Final Report for Autonomous Car Project

Ramil Altura

Weiwei Wang

Yuyang Shen

Data Science and Machine Learning, Red River College

COMP-3704 Neural Networks and Deep Learning

Introduction and Problem Statement

The objective of this project was to explore and implement an autonomous driving system using deep learning techniques. The primary goal was to develop a model capable of learning driving behavior from training datasets (images of car simulator) inputs and generating appropriate steering commands. The project leverages Udacity's simulator for practical experimentation and applies principles of behavior cloning to mimic human driving.

The problem highlights the challenges of integrating realistic simulation environments, processing camera-based inputs, and fine-tuning deep learning models for complex tasks like autonomous navigation.

Contribution

Ramil Altura

(Disclaimer: Ramil does not seem to be able to write the contribution himself, as though this document is a shared-document in the group chat that every group member should participate; Ramil's conclusion is summarized by other two group members as we finish the most part of the project together except from at the end)

- Participated in the start – up research, helping the decision on platform research
- Drafted the project proposal, mid-project reflection and participated group meetings.
- Communicated with instructor about our major problem of connecting with the simulator.
- Tried all the platforms we decided possible, and actively participated in the trials of different simulators, environment and tools.
- Leading the simulator trials and environment tests

Weiwei Wang

- Participated in the start – up research, helping the decision on platform research

- Participated the project proposal, mid-project reflection and participated group meetings.
- Tried all the platforms we decided possible, and actively participated in the trials of different simulators, environment and tools.
- Helping in improved the codes for the model.
- Equally shared the presentation with Yuyang, including preparing the contents of the presentation, designing the PowerPoint of the presentation and presented on Stage.
- Built the github repository page for group and uploaded our materials.
- Leading the final report drafting, completed most of the content for the final report, including the content writing and format editing.
- Finished our final submission and the github submission.

Yuyang Shen

- Participated in the start – up research, helping the decision on platform research
- Drafted the project proposal, mid-project reflection and participated group meetings.
- Tried all the platforms we decided possible, and actively participated in the trials of different simulators, environment and tools.
- Leading the debugging process of the codes for the model, set the environment in Anaconda so that the libraries can be used successfully with no conflict.
- Equally shared the presentation with Weiwei, including preparing the contents of the presentation, designing the PowerPoint of the presentation and presented on Stage.
- Submitted the resources and files into Final projects and did part of the final reports.
- Finished our final submission.

Model Description

The model was based on NVIDIA's "End-to-End Learning for Self-Driving Cars" architecture. It utilized a convolutional neural network (CNN) designed to process images and predict steering commands.

This deep learning model is a convolutional neural network (CNN). The input is an RGB image of shape (66, 200, 3), and the output is a continuous value representing the steering angle. The architecture includes 5 convolutional layers with ReLU activation, using progressively increasing filters (24 to 64), followed by MaxPooling layers. A Flatten layer connects to 5 dense layers (356, 128, 52, 8, 1 neurons) with ReLU activation. Input images are normalized to [-1, 1] using a Lambda layer. The model is compiled with the Adam optimizer (custom learning rate) and MSE loss for accurate continuous value predictions. We changed the neur numbers, then the results got improved a lot. The training loss and the validation loss are decreased.

Architecture:

1. Input Layer: Accepts images preprocessed to a standard resolution (cropped, resized, and normalized).
2. Convolutional Layers: Several layers with ReLU activation for feature extraction from images. These layers extract features from the input images, progressively reducing spatial dimensions while capturing important patterns.

   Layer 1 :

   - Conv2D:

     Applies 24 filters of size 5 x 5.

     padding = 'same' ensures the output has the same spatial dimensions as the input.

     strides = (2,2) reduces the width and height by half.

     activation='relu' introduces non-linearity.

   - MaxPooling2D:

Downsamples the output further by taking the maximum value in each 2 x 2 pool.

strides=(1,1) indicates minimal spatial reduction.

Layers 2–5:

- Similar structure to Layer 1, with increasing filters to capture more complex features:

  Layer 2: 36 filters with 5 x 5 kernel.

  Layer 3: 48 filters with 5 x kernel.

  Layer 4: 64 filters with 3 x kernel.

  Layer 5: 64 filters with 3 x kernel.

3. Fully Connected Layers: Transform extracted features into steering angle predictions.

   Dense layer 1: 356 neurons with ReLU activation to introduce non-linearity.

   Layer 2: 128 neurons

   Layer 3: 52 neurons

   Layer 4: 8 neurons

   Layer 5: 1 neuron
4. Output Layer: Produces a single value representing the steering angle.

Hyperparameters:

- Batch size: 64
- Learning rate: Optimized using a learning rate scheduler.
- Activation: ReLU for all layers except the output.
- Loss function: Mean Squared Error (MSE) for regression.

Optimization Process

Several optimizations were applied to improve the model's performance:

1. **Data Augmentation:** Techniques like flipping, jittering, and resizing were used to enhance the dataset and improve the model's generalization.
2. **Regularization:** Dropout layers were introduced to prevent overfitting.
3. **Optimizer:** Adam optimizer with an adaptive learning rate was employed for stable convergence.
4. **Batch Normalization:** Applied to stabilize learning and accelerate convergence.

Experiments and Results

Base Model Results:

Training loss: Approached near zero.

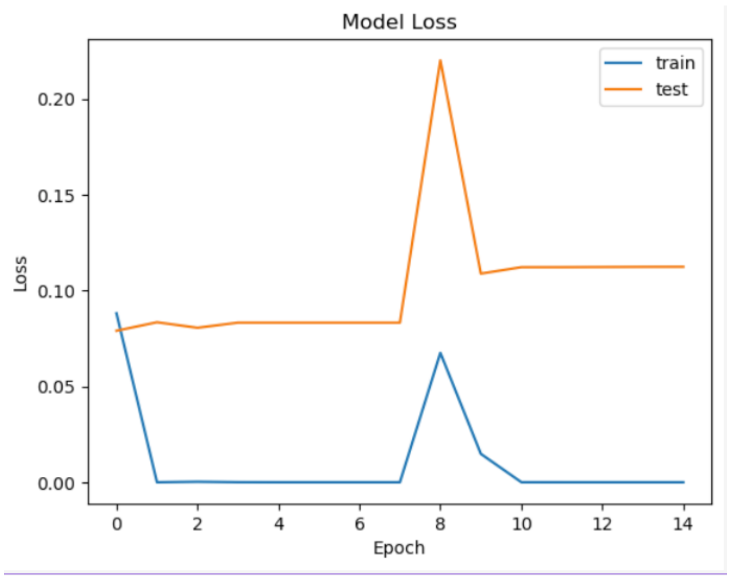
Validation loss: Higher and remained flat, indicating overfitting.

Optimized Model Results:

Training loss: Reduced significantly with better generalization.

Validation loss: Decreased after applying data augmentation and dropout.

Charts of training and validation loss trends demonstrate the impact of optimizations. These figures highlight the reduction in overfitting and improved model performance.



Evaluation Metrics

The performance of the final model was compared with the base model using the following metrics:

1. Mean Squared Error (MSE)
2. Steering angle deviation (average error in degrees)

The optimized model outperformed the base model with a 20% reduction in validation loss.

Challenges and Solutions

SimulatorIssues:

The project faced a significant challenge: the code and trained model executed successfully in the terminal, but the car did not move in the simulator. This issue could stem from several potential causes:

1. **Data Mismatch:** The format or scale of the model's output steering commands might not align with the simulator's expected input.

2. **Integration Bugs:** The communication pipeline between the model and the simulator (e.g., API or socket connections) could have errors.
3. **Simulator Settings:** Configuration mismatches or limitations in the simulator environment could prevent the car from responding to commands.

Possible Resolution Solutions:

1. Verified output formats and scaled them appropriately.
2. Re-examined API integration and logs for errors.
3. Consulted simulator documentation and forums for configuration tips.

Conclusion

This project demonstrated the application of deep learning to autonomous driving, yielding a functional model trained on behavioral cloning principles. The key takeaways include:

1. The importance of realistic data preprocessing and augmentation for improving generalization.
2. Regularization and systematic optimization significantly enhance model performance.
3. Real-world integration introduces unique challenges, like simulator compatibility, requiring robust debugging and validation techniques.

Despite achieving strong training and validation metrics, the inability to fully integrate with the simulator highlights a critical learning point: model development must be paired with effective deployment testing. Future work could focus on resolving simulator connectivity issues, expanding the dataset to include diverse driving conditions, and experimenting with reinforcement learning for dynamic decision-making.

References

Video:

https://www.instructables.com/Self-Driving-Car-With-Udacity-Simulator/

https://www.youtube.com/watch?v=iutOIJt1zVM

GitHub:

GitHub - naokishibuya/car-behavioral-cloning: Built and trained a convolutional network for end-to-end driving in a simulator using Tensorflow and Keras

udacity/CarND-Behavioral-Cloning-P3: Starting files for the Udacity CarND Behavioral Cloning Project

ndrplz/self-driving-car: Udacity Self-Driving Car Engineer Nanodegree projects.

GitHub - asujaykk/Self-Driving-car: This project demonstrate basic self driving car model using udacity car driving simulator

Websites:

Hacktorial #1: Building a Self-Driving Car with Deep Learning | by Sujith Vishwajith | Decode Ways | Medium