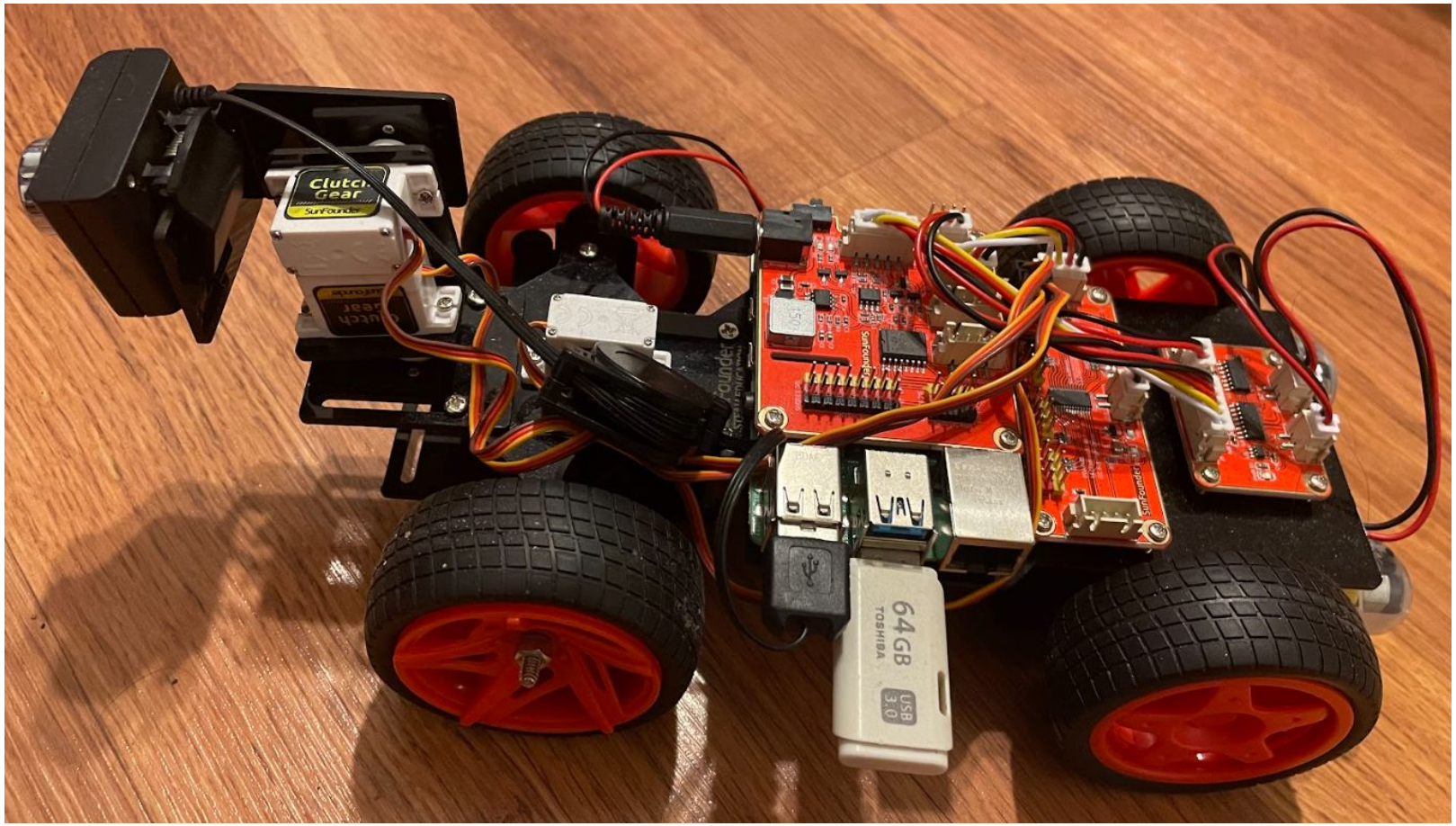
Automatic lane navigation of a robotic car using deep learning models



Lakshmikar R Polamreddy
Harsha Koduri
Jatin Kayasth

AI Final project
Dec 18, 2023

Agenda

- Introduction
- Relevant literature
- Approach
- Responsibilities
- Project set up
- Data collection
- Model training
- Results
- Challenges
- Conclusions
- Future work
- Codes and GitHub link
- References

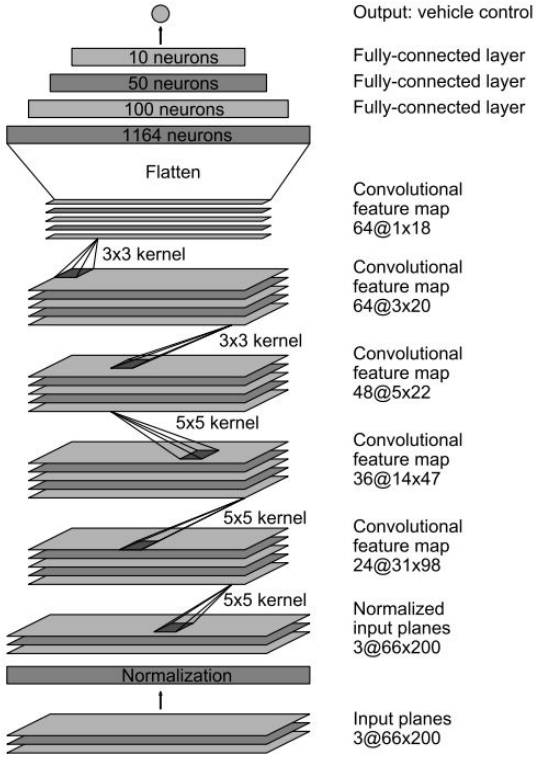
Introduction

- The focus of this project is to build a physically build a robotic car for autonomous lane navigation using deep learning models
- Gather our own training data using the robotic car
- Test the performance of the state-of-the-art NVIDIA model
- Build our own CNN model and compare its performance with that of NVIDIA model

Relevant Literature

- Bojarski et al. Bojarski et al. [1], as part of the NVIDIA research team, proposed a new convolutional neural network (CNN) architecture for end-to-end deep learning for self-driving cars.

- David Tian[2] proposed how to build our own physical, deep-learning, self-driving robotic car from scratch and to make our car detect and follow lanes.



NVIDIA model architecture [1]

Approach

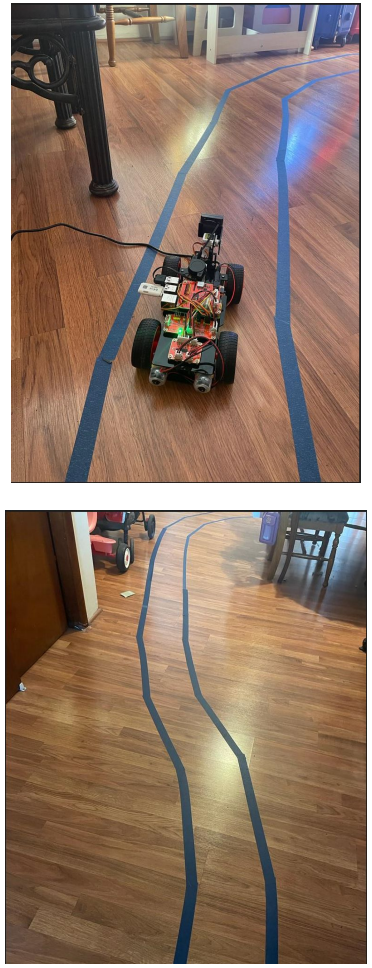
- Construct a physical robotic car
- Prepare a track on the wooden floor
- Drive the car autonomously using OpenCV and gather training data – images and steering angles
- Train the deep learning model with this training data
- Output from the model is the predicted steering angle
- Calculate regression loss between the actual and the predicted steering angles.
- Once the loss converges, save and use the model weight to test the performance in terms of R-squared value and compare the lane follower videos of deep learning and openCV

Responsibilities

Lakshmikar P	• Project proposal and project management • Purchase of hardware components • Physical construction of the robotic car • Physical lane preparation • Installation of necessary softwares and libraries • Data collection for training • Code for steering angle calculation • Custom model architecture • Training the NVIDIA model with smaller datasets (video-1 and videos-1,2) • Training our CNN model with smaller datasets (video-1 and videos-1,2) • Preparation of the report and presentation
Harsha K	• Physical construction of the robotic car • Physical lane preparation • Installation of necessary softwares and libraries • Codes for data collection using OpenCV • Data collection for training • Training the NVIDIA model with larger dataset (videos-1,2,3) • Comparison of the results of deep learning models with OpenCV • Preparation of the report
Jatin K	• Physical lane preparation • Installation of necessary libraries • Data collection for training • Training our CNN model with larger dataset (videos-1,2,3) • Preparation of the report

Project set up

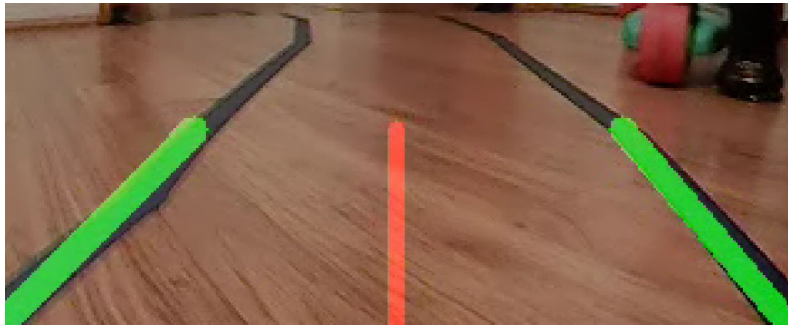
- **Hardware components -**
  - Raspberry Pi 4 computer – to load the software and libraries
  - USB Drive
  - SunFounder PiCar-V kit with camera module– main body of the robotic car
  - Camera module - 120 degree wide-angle camera with resolution 640x320 pixels
  - Few other accessories
  - Lane preparation with blue tape on the wooden floor
- **Software components -**
  - Operating system: Raspberry Pi OS
  - PuTTY, VNC Server/client
  - Python virtual environment setup – Python 3.9, OpenCV, Tensorflow
  - Camera module - to run deep learning models written in TensorFlow
  - Few other accessories
  - Lane preparation with blue tape on the wooden floor
  -
- **Testing the software and hardware components -**
  - Validate working of the front and the rear wheels by controlling them through the pi terminal
  - Make sure that OpenCV is also working by visualizing the black and color images of the lane on our laptop.



Images of the robotic car and the lane

Data collection

- On the prepared lane, robotic car is driven using OpenCV that detects the edges of the tracks
- Steering angle is calculated based on the following:
    - Consider the first and last elements of the list of the detected lanes (Green lanes)
    - Compute the average of these 2 elements and join these elements as shown like the red line segment
    - Then calculate the slope of this red line with respect to the frame width.
- Created 3 videos as part of data collection for training as shown below in Table. 1
- Images with respective steering angles are created from these videos
- Another test video is created and used for comparing the results of lane navigation of deep learning models with that of OpenCV



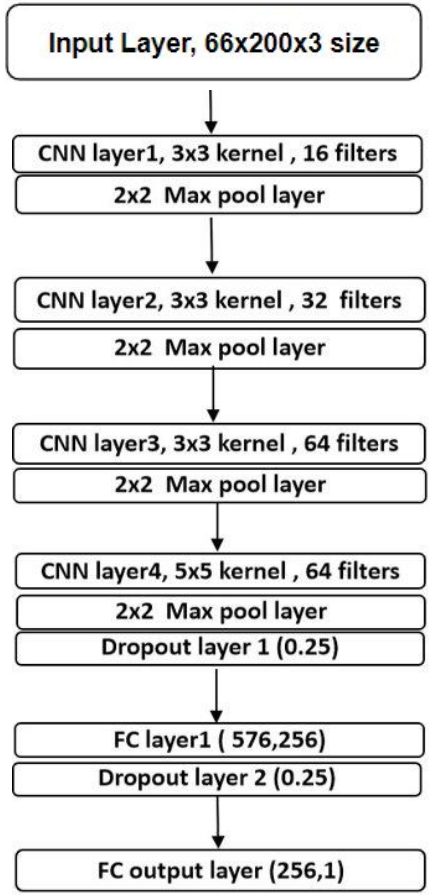
Track detection

Table 1: Dataset size

Video	Total number of images	Train data	Test data
Video 1	448	358	90
Video 1 and Video 2	807	701	106
Video 1, Video 2 and Video 3	1321	1056	265

Our CNN model architecture

- **4** Convolutional layers
- **2** fully connected layers
- Total trainable parameters : **208,481** (NVIDIA parameters - 252,219)



Our CNN model architecture

Model training

- Images and respective steering angles collected from the robotic car camera are passed onto the model
- Model training is done using the hyper parameters as shown below.
    - Number of epochs: 20
    - Number of steps per epoch: 300
    - Batch size: 100
    - Optimizer: Adam
    - Learning rate: 0.001
- Data Augmentation is done by using techniques like zoom, pan, adjusting brightness, crop, random flip, and blur
- Model outputs the predicted steering angle for test images
- Performance metrics: Regression loss in terms of MSE, R-squared value
- Trained both the models (NVIDIA and Our CNN model) for each of the three datasets

Results

- Our results are consolidated and shown in the Table. 2 and Table. 3
- We obtain these results after training the models for 20 epochs and then testing with 20 percent of the data from the images extracted from videos 1, 2, and 3
- We notice that both the models have shown similar performance in terms of R-squared when trained with the images of videos 1 and 2
- Our CNN model outperforms NVIDIA model when combined with images of all the three videos
- However, both the models show relatively poor results in the latter case. It is due to the reason that the loss did not converge in 20 epochs and so, we further train the model for 50 epochs and these results are shown in the Table.Table. 4
- In this case, Our CNN model shows better performance than that of NVIDIA model
- Result video using weights, showing the comparison of deep learning video and openCV video

Table 2: Performance of the models in terms of MSE(Mean Squared Error) with 20 epochs

Model	Video 1	Videos 1,2	Videos 1,2,3
NVIDIA model	5.5	6.6	36
Our CNN model	7.7	8.6	16

Table 3: Performance of the models in terms of R^2 with 20 epochs

Model	Video 1	Videos 1,2	Videos 1,2,3
NVIDIA model	93.4	94.2	65.2
Our CNN model	92.9	93.3	**80**

Table 4: Performance of the models in terms of R^2 with 50 epochs

Model	Training videos	MSE	R^2
NVIDIA model	1,2,3	28	65.6
Our CNN model	1,2,3	10	**90.6**

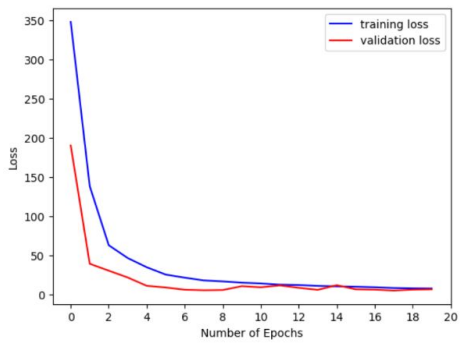


Fig. 6: Loss versus epochs for NVIDIA model when trained with images of video 1

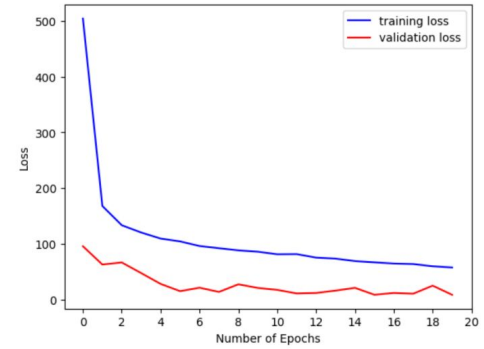


Fig. 7: Loss versus epochs for Our CNN model when trained with images of video 1

Challenges

- Setting up the front wheels
- Installation of OpenCV and Tensorflow on Raspberry pi
- Driving the car autonomously using OpenCV
- Failure of SD card
- Steering angle calculation
- Limited training data.

Conclusions

- Our CNN model with fewer parameters demonstrated similar performance to that of the NVIDIA in terms of R squared value for smaller datasets
- In case of larger datasets, our CNN model outperformed NVIDIA model

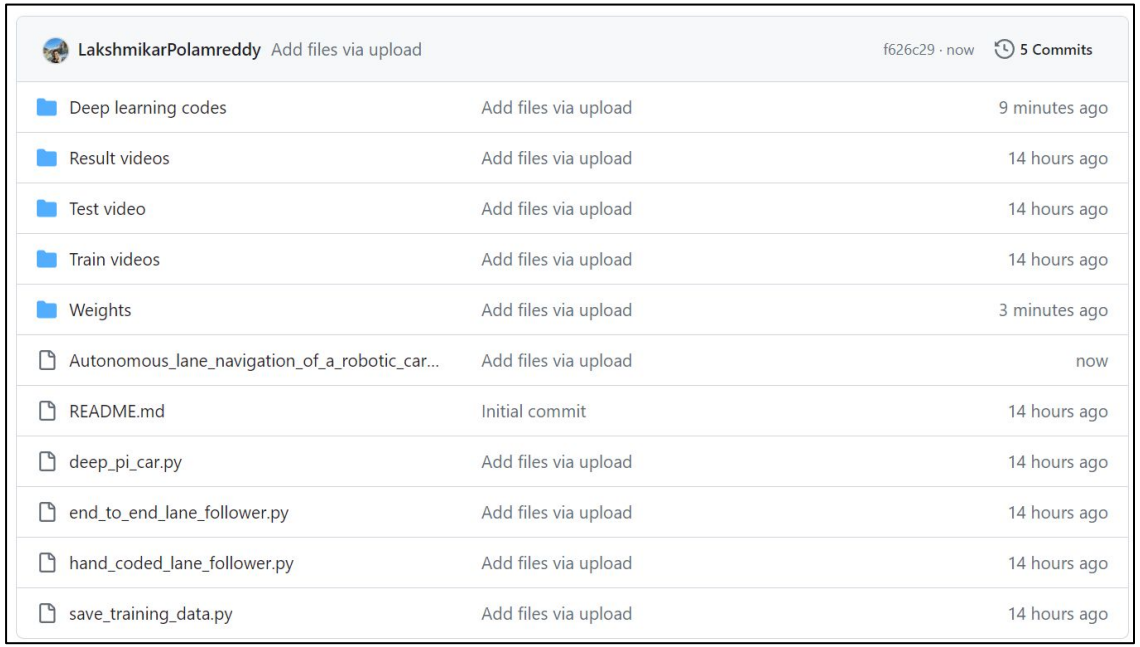
https://github.com/dctian/DeepPiCar

Codes and GitHub link

- Follow this link for dataset, codes, results and report

https://github.com/LakshmikarPolamreddy/Autonomous_lane_navigation_robotic_car_deep_learning_models

- **Deep learning codes** - contains 8 codes of NVIDIA and our CNN model training
- **deep_pi_car.py** - To drive the sun-founder car
- **hand_coded_lane_follower.py** - For steering angle calculation
- **save_training_data.py** - To create frames from the videos
- **end_to_end_lane_follower.py** - To compare the performance of the deep learning model with OpenCV videos by loading the model weight.



- Reference for the above 4 codes is listed in Ref[3]

Future work

- Test the performance of these models by deploying them on the physical robotic car
- Test the performance of these models by generating more training data on multiple tracks
- Extend this work beyond lane navigation to perform traffic sign and pedestrian detection and handling during navigation

References

1. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zieba, K. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

2. https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c

3. https://github.com/dctian/DeepPiCar

Questions?

Thank you!