National College of Ireland

Configuration Manual

MSc Research Project
MSc Artificial Intelligence

Rahul Goswami
Student ID: X23167572

School of Computing
National College of Ireland

Supervisor: Arundev Vamadevan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Rahul Goswami

Student ID: X23167572

Programme: MSc Artificial Intelligence **Year:** 2024

Module: Practicum

Lecturer: Arundev Vamadevan

Submission Due Date: 12/12/2024

Project Title: Advancements in Steering Angle Prediction: Deep Learning Approaches for Self-Driving Car

Word Count: 1038 **Page Count:** 5

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rahul Goswami

Date: 12/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rahul Goswami
Student ID: X23167572

1 Introduction

Configuration document: This provides a step-by-step guide to setting up the necessary software and hardware environment that will allow reproduction of the self-driving car project. In general, the work is focused on developing, training, and testing three deep learning models to predict steering angles from input data obtained using the Udacity Self-Driving Car Simulator. The following readme describes environment settings, library installation information, information about the dataset, code execution, and methodology for model performance evaluation.

2 Environmental Setup

2.1 System Hardware Configuration

- Processor: Intel Core i5 (4 cores)
- GPU: NVIDIA RTX 3050
- RAM: 16 GB
- Storage: 512 GB SSD

2.2 Software Requirements

- Operating System: Windows 10
- Python Version: 3.7.12 (Anaconda Environment)
- IDE: VSCode

2.3 **Libraries:**

The following libraries and modules are necessary to run the models and configure the environment. A requirements.txt file has been provided for easy setup. To install the dependencies, use:

```
>pip install -r requirements.txt
```

Contents of requirements.txt:

```
opencv-contrib-python
numpy
matplotlib
tensorflow
keras
python-socketio
python-engineio
Flask
Eventlet
```

Library Descriptions:

- **OpenCV** (opencv-contrib-python): Provides tools for image processing and augmentation.
- **NumPy** (numpy): Used for numerical computations.
- **Matplotlib** (matplotlib): For visualizing data distributions and training performance.
- **TensorFlow** (tensorflow==2.17.0) and Keras (keras==3.6.0): Essential for developing, training, and evaluating deep learning models.
- **Python-SocketIO & Python-EngineIO**: Facilitate communication between the drive.py script and the Udacity simulator.
- **Flask & Eventlet:** Required to create a local server for interacting with the simulator.
- **Scikit-Learn** (scikit-learn): Used for data preprocessing and model evaluation metrics.
- **Pandas** (pandas): For data manipulation and analysis.

3 Dataset Description

The dataset for this project was captured from the Udacity Self-Driving Car Simulator. It consists of images taken from three perspectives (center, left, and right cameras) along with a CSV log that records metadata, including steering angle, throttle, brake, and speed.

- **Image Folder Link**: Images
- **CSV File Link**: CSV File

Dataset Preparation:

- The video from the Udacity simulator was split into individual frames for model training.
- The dataset was divided into training (80%) and validation (20%) sets.

4 Model and Code Description

The project includes three models: CNN-RNN Model, NVIDIA Model, and PilotNet Model. Each model has its code and drive.py script. Below are the links to each model's code, trained model file, and drive.py:

1. **CNN-RNN Model**:
- Model, Code, and Drive Script: Google Drive Link

2. **NVIDIA Model**:
- Model, Code, and Drive Script: Google Drive Link

3. **PilotNet Model**:
- Model, Code, and Drive Script: Google Drive Link

5 Interface

To evaluate the trained models, the drive.py script is used. This script takes images from the simulator in real time, processes them using the trained model, and predicts the steering angle, which is then applied to control the vehicle.

- **Drive Script Path**: Refer to the Google Drive links in Section 4.

- **Simulator Requirements**: The Udacity Self-Driving Car Simulator must be used in Autonomous Mode to test the models.

- **Udacity Simulator Download**: Simulator .exe

- **Simulator Config Files**: Config Files

6 Evaluation

The models were compared regarding the capability to predict correct steering angles under different driving conditions provided by the Udacity simulator. With the aim to present a clear model comparison, the evaluation was performed using a metric like Mean Squared Error, along with an analysis of loss curves on training and validation.
Every model has its corresponding drive script and dataset used for evaluation:

- CNN-RNN Model: Evaluates temporal dependencies using an RNN layer.
- NVIDIA Model: uses an end-to-end learning based on a Convolutional Neural Network.
- PilotNet Model: Optimized architecture from NVIDIA for real-time steering prediction.

7 Running the Project

Follow these steps to run the trained models:

1. **Set Up the Environment**:
   - Ensure all necessary libraries are installed using the requirements.txt file.
   - Activate the Anaconda environment. Eg. 'car' as per my system.

```
conda activate car
```

```
pip install -r requirements.txt
```

2. **Launch the Udacity Simulator**:
   - Download and install the Udacity Self-Driving Car Simulator from the provided link in section 5.
   - Open the simulator and select **Autonomous Mode**.

3. **Run the Drive Script**:
   - Navigate to the directory containing the drive.py script and the trained model (model.h5).
   - Run the drive script for the respective model:

```
python drive.py
```

4. **Observe the Vehicle's Behaviour:**
   - Set the simulator to **Autonomous Mode** to observe how the model controls the vehicle in real time.

8 Code:

```python
def create_pilotnet():
    model = Sequential()
    # Input layer
    model.add(Input(shape=(66, 200, 3)))
    # Normalization layer
    normalization_layer = Normalization(axis=-1)
    normalization_layer.adapt(tf.random.normal((1, 66, 200, 3)))
    model.add(normalization_layer)
    # Convolutional layers
    model.add(Conv2D(24, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(36, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(48, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(64, (3, 3), activation='elu'))
    model.add(Conv2D(64, (3, 3), activation='elu'))
    # Flatten the output
    model.add(Flatten())
    # Fully connected layers
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    # Output layer for the steering angle
    model.add(Dense(1))
    return model
```

PilotNet

```python
def nvidia_model():
    model = Sequential()
    model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
    model.add(Conv2D(36, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(48, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(64, (5, 5), activation='elu'))
    model.add(Flatten())
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))

    # Use the full loss name 'mean_squared_error'
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model

# Train the model again
model = nvidia_model()
```

Nvidia Model

```python
def cnn_rnn_model(input_shape):
    inputs = Input(shape=input_shape)

    # Convolutional layers for feature extraction
    x = Conv2D(24, (5, 5), strides=(2, 2), padding='valid', activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)  # Added Dropout to reduce overfitting
    x = Conv2D(36, (5, 5), strides=(2, 2), padding='valid', activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)  # Increased Dropout
    x = Conv2D(48, (5, 5), strides=(2, 2), padding='valid', activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)  # Increased Dropout
    x = Conv2D(64, (3, 3), strides=(1, 1), padding='valid', activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.4)(x)  # Increased Dropout
    x = Conv2D(64, (3, 3), strides=(1, 1), padding='valid', activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.4)(x)  # Increased Dropout

    # Flatten the output of the CNN
    x = Flatten()(x)

    # Dense layers for intermediate processing
    x = Dense(100, activation='relu')(x)
    x = Dropout(0.5)(x)  # Increased Dropout
    x = Dense(50, activation='relu')(x)
    x = Dropout(0.5)(x)  # Increased Dropout
    x = Dense(10, activation='relu')(x)

    # Use Reshape layer instead of Lambda to reshape the tensor for LSTM input
    x = Reshape((1, 10))(x)  # Add a time dimension

    # Recurrent layers for temporal dependencies
    x = LSTM(50, return_sequences=False, dropout=0.2, recurrent_dropout=0.2)(x)  # Added dropout to LSTM

    # Final output layer
    outputs = Dense(1, activation='linear')(x)

    model = Model(inputs=inputs, outputs=outputs)
    return model
```

CNN-RNN Model

Figure 1. CNN Architecture of all three models

Figure 1 represents the architecture in PilotNet, Nvidia, and CNN-RNN hybrid models concerning autonomous driving. In the PilotNet model, a pure CNN architecture was designed for the task of end-to-end learning: the convolutional layers extract the spatial features, while the fully connected layers compute the steering angle. This model was trained using ELU for the activation and normalization. The Nvidia's improved model is a further extension of PilotNet, including more convolutional filters and layers, reduced learning rates, and higher capacity for feature extraction; thus, it performed better in various road conditions. Another exploitation of this idea is presented in the hybrid architecture of CNN-RNN, where the combination of CNN layers with an LSTM layer extracts both the spatial features and temporal dependencies. Such architecture involves batch normalization and dropout layers that enhance robustness and prevent overfitting; thus, it is apt for handling dynamic driving with temporal context involved. These models together carry an increasing level of sophistication in the methodology adopted for steering angle prediction.

9 Important Note:

The file paths used in this project are configured based on the directory structure of my system. To ensure the code runs successfully on your machine, please update all file paths in the code to match the directory structure of your system.

10 Conclusion

This configuration manual is supposed to let any person be able to reproduce research and test models developed in predicting the steering angle within an autonomous vehicle. Hyperlinks for all code, models, and data required for reproducing the results are provided within this manual. In addition, if further assistance is needed, use the documentation in the project that is located in associated Google Drive folders.