End-to-End Autonomous Driving Using Deep Learning and Cameras

1st Fengxiang Xue
dept. MPSYS
Chalmers University of Technology
Gothenburg, Sweden
xuefe@student.chalmers.se

2nd Xinying Wang
dept. MPSYS
Chalmers University of Technology
Gothenburg, Sweden
xinying@chalmers.se

***Abstract*—This study explores an end-to-end approach to predict steering angles in autonomous driving using camera images. We evaluate two architectures: ResNet with a fully connected (FC) layer and ResNet with LSTM layers, assessing spatial and temporal pattern capturing in steering predictions. Results show that CNN+LSTM excels in handling temporal sequences, while CNN+FC achieves faster convergence and effective spatial feature extraction, highlighting trade-offs in complexity and data dependency. These findings offer insights into model architecture choices for autonomous driving.**

***Index Terms*—Autonomous driving, steering prediction, ResNet, FC, LSTM, end-to-end learning**

I. Introduction

Recent advancements in deep learning have driven rapid evolution in autonomous driving. Unlike traditional rule-based systems, end-to-end learning directly maps raw inputs, like camera images, to control commands, simplifying system complexity.

This project focuses on predicting steering angles from images captured by a windshield-mounted camera, evaluating two architectures: (1) CNN with fully connected layers for spatial information and (2) CNN with LSTM layers for both spatial and temporal dependencies. This comparison examines the effect of temporal modeling, especially in data-limited settings.

Our contributions:

- We introduce preprocessing to filter redundant data, reducing noise and enhancing feature extraction with ResNet.
- We compare two end-to-end models: CNN+FC for efficient spatial extraction and CNN+LSTM for capturing temporal dependencies to improve steering accuracy.
- We analyze each model's impact on training efficiency, generalization, and robustness in limited data scenarios, offering guidance for model selection in autonomous driving.

A. Related Work

The field of autonomous driving has advanced significantly with deep learning, especially in computer vision and time-series analysis.

Goodfellow et al. (2017) [1] established the foundation for applying deep learning to complex control tasks.

Bojarski et al. (2016) [2] pioneered end-to-end learning for steering prediction, reducing reliance on hand-crafted features, though generalization issues remained. To address these, we incorporate advanced preprocessing and a hybrid model design.

He et al. (2016) [3] showed that deep neural networks are difficult to train. They proposed a residual learning framework to simplify the training of networks that are deeper than those previously used.

Donahue et al. (2015) [4]proposed the long-time recurrent convolutional network (LRCN), which combines the strengths of CNNs and LSTMs to excel in visual recognition and temporal data modeling.The spatio-temporal modeling capabilities of LRCNs make them ideal for generating driving commands from video inputs in automated driving tasks, providing theoretical and Technical Support.

Thoses research provides the theoretical foundation and technical advancements that inform our proposed hybrid ResNet-LSTM model, aiming to create an effective and efficient end-to-end solution for autonomous steering prediction in dynamic environments.

II. DATA

A. Data Collection

The dataset [5] used in this project was provided by the Udacity Challenge. We utilized two datasets based on ROSbag: one with steering angle data and one without. Training was conducted on the first dataset, while testing was performed on the second dataset.

B. Data Preprocessing

(labels)

- **Data Reading and Filtering**: First, read the CSV file containing the driving data using `pandas`. Filter out unnecessary camera data, keeping only the center camera data, and save this filtered data into a new CSV file.
- **Dataset Splitting**: Use a custom function `split_dataset` to divide the dataset into training, validation, and test sets according to a specified ratio.
- **Data Pair Construction**: Using the `read_input` function, read image paths and corresponding labels from the

CSV file and store them in a data pair format to facilitate further processing.

- **Image Preprocessing**: Resize the images, convert them to tensors, and apply normalization to make them suitable for model input requirements.
- **Custom Dataset and Data Loader**: Pack the images and labels into a PyTorch `Dataset` object, loading 5 frames per instance. After preprocessing, these are formed as inputs to the model. Use `DataLoader` to load data in batches, creating loaders for training, validation, and test sets to ensure the data is in the correct format and batch size for training and evaluation.

III. Model

A. Residual Network (ResNet50)

Deep networks are prone to vanishing gradients, particularly as depth increases. ResNet50 mitigates this with residual blocks that introduce skip connections, allowing gradients to bypass layers and maintain strength across the network. These skip connections improve stability and efficiency, making it possible to train deeper networks with reduced gradient loss, as shown in Figure 1. The propagation of gradients through these connections can be described by:

$$\frac{\partial \varepsilon}{\partial \alpha^{l1}} = \frac{\partial \varepsilon}{\partial \alpha^{l2}} \frac{\partial \alpha^{l2}}{\partial \alpha^{l2-1}} \cdots \frac{\partial \alpha^{l1+1}}{\partial \alpha^{1}} \tag{1}$$

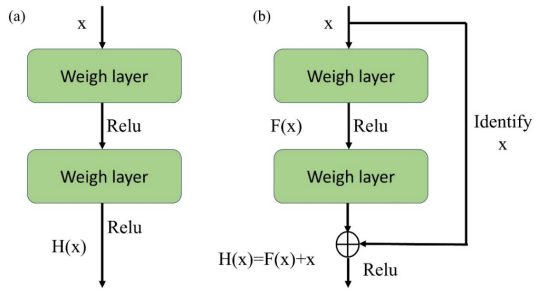

Fig. 1. (a) Standard network structure; (b) Residual network structure

B. Fully Connected (FC) Layer

In the CNN+FC model, ResNet's spatial features are flattened and passed through one or more fully connected (FC) layers, which map these features directly to steering angle outputs. This structure is straightforward, making it computationally efficient and faster to converge. While it effectively captures spatial features, the absence of temporal modeling limits its performance in scenarios where sequential data patterns, like road curvature or vehicle motion, are essential.

C. Long Short-Term Memory (LSTM) Networks

LSTM networks, a form of RNN, address the vanishing gradient issue via gating mechanisms: Forget Gate, Input Gate, and Output Gate. These gates selectively retain or discard information, allowing LSTMs to capture long-term dependencies across time steps. The core cell state accumulates relevant information over sequences, maintaining dependencies that are crucial for temporal data tasks:

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right), \quad C_t = f_t * C_{t-1} + i_t * C'_t \tag{2}$$

LSTM is ideal for sequential tasks in autonomous driving as it models patterns in steering adjustments across time, complementing ResNet's spatial features.

D. Loss Function

The Mean Squared Error (MSE) is chosen to measure the difference between predicted and actual steering angles:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y'_i)^2 \tag{3}$$

MSE is smooth, differentiable, and sensitive to large errors, making it effective for minimizing prediction deviations in critical scenarios. Its characteristics include:

- **Penalty for Large Errors**: Emphasizes significant deviations, suitable for safety-critical predictions.
- **Symmetry and Positive Definiteness**: Provides balanced predictions and is non-negative.
- **Sensitivity to Outliers**: Increases for large deviations, which may be addressed with alternative loss functions like MAE if data noise is a concern.

E. Optimizer

The Adam optimizer is used, leveraging adaptive learning rates and momentum to improve convergence by maintaining exponentially weighted averages of the gradients:

- **First Moment (Momentum)**: Tracks the exponential moving average of the gradients, helping the optimizer converge more efficiently by smoothing out updates.
- **Second Moment (Adaptive Rate)**: Tracks the average of squared gradients, allowing for adaptive adjustments to the learning rate.

The key steps in Adam's algorithm are:

1) **Initialization**: Set learning rate η, momentum coefficients β_1 and β_2, and initial parameters θ.
2) **Gradient Calculation**: Compute gradients g_t of the loss with respect to parameters.
3) **Moment Updates and Bias Correction**: Update first moment m_t and second moment v_t, apply bias correction to obtain $\hat{m}_t$ and $\hat{v}_t$.
4) **Parameter Update**: Use corrected moments to adjust parameters.

Adam's adaptive nature and efficient convergence make it well-suited for optimizing both CNN+FC and CNN+LSTM models in autonomous driving tasks.

IV. Train

A. Training Setup

Prior to the start of training, we calculated the maximum and minimum values of the steering angle in the training set to determine the range of the target variable. The range of steering angles was:

- Maximum value: 1.8261
- Minimum value: -1.2261

This helped us to compare the loss and MSE to the overall range of steering angles when calculating the loss and MSE to make it easier to understand the relative magnitude of the model error.

B. CNN(ResNet) + FC

ResNet50 as a Feature Extractor

Using `torchvision.models`, the pre-trained ResNet50 model is loaded, with the final two layers (fully connected layer and global average pooling layer) removed. ResNet50 provides an efficient convolutional neural network architecture for extracting features from input images.

Adaptive Average Pooling Layer

`nn.AdaptiveAvgPool2d((1, 1))` is used to convert the convolutional feature map to a fixed size (1×1), which makes it easier to connect to the fully connected layer.

Fully Connected Layer (FC)

- First Layer: A fully connected layer maps the 2048-dimensional features extracted by ResNet to a user-specified hidden layer dimension (`hidden_size`).
- Second Layer: The output layer, which maps the output of the hidden layer to a scalar (i.e., the predicted steering angle).

1) Training Loop: The process is described as follows:

```
for epoch in range(num_epochs):
    model.train()  # Set to training mode
    for inputs, labels in train_loader:
        # 1. Forward pass: inputs -> model -> outputs
        outputs = model(inputs)
        # 2. Compute loss: criterion(outputs, labels)
        loss = criterion(outputs, labels)
        # 3. Backward pass: loss.backward()
        loss.backward()
        # 4. Update parameters: optimizer.step()
        optimizer.step()

    # Evaluate performance on validation set (model.eval())
    model.eval()
```

2) Loss Tracking: The `MSELoss` function is chosen as the loss function to measure the mean squared error between the predicted steering angles and the true values.

The Adam optimizer is used for parameter updates, with a learning rate set to 0.0001.

```
Epoch 1/10, Train Loss: 0.0433, MSE: 0.0433
Validation Loss: 0.1180, MSE: 0.1180
Epoch 2/10, Train Loss: 0.0179, MSE: 0.0179
Validation Loss: 0.1037, MSE: 0.1037
Epoch 3/10, Train Loss: 0.0107, MSE: 0.0107
Validation Loss: 0.0616, MSE: 0.0616
Epoch 4/10, Train Loss: 0.0074, MSE: 0.0074
Validation Loss: 0.0614, MSE: 0.0614
Epoch 5/10, Train Loss: 0.0058, MSE: 0.0058
Validation Loss: 0.0619, MSE: 0.0619
Epoch 6/10, Train Loss: 0.0063, MSE: 0.0063
Validation Loss: 0.0805, MSE: 0.0805
Global Train MSE: 0.0119
Global Val MSE: 0.0718
Train MSE as percentage of angle range: 0.4099%
Val MSE as percentage of angle range: 2.4631%
```

Loss and MSE: In the first few rounds of training, the loss in the training and validation sets drops significantly, indicating that the model learns well early on. However, after round 4, the loss in the training set continues to decrease, while the loss in the validation set stabilizes and even increases slightly in round 6, which may indicate that the model is starting to overfit.

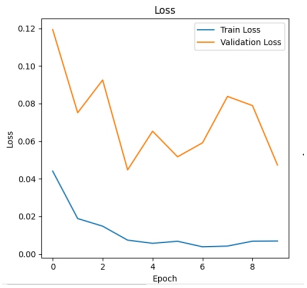

Fig. 2. ResNet_FC_Loss

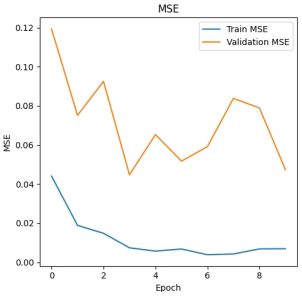

Fig. 3. ResNet+FC_MSE

After the complete training process, the model achieved a train MSE that is within 0.41% of the steering angle range, indicating a close fit to the training data. However, the validation MSE stands at 2.46% of the angle range, which is notably higher than the training error. This difference suggests that while the model captures patterns well in the training set, it may still have room for improvement in generalizing to unseen data

C. CNN(ResNet) + LSTM

The training loop involves several crucial steps: loading data, forward pass, computing loss, backpropagation, and parameter optimization. These steps are repeated for each epoch, refining the model's parameters iteratively. The process is described as follows

`CNN_LSTM_Model` is used for this task, which combines convolutional neural network (CNN) and long short-term memory (LSTM) layers: - The CNN component extracts meaningful features from each frame of the input sequence. - The LSTM component captures temporal dependencies across the sequence, making it well-suited for tasks involving sequential image data, like predicting steering angles in an autonomous driving scenario.

1) Training Loop: The process is described as follows:

```
for epoch in range(num_epochs):
    model.train()  # Set model to training mode
    for inputs, labels in train_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        # 1. Forward pass: pass inputs through the model
        outputs = model(inputs)
        # 2. Compute loss
        loss = criterion(outputs, labels)

        # 3. Backward pass: compute gradients
        optimizer.zero_grad()
        loss.backward()

        # 4. Update parameters using optimizer
        optimizer.step()

    # Evaluate on validation set at the end of each epoch
    model.eval()
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            val_loss = criterion(outputs, labels)
```

2) Loss Tracking: The `MSELoss` function is chosen as the loss function to measure the mean squared error between the predicted steering angles and the true values.

The Adam optimizer is used for parameter updates, with a learning rate set to 0.0001. **Steering Angle Range:** Max = 1.8261, Min = -1.2261

```
Epoch 1/10, Train Loss: 0.0745, MSE: 0.0745,
Validation Loss: 0.2193, MSE: 0.2193,
Epoch 2/10, Train Loss: 0.0683, MSE: 0.0683,
Validation Loss: 0.2157, MSE: 0.2157,
Epoch 3/10, Train Loss: 0.0660, MSE: 0.0660,
Validation Loss: 0.2196, MSE: 0.2196,
Epoch 4/10, Train Loss: 0.0647, MSE: 0.0647,
Validation Loss: 0.2171, MSE: 0.2171,
Epoch 5/10, Train Loss: 0.0623, MSE: 0.0623,
Validation Loss: 0.2190, MSE: 0.2190,
Epoch 6/10, Train Loss: 0.0647, MSE: 0.0647,
Validation Loss: 0.2148, MSE: 0.2148,
Epoch 7/10, Train Loss: 0.0630, MSE: 0.0630,
Validation Loss: 0.2118, MSE: 0.2118,
Epoch 8/10, Train Loss: 0.0628, MSE: 0.0628,
Validation Loss: 0.2174, MSE: 0.2174,
Epoch 9/10, Train Loss: 0.0622, MSE: 0.0622,
Validation Loss: 0.2174, MSE: 0.2174,
Epoch 10/10, Train Loss: 0.0632, MSE: 0.0632,
Validation Loss: 0.2147, MSE: 0.2147,
Global Train MSE: 0.0700
Global Val MSE: 0.2351
Train MSE as percentage of angle range: 2.2324%
Val MSE as percentage of angle range: 7.5012%
```

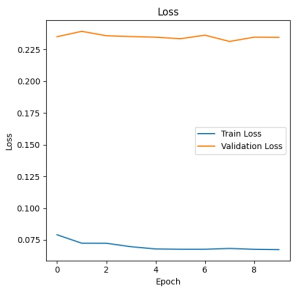

Fig. 4. ResNet+LSTM_Loss

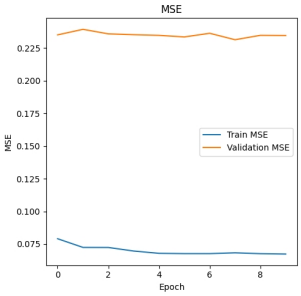

Fig. 5. ResNet+LSTM_MSE

Loss and MSE: The training loss gradually decreases, but the validation loss stays around 0.21 overall with small fluctuations, indicating that the model has limited generalization performance.

After training on small validation dataset, the MSE is about 7.5% of the steering angel range, which is large for the driving scenario. So the prediction on ResNet+ LSTM needs more time to be trained on full dataset, if we have more time and resource the output of this model will be more solid. On the other hand, it shows the ResNet+ LSTM turns out not perform as well as the ResNet+FC layer model, it might

V. Results and Discussion

The model's performance is evaluated using MSE, loss, and accuracy, as shown below:

Model Analysis:

- **CNN-FC Model:** Training loss and MSE decrease steadily, showing effective learning, but validation loss plateaus in later stages, indicating potential overfitting.
- **CNN-LSTM Model:** Training loss decreases, but validation loss fluctuates without consistent reduction, suggesting overfitting likely due to model complexity and limited data.

Model	Dataset	Loss	MSE	MSE in angel range
CNN-FC	Train	0.0058	0.0058	0.4099
	Validation	0.0619	0.0619	2.4631
CNN-LSTM	Train	0.0632	0.0632	2.2324
	Validation	0.2147	0.2147	7.5012

TABLE I
PERFORMANCE OF CNN-FC AND CNN-LSTM MODELS ON TRAINING AND VALIDATION DATASETS

Overfitting Causes and Optimization:

- **Data Limitation:** Limited autonomous driving data makes it challenging for complex models, like LSTM, to generalize effectively.
- **Model Complexity:** High-dimensional LSTM layers may overfit on smaller datasets.
- **Optimization Recommendations:**
  - **Simplify LSTM Layers:** Reduce LSTM layers or use GRU to decrease complexity and mitigate overfitting.
  - **Learning Rate Tuning:** Use a learning rate scheduler (e.g., `ReduceLROnPlateau`) to adjust learning rates as validation loss stabilizes.
  - **Data Augmentation and Regularization:** Apply data augmentation techniques, such as random cropping and brightness adjustment, to increase data diversity; add dropout layers (0.2–0.5) between CNN and LSTM layers to reduce neuron dependency.
  - **Early Stopping:** Stop training if validation loss does not improve after several epochs to avoid unnecessary computation.

VI. Conclusions

From the above analysis, we conclude that while the CNN-LSTM model demonstrates a notable advantage in capturing temporal dependencies, its generalization performance is constrained by data volume and model complexity, leading to consistently higher validation errors in data-limited scenarios. The CNN-FC model, although less capable in handling temporal dependencies, is simpler in structure and converges more rapidly in early training, showing stronger generalization on smaller datasets.

Applicability Analysis: The CNN-FC architecture, with its straightforward structure and lower computational demand, is particularly suitable for deployment in resource-constrained embedded systems where real-time processing and power efficiency are critical. Its faster convergence and reduced tendency to overfit on limited data make it a practical choice for applications with primarily spatial feature requirements and low sequential dependency.

In contrast, the CNN-LSTM model is well-suited to tasks involving complex spatial-temporal dependencies, such as scenarios with significant road curvature or variable driving speeds, where capturing sequential patterns in the data is essential. However, this model requires larger datasets and greater computational resources, making it more appropriate for environments where data availability and processing power are sufficient to support its higher complexity.

Future Optimization Directions: To enhance model performance, we recommend exploring data augmentation, regularization, and hyperparameter optimization techniques. Additionally, using simpler temporal models, such as GRU, can help balance model complexity with computational efficiency, especially for applications with moderate sequential dependency. Implementing transfer learning or ensemble learning strategies may further improve model generalization and robustness, particularly in low-data contexts.

REFERENCES

[1] J. Heaton, "Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1-2, 2017.

[2] M. Bojarski, D. W. del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *ArXiv*, vol. abs/1604.07316, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:15780954

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[4] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, "Long-term recurrent convolutional networks for visual recognition and description," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 2625–2634.

[5] udacity, "self-driving-car," https://github.com/udacity/self-driving-car/tree/master/datasets, 2018.