network {:.0f}
FPS".format(net.GetNetworkName(), net.GetNetworkFPS()))

                # print out performance info
                net.PrintProfilerTimes()

                # exit on input/output EOS
                if not input.IsStreaming() or not output.IsStreaming():
                        ser.write(bytes('Q\n','utf-8'))
                        print('Stop')

                        break
```

Reference

Masmoudi, M., Ghazzai, H., Frikha, M., & Massoud, Y. (2019). Autonomous car-following approach based on real-time video frames processing. *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. https://doi.org/10.1109/icves.2019.8906299

NVIDIA TensorRT. NVIDIA Developer. (2021, August 17). https://developer.nvidia.com/tensorrt.

Petrović, Đ., Mijailović, R. & Pešić, D. (2020). Traffic Accidents with Autonomous Vehicles: Type of Collisions, Manoeuvres and Errors of Conventional Vehicles' Drivers. *Transportation Research Procedia, 45*, 161–168. https://doi.org/10.1016/j.trpro.2020.03.003

Sahni, Yuvraj, Jiannong Cao, Shigeng Zhang, & Lei Yang. (2017). Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things. *IEEE Access*, *5*, 16441–16458. https://doi.org/10.1109/ACCESS.2017.2739804

Wasserman, T., & Wasserman, L. D. (2019). Understanding Functional Neural Networks. In *Therapy and the neural network model* (pp. 27–43). Springer International Publishing.