Developing an Autonomous Driving Model Based on Raspberry Pi

Yuanqing Suo, Song Chen*, Mao Zheng
Department of Computer Science
*Department of Mathematics and Statistics
University of Wisconsin-La Crosse
La Crosse WI, 54601
mzheng@uwlax.edu

Abstract

In this project, we train a convolutional neural network (CNN) based on the VGG16 using the lane image frames which are obtained from a front-facing camera mounted on a robotic car. We then use our trained model to drive the car in a predefined environment. The model will give the steering commands, such as forward, left-turn and right-turn, to make the car drive in the lane.

The dataset we used to train and test the model contains more than 120 minutes of driving data. In a pre-trained real environment, the model can operate the robotic car to continuously drive in the lane. We trained the self-driving model by giving raw pixels of the lane image frame. The model will predict the next steering command based on the current frame state. This end-to-end approach significantly relies on the quality of the training dataset. To collect the training dataset, we chose a pi camera version one module with 5 megapixels resolution instead of a USB 2.0 8 megapixels (MP) resolution web camera. Although the webcam has a higher number of megapixels, it has a low frame rate and no GPU encoding. On the other hand, the pi camera has GPU encoding, and 5 megapixels resolution which produces decent quality image frames. In addition, the pi camera has a wide field of view in images. With image preprocessing, such as cropping and grayscaling, the testing accuracy of the transferred model was approximately 80%. This project empirically demonstrated that we can train CNN models to give the steering commands to operate the car in the lane.

We used the free Cloud Tensor Processing Units (TPU) resource provided by google Colaboratory for training and a Sunfounder smart video car based on Raspberry Pi 3 model B for testing. The system operates at ten frames per second (FPS).

1. Introduction

Self-driving became a popular sub-field in the machine learning field for the past decade. Many companies started developing their own self-driving product such as, Waymo, Tesla, Audi, Ford, Volvo etc. The most basic idea behind the self-driving technology probably is to predict the following action based on current input data obtained from sensors, for example, the lane image from the camera sensor or the distance information from the radar sensor.

The NVIDIA company published their work on training an end-to-end self-driving model based on the raw road frames [1]. The end-to-end approach refers to the model outputs from the prediction based on the raw data obtained from sensor directly. The NVIDIA company trained their CNN model by using the raw pixel road picture as the input data to predict the steering wheel angle of the car. Their contribution on the self-driving car provides us with the basic idea for training our model.

The goal of this paper is to introduce the work on training a robotic car and letting it drive itself in the lane. The model is based on the lane image frames which were captured from the mounted front facing camera. The self-driving model was deployed on a robotic car based on Raspberry pi. The raspberry pi is the core controller of the robotic car. It connects with a servo controller and a motor driver. The servo controller is built based on a 16-channel LED controller with I2C bus interface to control the speed of the car. The motor driver allows raspberry pi to control the motor of the car directly.

In order to train our model, we collected the lane images and their corresponding labels. Details of the training data will be introduced in Section 2. We used the transfer learning when training our self-driving model. The architecture detail of the model will be introduced in Section 3. The main goal of our model is using lane images as inputs, then output the predicted action to drive the robotic car in the lane.

Once the action prediction model is trained, we saved the model weights and model architecture together into a .h5 file. The reason we saved the model in a .h5 file is that a h5 file can save data in a hierarchical way. In addition, it is also supported by the most popular open source machine learning platform such as TensorFlow or open source neural network library Keras. We also saved the class label dictionary of our training dataset into a python pickle file. Python pickle module is used for saving objects by serialization [2]. We used the pickle to save and reload the action labels of the training data. We found that saving the class labels of the model in the pickle file will be easy to reload to get the same class sequence which were used in training process. Reloading the class sequence correctly is very important for us to determine the most likely next action for the current state.

Section 2 describes our work on data collection of the labeled training dataset. In section 3, we will introduce the prediction models. Test environment and the platform will be described in section 4. Results of the different datasets we trained will be shown in section 5. The last section concludes the paper and gives the directions for future work.

2. Data Collection

There are many open datasets for self-driving, such as Waymo open dataset [3], Berkeley DeepDrive BDD 100k [4]. These open datasets are large scaled and contain independently generated labels for sensors such as lidar and camera. The datasets are used for a real car to do the object detection and tracking. These popular open datasets are usually used to output the steering angle for the driving steer angle sensor. However, the robotic car we used can only remotely be operated by receiving forward, backward, left-turn or right-turn commands from a client-end, which is different from a real car. The Sunfounder company [5] provides a client-end Python program to control the robotic car remotely. When the user presses some specific keys on the keyboard or clicks the control button on the user interface, the server-end of the car will receive the steer commands and send these commands to the servo controller to control the motor of the car. In this project, we modified the client-end a little bit. We keep the steering button in the interface and added the button 'Begin self-control' for the self-driving mode. When we collect the datasets, we used the Forward, Left and Right buttons to save the steering commands and the associated image pictures. We did not use the Home button and Backend in the data collection process. The Home button controls the direction of the tire. After the robotic car turns left or right, pressing the Home button straightens the tire direction. Moreover, due to the fact that we only mounted one front-faced camera in the robotic car, we did not train the model to do a backward prediction. The modified user interface is shown in Figure 1. In this project, in order to collect a training dataset by ourselves, we added code in the client-end program to store the action array into a file when the steering functions are triggered. Meanwhile, we also captured current frames when the steering function is triggered. Finally, we got sets of lane frames and their corresponding action labels.

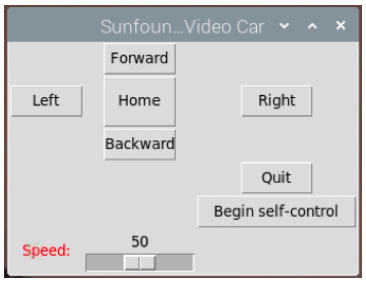


Figure 1 GUI of the Robotic Car

In order to collect the training data with reasonable quality, we tried two different types of cameras and compare the captured images. The first one is the USB 2.0 camera with 8 MP resolution, and another is the pi camera version one with 5 MP resolution. The car kits in Sunfounder has a USB 2.0 camera. But after comparing the two sets of images, we decided to use the pi camera. The main reason that we choose the training datasets collected by pi camera is that the pi camera provides the lane frame with a wide field of view. The webcam's front-end blind spots are almost two-length of the robotic car. In addition, the captured image resolution of pi camera was also better than webcam. This is because that the images captured by pi camera are encoded by GPU in raspberry pi. Although the webcam has a higher image resolution, it is only available in CPU. So, when comparing the image captured by the two types of camera, we found that the frames obtained by the pi camera are better than the webcam.

In this paper, we collected the training dataset by manually operating the robotic car through the client-end driving in the lane for more than 120 minutes. We collected the dataset in different light environment and different interior backgrounds, for example, with/without a sofa or with/without chairs etc. At the same time, we kept the camera position and angle unchanged. Before training our action prediction model, we split the labeled lane frames into training, validation and testing sets. The total number of images in each set are 1425, 405 and 405, respectively. We should note that, in order to have a clear baseline of the testing accuracy, each set (training, validation and test) should contain the same number of images in each action class. For example, in the training set, we have 475 images under class "forward" , 475 under class "left" and 475 under class "right". The validation set contains 135 forward images, 135 left images and 135 right images. The construction of the training dataset is shown in Figure 2.

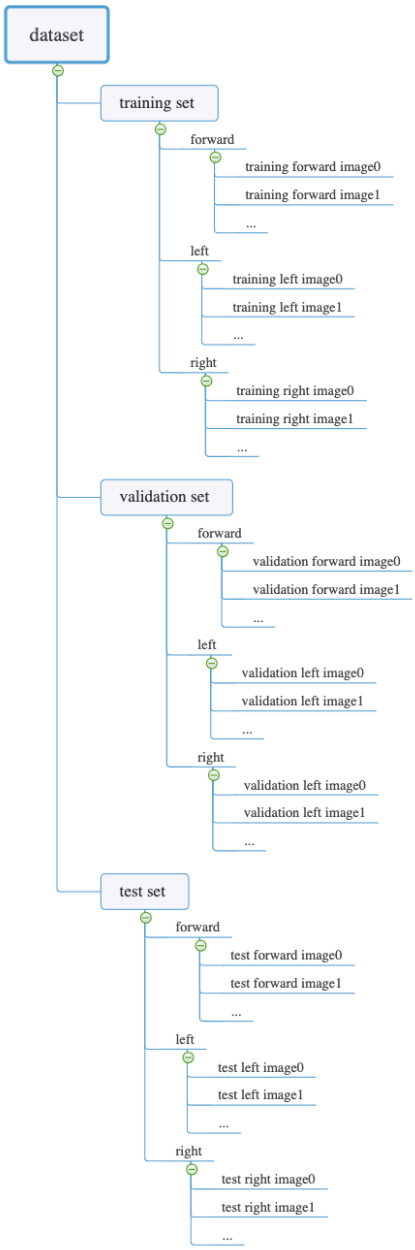


Figure 2 Structure of the Dataset

In addition, we also conducted the data preprocessing before training the model. In order to focus on the region of interest, which is the lane in each frame, we cropped out the top half of the picture to remove the chair or sofa from the image. After cropping, we use 255 which is the maximum edge value of the RGB subtract the pixel value of current image for grayscale processing. The commonly used grayscale processing in Open Source Computer Vision Library (OpenCV) is based on the formula: $Y = 0.2989R + 0.5870G + 0.1140B$ [6]. This process will convert red, green, blue three channels into one channel to decrease the complexity of the image. However, the input of our model which will be discussed in detail below requires the input image using three channels. To solve this conflict, we decided to reverse color of each channel in each image. Thus, we can also get the gray-liked image by keeping the image in three channels. That is the reason why we used 255 subtracting the pixel value of every images. The process of data preprocessing is shown in Figure 3.

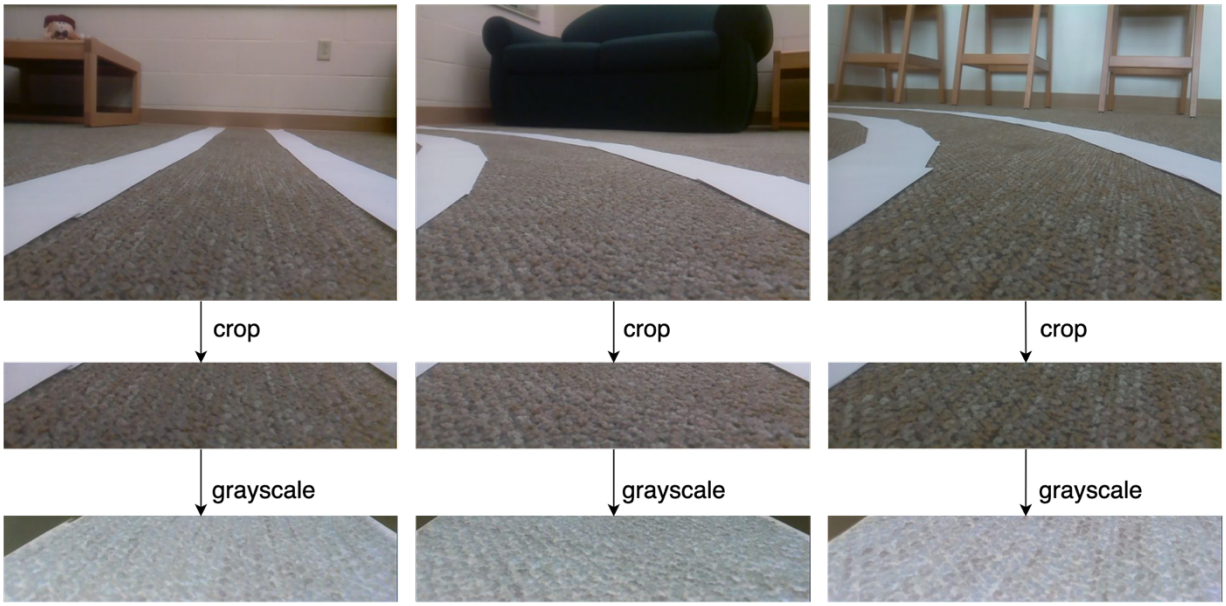


Figure 3 Process of Data Processing

3. Model Training

In order to map the lane frames with the steering commands, we choose the CNN architecture. CNN is a most popular deep learning architecture inspired by the visual perception mechanism [7] which is widely used in computer version, speech and natural language processing. In addition, there are many pre-trained CNN models in image classification, like VGG16 [8], ResNet [9], MobileNet [10] etc. We choose Keras, which is a widely used Python deep learning library to train our model. Keras provides three open source machine learning framework backends. They are TensorFlow, CNTK and Theano [11]. In this project, we selected TensorFlow as the tensor manipulation library, which is also the default backend in Keras.

3.1 Transfer Learning

The goal of our project is to map the lane frame to the fixed number of steering commands. This goal is achieved by using the image classification technique. We choose the pre-trained image classification model VGG16 and modify it in our project. Generally speaking, transfer learning is the process of using the information learned from one neural network into another neural network [12]. VGG16 will output the probabilities of 1000 known classes. In this project, we use VGG16 as a kind of features extractor. To implement that, when we trained our model, we would freeze the convolutional layers and modify the top layers to fit our goal.

3.2 Architecture of the Prediction Model

The prediction model is based on VGG16. Basically, in total it has 13 convolutional layers, and five max-pooling layers. We added one flatten layer, three dense layers and two dropout layers before the output layer. The overview of the architecture is shown in Figure 4. The white layers are the structure of VGG16, the blue ones are the layers we modified.

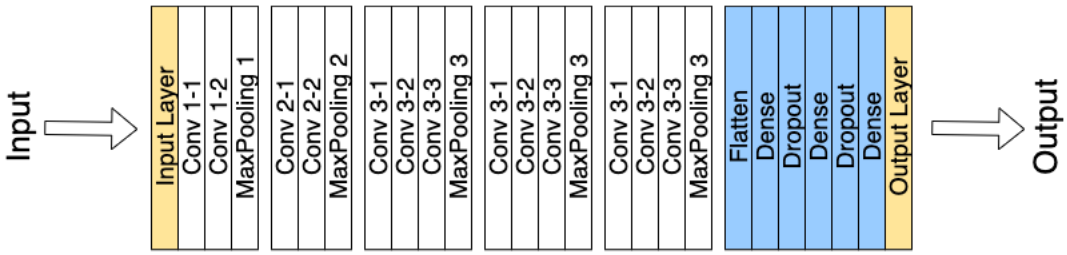


Figure 4 Architecture of the Model

3.3 Training Detail

As we mentioned before, we split all labeled lane frames into training set, validation set and test set. In this project, we trained the model by using the training set and validation set. Generally, the training set is used for training the model to find out the fitted parameters to do classification. The validation set is used to tune the trained parameters. Finally, the test set is used only to assess the performance of the classifier [13]. In order to have a clear baseline to evaluate the performance of our model, we kept the same class distribution in each set.

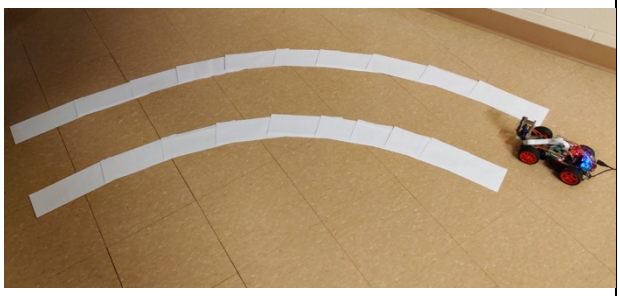
4. Test Environment

We tested the self-driving model online and in the real world. In order to have an overview of the performance of the model, we tested the model online using the test set. We fed every lane image into the model and got the predicted output of each lane picture. After that, we compared the predicted output with the actual label of the image to get the accuracy of the model. It should be noted that, the test set was not used in the model training process. So, every image used for testing is new for the model. This is important, because we want to get an unbiased evaluation of the prediction model [14].

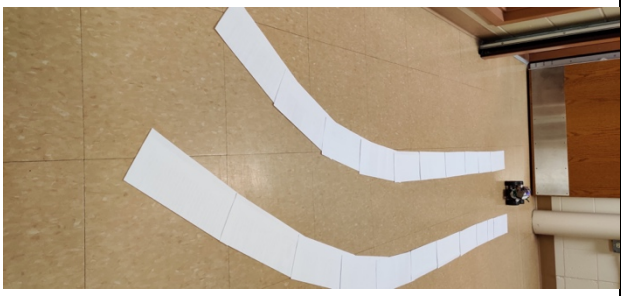
In addition to the online testing, we also tested our model in the real world. We deployed the model on the robotic car that is based on raspberry pi. In the first stage, we tested the model indoors since the outdoor environment is more complicated. We drew the lane indoor manually and tested the model in two different road conditions: with carpet and without carpet, and two lighting conditions: sufficient light and insufficient light. We set these different backgrounds to test the robustness of the model. The indoor test environment is shown in Figure 5.

carpet ground with insufficient light

carpet ground with sufficient light



floor tile ground with insufficient light



floor tile ground with sufficient light

Figure 5 Indoor Test Environments

5. Test Results

We trained the model by using different amounts of images. We named these datasets as pi-00, pi-01 and pi-02 dataset. As the increment of the number of training image, the online accuracy of prediction models is not improved. The results are shown in the table below:

		pi-00	*pi-01*	*pi-02*
train	Forward	178	210	475
	Left	178	210	475
	Right	178	210	475
validation	Forward	45	70	135
	Left	45	70	135
	Right	45	70	135
test	Forward	45	70	135
	Left	45	70	135
	Right	45	70	135
accuracy		70.00%	80.00%	68.40%

Table 1 Online accuracy for different scale of datasets

We initially speculated that increasing the amount of training data may help in improving the prediction accuracy. But the online test results illustrate the accuracy decreased after we expanded the training data.

The indoor test result in real environment are described as follows:

- Carpet ground with insufficient light:
  - Test driving forward then turn right: total lane length is around 2.3 meters, and lane width is 0.25 meter. The model can drive the robotic car forward. However, when the car drove approximately 1.6 meters that is before entering the right curve, the model mistakenly predicted a left-turn instead of the right-turn. This is because of the limited light, the captured frame almost has no sign of lane.
  - Test driving turn left then forward, total lane length is approximately 2.3 meters, and lane width is 0.25 meter. The model can drive the robotic car to correctly finish the left curve and then drive forward in the lane.
- Carpet ground with sufficient light:
  - Test driving forward then turn right, total lane length is approximately 2.3 meters, and lane width is 0.25 meter. The model can drive the robotic drive forward and correctly finish the right curve. But, when the car approached the end of the lane, the left tire of the car overlapped the left lane, and then mistakenly turned left. The car departed from the lane.
  - Test driving turn left then forward, total lane length is around 2.3 meters, and lane width is 0.25 meter. The model can drive the robotic car in a manner which finished the left curve and then successfully drove forward in the lane.
- Floor tile ground with insufficient light:
  - Test driving forward then turn right, total lane length is around 1.97 meters, and the lane width is 0.25 meters. The right tire of the robotic car overlapped with the right lane when the model drove it at approximately 0.4 meters from the starting point. After the model drove the car at the end of the right curve, the model mistakenly

predicted turn right as going forward, and drove the robotic car rushing out of the lane.
    - Test driving turn left then forward, total lane length is around 1.97 meters, and the lane width is 0.25 meters. The model drove the car and finished the test successfully.
- Floor tile ground with sufficient light:
    - Test driving forward then turn right, total lane length is around 2.4 meters, and lane width is 0.25 meter. The model mistakenly predicted forward as turn left and left the lane.
    - Test driving turn left then drive forward, total lane length is around 2.4 meters, and lane width is 0.25 meter. The model can drive the car turn left in the left curve, but the left tire of the robotic car overlapped the left lane. It drove the car forward with the left tire continuing to overlap with the left lane.

Because we originally collected our dataset in the carpet ground, we assumed the model may have a limitation that can only be used in a familiar environment. We also found due to the limited computing power of the raspberry pi, even if we set the captured frames per second (fps) as 10 in the code, the actual fps in the indoor test is only about 1.

6. Conclusion and Future Work

This project provides us the experience of training a simple autonomous driving model based on machine learning. Comparing the online testing accuracy results between the model built by ourselves and the transferred model, using transfer learning can help the model improve the prediction accuracy. In addition, in the approach of improving our model, we also learned the basic skills to do the error analysis.

For future work, we plan to collect more data and train the model to recognize obstacles or crossroad on the road, and then makes corresponding responses. As we are approaching more complex goals, we need to learn more detail about the machine learning techniques.

Reference

[1] Bojarski, M., Testa, D.D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to End Learning for Self-Driving Cars. ArXiv, abs/1604.07316.

[2] Python Pickle module document: https://docs.python.org/3/library/pickle.html

[3] Waymo open dataset: http://waymo.com/open

[4] BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling, Fisher Yu et. https://arxiv.org/abs/1805.04687

[5] Sunfounder company: https://www.sunfounder.com/

[6] OpenCV color conversions document: https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html

[7] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. Pattern Recognition, 77, 354-377.

[8] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[10] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[11] Keras document: https://keras.io/#configuring-your-keras-backend

[12] Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), 1345-1359.

[13] Hjort, N. L. (1996). Pattern recognition and neural networks. Cambridge university press.

[14] About Train, Validation and Test Sets in Machine Learning : https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7