PROGRESS REPORT: AUTONOMOUS CAR USING CNN

Rudra Dey
Student# 1010124866
rudra.dey@mail.utoronto.ca

Pravin Kalaivannan
Student# 1010141295
pravin.kalaivannan@mail.utoronto.ca

Aadavan Vasudevan
Student# 1010101514
aadavan.vasudevan@mail.utoronto.ca

Abishan Baheerathan
Student# 1010218756
abishan.baheerathan@mail.utoronto.ca

ABSTRACT

This document presents our team's procedure to create the baseline model code for an autonomous self-driving car. ...

1 PROJECT DESCRIPTION

2 INDIVIDUAL CONTRIBUTIONS AND RESPONSIBILITIES

2.1 INDIVIDUAL CONTRIBUTIONS

3 DATA PROCESSING

Data was collected from the CARLA simulator, including images and corresponding steering values. We then normalized the images allowing us to train our CNN. Further, data augmentation was applied to increase the size and diversity of the dataset, leading to better generalization of the model.

3.1 DATA COLLECTION AND CLEANING

In CARLA, there is vehicles that have autonomous functionality, which we used to collect data. We attached an RGB Camera Sensor to the car to record 10 frames per second (FPS) as the car travels around a designated map. In addition to the images, data for the steering values and turn signals of the car was collected allowing for use as ground truth labels for training the model. This data collection was done on diffreent maps with different weather conditions to ensure a diverse dataset. We collected samples on three maps, in which the car collected data for 30 minitues, resulting in 54000 total images.

To clean the data, we removed any images that were all black, duplicates or not the correct resolution of 320x240 leaving us with 53457 images. We then saved the data as numpy arrays, zipped and uploaded to a shared Google Colab file along, ensuring access for all members.

Procedure:

1. Spawn in a vehicle
2. Attach an RGB Camera Sensor (front-facing) to the dash of the car.
3. Collect image data (10 fps) along with steering data and turn signals.

4. Clean the data by removing corrupted images and duplicates.
5. Save the data as numpy arrays for easy access and processing.

3.2 Data Preprocessing

All collected images were resized to a resolution of 160x120 pixels, with the pixel values normalized to a range of [0, 1]. This was done to ensure consistency in the input data, allowing the model to learn effectively from the images.

3.3 Data Distribution

We decided to do a 80/10/10 split for the training, validation and test sets respecivelty. This was done to ensure that have a large set of data for training, while also having enough data for validation and testing. Table 3 shows the distribution of the data.

Dataset Split	**Total Images**
Training Set	42766
Validation Set	5345
Test Set	5346
Total Images	53457

Table 1: Dataset Distribution

3.4 Data Agumentation

Data augmentation was applied to the training set to increase the size and diversity of the dataset. We applied two tenchniquies to generalzie the model and forces it to learn critical features.

- **Greyscale Conversion:** All training images were converted to greyscale reducing model's reliance on color-based features, encouraging learning of more genreal visual features/road patterns. This will allow the model to improve, and work in different lighting conditions or if the camera is distorted.
- **Cutout Augmentation:** A rectangular region of each training image was masked out to simulate real-world situations (e.g. raindrop or debris on the camera lens). This encoruses the model to learn relevant features from different parts of the image making it more robust overall.

These agumentations lead to the training set being increased to 128298 images, with each image having two copies with each augmentations applied respecivelty.

Dataset Split	**Total Images**
Training Set	42766 x 3 = 128298
Validation Set	5345
Test Set	5346
Total Images	138989

Table 2: Dataset Distribution after Training Set Augmentation

3.5 Plan for Testing on Unseen Data

In order to evaluate the model's performance, we will be testing it on unseen data which we reserved 10% of the dataset for. Given how the data was collected sequentially across different maps and weather conditions, the test set will inheritely contain frames the model has not encountered before. This will provide the model differnt roads and turns to encounter and we can see how well it generalizes to new situations.

Additionally, to further test the model's generalization, we can load the model in CARLA and make it drive around a new map where data was not collected from, and see how well the model performs in real-time. This will allow us to visualize the model's performance and see how well it can adapt to new environments.

3.6 Challeneges Faced and Solutions

During the data collection process, we faced several challenges and overcame them with the following solutions:

- **Slow Initial Data Collection:** The simulator originally ran in real time, meaning that collecting data for 30 minitues would take that long in real time. To solve this issue, we sped up the simulation, allowing us to collect data at a faster rate and collect more data in a shorter time frame.

- **Turn Signal Data Collection:** Collecting turn signal data was initially problematic, because the vehicle's indicators were not being used by the automated vehicle in CARLA. We tried to resolve this through manual activation based on steering angle, but this proved to be innacurate and inconsistent. We resolved this by enabling the CARLA Traffic Manager's automatic lights, allowing turn signals to activate based on the vehicle's navigation. Allowing us to have accurate turn signal data for the training set.

4 Baseline Model

Using a Ridge Regression Algoirthm we created a baseline model that predicts steering angles for a self-driving car from grayscale camera images and turn signal inputs. This baseline model serves as a simple, interpretable baseline such that we can compare against our more complex primary neural network model later. To collect data, we used a simulator called CARLA, to obtain camera images, turn signal inputs, and our ground truth label the steering angles.

4.1 Ridge Regression with Image Features

Initially, the grayscale images (of shape 160x120) were flattened into 1D feature vectors and normalized. The features themselves represent the visual input of the car's front-facing camera. Additionally, left and right turning signals were captured as an additional feature; combining this with our flattened grayscale images we were left with a numpy array of shape $N \times 19201$, where N represents the number of images, and 19201 are the number of feature column vectors. The Ridge Regression algorithm was selected because it penalized large coefficients (L_2 norm), allowing for a generalized model. A series of models were trained with different regularization strengths (α), and the performance was evaluated using Mean Squared Error (MSE) and R^2 Score on a held-out 20% test set.

4.2 Minimal Tuning of α

The only hyperparameter needing to be tuned was the regularization strength for the Ridge Regression Algorithm. This was manually tuned, and thus no validation set was used. A small set of candidate values for α was chosen: $\alpha \, \epsilon \, \{1, 10, 100, 250, 1000, 8000\}$ With each value of α, the model was trained and tested with a dataset of 15 minutes of simulated driving. By obtaining the average MSE and the average R^2 score, our fourth model of $\alpha = 100$ came out to be the best.

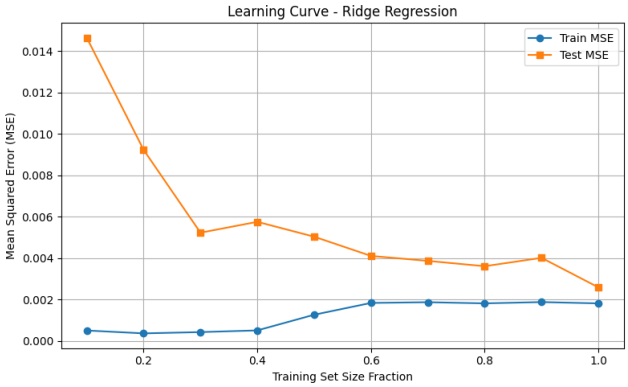

Figure 1: This is the learning curve for "Model 4" a Ridge Regression Model with $\alpha = 100$

4.3 Overfitting and Model Quality

As shown in Figure 1, the results are clear.

4.4 Qualitative Observations

The following libraries will be used:

1. numpy: For numerical operations on arrays
2. matplotlib: For visualization of data
3. scikit-learn: For machine learning components
   (a) Ridge: A regularized linear regression model
   (b) MSE: Mean Square Error for evaluation metrics
   (c) train_test_split: For data splitting

The data will be loaded using two numpy arrays: images.npy and angles.npy. The images.npy will contain N grayscale images of resolutions 160x120, and the angles will have the corresponding images steering angles (of the car) in a range from [-1,1]. This makes it so the image data is represented as 3D arrays of samples x hieght x width as the dimension. The steering angles are continuous (regression problem).

The images will be flattened from their 2D representation of 160x120 matrix to a 1D representation of 19200 vectors. Where each image becomes a single row in a feature matrix. Additionally, we will split do a 80-20 train-test split.

The baseline model will be a ridge regression model. The ridge regression model is basically, the same as a linear regression model except, it regularizes large weights. It adds an L2 penality to discourage large weights, this helps with overfitting. The same as linear regression as we try to fit: $\hat{y} = Xw + b$, but now we try to minimize the loss function: $LOSS = MSE + \alpha||\omega||_2^2$. The model will learn the weights corresponding to each pixel's contribution to steering.

We will compute two metrics, the mean square error (MSE) and the R^2 score (proportion of variance).

This Ridge Regression Algorithm is a linear regression technique which introduces a L_2 penalty which discourages the model from using very large weights, and thus improving generalization to unseen data. This model was chosen due to its simplicity and fast training time but also allowed quick adjustment to an α value which allowed us to adjust the regularization effect on the performance. By not using a plain linear regresion model, we were able to prevent major overfitting, and generalize better.