Self-driving toy car based on deep learning module

Peixin Li pxli@bu.edu
Lijun Xiao ljxiao@bu.edu

1 Discussion of failure

1.1 Video Streaming

In previous project proposal , we planed to broadcast running video of car via webpage whilst it analyzing road condition to make decision on the server side. During the implementation process, we used multithreading that one thread is that we processed image frame by frame after picam captured image data into buffer, the other thread is sending image to front-end webpage to show. But, somehow, when we run this part of code, we found that one of threads would be suspended and ran into deadlock. Due to the time limitation, we didn't find the right solution to better solve this problem.

1.2 Classification

The road shape is very simple, due to the very time-consuming training process since we need to try different parameters and either adding or deleting certain layers and the procedure of gathering images. We did gathered over 30,000 pieces of images on the previous lane but the procedure of filtering image and training process is too long so we decide to change the lane to a simpler one. Basically the thing that we did not implemented in original proposal is that what we suppose to do is to classify the images into 5 classes which are forward, backward, left, and stop. But we simplify the model into classify image into 2 classes which are forward and right.

1.3 Object detection

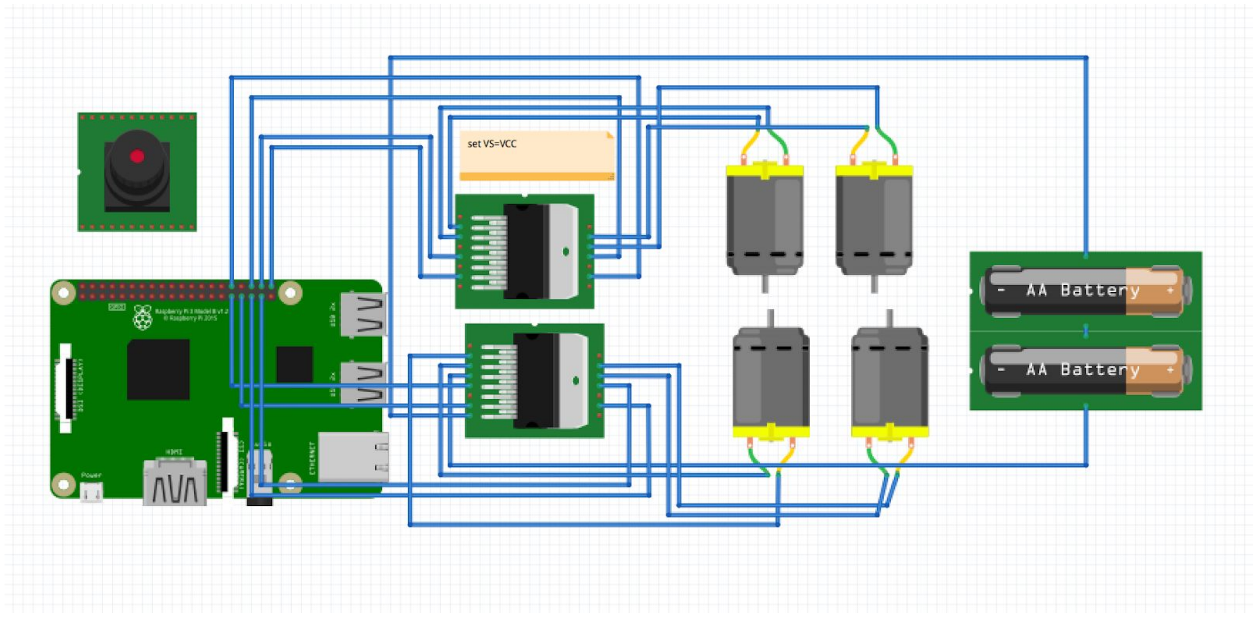
We did look at algorithms like yolo and surf, but due to the complexity and the lack of time we did not actually include these algorithms and also we did not have time to add some road signs and blocks for the model to detect. so in the end what we did is the basic CNN model. If we simulate real road signs, then we probably need to take photos like red lights from far away and very close distances and from all angles.

2. Technical report of success

2.1 Software/Hardware architecture

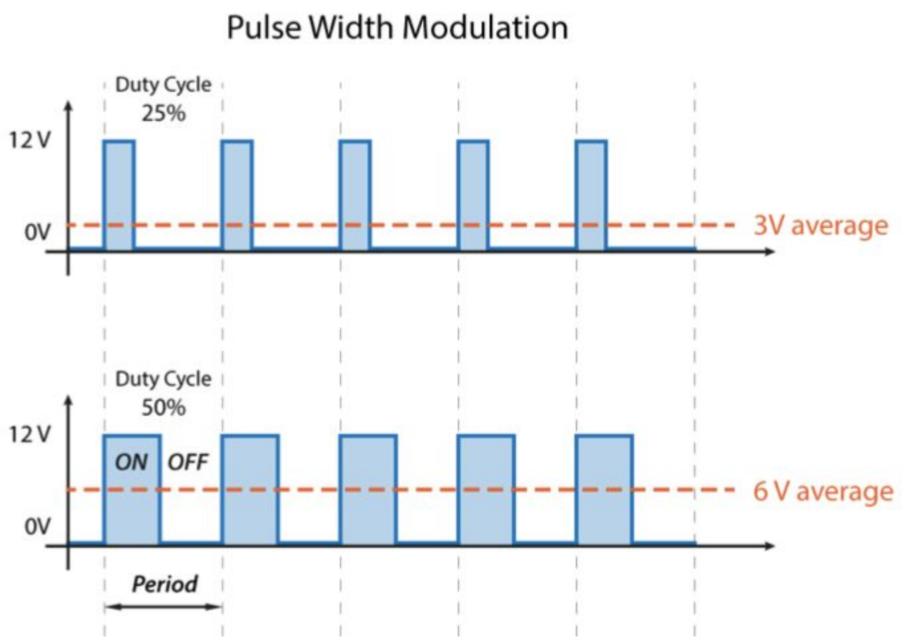
2.1.1 Hardware architecture

Here is the overall hardware architecture of our self-driving car. We used Raspberry Pi 3 B+ and L298N as motor driver, Pi camera as image caption hardware. We connected raspberry pi to wifi and in order to control raspberry pi we used VNC viewer to look into raspberrypi.local.



2.1.1.1 Driving Component

We used L298N H bridge to control the rotation direction of DC motors. The basic idea of changing direction in a common RC car is letting two motors working in same or reverse direction. By using H bridge circuit, we can activate two particular switches at the same time to change the flow direction. Besides, we can control the speed of the DC motor by simply controlling the input voltage to the motor and the most common method of doing that is by using PWM(pulse width modulation) signal. By controlling the duty cycle, we can get an average voltage that scaled down from the previous value.



2.1.1.2 Video and Image Caption Component

We use PiCamera in our project to capture image data and video. Picam have a python library which is easy to apply. We can just import picamera and run *camera.capture* to store image data either in file format or bytes buffer.

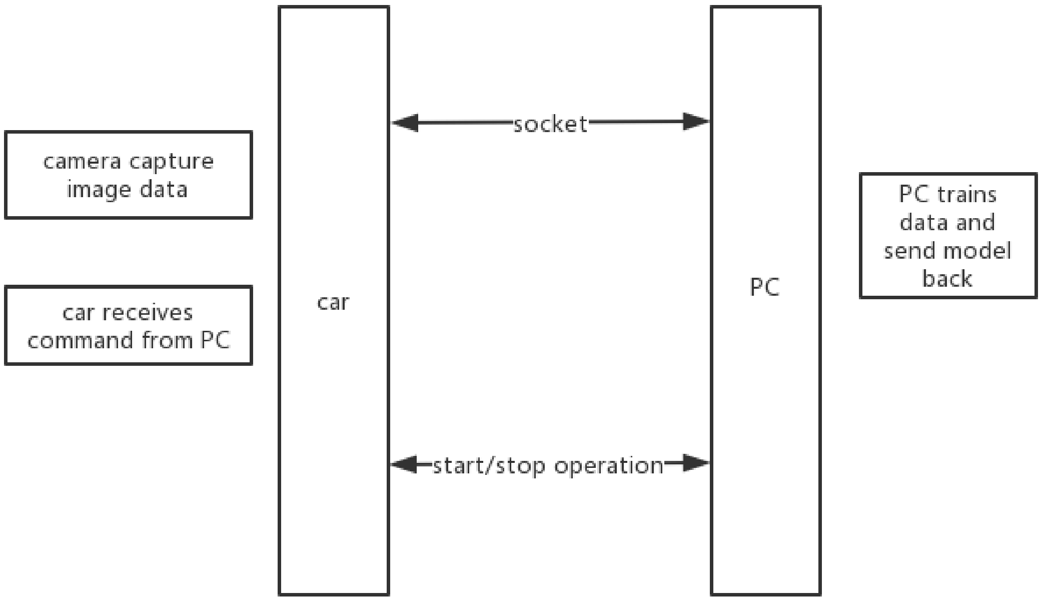
2.1.2 Software Architecture

2.1.2.1 Software Requirement

- Image collection by car
- PC trains model and send it back to toy car
- PC side sends start operation to run project
- Toy car receives command and analyze input image
- Toy car runs trained model and make decision

2.1.2.2 Design architecture

We first control toy car using imported pygame library in python. Pygame can detect keyboard pressing, car would change running direction based on user input. In the same time, pi camera could capture corresponding image data using multithreading programming. After these, we use deep learning library to analyze image and return the prediction with the highest probability among four directions. Then, we implemented socket to communicate in full-duplex.


camera capture image data
car receives command from PC
car
socket
start/stop operation
PC
PC trains data and send model back

2.2 Video and Image Capture and Process

2.2.1 Image Collecting

In the data-collecting phase, we use two threads, one is for controlling car through keyboard, the other is to use picamera to record image frames at framerate = 30. We set the pi camera capturing thread as User Thread and car_control thread as Daemon Thread. The car engine is driven by L298N board, and raspberry pi provides control signals according to keyboard input (in this part we used pygame which is preinstalled in the raspberry pi to detect keyboard input). Keys and their corresponding direction are as follows: w - forward, s - backwards, a - left, d - right, q - stop. The keyboard input is recorded and saved as part of the image name. We name the images collected in every round as 'image+#image+#round+corresponding direction' so that we can change the round number for the images we collect every circle to make sure they do not overlap.

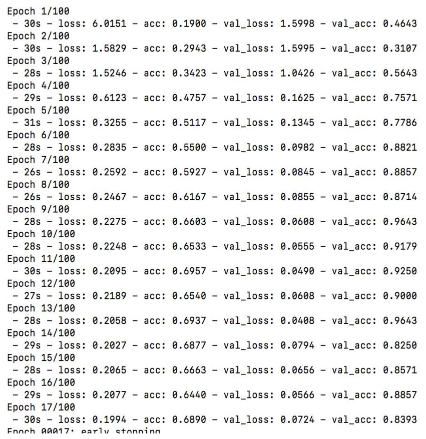
2.2.2 Processing images

- First we convert input images to npz format, npz is a simple zip archive, which contains numpy files.

- Manually delete some pictures that either captured image that does not contain road information or the direction is not what the image should corresponds to.

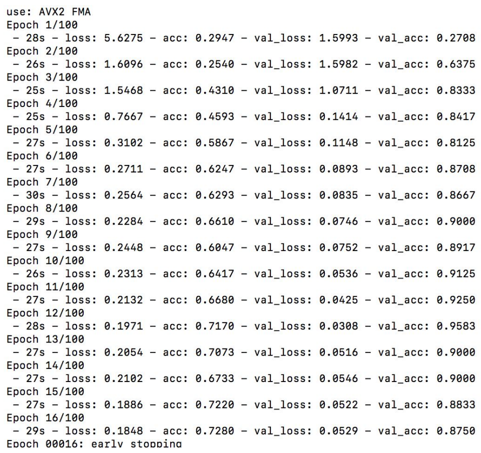
2.3 Training model

2.3.1 The accuracy of our model



```
Epoch 1/100
 - 30s - loss: 6.0151 - acc: 0.1900 - val_loss: 1.5998 - val_acc: 0.4643
Epoch 2/100
 - 30s - loss: 1.5829 - acc: 0.2943 - val_loss: 1.5995 - val_acc: 0.3107
Epoch 3/100
 - 28s - loss: 1.5246 - acc: 0.3423 - val_loss: 1.0426 - val_acc: 0.5643
Epoch 4/100
 - 29s - loss: 0.6123 - acc: 0.4757 - val_loss: 0.1625 - val_acc: 0.7571
Epoch 5/100
 - 31s - loss: 0.3255 - acc: 0.5117 - val_loss: 0.1345 - val_acc: 0.7786
Epoch 6/100
 - 28s - loss: 0.2835 - acc: 0.5500 - val_loss: 0.0982 - val_acc: 0.8821
Epoch 7/100
 - 26s - loss: 0.2592 - acc: 0.5927 - val_loss: 0.0845 - val_acc: 0.8857
Epoch 8/100
 - 26s - loss: 0.2467 - acc: 0.6167 - val_loss: 0.0855 - val_acc: 0.8714
Epoch 9/100
 - 28s - loss: 0.2275 - acc: 0.6603 - val_loss: 0.0608 - val_acc: 0.9643
Epoch 10/100
 - 28s - loss: 0.2248 - acc: 0.6533 - val_loss: 0.0555 - val_acc: 0.9179
Epoch 11/100
 - 30s - loss: 0.2095 - acc: 0.6957 - val_loss: 0.0490 - val_acc: 0.9250
Epoch 12/100
 - 27s - loss: 0.2189 - acc: 0.6540 - val_loss: 0.0608 - val_acc: 0.9000
Epoch 13/100
 - 28s - loss: 0.2058 - acc: 0.6937 - val_loss: 0.0408 - val_acc: 0.9643
Epoch 14/100
 - 29s - loss: 0.2027 - acc: 0.6877 - val_loss: 0.0794 - val_acc: 0.8250
Epoch 15/100
 - 28s - loss: 0.2065 - acc: 0.6663 - val_loss: 0.0656 - val_acc: 0.8571
Epoch 16/100
 - 29s - loss: 0.2077 - acc: 0.6440 - val_loss: 0.0566 - val_acc: 0.8857
Epoch 17/100
 - 30s - loss: 0.1994 - acc: 0.6890 - val_loss: 0.0724 - val_acc: 0.8393
Epoch 00017: early stopping
```

previous lane



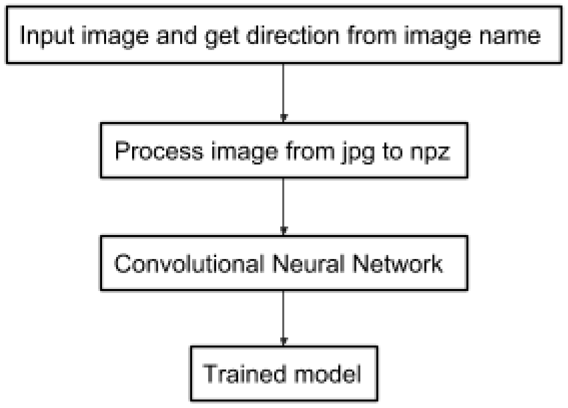
```
use: AVX2 FMA
Epoch 1/100
 - 28s - loss: 5.6275 - acc: 0.2947 - val_loss: 1.5993 - val_acc: 0.2708
Epoch 2/100
 - 26s - loss: 1.6096 - acc: 0.2540 - val_loss: 1.5982 - val_acc: 0.6375
Epoch 3/100
 - 25s - loss: 1.5468 - acc: 0.4310 - val_loss: 1.0711 - val_acc: 0.8333
Epoch 4/100
 - 25s - loss: 0.7667 - acc: 0.4593 - val_loss: 0.1414 - val_acc: 0.8417
Epoch 5/100
 - 27s - loss: 0.3102 - acc: 0.5867 - val_loss: 0.1148 - val_acc: 0.8125
Epoch 6/100
 - 27s - loss: 0.2711 - acc: 0.6247 - val_loss: 0.0893 - val_acc: 0.8708
Epoch 7/100
 - 30s - loss: 0.2564 - acc: 0.6293 - val_loss: 0.0835 - val_acc: 0.8667
Epoch 8/100
 - 29s - loss: 0.2284 - acc: 0.6610 - val_loss: 0.0746 - val_acc: 0.9000
Epoch 9/100
 - 27s - loss: 0.2448 - acc: 0.6047 - val_loss: 0.0752 - val_acc: 0.8917
Epoch 10/100
 - 26s - loss: 0.2313 - acc: 0.6417 - val_loss: 0.0536 - val_acc: 0.9125
Epoch 11/100
 - 27s - loss: 0.2132 - acc: 0.6680 - val_loss: 0.0425 - val_acc: 0.9250
Epoch 12/100
 - 28s - loss: 0.1971 - acc: 0.7170 - val_loss: 0.0308 - val_acc: 0.9583
Epoch 13/100
 - 27s - loss: 0.2054 - acc: 0.7073 - val_loss: 0.0516 - val_acc: 0.9000
Epoch 14/100
 - 27s - loss: 0.2102 - acc: 0.6733 - val_loss: 0.0546 - val_acc: 0.9000
Epoch 15/100
 - 27s - loss: 0.1886 - acc: 0.7220 - val_loss: 0.0522 - val_acc: 0.8833
Epoch 16/100
 - 29s - loss: 0.1848 - acc: 0.7280 - val_loss: 0.0529 - val_acc: 0.8750
Epoch 00016: early stopping
```

ultimate lane

2.3.2 Convolutional Neural Network

The convolutional neural network we build here includes Sequential, Conv2D, Dropout, Flatten, Dense layers.

We use elu as activation function. The whole dataset is divided into training set and testing set using train_test_split function from sklearn.model_selection, the test set size is set to be 20 percent. The model is trained from the training set and the prediction accuracy is calculated by the comparing the difference between the predicted result and the actual result in the test set.


Input image and get direction from image name
Process image from jpg to npz
Convolutional Neural Network
Trained model

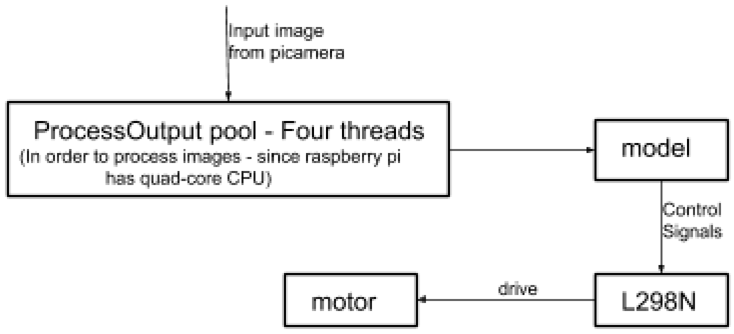
Training Model

2.3.3 Keras modules

We first refer to the keras hadwritten digit recognition example (which can be found in https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py). This dataset can be used by *from keras.datasets import mnist*. We build a similar convolutional neural network and tried different ModelCheckpoint: Save the model after every epoch.

During the training process we monitor the val_loss, if this value stops improving then the training process stops. The val_acc parameter is the accuracy on the testing set. In general, if loss and acc keeps improving but the val_acc and val_loss stops improving then this model might be overfitting. So we set the monitor value to be the val_loss to prevent the model from overfitting. In the above left side figure, the model trained from epoch 9, 11, 13 gets the acc around 0.69 and validation accuracy around 0.96. So we pick these three models and tried these on the car. In real practice situation, model 13 seems to be the best one, and it is indeed the stopping point we get from the function of earlystopping. Figure on the right side above is the model we get from a different set of parameters. These two set of parameters works best among all the parameters we tried among the process.

2.4 Self-driving

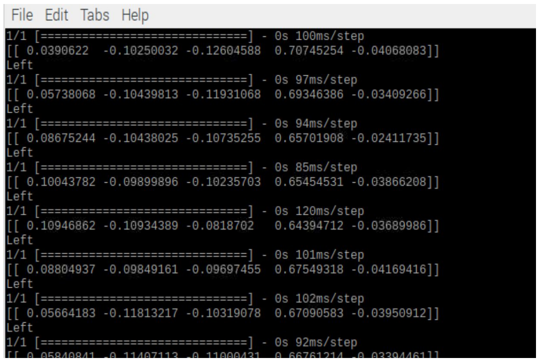

Input image
from picamera
ProcessOutput pool - Four threads
(In order to process images - since raspberry pi has quad-core CPU)
model
Control Signals
L298N
drive
motor

Self-driving

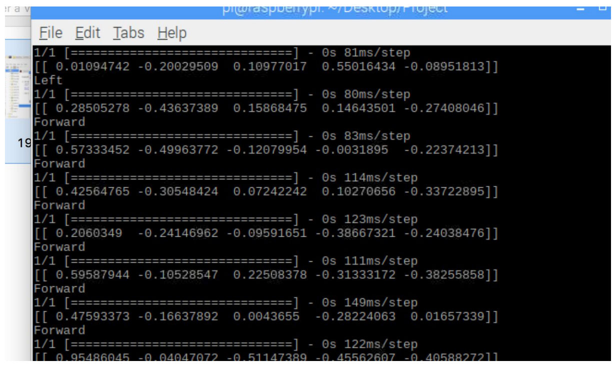
When trying the module we find at first that even though the validation accuracy seems to be pretty good, the predicted direction is unbelievably bad. After trying for pretty long time we decide to simplify our 'road' to make it easier for the model to predict directions.

During the self-driving process, we first use the Picamera.start_recording function which takes the custom output file-like object ProcessOutput(), in this object there are four ImageProcessor threads in a pool, the four threads process image to numpy array and use loaded model as a global variable and get the corresponding direction of max probability and give it to action number variable. Then the car control function controls the car according to the action number.

2.4.1 Prediction results during the self-driving process



```
1/1 [==============================] - 0s 100ms/step
[[ 0.0390622  -0.10250032 -0.12604588  0.70745254 -0.04068083]]
Left
1/1 [==============================] - 0s 97ms/step
[[ 0.05738068 -0.10439813 -0.11931068  0.69346386 -0.03409266]]
Left
1/1 [==============================] - 0s 94ms/step
[[ 0.08675244 -0.10438025 -0.10735255  0.65701908 -0.02411735]]
Left
1/1 [==============================] - 0s 85ms/step
[[ 0.10043782 -0.09899896 -0.10235703  0.65454531 -0.03866208]]
Left
1/1 [==============================] - 0s 120ms/step
[[ 0.10946862 -0.10934389 -0.0818702   0.64394712 -0.03689986]]
Left
1/1 [==============================] - 0s 101ms/step
[[ 0.08804937 -0.09849161 -0.09697455  0.67549318 -0.04169416]]
Left
1/1 [==============================] - 0s 102ms/step
[[ 0.05664183 -0.11813217 -0.10319078  0.67090583 -0.03950912]]
Left
1/1 [==============================] - 0s 92ms/step
[[ 0.05840841 -0.11407113 -0.11000431  0.66761214 -0.03394461]]
```



```
1/1 [==============================] - 0s 81ms/step
[[ 0.01094742 -0.20029509  0.10977017  0.55016434 -0.08951813]]
Left
1/1 [==============================] - 0s 80ms/step
[[ 0.28505278 -0.43637389  0.15868475  0.14643501 -0.27408046]]
Forward
1/1 [==============================] - 0s 83ms/step
[[ 0.57333452 -0.49963772 -0.12079954 -0.0031895  -0.22374213]]
Forward
1/1 [==============================] - 0s 114ms/step
[[ 0.42564765 -0.30548424  0.07242242  0.10270656 -0.33722895]]
Forward
1/1 [==============================] - 0s 123ms/step
[[ 0.2060349  -0.24146962 -0.09591651 -0.38667321 -0.24038476]]
Forward
1/1 [==============================] - 0s 111ms/step
[[ 0.59587944 -0.10528547  0.22508378 -0.31333172 -0.38255858]]
Forward
1/1 [==============================] - 0s 149ms/step
[[ 0.47593373 -0.16637892  0.0043655  -0.28224063  0.01657339]]
Forward
1/1 [==============================] - 0s 122ms/step
[[ 0.95486045 -0.04047072 -0.51147389 -0.45562607 -0.40588272]]
```

2.5 Web application:

We build our front-end website using React.js, implement user authentication with firebase. Socket.io is a node.js library that can browser talk with Node.js server in real-time. When socket is listening for a port, both client or server can emit an event to publish topics and subscribe a topic using socket.on method. Then user render the page of socket communication, he/she can press the start button to let server side toy car call for a subprocess to run python script, analyzing image data and getting the prediction.

3. Libraries/Packages

- RPi.GPIO: We use this module to output control signals through the gpio on raspberry pi, and we control the speed of the car by change the duty cycle of the pulse by using the ChangeDutyCycle function in GPIO.PWM
- pygame: pre-installed in raspberry pi, used to detect keyboard input in the data collecting phase.

- threading: We import this library in order to accomplish the two processes in the data collecting phase and four processes in the self-driving phase.
- picamera: necessary library to use picamera
- PIL: Use this library to read image from A stream implementation using an in-memory bytes buffer.
- numpy: process the image as numpy arrays.

  keras
  - Sequential model
  - (keras.layers) Lambda, Conv2D, MaxPooling2D, Dropout, Dense, Flatten: We add several Dropout layers to prevent the training model from overfitting.
    - pandas
    - h5py: Use this library to save the trained model as .h5 file.

  What we learn from this project
    - Hardware connection: GPIO output to L298N
    - Raspberry pi
      - picamera
      - socket
    - Multithreading
    - Machine learning module and adjusting parameters
    - Web app

4. References

https://picamera.readthedocs.io/en/release-1.13/recipes2.html#rapid-capture-and-streaming
https://picamera.readthedocs.io/en/release-1.13/
https://wizardforcel.gitbooks.io/hyry-studio-scipy/content/2.html
https://keras.io/callbacks/#earlystopping
http://image-net.org/challenges/posters/JKU_EN_RGB_Schwarz_poster.pdf
https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/

5. Github Link

https://github.com/BUConnectedWorld/Group12