VIETNAMESE - GERMAN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEER

Introduction to Computer Science Report

Group no: **Self-driving car**
Student 1: Do Minh Quang
Student 2: Doan Quan Tien
Student 3: Hoang Minh Phi
Student 4: Mai Nguyen Viet Phu
Student 5: Nguyen Nho Minh Tri
Student 6: Hoang Minh Tri
Student 7: Nguyen Phuoc Thien Phu

Content

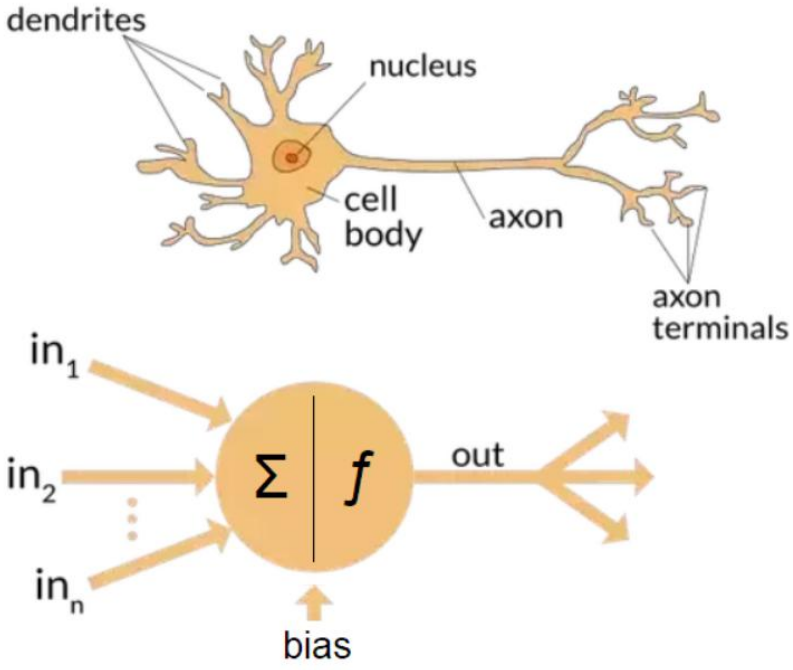
1 Introduction

Nowadays, traffic has become extremely complicated. There are a lot of accidents and most of them are caused by human error. So our group wants to introduce self-driving car - an application of neural networks. In this project, we will summarize the implementation of a project for the Introduction for Computer Science course. There will be 3 main parts in this report: An introduction to Artificial Neural Network, Convolution Neural Network and finally, the implementation of neural network in self-driving car.

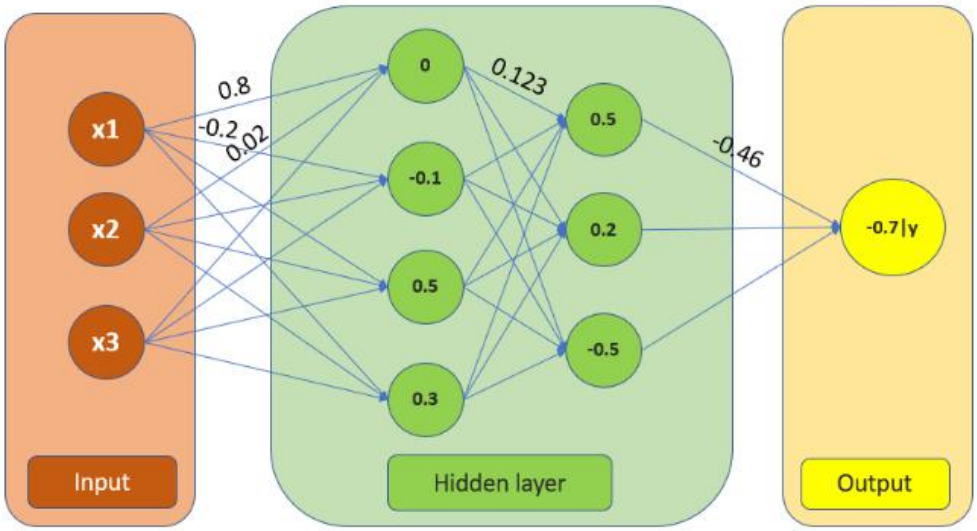
2 ANN - Artificial Neural Network

2.1 Definition

Artificial Neural Network is a Deep Learning model where the algorithms are inspired by the neural structure of the human brain. ANN takes data, trains themselves to recognize the patterns in this data and then predicts the outputs for a new set of similar data.

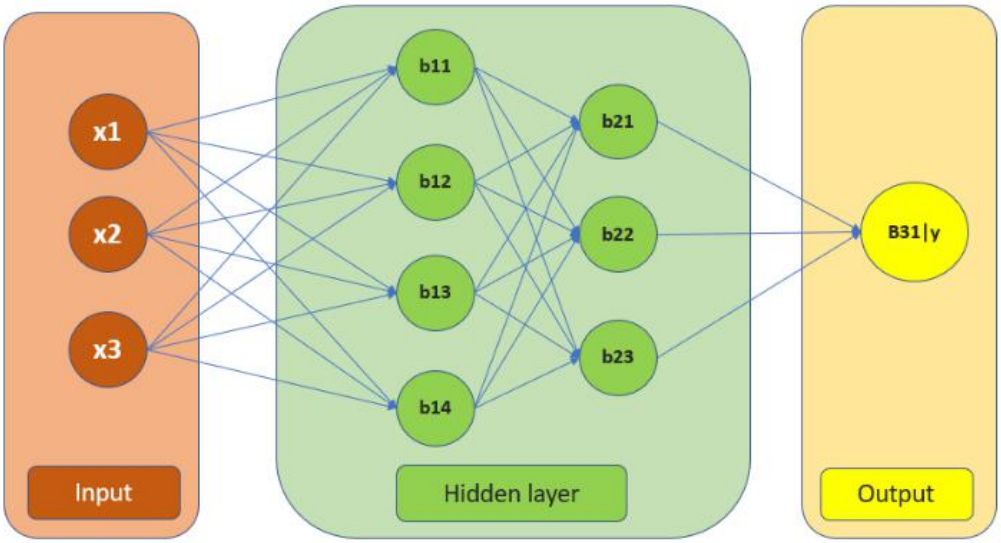


ANNs are complex structures containing interconnected adaptive elements known as artificial neurons that can perform large computations for knowledge representation.

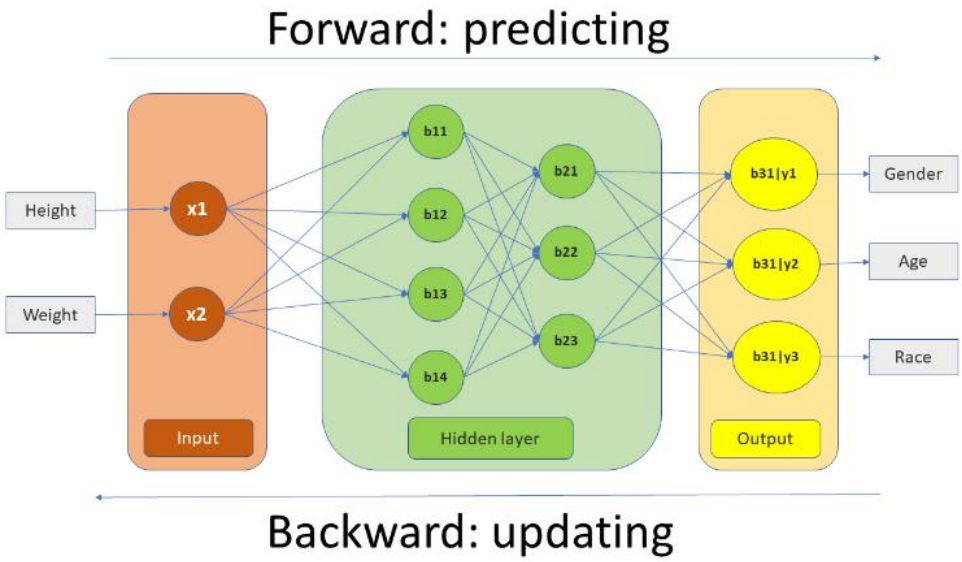


All of the neurons are fully connected to each other. Each of these lines is called a weight, which is a number that shows how strong the connection between 2 neurons and each neuron has its own number which is called bias. Each bias demonstrates how a neuron bias its prediction.

The first layer is the input layer which receives the inputs; the last layer is the output layer that predicts the final output; in between these 2 layers exist the hidden layers which perform most of the computations required by our network.



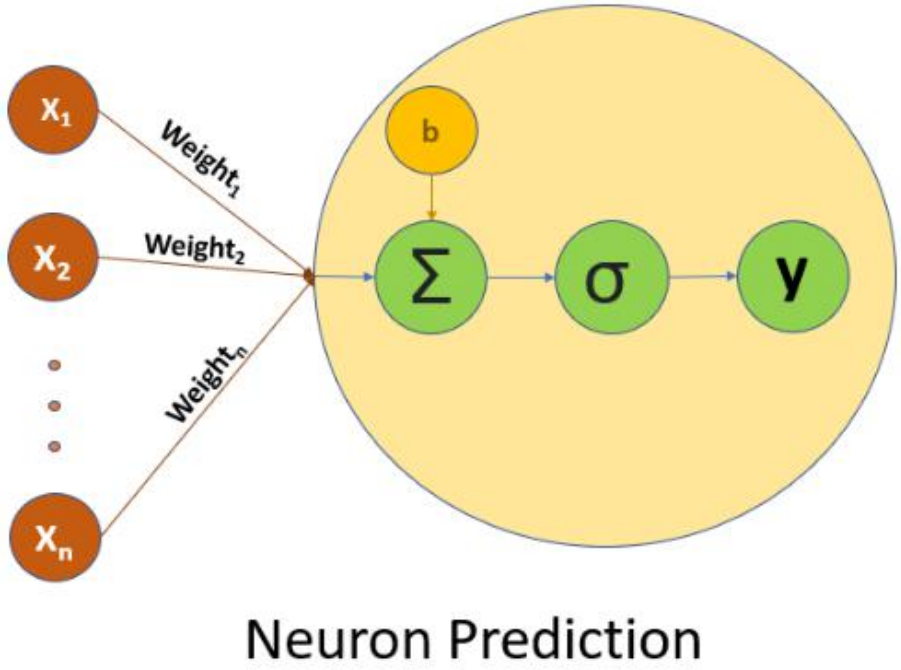
For example: we can use neural networks to predict a person's gender, age and race based on their input height and weight.



Overall, the neural network will try to predict an output from an input (this is called forward propagation) and then adjust the weights so that the prediction error can get smaller (and this process is called backward propagation).

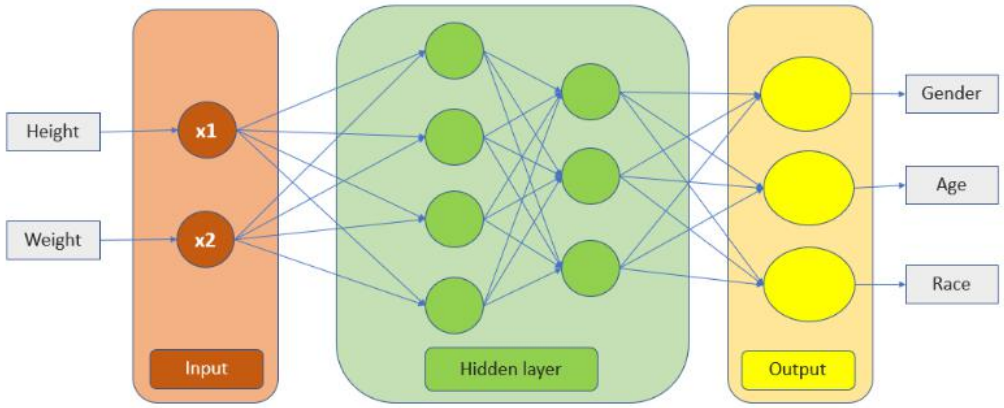
2.2 Forward propagation

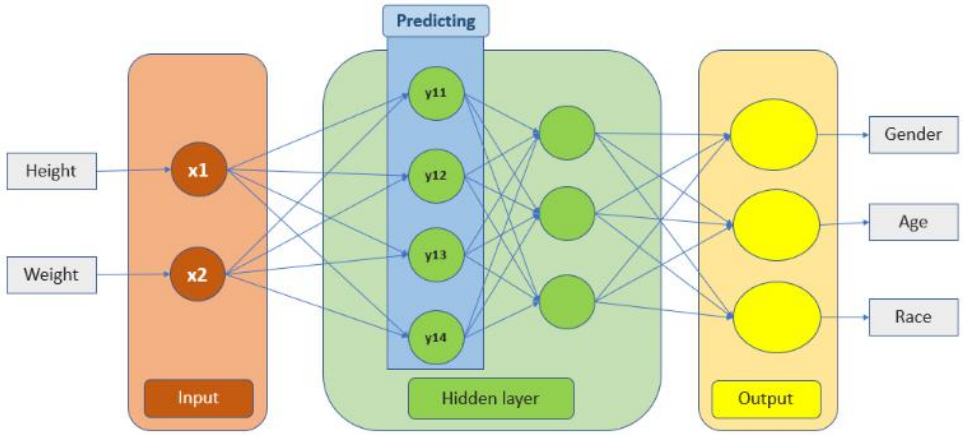
In forward step, each of the neuron predict by sum the weight multiplied by the value of every previous neuron plus its bias. Then this result is passed into an activation function and the neuron stores this final result.

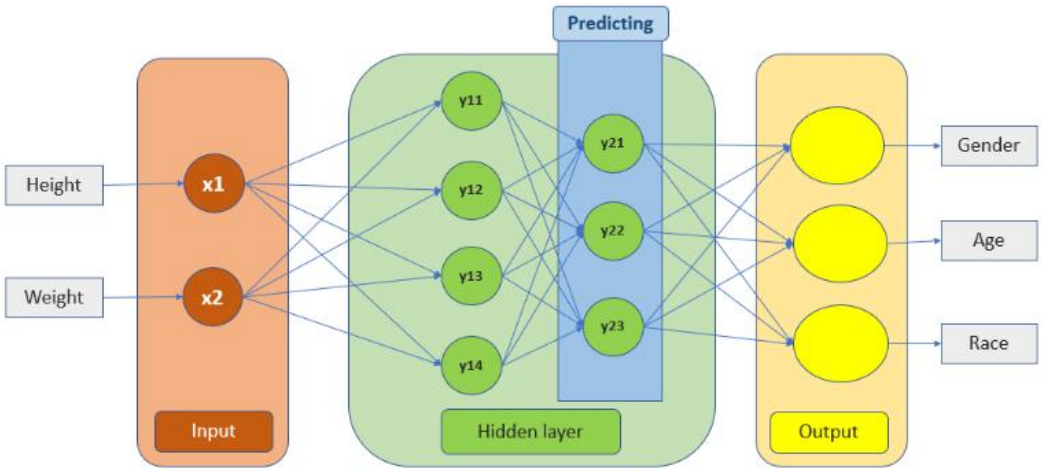


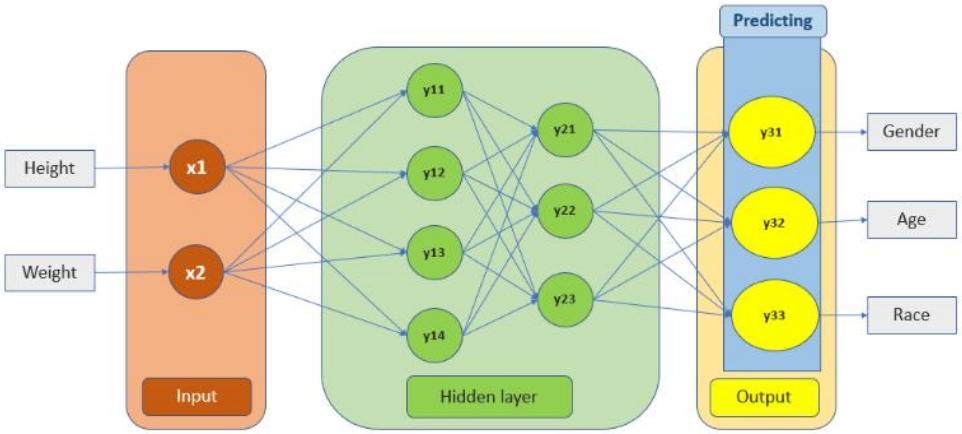
Each of the neurons in the same layer together predict and then they have their own predicted

values. After that, the next layer uses their value to predict and the next next layer continues to use the previous layer value to predict all the way to the output layer.



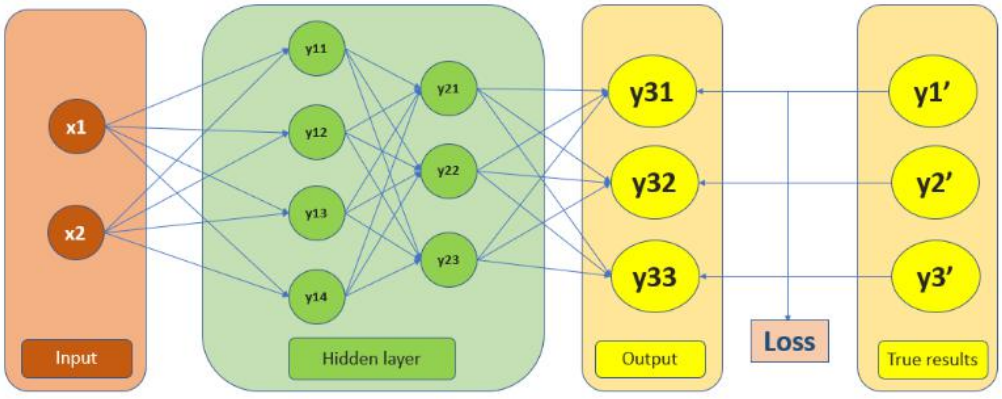






2.3 Back propagation

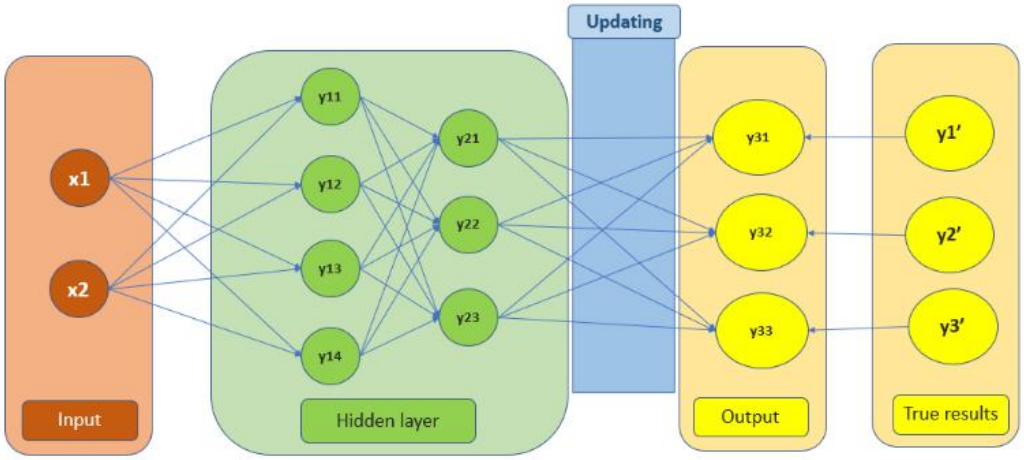
Now, the network will try to measure the error between the predicted result and the actual result. Its goal is to minimize this error function. To do this, it tells the weight and bias in the previous layers to minus the derivative of this error function with regard to itself. By doing this, the weights and the biases will change in a way that the error or Loss function will decrease until it reaches minimum.

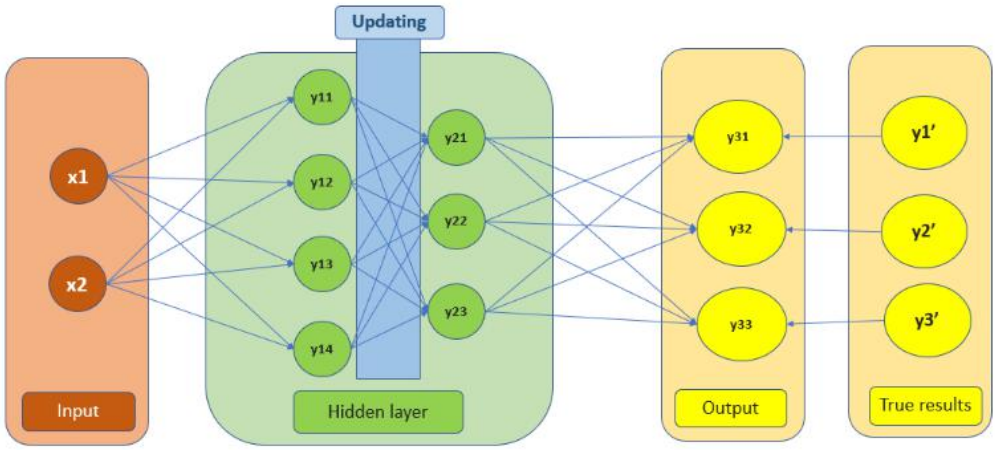


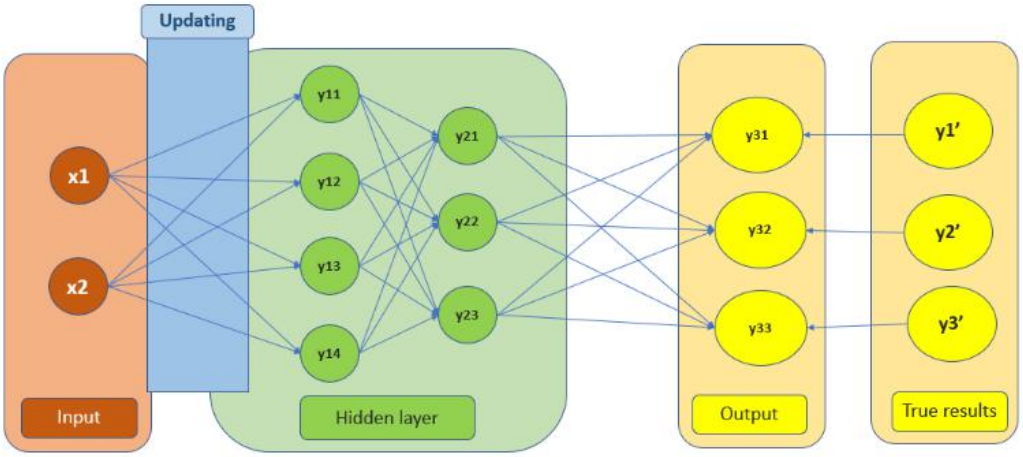
$$Error = Loss = \frac{1}{3}\left(|y_{31} - y'_1| + |y_{32} - y'_2| + |y_{33} - y'_3|\right)$$

$$new\_weight_{ij} = old\_weight_{ij} - \eta * \frac{d_{Loss}}{d_{old\_weight_{ij}}}$$

$$new\_bias_i = old\_bias_i - \eta * \frac{d_{Loss}}{d_{old\_bias_i}}$$





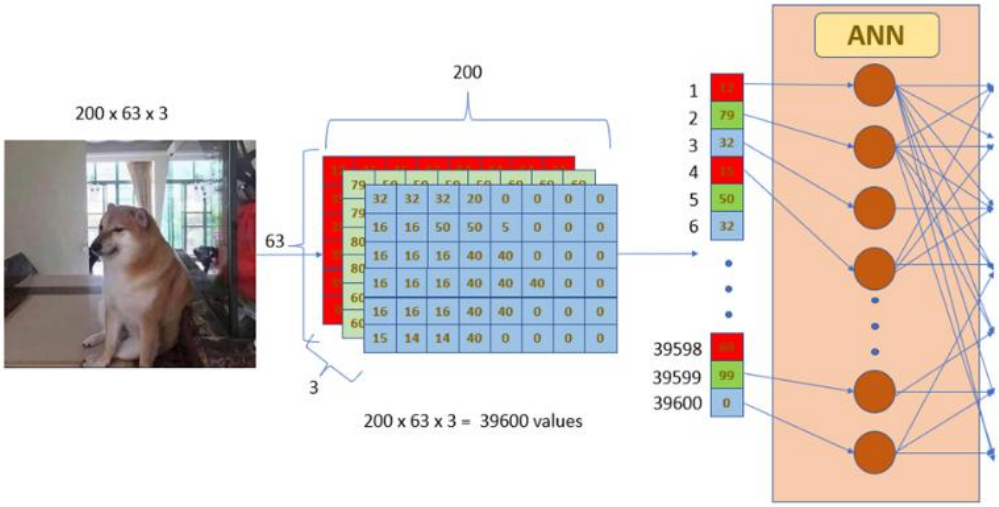


3 CNN - Convolution Neural Network

3.1 Definition

So basically, now we have a complete neural network to train with image recognition. However, it is not very efficient and we can't just use an image as a single input to the network.

Instead, an image is a collection of color pixels. So we have to consider each color value in each pixel of the image to be an input x so. This image size is 66*200 and rgb if we break this image down, it will be a total of 63*200*3 = 39600 different inputs.

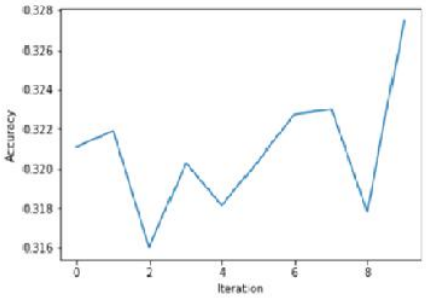


Because there are too many inputs, there will be a lot of weights and biases to train in the network, which will cause the network to train for a very long time and the network doesn't even promise that It will work effectively. For example, in our project, when we only use the

basic neural network, the prediction accuracy is only around 0.3.

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 39600)	0
dense_8 (Dense)	(None, 100)	3960100
dense_9 (Dense)	(None, 50)	5050
dense_10 (Dense)	(None, 10)	510
dense_11 (Dense)	(None, 1)	11

Total params: 3,965,671
Trainable params: 3,965,671
Non-trainable params: 0



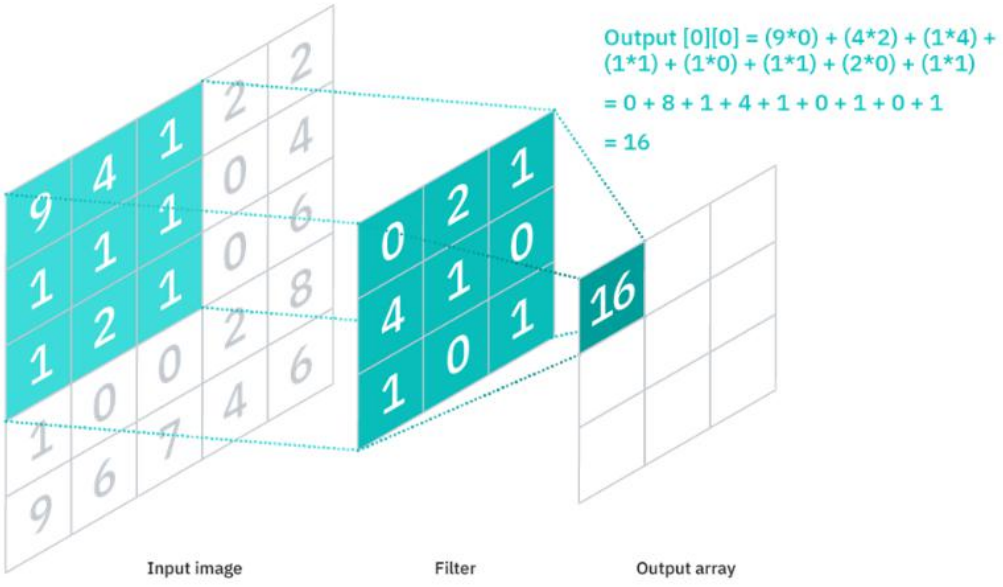
So to cope with this, we have to use CNN. A convolution neural network (CNN) is a type of artificial neural network used primarily for image recognition and processing, due to its ability to recognize patterns in images. The main advantage of CNN is that it can detect important features of an image. This helps to solve the main problem of basic neural networks, which is that they have too many inputs. So we add a CNN network between the input and the neural network.

Overall:

- CNN summarizes the existing features of an original image to a smaller set of features input and passes it to ANN.
- ANN then uses those inputs and starts processing to predict and to learn.

3.2 Convolution layers

The convolution stage performs the feature extraction. It uses a kernel (which is like a filter) to scan through an image to detect patterns and then summarize those patterns into a new smaller image.



For example: Vertical kernels (filters) when applied to the image can highlight features that are made up of vertical edges and remove those features that don't have vertical edges.

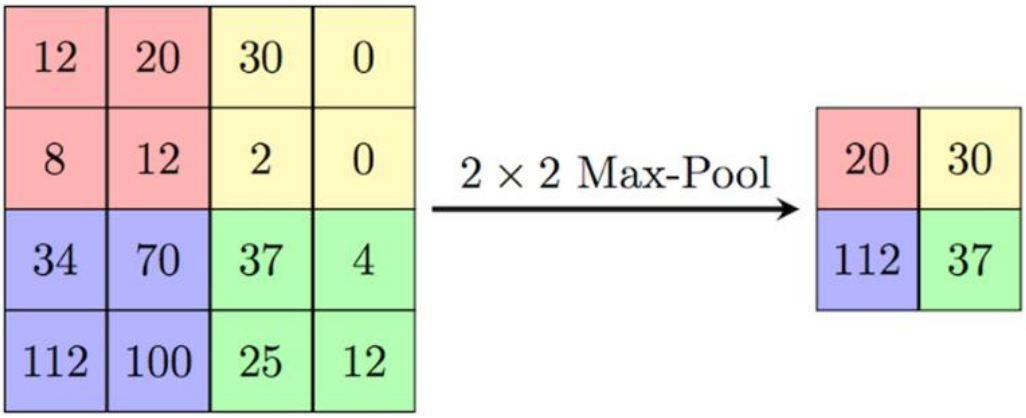
-1	0	+1
-1	0	+1
-1	0	+1

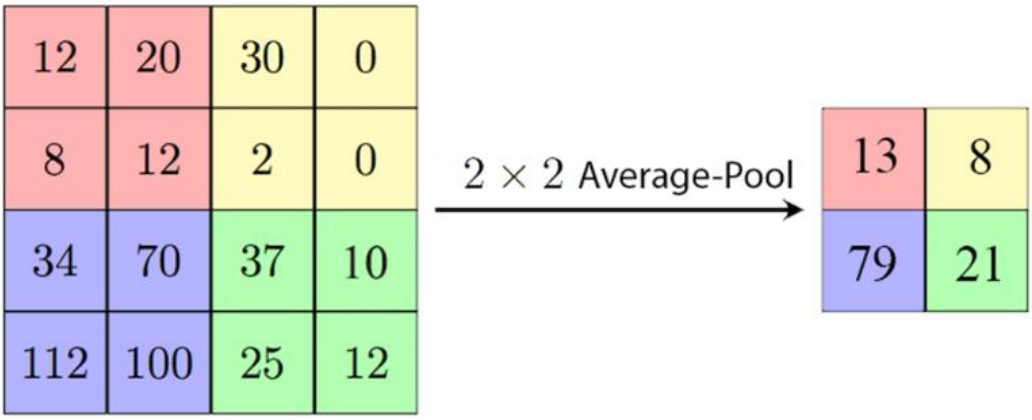
-1	0	+1
-2	0	+2
-1	0	+1

3.3 Pooling layers

The next step is the pooling stage. The main purpose is to significantly reduce the size of the image and condense the feature of that image. Pooling usually uses a 2x2 kernel.

There are two common types of pooling: max and average. Max pooling uses the maximum value of each local cluster of neurons in the feature map while average pooling takes the average value.





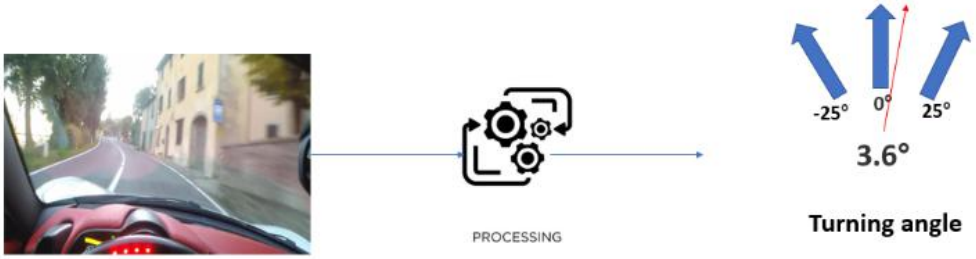
After that, the feature map is then flattened before inputting into fully connected layers

(ANN) to transform from 2D tensor into 1D.

4 Self-driving car and neural network

Use case: path navigation allows self-driving car to know how to stay on the road with the front camera input

So to wrap it up, our mission is to create a neural network that inputs an image of the road and outputs a number which tells the car how to stay on the road.

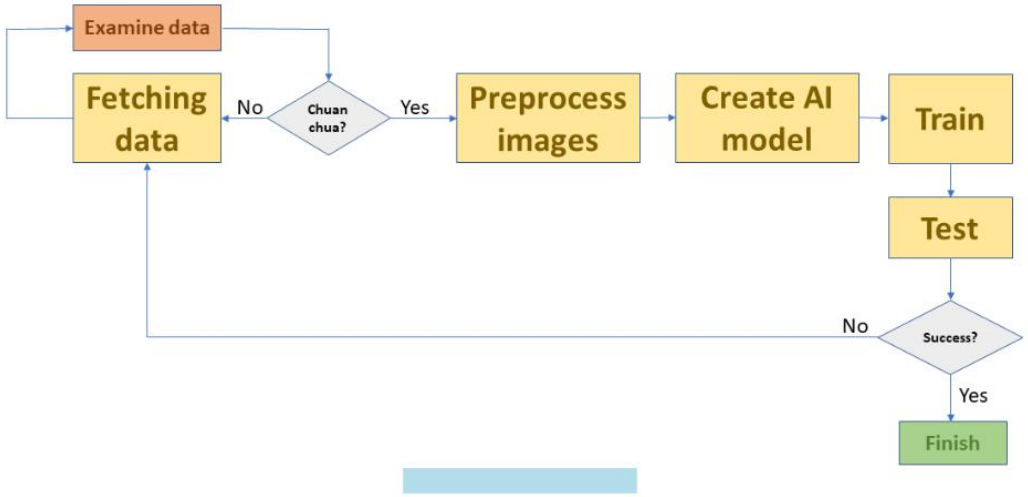


4.1 Components

To conduct our project, we need 4 main components:

- NVIDIA' scientific paper: End to end learning for self-driving cars.
- Numpy, pandas, matplotlib, opencv library for data and image processing.
- Tensorflow library for creating neural network models.
- Car simulator from Udacity open-source github and socket.io library for communication between AI model and the simulator .

4.2 Plan



In our project, we divide it into 5 main steps:

- Fetch training data, in this step, we fetch the data and check if the data is not good then we delete the data and do it again until the data is good enough.

  So firstly, we use our simulator and try to drive the car for a few minutes. After that, we record our driving experience into the data. The recording contains an image folder and a csv file, which contains the path to the corresponding image with its turning angel.

driving_log.csv
IMG

Image Path	Turning angle
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_44_903.jpg	0
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_44_975.jpg	-0.1
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_045.jpg	-0.3
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_118.jpg	-0.5000001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_189.jpg	-0.6500001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_264.jpg	-0.8500001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_330.jpg	-0.6500001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_406.jpg	-0.6500001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_482.jpg	0
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_556.jpg	-0.6500001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_633.jpg	0
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_706.jpg	0
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_779.jpg	0.05
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_852.jpg	0.2
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_923.jpg	0.4
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_45_998.jpg	0.6000001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_070.jpg	0
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_140.jpg	0
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_215.jpg	0
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_285.jpg	0
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_359.jpg	-0.1
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_432.jpg	-0.25
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_504.jpg	-0.6500001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_579.jpg	-0.6500001
C:\Users\quang\OneDrive\Desktop\IMG\center_2022_12_11_20_16_46_651.jpg	0

- Pre-process image data

  Before using the data to train, we randomly edit the images to different conditions. The reason why we do this is because we expect the car to meet unfavorable conditions when running on the road.

```
1 if np.random.rand() < 0.5:
2     pan = iaa.Affine(translate_percent = {'x':(-0.1,0.1),'y'↩
          :(-0.1,0.1)})
3     img = pan.augment_image(img)
```

  Shifting: when running, the car doesn't run really smoothly, sometimes it will suddenly stop or turn, this will cause the camera to shake and make the image look like it is shifted.

So that's why we need a shifting effect in our image.

```
if np.random.rand() < 0.5:
    zoom = iaa.Affine(scale=(1,1.2))
    img = zoom.augment_image(img)
```

Zooming: You can see that there is a turning left at the end of the road but it is too far so it's really small and the model can not see. By zooming, the model can see the turning angle and then can predict a to turn left successfully.

```
if np.random.rand() < 0.5:
    brightness = iaa.Multiply((0.2,1.2))
    img = brightness.augment_image(img)
```

Adjust brightness also helps the model reaction when the road is too bright or too dark

```
if np.random.rand() < 0.5:
    img = cv2.flip(img, 1)
    steering = -steering
```

Flip helps the model to be more flexible to both the left and right image.

- Create AI model

  The model will consist of 5 convolutional layers and 4 dense layers. And also it will use Adam for optimizer in backpropagation and the error function is MSE, mean square error

```
model = Sequential()

model.add(Convolution2D(24, (5,5), (2,2), input_shape=(66,200,3), ↩
    activation ='elu'))
model.add(Convolution2D(36, (5,5), (2,2), activation = 'elu'))
```

```
model.add(Convolution2D(48, (5,5), (2,2), activation = 'elu'))
model.add(Convolution2D(64, (3,3), activation = 'elu'))
model.add(Convolution2D(64, (3,3), activation = 'elu'))

model.add(Flatten())

model.add(Dense(100, activation = 'elu'))
model.add(Dense(50, activation = 'elu'))
model.add(Dense(10, activation = 'elu'))
model.add(Dense(1, activation = 'elu'))

model.compile(Adam(learning_rate = 0.0001), loss = 'mse')
```

- Train model

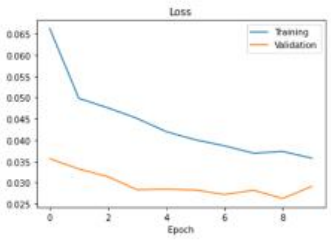
  Now before training, we split the data into 0.8 for training and 0.2 for validation. The training data is for training and the validation data is for checking how good the model is while training.

```
xTrain, xVal, yTrain, yVal = train_test_split(imagesPath, steerings, test_size = 0.2, random_state = 5 )
```

  And after that, we will train the model.

```
history = model.fit(batchGen(xTrain, yTrain, 100,1), steps_per_epoch = 300, epochs = 10, validation_data = batchGen(xVal, yVal, 100, 0), validation_steps=200)
```

```
Epoch 1/10
300/300 [==============================] - 130s 433ms/step - loss: 0.0662 - val_loss: 0.0356
Epoch 2/10
300/300 [==============================] - 128s 427ms/step - loss: 0.0498 - val_loss: 0.0332
Epoch 3/10
300/300 [==============================] - 128s 427ms/step - loss: 0.0476 - val_loss: 0.0314
Epoch 4/10
300/300 [==============================] - 128s 427ms/step - loss: 0.0451 - val_loss: 0.0283
Epoch 5/10
300/300 [==============================] - 128s 428ms/step - loss: 0.0420 - val_loss: 0.0285
Epoch 6/10
300/300 [==============================] - 127s 424ms/step - loss: 0.0401 - val_loss: 0.0283
Epoch 7/10
300/300 [==============================] - 128s 426ms/step - loss: 0.0387 - val_loss: 0.0272
Epoch 8/10
300/300 [==============================] - 127s 426ms/step - loss: 0.0369 - val_loss: 0.0282
Epoch 9/10
300/300 [==============================] - 128s 428ms/step - loss: 0.0373 - val_loss: 0.0263
Epoch 10/10
300/300 [==============================] - 129s 433ms/step - loss: 0.0358 - val_loss: 0.0291
Model Saved
```



- Test model, After training, if the test on the simulator is unsuccessful, we will repeat the process again until we are done.

  There are 3 things we can interfere to increase the accuracy of the model:

- Model architecture

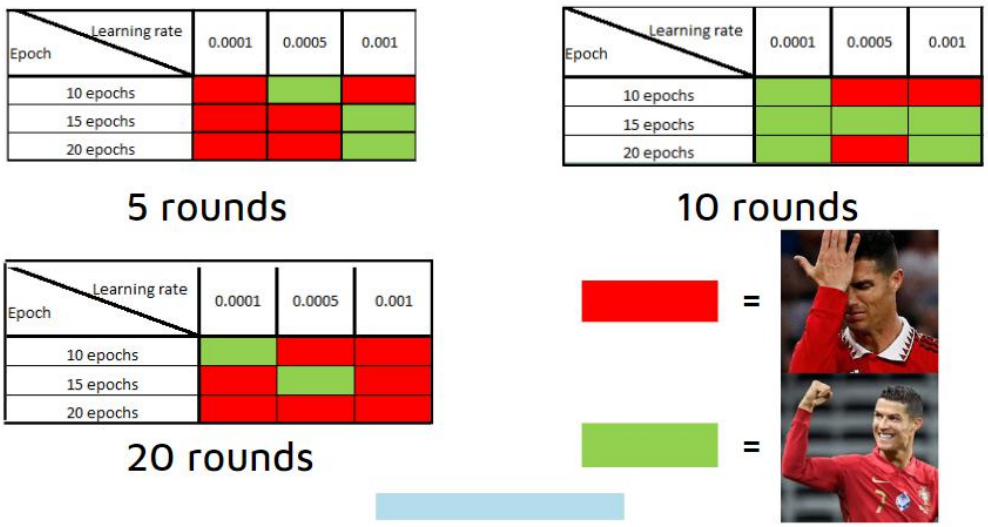
  Changing the model means that we can add more CNN or ANN layers or more neurons or more kernels. However, in the paper of NVIDIA, this model is the best they could obtain so we should not change it.

- Train data

  Data for training also affects how the model learn. So after numerous experiments, we finally conclude that data should balance between left and right and focus most on going straight. This will result in the car with the ability of going stably on the road.

- Hyper parameters

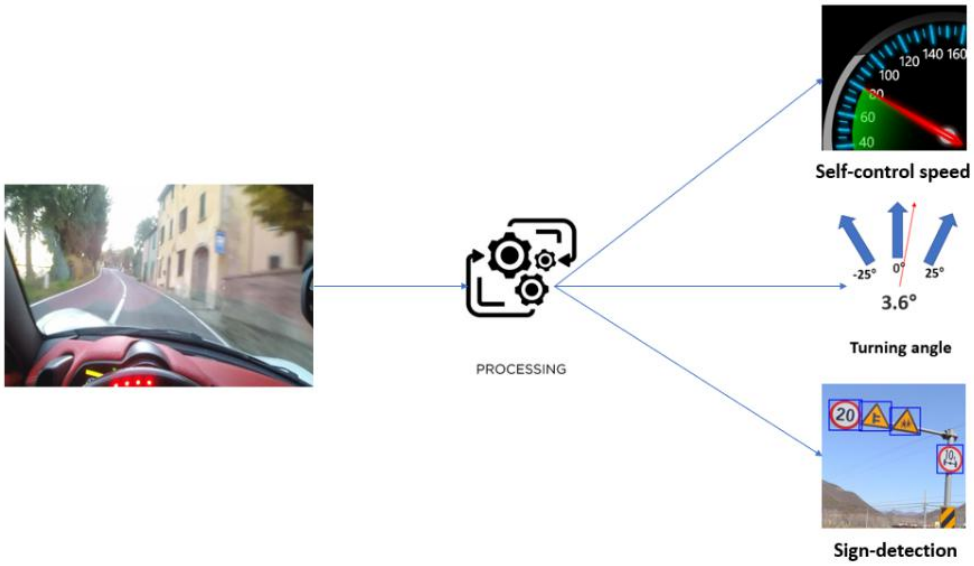
  Hyperparameters represent how we want to train the model such as epoch (how many times to train), learning rate (how much to learn in each update) and batch-size (how much data to train each turn). But out of the 3, only epoch and learning rate is effective in training the model.



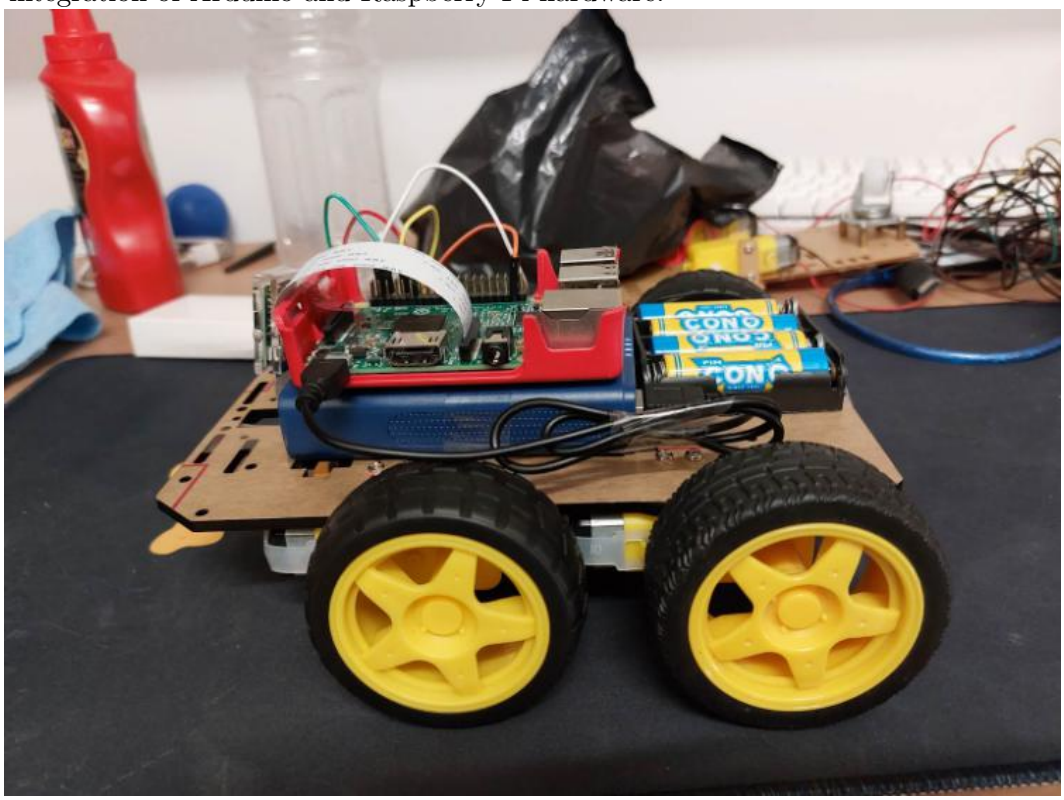
After some experiments with different combinations of the 2 factors above, we finally found some successful model. From the picture, we conclude that the best conditions for training is using the 10 rounds dataset with the learning rate of 0.0001 and 15 epochs.

5 Future improvement

Right now, our project is complete but very simple so in the future we can add more features to the model such as sign detection and self-control speed. We will also try to implement a mechanics to detect exceptions. To be more detailed, in our project, we only train the model to output whether to turn left, right or go straight. So if our model encounters a dog, it will still continue to go instead of stopping.



Furthermore, we will also try to implement this idea from simulation to real-world use with the integration of Arduino and Raspberry Pi hardware.



6 Conclusion

To sum up, our report has introduced the fundamentals of neural networks and applied it into our project. Throughout this project, we would like to show the readers a better understanding of AI, specifically how it works and also what it is like to work and deal with problems in an AI project.

7 Further Resources

- Project Github: Click me - Project slide: Click me