Self-Driving Car using Deep CNN

Abstract

This project focuses on developing a self-driving car model using deep learning techniques, specifically employing behavior cloning with a Convolutional Neural Network (CNN). The model is trained to predict steering angles from camera images captured by the Udacity open-source car simulator. Inspired by NVIDIA's End-to-End Deep Learning for Self-Driving Cars. This system maps raw pixel data directly to steering commands, without relying on explicit lane detection or path planning, the model employs image preprocessing and data augmentation to improve training. The performance of the model is evaluated based on its ability to autonomously drive within the simulator, achieving reliable steering angle predictions.

Introduction

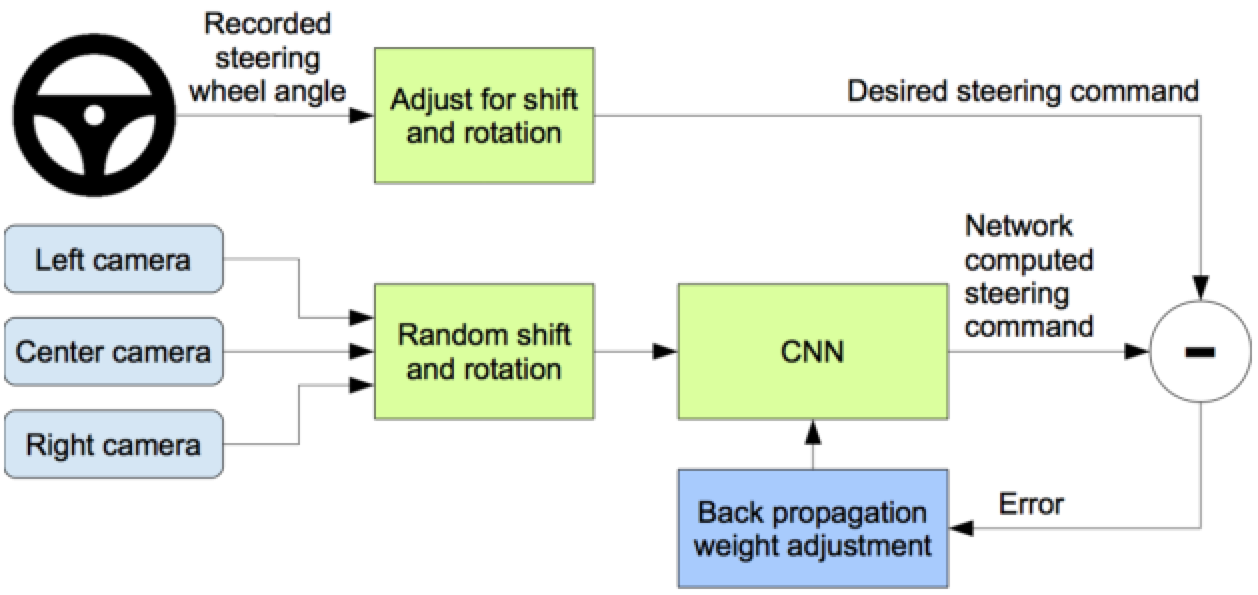
Self-driving cars have become one of the most promising applications of artificial intelligence (AI) in recent years, with the potential to revolutionize transportation, improve safety, and reduce human error. Autonomous driving systems require the ability to interpret and respond to dynamic environments, a task traditionally achieved using a combination of computer vision, path planning, and control systems. However, an emerging approach—end-to-end deep learning—aims to simplify this by directly mapping sensory data to driving actions.

This project leverages behavior cloning, a form of supervised learning, to train a deep neural network model to predict steering angles from camera images. Inspired by NVIDIA's end-to-end deep learning approach for self-driving cars, the model is trained using images from the Udacity open-source car simulator, which provides a virtual environment for developing and testing autonomous driving systems. The model utilizes three cameras (center, left, and right) as inputs to provide a wider field of view, enhancing its ability to navigate complex driving scenarios.

While previous approaches often rely on explicit algorithms for tasks like lane detection or path planning, this project focuses on training the model to learn driving behaviors directly from human steering and throttle data, without the need for hand-crafted features. The goal of this project is to demonstrate the feasibility of using deep learning for autonomous driving, while also highlighting the challenges and potential solutions when working with real-time driving simulations.

Methods

To develop a self-driving car model, I employed a behavior cloning approach, where a deep learning model learns to predict steering angles and throttle based on the speed and camera images or feed. The model architecture is inspired by NVIDIA’s End-to-End Deep Learning for Self-Driving Cars, which uses a Convolutional Neural Network (CNN).


Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
−
Back propagation weight adjustment
Error

The project used the Udacity open-source car simulator to generate data for training and testing. The simulator provided images captured by three cameras—center, left, and right—along with the corresponding steering angles, throttle, speed and brake. These images served as the input to the model, which learned to predict the appropriate steering angles based on the visual data provided by the cameras.

Data Balancing

To address the imbalance in the dataset, where certain steering angles (especially near-zero angles) were overrepresented, I used a data balancing technique. This involved dividing the steering angles into multiple bins and ensuring that each bin contained a balanced number of samples. The steps involved:

1. Histogram Creation: The steering angles were divided into 31 bins, and the number of samples in each bin was computed.
2. Sample Removal: For bins with more than 1200 samples, random samples were removed to ensure that each bin had a more balanced representation.
3. Shuffling: The removed samples were randomly selected to prevent any bias during training.

By applying this technique, I was able to create a more balanced dataset, which helped the model generalize better by preventing it from overfitting to the most frequent steering angles.

Preprocessing and Data Augmentation

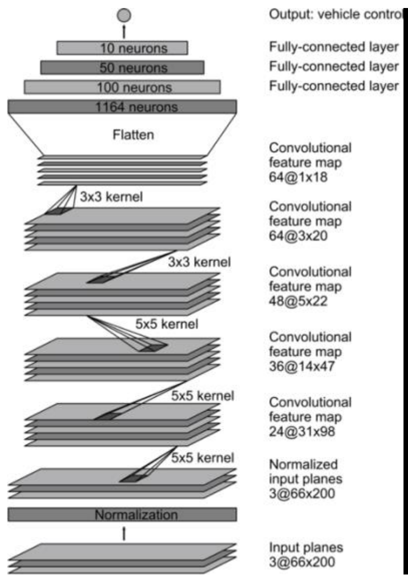
I applied several preprocessing techniques to ensure that the model could learn effectively from the images.

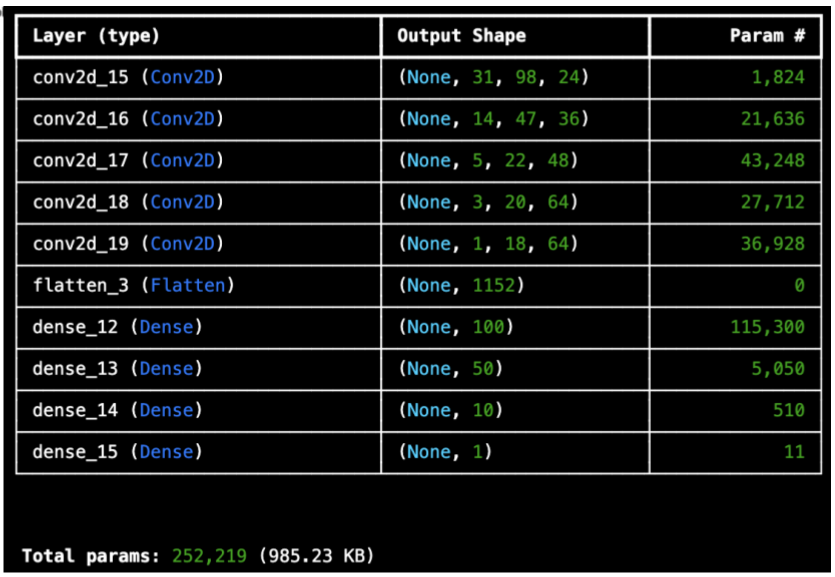
1. Cropping: Cropped the images to focus on the relevant area of the road, removing unnecessary parts such as the sky and the car's exterior.

2. Color Space Conversion: The images were transformed from RGB to YUV color space to improve the model's ability to better interpret the road and surrounding environment, which was also done by NVIDIA.
3. Normalization: The pixel values were normalized to a range of [0, 1] by dividing by 255.
    a. Resizing: Resized the images to 200x66 as recommended by NVIDIA.
4. Image Augmentation: To enhance the model's ability to generalize, I applied random transformations to the images during training, with each transformation having a 40% probability of being applied. These transformations included panning, zooming, adjusting brightness, and flipping images horizontally, with corresponding adjustments made to the steering angles.

Model Architecture

The model used a Convolutional Neural Network (CNN) with a similar architecture to NVIDIA's end-to-end deep learning model. The CNN was designed to learn spatial features from the images and map them to the predicted steering angle. The model consisted of several convolutional layers followed by fully connected layers





```
Layer (type)            Output Shape              Param #
conv2d_15 (Conv2D)      (None, 31, 98, 24)          1,824
conv2d_16 (Conv2D)      (None, 14, 47, 36)         21,636
conv2d_17 (Conv2D)      (None, 5, 22, 48)          43,248
conv2d_18 (Conv2D)      (None, 3, 20, 64)          27,712
conv2d_19 (Conv2D)      (None, 1, 18, 64)          36,928
flatten_3 (Flatten)     (None, 1152)                    0
dense_12 (Dense)        (None, 100)               115,300
dense_13 (Dense)        (None, 50)                  5,050
dense_14 (Dense)        (None, 10)                    510
dense_15 (Dense)        (None, 1)                      11

Total params: 252,219 (985.23 KB)
```

left: NVIDIA's architecture right: My Model

The network included several Conv2D layers with ELU activations to extract features from the input images. These layers progressively reduced the spatial dimensions of the images while increasing the depth of the feature maps.

After the convolutional layers, the output was flattened and passed through fully connected layers, which helped the model learn higher-level abstractions of the features.

The final layer consisted of a single neuron that predicted the steering angle, without an activation function, as this is a regression task.

The model was implemented and trained using TensorFlow-Keras. The model was trained with Mean Squared Error (MSE) loss, and the Adam optimizer was selected for its adaptive learning rate. For evaluation, Mean Absolute Error (MAE) was used as the performance metric, which is common for regression tasks.

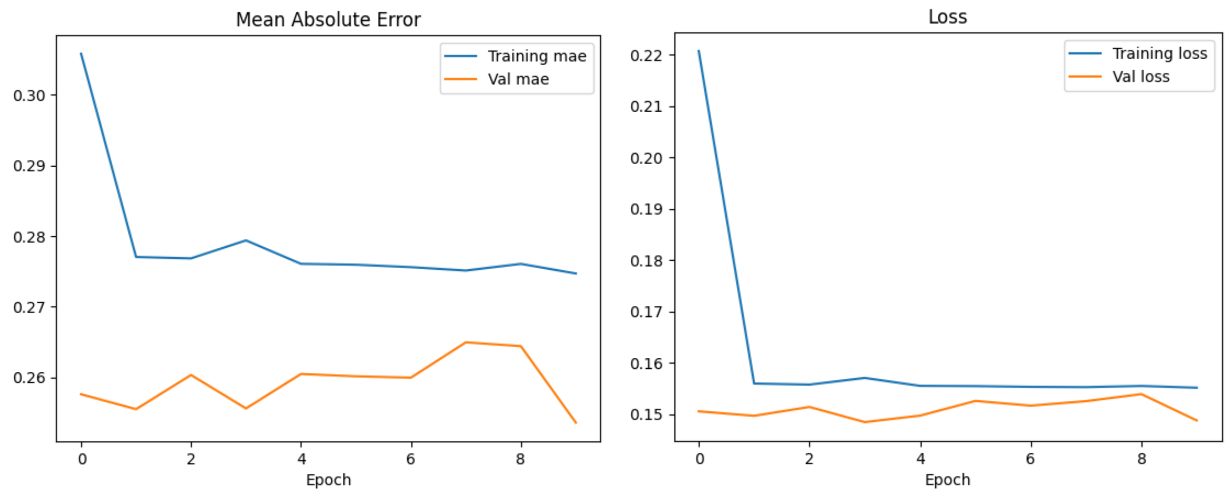
After training, the model is saved as ‘model.h5’ file. This allows the model to be easily loaded and used for inference without retraining. This contains the architecture, weights, and training configuration, allowing the model to be reloaded and used at any time.

Autonomous Driving

To enable autonomous driving, the saved model is loaded in the Autonomous.py script which interacts with the simulator through a Flask server and SocketIO for real-time communication. The server listens for telemetry data from the simulator, which includes images from three cameras (center, left, and right) and speed information. Upon receiving the data, the images are decoded, preprocessed (cropped, converted to YUV color space, blurred, resized to 200x66 pixels, and normalized) as recommended by NVIDIA, and then passed to the model for steering angle prediction.

The simulator continuously provides updated telemetry, including new images and vehicle status, which allows the model to adjust its predictions and refine control commands, creating a feedback loop for autonomous driving within the simulator.

Results



Both the training MAE and loss decreases rapidly in the initial epochs, indicating that the model is learning quickly from the training data. After a few epochs, the training MAE and Loss stabilizes and fluctuates around a relatively low value. This suggests that the model has learned the underlying patterns in the training data and is not overfitting significantly.

The validation graph also decreases initially, but it plateaus earlier than the training graph. This indicates that the model's performance on unseen data is not improving as much as on the training data.

In this case, the difference between the training and validation graph is relatively small, indicating that overfitting is not a major issue.

Challenges and Issues

Package Importing Issues: During development, I encountered issues with importing the necessary packages, which prevented me from running the model autonomously with the simulator. This hindered my ability to perform end-to-end testing of the car in the simulated

Training: Due to some issues on my MacBook, I was unable to train the model locally. Instead, I used Google Collab, where I was able to install the compatible version of TensorFlow (2.12) and train the model. Training the model on nearly 30,000 images took a significant amount of time due to the large dataset and the computational complexity of the model.

Operating System: As my laptop runs on an ARM-based MacBook M1 chip, and the simulator was not compatible natively, I had to set up a Windows virtual machine to run the Udacity car simulator. This workaround enabled me to use the simulator for data generation and testing.