SteerNet: Improving the accuracy of self-driving cars through deep learning

Chen Hao Zhang
Department of Electrical
and Computer Engineering
University of Toronto
`chenhao.zhang@mail.utoronto.ca`

Lang Sun
Department of of Electrical
and Computer Engineering
University of Toronto
`lang.sun@mail.utoronto.ca`

Lya Chiñas Serrano
Department of Chemical Engineering
and Applied Chemistry
University of Toronto
`lya.chinas@mail.utoronto.ca`

Abstract

This project presents SteeringAngleCNN, a deep learning model designed for lane-keep assist in autonomous driving. The model addresses limitations in rule-based methods to capture complex conditions in driving tasks. The comma2k19 dataset is used for training and testing, to predict the future steering wheel angle. Four architectures were evaluated: a baseline CNN inspired by NVIDIA's PilotNet, an enhanced CNN with temporal steering history, a MobileNetV2-based CNN, and an RNN with ConvLSTM. All models performed similarly, with MobileNetV2-based CNN demonstrated the highest accuracy (MAE 0.57°). These findings suggest that optimizing visual feature extraction using CNN with historical inputs can perform just as well as temporal modeling for short-term steering prediction using RNN.

Keywords: autonomous driving, steering wheel angle, CNN, ConvLSTM, historical input

Assentation of Teamwork

Chen Hao Zhang: Primary code writing, contribution to report, results analysis.
Lang Sun: Primary code writing, contribution to report, results analysis.
Lya Chiñas Serrano: Primary report writing, contribution to code, results analysis.

1 Introduction

Autonomous driving is a revolutionary field, but it cannot rely on rule-based algorithms alone, as they fail to handle complex conditions such as lighting, obstacles, and varying road curvature. Deep learning offers a solution to learn complex patterns directly from data, rather than following explicit instructions. Through hierarchical feature extraction and representation learning, neural networks can capture non-linear relationships between sensory inputs and driving decisions, adapting to the changing context [1].

2 Preliminaries and Problem Formulation

The problem this project addresses is a predictive learning task for autonomous driving. Rather than explicitly programming rules for steering control, a deep learning model will be trained to anticipate and replicate human driving behaior.

Previous deep learning approaches for this task include NVIDIA's PilotNet [2, 3], which demonstrated that convolutional neural networks (CNNs) could map raw camera pixels directly to steering commands without explicitly programmed rules. Moreover, the idea of incorporating temporal information through recurrent neural networks (RNNs) has demonstrated significant improvements, especially those combining Long Short-Term Memory (LSTM) networks with convolutional architectures to form ConvLSTMs [4]. The RNN is expected to capture steering behaviors that require temporal reasoning, such as gradual turns, lane changes, and situations where objects enter or exit the vehicle's path.

The deep learning model for this project will take video frames and sensor readings as input, to predict future steering wheel angles. The model is expected to identify relevant visual cues (such as lanes and their curvature) and anticipate changes in road conditions ahead of time. Model architectures for this task must simultaneously extract temporal and visual information (e.g., ConvLSTM), or whenever the architecture cannot capture this (e.g., CNNs), it's critical to include an input with temporal information, such as past video frames.

3 Solution via Deep Learning

Dataset: The dataset used in this project is derived from the comma2k19 dataset [5], containing 33 hours of driving data on California's Highway 280, covering a 20km stretch between San Jose and San Francisco.

The dataset is organized into 10 chunks of approximately 200 minutes of driving. Every chunk contains a video file paired with sensor data logs collected during the corresponding driving session. The videos are recorded at 20 fps.

The sensor data includes readings on: **Steering wheel angles** in degrees, where positive values are for right turns, negative are for left turns; **vehicle speed** in meters per second; **acceleration** in m/s² on three axes; **gyroscope readings** in rad/s on three axes; **GPS coordinates**; and **CAN data** including wheel speeds and radar measurements.

For the development of SteerNet, the following components were identified:

- **Input data, x_i:** Camera video frames, and the synchronized data of the vehicle sensors, including steering wheel angles, vehicle speed, and gyroscope readings. These sensor readings should provide a representation of the vehicle's dynamic state and position at the current time step $[t]$.
- **Target data, y_i:** Steering wheel angles in the future timestep $[t + 1]$ will be labeled outputs for supervised training. With this, the model is expected to predict a driver's decision in the future, based on conditions at the current time step $[t]$.

4 Implementation

Data preprocessing: The downloading of the dataset occurs in `download_dataset.py`. For the scope of this project, only the first segment is used, corresponding to three hours of driving footage and approximately 240,000 data points. Preprocessing of the datapoints occurs in `data_prep.py`. Here, each video frame is resized to 346×260 pixels; then, each frame is synchronized with its corresponding sensor data based on timestamps to ensure temporal alignment; lastly, the steering angle at the future 200 ms ($t + 200ms$) is extracted to be used as the prediction target.

Models: Four architectures were implemented. Each can be referred to their corresponding `.py` file:

- **v4_CNN:** The architecture for this model is inspired in the one from NVIDIA's PilotNet [3]. It processes video frames through a series of 5 convolutional layers to extract visual features.

In parallel, current sensor data is processed through fully connected layers, which includes the current steering angle at time $[t]$, and current speed. The visual and sensor features are then concatenated and processed through additional fully connected layers to predict the future steering angle.

- **v4_CNN2:** Building upon the basic CNN, the enhanced version incorporates temporal information by including historical steering angles (previous 100ms and 200ms) as additional input features. This modification allows the model to consider not just the current state but also the recent trajectory, improving prediction consistency and accuracy.
- **v4_CNN_MobileNetV2:** This uses the pre-trained MobileNetV2 architecture as its feature extractor backbone. MobileNetV2 was chosen due to its lightweight but efficient CNN designed for mobile vision applications [6]. The model extracts features from input frames through an inverted residual structure (ResNet), and uses average pooling to reduce spatial dimensions. The visual features are then concatenated with sensor data (speed and current steering angle) and processed through fully connected layers to predict the future steering angle.
- **v4_RNN:** This model uses a CNN in combination with RNN in order to capture the temporal dynamics of driving. It processes sequences of video frames and sensor data. A CNN base extract spatial features, concatenated with sensor data, form the input sequence to a single-layer LSTM. Each sequence consists of 10 consecutive frames.

Specific details on model architectures are available in Table 1.

Table 1: Tested model architectures

Model	**v4_CNN**	**v4_CNN2**	**v4_CNN (MobileNetV2)**	**v4_RNN**
Architecture	Custom CNN	Custom CNN	MobileNetV2 (pre-trained)	Custom CNN + LSTM
Descrip	Base steering angle prediction model	Includes steering history inputs	Uses pretrained MobileNetV2 for efficiency	Processes sequence of frames for temporal context
Input Data	• RGB image • Speed • Steer angle (t)	• RGB image • Speed • Steer angle $[t]$ • Prev steer angle $[t-100ms]$ • Prev steer angle $[t-200ms]$	• RGB image • Speed • Steer angle $[t]$	• Sequence of RGB images • Sequence of Speed • Sequence of steer angle $[t...t-n]$
Visual Encoder	5 convolutional layers*	5 convolutional layers*	1 convolutional layer, 19 ResNet layers	5 convolutional layers*; 1 unidirectional layer LSTM with 64 hidden units
Sensor Encoder	Linear(2→8)	Linear(4→8)	None, directly concatenates with visual features in a 128-neuron layer	None, directly connects to hidden layer
Regularization	• Dropout (0.15, 0.2) • BatchNorm • Weight decay: 0.01	• Dropout (0.15, 0.2) • BatchNorm • Weight decay: 0.0001	• Dropout (0.2) • BatchNorm • Weight decay: 0.0001	• Dropout (0.15) • BatchNorm • Weight decay: 0.0001

* Conv(3→24, 5×5, stride=2) → Conv(24→36, 5×5, stride=2) → Conv(36→48, 5×5, stride=2) + MaxPool(2×2) → Conv(48→64, 3×3, stride=1) + MaxPool(2×2) → Conv(64→64, 3×3, stride=1)

Table 2: Comparison of model performance

Model	MSE (deg^2)	MAE (deg)	Within 1°	Within 3°	Within 5°
v4_CNN	1.0587	0.5519	85.3%	98.7%	99.4%
v4_CNN2	1.0513	0.6281	79.1%	98.6%	99.6%
v4_CNN_MobileNetV2	0.9494	0.5714	85.9%	98.9%	99.6%
v4_RNN	1.1398	0.6828	79.9%	97.6%	99.6%

Optimization: All models were trained using the AdamW optimizer with a learning rate of 0.0001. The mean squared error (MSE) was used as a loss function to quantify the error between the model's predicted steering angle (v_i) against the ground truth (y_i), MSE penalizes larger prediction errors more heavily due to its quadratic term, which is relevant in the steering angle prediction task where large deviations could result in unsafe driving behavior:

$$\mathcal{L}(f(v_i \mid \mathbf{w}), y_i) = (f(v_i \mid \mathbf{w}) - y_i)^2$$

Data splitting: The driving footage required preprocessing to handle out-of-distribution data, such as U-turns, which could potentially lead to higher prediction errors. This preprocessing was implemented within the `DrivingDataset` class, where significant steering angles (beyond ±45°) were excluded from the dataset to ensure model training focused on typical driving scenarios.

Since driving data is sequential, a random split would risk including frames from the same driving segment being leaked across sets. Thus, the complete dataset was first concatenated in its original temporal order, then divided chronologically: the first 70% of sequential frames were used for training, the next 15% were used for validation, and the final 15% of frames were used for the test set. This approach ensures that the model is evaluated on completely unseen driving scenarios.

Training: During training, the data is processed in batches of 64 samples, for 10 epochs, and early stopping is implemented to prevent overfitting. Three CNN models were trained on a total of 100 minutes of driving footage, while the RNN was trained on a total of 20 minutes of footage. The RNN model was significantly slower in training speed, and required high VRAM usage.

Evaluation: The evaluation metrics of choice were MSE and Mean Absolute Error (MAE) to assess both the average deviation and the impact of larger errors, respectively. MAE offers an intuitive measure that directly corresponds to the physical steering wheel deviation. Moreover, the percentage of predictions within 1°, 3°, and 5° of the true steering angles is used to assess practical accuracy; where predictions within the 1° threshold represent high-precision steering control. For visual evaluation, we generated time-series plots showing the alignment between predicted and ground truth steering angles over consecutive frames (Figure 2), and correlation plots comparing predicted versus actual steering angles (Figure 3). Additionally, qualitative analysis was performed by visualizing the most challenging predictions (those with the highest error) alongside their corresponding input images to study specific scenarios (Figure 5).

Frame rate performance analysis: An additional experiment was conducted to determine the optimal frame rate to use. This was done to evaluate the trade-off between prediction accuracy and the high computational requirements needed for this task. Models were trained and evaluated across multiple frame rates ranging from 1fps to 20fps. For each frame rate, the model was trained using the same splitting strategy, with the same number of frames per video sequence to ensure fair comparison. Performance was measured using the standard metrics (MSE, MAE), while simultaneously tracking the number of frames processed (Figure 4).

Code availability The implementation is available in the GitHub Repository: https://github.com/gh0stintheshe11/SteerNet.

5 Numerical Experiments

As highlighted in Table 1, the MobileNetV2-based CNN model achieved the best overall performance with the lowest MSE (0.9494) and strong accuracy within the 1°, 3°, and 5° thresholds. However, all

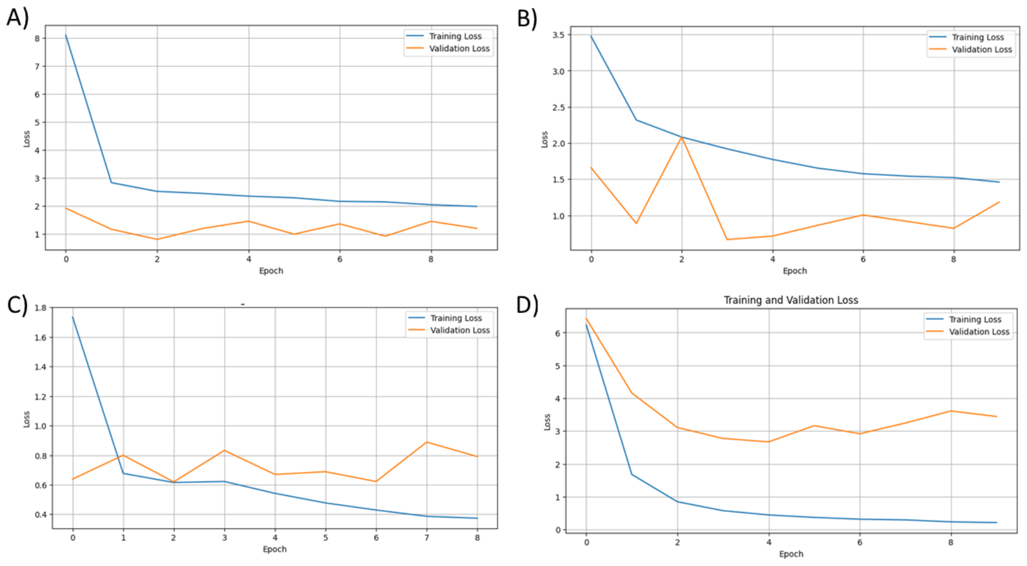

Figure 1: Learning curves for **A)** v4_CNN, **B)** v4_CNN2, **C)** v4_CNN (MobileNetV2), **D)** v4_RNN . Training performed with batch size=64, Adam optimizer (lr=0.0001), MSE loss, for 10 epochs. CNN models trained on 100 minutes of data, RNN trained on 20 minutes of data.

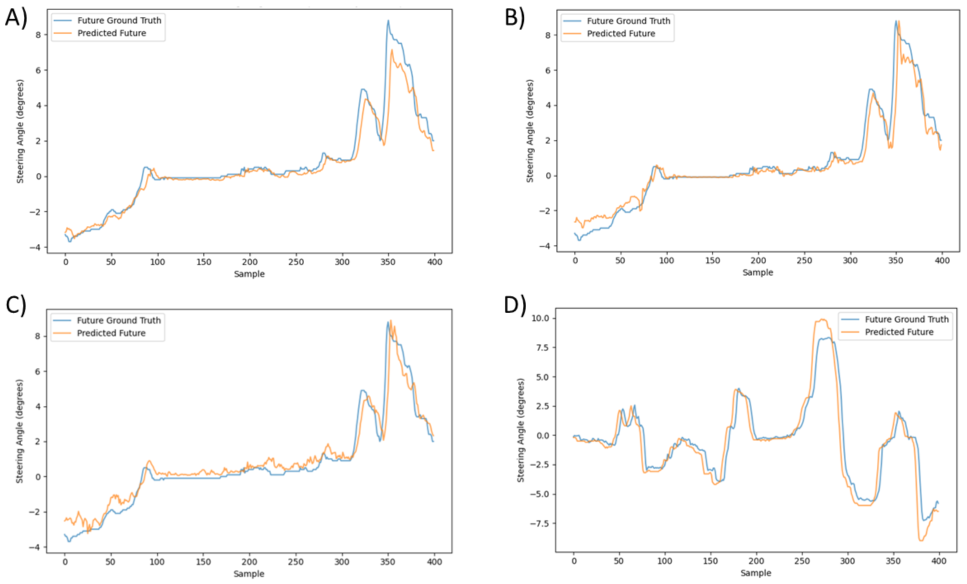

Figure 2: Predictions vs Ground Truth of future steering angles at $t + 200ms$. Test set results for **A)** v4_CNN, **B)** v4_CNN2, **C)** v4_CNN_MobileNetV2, **D) v4_RNN**. Training performed with batch size=64, Adam optimizer (lr=0.0001), MSE loss, for 10 epochs. CNN models trained on 100 minutes of data, RNN trained on 20 minutes of data.

models performed similarly. The learning curves in Figure 1 show that the architecture with the most effective feature extraction capabilities is the MobileNetV2-based model (Panel C).

The results for the frame rate analysis (Figure 4) show that the prediction accuracy does improve with higher frame rates, but will significantly diminish beyond 15 fps. This indicates an optimal operating point that balances accuracy with computational efficiency. However, for the critical requirements of self-driving cars, higher frame rates may still be justified despite the small performance gains.

The superior performance of CNN-based architectures emphasizes the importance of visual feature extraction in this task. This can be attributed to the fact that driving decisions strongly depend on the surrounding visual cues. Moreover, v4_CNN_MobileNetV2 shows the efficiency of incorporating residual connections, linear bottlenecks, and a depth-wise convolution stage. This combination allows the gradient flow to be better preserved, and temporal information to be effectively captured without explicit recurrent structures.

The convolutional LSTM used in v4_RNN was not able to perform better than CNN models. This could be attributed to the complex architecture or the relatively short-term prediction required in this task (200ms ahead). Additionally, sequence length of 10 frames might be insufficient to capture temporal relationship.

Future improvement of this framework includes the incorporation of imitation (IL) and reinforcement learning (RL) techniques [7]. IL aims to train a log-likelihood policy to match an expert's actions given the observed states, aligning well to mimic human driving behavior. RL incorporates a reward function to penalize large deviations, which is useful to handle rare driving scenarios (e.g. collisions). The combination of these two could enhance the model's performance for the task of self-driving.

6 Conclusion

This project presented a predictive learning framework for the lane-keep assist function, that by integrating both spatial and temporal data, successfully generalizes in diverse driving conditions. The findings reveal that optimizing visual feature extraction in CNN architectures might be more valuable than complex temporal modeling, especially when predicting relatively short time horizons. For this type of model, a temporal element is necessary in the input to provide context for motion prediction. Moving forward, the integration of imitation learning and reinforcement learning techniques presents a promising direction to achieve more reliable models that can capture human-like driving patterns.

References

[1] Anjali Gupta, Alagan Anpalagan, Ling Guan, and Ahmed S. Khwaja. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array*, 10:100057, 2021. doi: 10.1016/j.array.2021.100057. URL `https://doi.org/10.1016/j.array.2021.100057`.

[2] Mariusz Bojarski, Chenyi Chen, Jeremy Daw, Alperen Değirmenci, Julien Deri, Bernhard Firner, Beat Flepp, Sachin Gogri, Jesse Hong, Lawrence Jackel, Zhenhua Jia, Biao Lee, Bo Liu, Fei Liu, Urs Muller, Samuel Payne, N. K. Narasimha Prasad, Anton Provodin, John Roach, Timur Rvachov, Nikhil Tadimeti, Jesper Van Engelen, Haoran Wen, Eric Yang, and Zongyi Yang. The nvidia pilotnet experiments. *arXiv preprint arXiv:2010.08776*, 2020.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016. URL `https://arxiv.org/abs/1604.07316`.

[4] Marcel Schreiber, Stefan Hoermann, and Klaus Dietmayer. Long-term occupancy grid prediction using recurrent neural networks, 2019. URL `https://arxiv.org/abs/1809.03782`.

[5] Comma.ai. comma2k19 dataset. `https://github.com/commaai/comma2k19`, n.d. Accessed: 2024-03.

[6] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019. URL `https://arxiv.org/abs/1801.04381`.

[7] Yiren Lu, Justin Fu, George Tucker, Xinlei Pan, Eli Bronstein, Rebecca Roelofs, Benjamin Sapp, Brandyn White, Aleksandra Faust, Shimon Whiteson, Dragomir Anguelov, and Sergey Levine. Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios, 2023. URL `https://arxiv.org/abs/2212.11419`.

Appendix

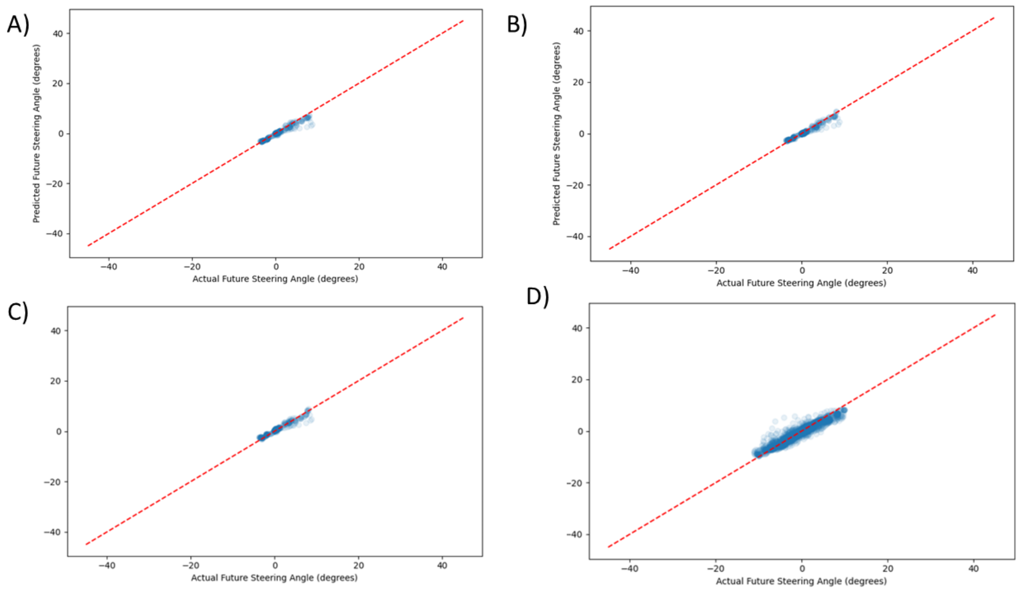

Figure 3: Correlation of predictions vs ground truth of future Steering angles at $t + 200ms$. Test set results for **A)** v4_CNN, **B)** v4_CNN2, **C)** v4_CNN (MobileNetV2), **D)** v4_RNN. Training performed with batch size=64, Adam optimizer (lr=0.0001), MSE loss, for 10 epochs, using 100 minutes of data.

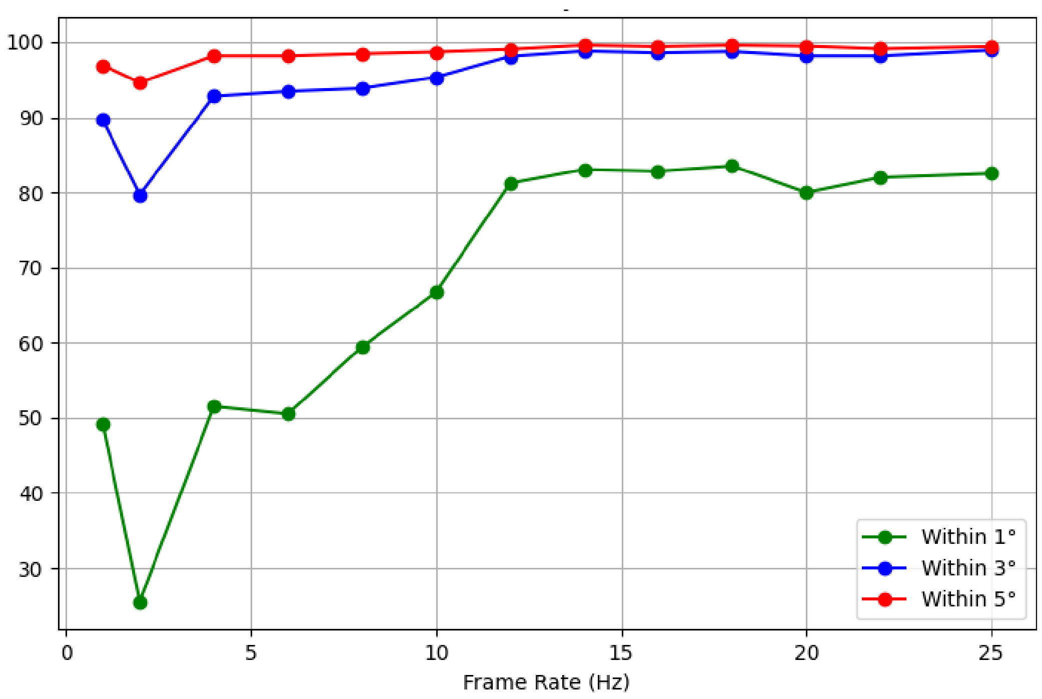

Figure 4: Frame rate importance analysis for model v4_CNN. Performance with training using batch size=64, Adam optimizer (lr=0.0001), MSE loss, for 10 epochs, using 20 minutes of data.

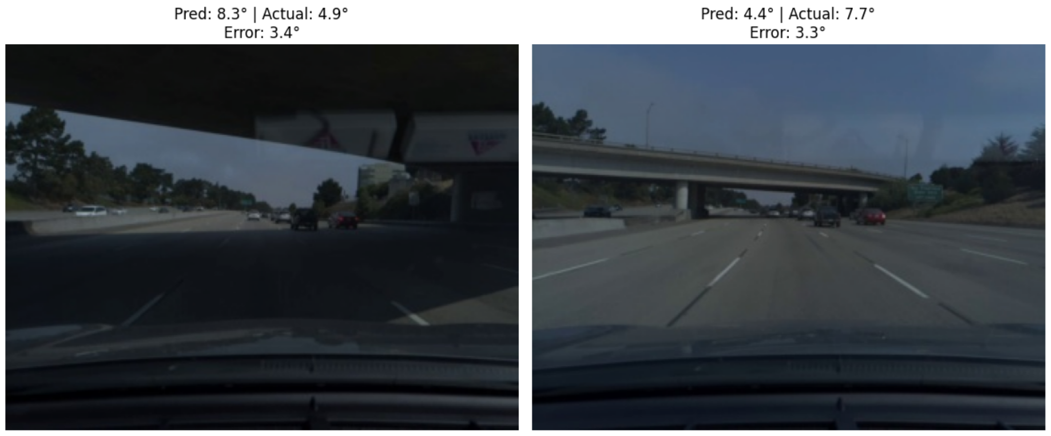

Figure 5: Visual examples of frames with the highest prediction error in v4_CNN. The discrepancies can be attributed to unusual driving behaviors, such as changing lanes.