PRISTINI
SCHOOL OF AI

Group #1

Autonomous Driving System Simulation

Proposed By:

Nour Yaakoub

Eya Ben Ismail

Aziz Ben Youssef

Bocar BA

Med Amine Dahmen

Proposed to:

Ms. Besma Ben Salah

Abstract

In this project, we built a deep learning model that can predict the steering angle of a car using images from a front-facing camera—an essential component in self-driving systems. Our main focus was on preparing the dataset, making it more balanced, and applying image augmentation techniques to mimic real-life driving situations. Then, we trained a convolutional neural network (CNN) to handle these scenarios better. This helped us improve how well the model works, especially when facing turns or changes in lighting and road conditions.

Keywords:

Self-driving car, Deep Learning, CNN, Steering angle, Data augmentation, TensorFlow

Table of Contents

1. Introduction

Self-driving technology is becoming more common and could reshape transportation in the near future. One key part of making it work is teaching cars how to make driving decisions—like when and how much to turn the wheel. In this project, we developed a system that takes in camera images and predicts the right steering angle using deep learning. Our goal was to make this prediction as accurate as possible, even in less-than-ideal conditions.

2. Project Context

2.1 Problem Statement

Most driving datasets have a lot of straight driving and not enough data for curves or sharp turns. This creates a bias that makes deep learning models perform poorly when faced with turns.

2.2 Objective

We set out to build a model that could predict steering angles more accurately, especially in varied and realistic driving conditions. To do that, we cleaned and balanced our dataset, and added image augmentations to better reflect real-world situations.

2.3 Methodology

We followed an agile methodology to divide the work into sprints

- Sprint 1 :
  - Data Loading and Processing
- Sprint 2 :
  - Data Balancing
  - Training and Validation split
  - Data Augmentation
  - Model Training and Evaluation
- Sprint 3:
  - image preprocessing
  - batch generator
  - implementing Nvidia model
- Sprint 4:
  - model training
  - environment and requirements setup
  - testing the model

2.3.1 Product Backlog

The product backlog outlines the main tasks prioritized by value and difficulty, while each sprint backlog details the tasks completed during that phase.

ID	User Story	Feature	Priority	Estimation (Story Points)
1	As a developer, I want to load and clean the driving dataset	Data Processing	High	5
2	As a data scientist, I want to balance the dataset to avoid steering angle bias	Data Balancing	High	8
3	As a developer, I want to apply realistic augmentations to improve generalization	Data Augmentation	High	8
4	As a model builder, I want to define and train a CNN to predict steering angles	Model Architecture & Training	High	13
5	As a tester, I want to evaluate the model and visualize predictions	Evaluation & Plotting	Medium	5
6	As a developer, I want to build a custom data generator for efficient training	Batch Generator	Medium	8
7	As an engineer, I want to implement Nvidia's CNN architecture for better performance	Nvidia Model Integration	Medium	13
8	As a developer, I want to package the environment and run final tests	Testing & Deployment	Low	5

2.3.2. Sprint Backlog

Sprint 1:

Task	**Status**	**Story Points**
Load and normalize data paths	Done	2
Visualize raw steering angles	Done	3

Sprint 2:

Task	**Status**	**Story Points**
Histogram analysis of angle distribution	Done	2
Undersample straight-driving samples	Done	4
Apply zoom, pan, brightness, flip	Done	4
Train baseline CNN	Done	8
Evaluate model (loss, plots)	Done	3

Sprint 3:

Task	**Status**	**Story Points**
Implement image preprocessing pipeline	Done	5
Develop YUV color space conversion	Done	3
Create efficient batch generator	Done	8
Integrate preprocessing with training pipeline	Done	3
Evaluate preprocessing impact on model performance	Done	2

Sprint 4:

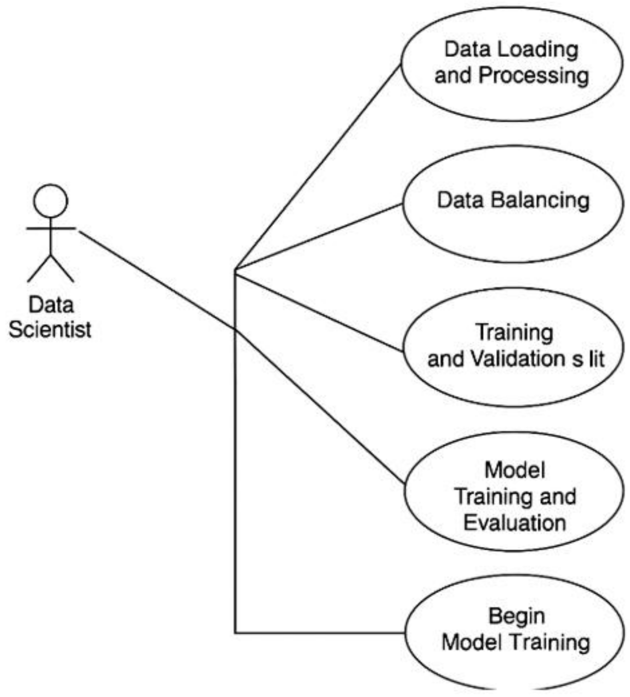
Task	Status	Story point
Prepare the environment and Requirements Setup	Done	5
Testing	Done	3

3. Tools and Technology Stack

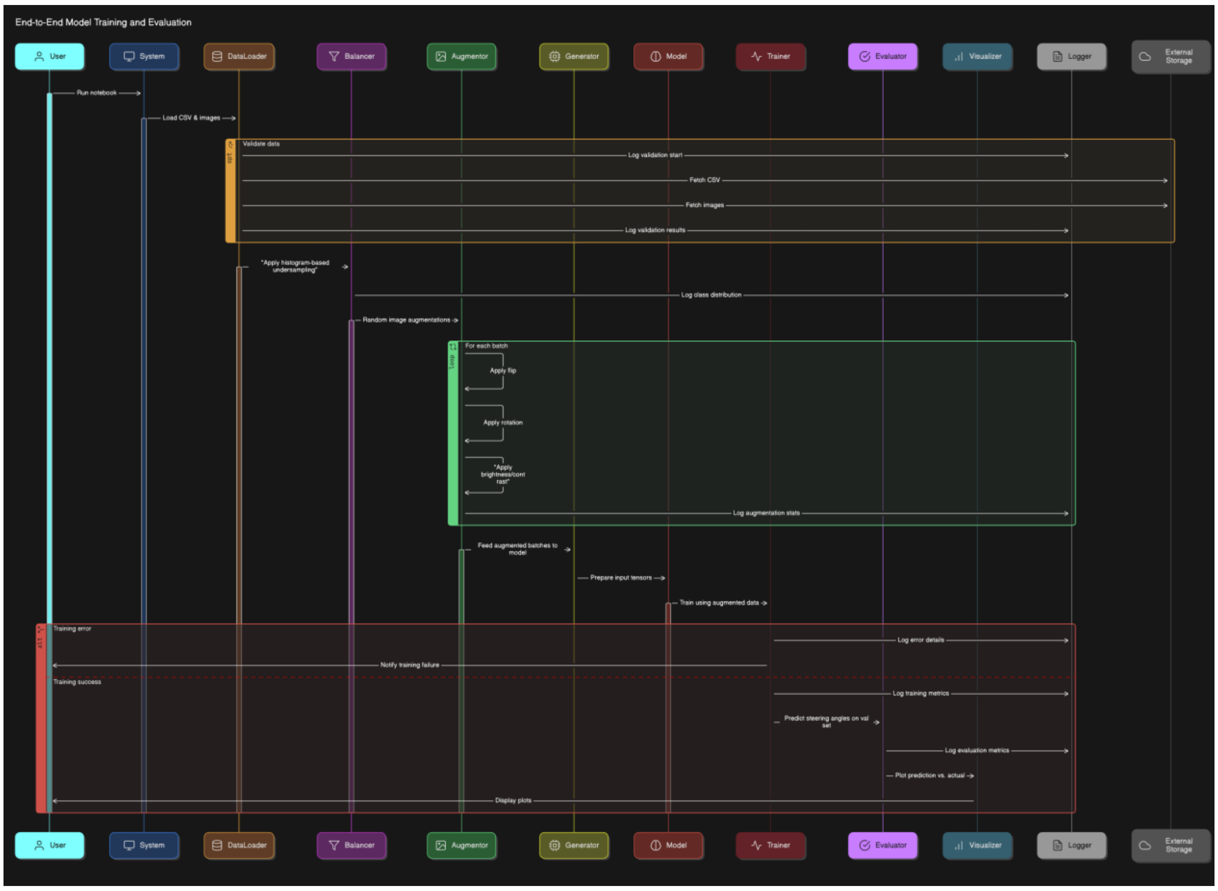
- Languages & Frameworks: Python, TensorFlow, Keras
- Libraries: Pandas, NumPy, Matplotlib, OpenCV, Albumentations
- Platform: Google Colab
- Version Control: Git

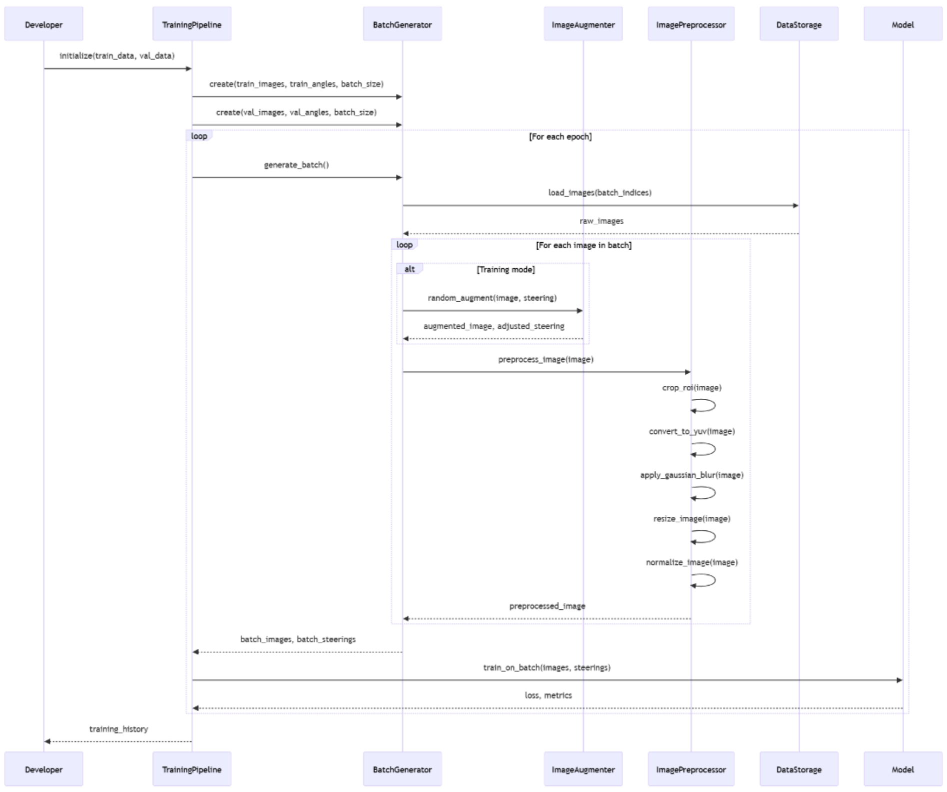
4. System Design and Architecture

4.1. Use Case Diagram

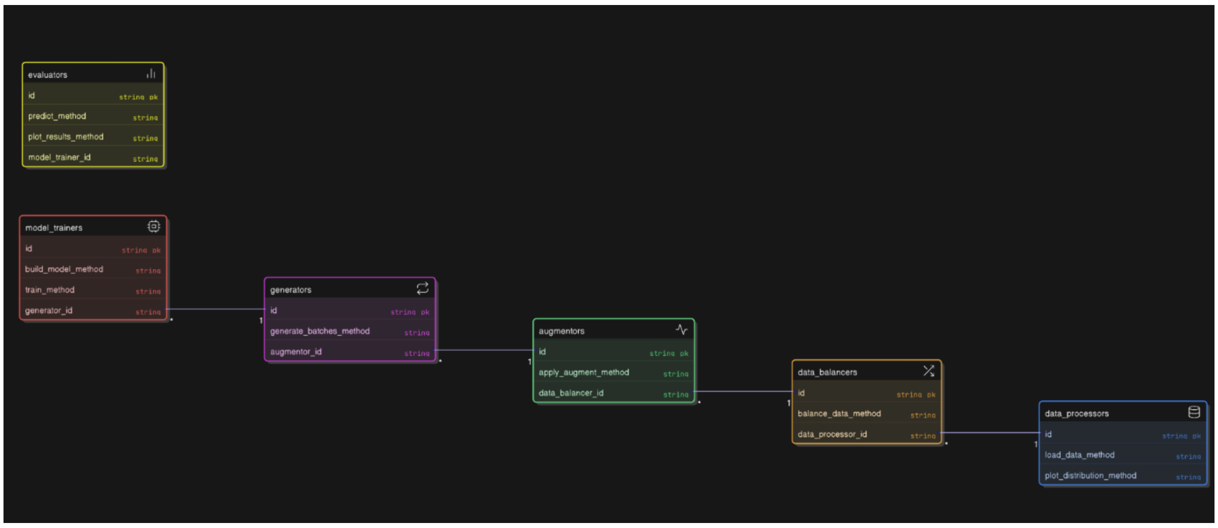

Data Scientist
Data Loading and Processing
Data Balancing
Training and Validation s lit
Model Training and Evaluation
Begin Model Training

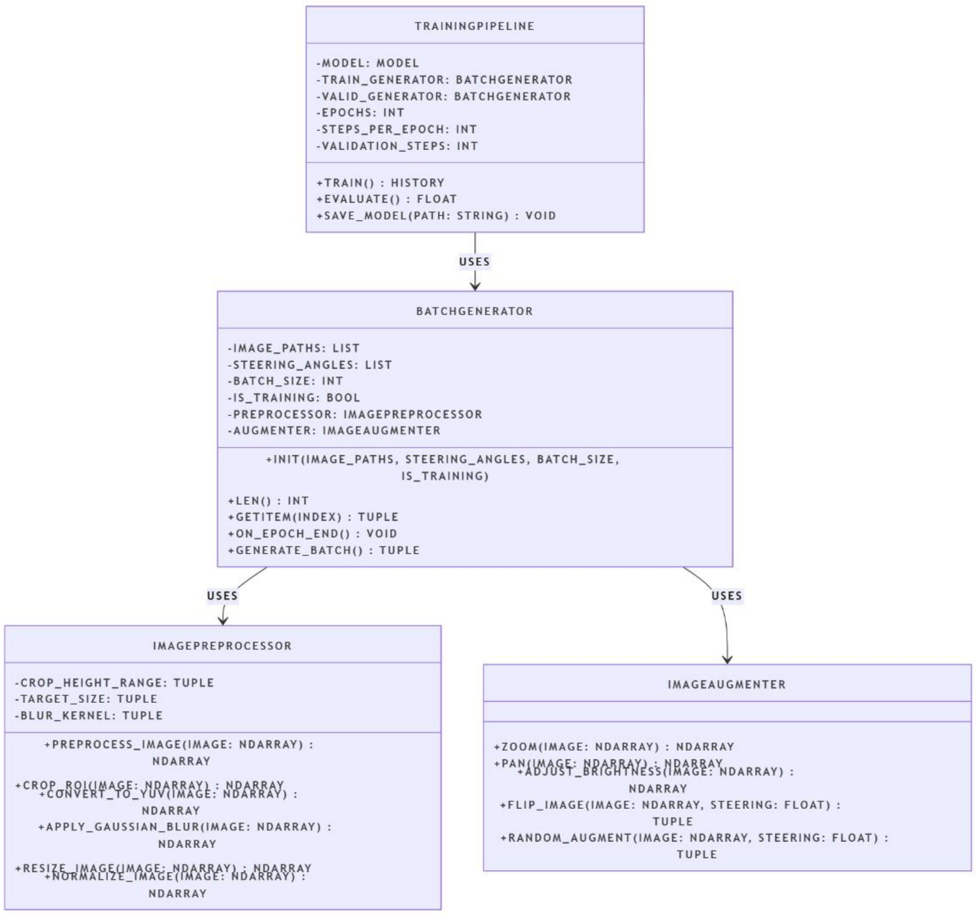
4.2. Sequence diagram


End-to-End Model Training and Evaluation
User
System
DataLoader
Balancer
Augmentor
Generator
Model
Trainer
Evaluator
Visualizer
Logger
External Storage
Run notebook
Load CSV & images
opt
Validate data
Log validation start
Fetch CSV
Fetch images
Log validation results
"Apply histogram-based undersampling"
Log class distribution
Random image augmentations
loop
For each batch
Apply flip
Apply rotation
"Apply brightness/contrast"
Log augmentation stats
Feed augmented batches to model
Prepare input tensors
Train using augmented data
alt
Training error
Log error details
Notify training failure
Training success
Log training metrics
Predict steering angles on val set
Log evaluation metrics
Plot prediction vs. actual
Display plots


Developer
TrainingPipeline
BatchGenerator
ImageAugmenter
ImagePreprocessor
DataStorage
Model
initialize(train_data, val_data)
create(train_images, train_angles, batch_size)
create(val_images, val_angles, batch_size)
loop
[For each epoch]
generate_batch()
load_images(batch_indices)
raw_images
loop
[For each image in batch]
alt
[Training mode]
random_augment(image, steering)
augmented_image, adjusted_steering
preprocess_image(image)
crop_roi(image)
convert_to_yuv(image)
apply_gaussian_blur(image)
resize_image(image)
normalize_image(image)
preprocessed_image
batch_images, batch_steerings
train_on_batch(images, steerings)
loss, metrics
training_history

4.3. Class diagram




TRAININGPIPELINE
-MODEL: MODEL
-TRAIN_GENERATOR: BATCHGENERATOR
-VALID_GENERATOR: BATCHGENERATOR
-EPOCHS: INT
-STEPS_PER_EPOCH: INT
-VALIDATION_STEPS: INT
+TRAIN() : HISTORY
+EVALUATE() : FLOAT
+SAVE_MODEL(PATH: STRING) : VOID
USES
BATCHGENERATOR
-IMAGE_PATHS: LIST
-STEERING_ANGLES: LIST
-BATCH_SIZE: INT
-IS_TRAINING: BOOL
-PREPROCESSOR: IMAGEPREPROCESSOR
-AUGMENTER: IMAGEAUGMENTER
+INIT(IMAGE_PATHS, STEERING_ANGLES, BATCH_SIZE, IS_TRAINING)
+LEN() : INT
+GETITEM(INDEX) : TUPLE
+ON_EPOCH_END() : VOID
+GENERATE_BATCH() : TUPLE
USES
USES
IMAGEPREPROCESSOR
-CROP_HEIGHT_RANGE: TUPLE
-TARGET_SIZE: TUPLE
-BLUR_KERNEL: TUPLE
+PREPROCESS_IMAGE(IMAGE: NDARRAY) : NDARRAY
+CROP_ROI(IMAGE: NDARRAY) : NDARRAY
+CONVERT_TO_YUV(IMAGE: NDARRAY) : NDARRAY
+APPLY_GAUSSIAN_BLUR(IMAGE: NDARRAY) : NDARRAY
+RESIZE_IMAGE(IMAGE: NDARRAY) : NDARRAY
+NORMALIZE_IMAGE(IMAGE: NDARRAY) : NDARRAY
IMAGEAUGMENTER
+ZOOM(IMAGE: NDARRAY) : NDARRAY
+PAN(IMAGE: NDARRAY) : NDARRAY
+ADJUST_BRIGHTNESS(IMAGE: NDARRAY) : NDARRAY
+FLIP_IMAGE(IMAGE: NDARRAY, STEERING: FLOAT) : TUPLE
+RANDOM_AUGMENT(IMAGE: NDARRAY, STEERING: FLOAT) : TUPLE

5. Implementation

5.1 Data Processing

- Loaded the CSV logs with image paths and steering angles

- Cleaned and normalized the paths
- Plotted the distribution of steering angles to understand the dataset

5.2 Data Balancing

- Histogram analysis of angle frequency
- Used histogram-based undersampling to reduce the number of straight-driving samples

5.3 Data Augmentation

To simulate different driving conditions, we applied several image transformations:

- **Zooming:** To imitate a vehicle getting closer to objects
- **Panning:** Shifted the image horizontally and vertically
- **Brightness Adjustments:** To simulate different lighting scenarios
- **Flipping:** Horizontally flipped images and inverted the steering angle accordingly
- **Random Combinations:** We applied these in random order to add more variation

5.4 Model Architecture

- Built a CNN using Keras
- Layers included Conv2D, MaxPooling, Dropout, Flatten, and Dense
- Used Mean Squared Error as the loss function
- Optimized with the Adam optimizer

6. Evaluation and Results

- We tracked training and validation loss throughout the training process
- Plotted predicted steering angles and compared them with actual ones
- Observed a noticeable improvement after balancing and augmenting the data
- Discussed how well the model generalizes, and how overfitting was addressed

7. Conclusion and Future Work

Summary

We were able to successfully train a model that predicts steering angles from camera input. Thanks to data balancing and augmentation, the model became much more reliable in different driving scenarios—not just going straight.

8. References

- Youtube Channel: deDSwithBappy (link:https://youtube.com/playlist?list=PLkz_y24mlSJawbZzfJrrxZi0QNmJ5aP6&si=pqcxVFW0OXLOouoN)