CNN Implementation for Driver Behavior Classification

Nguyen Tung Lam[1]
[1] ICTLab, University of Science and Technology of Hanoi, *

Abstract

This report presents a comprehensive analysis of a custom-built Convolutional Neural Network (CNN) implemented from scratch for driver behavior classification. The system employs pure Python programming with minimal external dependencies, demonstrating fundamental deep learning principles without relying on high-level frameworks such as TensorFlow or PyTorch. The neural network architecture classifies driver behaviors into five distinct categories: other activities, safe driving, talking on phone, texting on phone, and turning maneuvers. The implementation features a sequential CNN architecture comprising a single convolutional layer with 8 filters of 3×3 kernel size, ReLU activation function, 2×2 max pooling operation, flattening layer, and a fully connected dense layer for final classification. The model processes 32×32 grayscale images with a dataset containing up to 50 samples per behavioral category. Training is conducted using stochastic gradient descent with a learning rate of 0.001 over 10 epochs, employing cross-entropy loss function for optimization. Key findings indicate that while the implementation successfully demonstrates core CNN concepts and achieves functional classification capabilities, several architectural limitations exist, including incomplete backpropagation (limited to dense layer only), single-channel image processing, and scalability constraints inherent to list-based tensor operations. The research contributes to educational understanding of neural network fundamentals and provides a foundation for more sophisticated driver monitoring systems in automotive safety applications. The study concludes that the minimalistic approach effectively illustrates essential deep learning concepts while highlighting areas for enhancement, including complete gradient computation implementation, data augmentation techniques, and architectural improvements for production-level deployment.

*Vietnam Academy of Science and Technology, 18 Hoang Quoc Viet, Cau Giay, Hanoi, Vietnam

1 Introduction

1.1 Background and Motivation

Driver distraction represents one of the most significant contributing factors to road traffic accidents globally, with the World Health Organization reporting that human error accounts for approximately 94Modern automotive safety systems have evolved from passive safety measures to active intervention technologies. Advanced Driver Assistance Systems (ADAS) now incorporate various sensors and machine learning algorithms to enhance vehicle safety. Computer vision-based driver behavior classification represents a critical component of these systems, offering the potential to identify risky behaviors before they result in accidents.

1.2 Problem Statement and Objectives

The primary challenge addressed in this research is the development of an interpretable, educational CNN implementation for driver behavior classification that demonstrates fundamental deep learning concepts without the abstraction layers of modern frameworks. Traditional implementations using TensorFlow, PyTorch, or similar libraries often obscure the underlying mathematical operations and algorithmic processes, limiting educational value for students and researchers seeking to understand neural network mechanics. This project aims to bridge the gap between theoretical understanding and practical implementation by creating a transparent, modifiable CNN architecture that processes real-world driver behavior image data. The implementation serves dual purposes: providing a functional classification system while maintaining complete visibility into all computational processes.

2 Dataset and Data Processing

2.1 Dataset Structure

The implementation processes image data from five different categories of driver behavior.

cc

Category	Label	Description	Max Images per Category
Other Activities	0	Non-driving related activities	50
Safe Driving	1	Normal, attentive driving	50
Talking on Phone	2	Driver engaged in phone conversation	50
Texting on Phone	3	Driver texting while driving	50
Turning	4	Driver performing turning maneuvers	50

Table 1: Driving behavior categories and label mapping

2.2 Data Preprocessing Pipeline

The data loading mechanism implements several pre-processing steps: Image Normalization: The pixel values are normalized to the range [0,1] by dividing by 255.0, ensuring

consistent input scaling and improved convergence during training. Grayscale Conversion: All images are converted to grayscale using PIL's convert(L') method, reducing computational complexity while maintaining essential spatial features for behavior classification. Image resizing: The images are standardized to 32x32 pixels, creating uniform input dimensions while balancing computational efficiency with feature preservation. Data Structure: The images are formatted as [1] [H] [W] tensors, representing single-channel (grayscale) height-width matrices suitable for convolution operations. The pre-processing pipeline ensures data consistency and optimal format for the custom CNN architecture while maintaining the essential visual information required for accurate classification.

3 Model Architecture Analysis

3.1 Network Design Philosophy

The implemented CNN follows a classic architecture pattern with convolution, activation, pooling, and fully connected layers. The design prioritizes simplicity and interpretability while maintaining sufficient complexity for the classification task.

3.2 Layer-by-Layer Analysis

Convolutional Layer (Conv2D)

- Input Channels: 1 (grayscale images)
- Kernel Size: 3x3 filters
- Number of Filters: 8
- Weight Initialization: Random uniform distribution [-0.1, 0.1]
- Bias Initialization: Random uniform distribution [-0.05, 0.05]

The convolutional layer serves as the feature extraction mechanism, applying 8 different 3x3 kernels to detect various spatial patterns in the input images. The relatively small number of filters keeps the model lightweight while providing sufficient feature diversity for the classification task. Activation Function (ReLU) The Rectified Linear Unit activation function introduces nonlinearity while maintaining computational

simplicity. The implementation handles both single vectors and multi-dimensional arrays through recursive processing. Pooling Layer (MaxPool2D)

- Pool Size: 2x2
- Operation: Maximum pooling
- Purpose: Spatial dimension reduction and feature consolidation

The max pooling operation reduces spatial dimensions by factor of 2, creating translation invariance and reducing computational requirements for subsequent layers. Flattening Layer Converts the 3D feature maps from convolutional operations into 1D vectors suitable for dense layer processing. The implementation uses nested list comprehension for efficient tensor reshaping. Dense Layer

- **Input Features:** Calculated based on spatial dimensions after convolution and pooling.
- **Output Features:** 5 (number of classes).
- **Weight Initialization:** Random uniform distribution $[-0.1,\ 0.1]$.
- **Bias Initialization:** Random uniform distribution $[-0.05,\ 0.05]$.

3.3 Mathematical Foundation

The forward pass computation follows standard CNN mathematics:

For convolution operation:

$$\text{Output}[f][i][j] = \sum_{c,ki,kj} \text{Input}[c][i{+}ki][j{+}kj] \cdot \text{Kernel}[f][c][ki][$$

For max pooling:

$$\text{Output}[c][i][j] = \max(\text{Input}[c][ix : ix{+}P,\ jx : jx{+}P])$$

4 Training Methodology

4.1 Training Configuration

cc

Parameter	Value	Justification
Epochs	10	Sufficient for convergence on small dataset
Learning Rate	0.001	Conservative rate preventing overshooting
Batch Size	1	Single sample processing (online learning)
Train/Test Split	80% and 20%	Standard split for small datasets
Optimization	Stochastic Gradient Descent	Simple, interpretable optimization

4.2 Loss Function and Optimization

Cross-Entropy Loss: The implementation uses categorical cross-entropy loss, appropriate for multi-class classification problems. The loss function includes epsilon smoothing (1e-10) to prevent log(0) numerical instability.

Gradient Computation: The backward pass implements analytical gradients for the dense layer:

$$\frac{\partial \mathcal{L}}{\partial W[i][j]} = \text{gradient\_output}[i] \cdot \text{input}[j]$$

$$\frac{\partial \mathcal{L}}{\partial b[i]} = \text{gradient\_output}[i]$$

Weight Updates: Standard gradient descent update rule

$$W[i][j] = W[i][j] - \text{learning\_rate} \cdot \text{gradient}[i][j]$$

4.3 Training Process Analysis

The training loop processes each sample individually, computing forward pass, loss, and backward pass. This online learning approach provides several advantages:

- Immediate weight updates after each sample
- Lower memory requirements
- Suitable for small datasets
- Simple implementation without batch processing complexity

5 Performance Analysis and Results

5.1 Training Dynamics

The training process monitors loss reduction across epochs, providing insights into model convergence. The implementation tracks average loss per epoch, enabling assessment of learning progress and potential overfitting.

5.2 Evaluation Methodology

Test Set Evaluation: The model performance is evaluated on a held-out test set representing 20% of the total data.

Visual Inspection: The final code block generates visual predictions, displaying test images alongside predicted and true labels for qualitative assessment.

Prediction Mechanism: Classification decisions are made by selecting the class with maximum probability from the softmax output.

5.3 Model Limitations and Considerations

Limited Backpropagation: The current implementation only includes backward pass for the dense layer, limiting learning capacity of convolutional features.

Single Channel Processing: Grayscale conversion discards color information that might be relevant for driver behavior classification.

Fixed Architecture: The network architecture is not easily configurable, requiring code modifications for architectural changes.

Scalability Constraints: List-based tensor operations limit performance on larger datasets or higher-resolution images.

6 Conclusion

This CNN implementation demonstrates fundamental deep learning concepts through a practical driver behavior classification application. While the model has architectural limitations compared to modern frameworks, it successfully illustrates core neural network principles including feature extraction, non-linear activation, spatial pooling, and gradient-based optimization.

The implementation achieves its educational objectives by providing transparent, interpretable code that can be studied and modified for learning purposes. For production applications, the architecture would benefit from complete backpropagation implementation, enhanced data preprocessing, and performance optimizations.

The project establishes a solid foundation for understanding CNN mechanics and provides a starting point for more sophisticated driver monitoring systems. The modular design facilitates future enhancements while maintaining code clarity and educational value.

The combination of practical application domain (driver safety) with fundamental machine learning concepts makes this implementation valuable for both educational and research purposes, contributing to the broader understanding of computer vision applications in automotive safety systems.

Link dataset: https://www.kaggle.com/datasets/robinreni/revitsone-5class