**“Aggressive Driving Decoded: A Deep Learning Showdown with LSTM and RNN approaches”**

Adhoksh Milind Sonawane

“Information Systems (MSIS)”

“MIS 790 Culminating Experience”

“Prof. Carlos D. Paternina-Arboleda”

“March 15, 2025”

**“Comprehensive Overview: Importance and Previous Studies”**

Aggressive driving is described as a series of moving traffic violations that endanger other motorists or property (National Highway Traffic Safety Administration [NHTSA],2023). These actions include tailgating, irregular lane changes, and failing to yield. Understanding these habits is critical for increasing road safety and lowering accident risk. Verschuur and Hurts (2007) extended on prior models of accident causation by including psychological and physical antecedents like stress, exhaustion, and health issues, all of which have a substantial influence on dangerous driving behavior. Their research established a safety-related driver behavior model, which investigated elements such as strategic driving decisions, attitudes toward infractions, and self-reported accident participation. These psychological and behavioral characteristics are critical for predicting driving mistakes and infractions, implying that treatments might aid in improving road safety.

Verschuur and Hurts (2007) combined established frameworks, such as Reason's accident causation model and the Driver Behavior Questionnaire (DBQ), with the Theory of Planned Behavior (TPB) to investigate how attitudes, norms, and perceived behavioral control influence violations like speeding and drunk driving. Their research revealed that psychological antecedents and risky attitudes account for up to 9% of the variation in driving infractions and mistakes. This was a building study that has influenced further research in this area.

The National Highway Traffic Safety Administration (NHTSA) defines aggressive driving as tailgating, irregular lane-changing, speeding, and failing to yield (NHTSA, 2023). Tailgating is defined as following another vehicle excessively closely, which reduces response time and increases the danger of collision. Erratic lane-changing refers to frequent, sudden switches between lanes without signaling, which interrupts traffic flow and endangers other vehicles. Speeding is defined as driving beyond the posted limit or too fast for the road conditions, and it is a primary cause of accidents. Failure to yield happens when automobiles do not grant the right-of-way, which frequently results in unsafe junctions and pedestrian accidents. These actions not only increase the danger of accidents but also contribute to traffic congestion and environmental damage as a result of inefficient driving patterns (Wikipedia, 2023). Addressing these aggressive driving behaviors is critical for increasing road safety and traffic efficiency. As a result, they highlight the importance of developing techniques that address both the physical and psychological elements that influence driving.

Sensor data is crucial in identifying and assessing aggressive driving behaviors. Modern automobiles and road monitoring systems employ sensors to detect driver actions, vehicle motions, and environmental elements in real time. Accelerometers, GPS sensors, cameras, and computer vision systems are important sensors in this process (NHTSA, 2023). Furthermore, LiDAR and radar sensors monitor following distance, which aids in the identification of tailgating, and steering angle sensors detect abnormal steering associated with aggressive movements. Brake pressure sensors detect sudden braking patterns, which may indicate aggressive driving behavior.

Integrating sensor data with machine learning and artificial intelligence models enables real-time identification and response to aggressive driving. Furthermore, automatic devices such as adaptive cruise control or emergency braking can help to reduce irresponsible driving. By exploiting these technology improvements, it is feasible to limit aggressive driving and avoid accidents, contributing to overall road safety.

**“Data Preprocessing, Feature Engineering, and Methodology”**

Accurate identification of aggressive driving behavior is crucial for improving traffic safety and promoting social responsibility, and it is also a key component in the realization of intelligent transportation systems (Cai, 2024). This study aims to classify driving behavior into three categories—SLOW, NORMAL, and AGGRESSIVE—using sensor data from a Samsung Galaxy S21 smartphone. The dataset consists of accelerometer and gyroscope readings collected at a rate of two samples per second. Given the sequential nature of the data, deep learning models, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, were employed to capture temporal dependencies.

Traditional machine learning methods struggle with sequential data, making deep learning approaches, particularly RNNs and LSTMs, more suitable. This study compares these two architectures to determine their effectiveness in recognizing driving behavior patterns.

**“Dataset Description and Preprocessing”**

The dataset comprises the following features:

* **AccX, AccY, AccZ**: Acceleration along the X, Y, and Z axes (m/s²)
* **GyroX, GyroY, GyroZ**: Rotation rate along the X, Y, and Z axes (°/s)
* **Class**: Target variable indicating driving behavior (SLOW, NORMAL, AGGRESSIVE)
* **Timestamp**: Time of data recording (seconds)

**“Figure 1”**

*“Distribution of Accelerometer and Gyroscope Data”*

A group of blue and red graphs

AI-generated content may be incorrect.

This figure explains the unique feature distribution of the Accelerometer and Gyroscope data graphically with respect to frequency and value.

Each subplot represents the frequency of sensor readings, demonstrating the natural variability in the collected data. The histograms show the spread of acceleration (AccX, AccY, AccZ) and gyroscope (GyroX, GyroY, GyroZ) readings, indicating how each feature varies. These variations necessitate standardization to ensure uniform model input. Understanding feature distribution helps fine-tune models to avoid overfitting to dominant features.

**“Handling Missing and Erroneous Data:”**

To ensure data consistency, all feature columns were converted to numeric format. Missing values were imputed using the mean of respective columns to maintain the dataset’s statistical integrity. Duplicate entries were identified and removed to prevent redundant patterns from affecting model learning.

**“Feature Standardization”**

Since sensor data values vary in scale, StandardScaler was used to standardize all numerical features. This transformation ensures a uniform range, improves model stability, and accelerates convergence during training.

**“Time-Series Transformation:”**

To incorporate temporal dependencies, a time difference feature was introduced by calculating the difference between consecutive timestamps. The dataset was then segmented into 30-sample windows (15 seconds per sequence) to provide a structured input format, allowing the model to learn meaningful patterns over time.

**“Addressing Class Imbalance:”**  
​ In their seminal work, Chawla et al. (2002) introduced the Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalance by creating synthetic examples for the minority class, thereby improving classifier performance. To prevent biased learning, SMOTE (Synthetic Minority Over-sampling Technique) was applied to generate synthetic samples for underrepresented driving behavior classes. This approach ensured a balanced dataset while maintaining realistic feature distributions, improving classification accuracy.

**“Model Selection and Justification:**”

Given the sequential nature of driving behavior data, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were selected. An LSTM recurrent neural network is proposed for the classification of abnormal driving maneuvers." (Zhang et al., 2021). RNNs effectively process time-series data, while LSTMs mitigate the vanishing gradient problem, making them more suitable for learning long-term dependencies in driving patterns.

The table below compares “**Long Short-Term Memory (LSTM)”** networks and “**Recurrent Neural Networks (RNN)”** in the context of sequential data processing, specifically for driving behavior classification.

**“Table 1”**

*“Comparison of LSTM vs. RNN”*

|  |  |  |
| --- | --- | --- |
| **Feature** | **LSTM** | **RNN** |
| Handles long-term dependencies | Yes | No |
| Computational efficiency | Higher | Lower |
| Memory cell mechanism | Yes | No |
| Vanishing gradient problem | Minimal | Severe |
| Suitable for short sequences | Yes | Yes |

This table compares various features of LSTM and RNN models.

LSTMs and RNNs are both used for sequential data modeling, but they differ in handling long-term dependencies. LSTMs retain information over long sequences due to their memory cell mechanism, whereas RNNs struggle with long-term dependencies due to the vanishing gradient problem. LSTMs are computationally more expensive than RNNs but better capture complex temporal patterns. They also include gating mechanisms (input, forget, and output gates) that regulate information flow, making them more effective for tasks requiring memory over extended sequences. Both models were implemented to assess trade-offs between accuracy and efficiency.

RNNs were used as a baseline model due to their simplicity and faster computation, helping evaluate whether short-term dependencies were sufficient for classification. LSTMs were tested to determine if their memory cells improved accuracy by capturing gradual changes in driving behavior. Since LSTMs require higher computational resources, testing both models helped analyze whether their benefits outweigh the cost in real-time applications. The evaluation also ensured the model generalizes well across different sequences without bias toward shorter time frames. By comparing both, the study provided insights into whether long-term dependencies are essential for driving behavior classification, leading to the selection of the most effective architecture.

There are a lot of features that helped in the findings for this study. Chicco and Jurman (2020) stated that the Matthews correlation coefficient (MCC) is more informative and reliable than accuracy and F1 score in evaluating binary classification problems, especially with imbalanced datasets. According to Goodfellow, (Bengio and Courville, 2016), Dense layers, also known as fully connected layers, are fundamental components in neural networks where each neuron receives input from all neurons of the previous layer (p. 220). Dense layers, also known as fully connected layers, are fundamental components in neural networks where each neuron receives input from all neurons of the previous layer.

Considering these factors, both RNN and LSTM-based models were implemented for performance evaluation.

**“Model Architectures and Training Details”**

The implementation of the driving behavior classification model was partially inspired by the Kaggle dataset 'Driving Behavior' (Outofskills, n.d.), which provided valuable insights into structuring the preprocessing pipeline and model design.

**“LSTM-Based Model”**

model = Sequential([

LSTM(128, return\_sequences=True, input\_shape=(30, 6)),

Dropout(0.3),

LSTM(64, return\_sequences=False),

Dropout(0.3),

Dense(64, activation='relu'),

Dense(3, activation='softmax') ])

The first LSTM layer captures temporal dependencies and returns sequences, allowing deeper layers to refine feature representations. Dropout is one of the most popular regularization methods in the scholarly domain for preventing a neural network model from overfitting in the training phase. (Park, C., & Kwak, N. 2023). Dropout layers prevent overfitting, and dense layers map high-level representations to class probabilities.

**“RNN Model”**

model = Sequential([

RNN(64, return\_sequences=True, activation='relu'),

Dropout(0.5),

RNN(32, activation='relu'),

Dropout(0.5),

Dense(32, activation='relu'),

Dense(3, activation='softmax')

])

The RNN model, though simpler, lacks memory cells, necessitating a higher dropout rate to mitigate overfitting.

**“Training and Optimization**”

In their foundational work, Kingma and Ba (2014) introduced the Adam optimizer, stating that it is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The Adam optimizer was chosen for its adaptive learning rate and faster convergence capabilities. Training was conducted with a batch size of 32 for computational efficiency and model performance balance. Each model was trained for 50 epochs, with early stopping monitoring validation loss to prevent overfitting. Additionally, hyperparameter tuning was performed using Keras Tuner to optimize LSTM units, dropout rates, and learning rates.

**“Assumptions”**

1. The dataset accurately represents real-world driving behaviors.
2. Sensor readings were free from external noise and interference.

**“Experimental Results and Evaluation Metrics”**

**“Model Performance Evaluation”**

The LSTM and RNN models were evaluated using multiple performance metrics, including accuracy, precision, recall, F1-score, and confusion matrices. The results provide insight into the effectiveness of each model in classifying driving behaviors into SLOW, NORMAL, and AGGRESSIVE categories.

**“LSTM Model Results”**

The best validation accuracy achieved by the LSTM model was **61.07%**, with a test accuracy of **61.07%**. The classification report shows:

* **AGGRESSIVE driving** had the highest **precision (0.71)** and recall (**0.64**), leading to an **F1-score of 0.68**. This suggests that the model was relatively effective in detecting aggressive driving patterns but still had some misclassification.
* **NORMAL driving** was the most challenging to classify, with **a lower recall (0.31)** and **F1-score of 0.39**. This means that many normal driving instances were incorrectly predicted as either slow or aggressive, reducing the model's ability to differentiate this class accurately.
* **SLOW driving** had a recall of **0.82**, indicating that the model was better at identifying this class.
* The overall **balanced accuracy was 59.2%**, with an **MCC (Matthews Correlation Coefficient) of 0.4047**, suggesting a moderate correlation in predictions.

**“Figure 2”**

*“LSTM Confusion Matrix”*

A chart with blue squares

AI-generated content may be incorrect.

The **confusion matrix** shows that the model misclassified a significant number of NORMAL samples as AGGRESSIVE or SLOW, indicating challenges in differentiating between smooth and moderately aggressive driving patterns.

* 25 SLOW instances were correctly classified, but 11 were misclassified as AGGRESSIVE, suggesting some overlap in feature patterns.
* 15 NORMAL instances were correctly classified, but 9 were misclassified as SLOW and 24 as AGGRESSIVE, showing that this category is the hardest to distinguish.
* 51 AGGRESSIVE instances were correctly classified, but 10 were misclassified as NORMAL and 1 as SLOW, indicating high accuracy in detecting aggressive driving.

**“RNN Model Results”**

The RNN model showed a validation accuracy of **46.73%**, with lower generalization capabilities compared to LSTM. The classification report reveals:

* **The AGGRESSIVE** class performed the best, with a **precision of 0.66** and recall of **0.83**. This indicates that the RNN model was more confident in classifying aggressive driving correctly compared to other classes.
* **NORMAL** driving had a recall of **0.56** but a significantly lower **precision (0.34)**, meaning that many instances labeled as NORMAL were misclassified as either SLOW or AGGRESSIVE.
* **SLOW** driving had the lowest recall **(0.18)**, indicating that the model struggled to detect slow driving behaviors, leading to a poor **F1-score**.
* The **balanced accuracy was 51.95%**, with an **MCC** of **0.2416**, further confirming that the model lacked robustness in classification.

**“Figure 3”**

*“RNN Confusion Matrix”*

A chart with numbers and a number in a row

AI-generated content may be incorrect.

The confusion matrix highlights that a significant portion of SLOW driving was misclassified as NORMAL, and many NORMAL instances were predicted as SLOW, reducing classification confidence.

* 647 AGGRESSIVE instances were correctly classified, but 128 were misclassified as NORMAL and 9 as SLOW, indicating better performance in detecting aggressive driving.
* 554 NORMAL instances were correctly classified, but 236 were misclassified as SLOW, and 207 were misclassified as AGGRESSIVE, highlighting significant misclassification.
* 226 SLOW instances were correctly classified, but 949 were misclassified as NORMAL, showing that the RNN model struggled heavily with slow-driving patterns.

**“Findings and Discussion”**

The LSTM model outperformed the RNN, particularly in handling long-term dependencies. The results indicate that LSTM’s memory cell mechanism helps retain temporal patterns better, leading to improved classification of gradual changes in driving behavior. However, both models struggled with distinguishing NORMAL driving from SLOW or AGGRESSIVE, which suggests additional feature engineering (e.g., speed variations, road conditions) could enhance classification performance.

Despite its superiority, both models struggled with distinguishing NORMAL driving from SLOW or AGGRESSIVE, suggesting that additional feature engineering is required. One possible explanation for the lower accuracy in both models is overlapping feature distributions between classes. The application of SMOTE for class balancing may have introduced synthetic samples that were difficult for the models to distinguish effectively.

In summary, while LSTMs demonstrated superior classification performance, further improvements in feature engineering, additional data augmentation, and alternative deep learning architectures could further enhance model accuracy and robustness in real-world driving behavior classification.

**“Practical Implications of the Model’s Results”**

Despite advances in autonomous driving technology, traffic accidents remain a problem to be solved in the transportation system (Lee et al., 2021). The ability to classify driving behavior accurately has significant implications for road safety policies, insurance, and driver monitoring applications. Governments and transportation agencies can leverage such models to develop data-driven policies aimed at reducing road accidents by identifying aggressive driving patterns in real time. Traffic law enforcement could use these models to trigger alerts when a driver exhibits prolonged aggressive behavior, potentially preventing accidents and improving road safety.

Insurance companies could integrate these classification models into usage-based insurance (UBI) policies, where premiums are adjusted based on a driver's actual behavior rather than static risk profiles. Drivers demonstrating safe behavior (classified as SLOW or NORMAL) may benefit from lower insurance premiums, while consistently aggressive drivers may be flagged for increased premiums, creating an incentive for safer driving habits.

For commercial fleet management, these models can be used to monitor and assess driver behavior. Companies operating large vehicle fleets can track their drivers' behaviors and intervene when risky driving is detected. This not only enhances road safety but also improves vehicle longevity and reduces maintenance costs.

Additionally, in the era of autonomous driving, incorporating driving behavior classification models into advanced driver-assistance systems (ADAS) can help vehicles recognize hazardous driving environments and adjust accordingly. By detecting aggressive behaviors in surrounding traffic, autonomous vehicles can take proactive safety measures, such as maintaining safer distances or adjusting driving speed.

In summary, the findings of this study demonstrate the potential for deep learning models to contribute to safer roads, fairer insurance policies, and better driver accountability. While classification accuracy can still be improved, these models provide a foundation for real-world applications in transportation safety and risk assessment

**“References”**

“Cai, G. (2024). *Aggressive driving behavior recognition based on K-means clustering*. Scholarly Review Journal. https://www.scholarlyreview.org/article/127166-aggressive-driving-behavior-recognition-based-on-k-means-clustering”

“Lee, S., Kim, D., & Lee, J. (2021). *Driving behavior classification and sharing system using CNN-LSTM-based deep learning*. *Applied Sciences, 11*(21), 10420. https://www.mdpi.com/2076-3417/11/21/10420”

“Zhang, H., Nan, Z., Yang, T., Liu, Y., & Zheng, N. (2021). *A driving behavior recognition model with Bi-LSTM and multi-scale CNN*. *arXiv preprint arXiv:2103.00801*. <https://arxiv.org/abs/2103.00801>”

“Wikipedia(2023) Aggressive driving. *Wikipedia, The Free Encyclopedia.* Retrieved *Month Day, Year*, <https://en.wikipedia.org/wiki/Aggressive_driving>”

“National Highway Traffic Safety Administration. (2023). *Aggressive driving*. U.S. Department of Transportation. <https://www.nhtsa.gov>”

“Verschuur, W. L. G., & Hurts, K. (2007). *Modeling safe and unsafe driving behavior*. Accident Analysis & Prevention, 39(3), 481-488. <https://doi.org/10.1016/j.aap.2006.08.010>”

“Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over the F1 score and accuracy in binary classification evaluation. *BMC Genomics, 21*(1), 6. https://doi.org/10.1186/s12864-019-6413-7”

“Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research, 16*, 321–357. <https://doi.org/10.1613/jair.953>”

“Park, C., & Kwak, N. (2023). A review on dropout regularization approaches for deep neural networks. *Electronics, 12*(14), 3106. <https://doi.org/10.3390/electronics12143106>”

“Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. <https://www.deeplearningbook.org/>”

“Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980. <https://doi.org/10.48550/arXiv.1412.6980>”

“Outofskills. (n.d.). *Driving behavior dataset*. Kaggle. <https://www.kaggle.com/datasets/outofskills/driving-behavior>”

“Code Appendix”

“LSTM  
# Import necessary libraries

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.layers import LSTM, Dense, Dropout

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE

from tensorflow.keras.utils import to\_categorical

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, matthews\_corrcoef, cohen\_kappa\_score, jaccard\_score

import keras\_tuner as kt

# Load dataset filenames

data\_train\_file = "train\_motion\_data.csv"

data\_test\_file = "test\_motion\_data.csv"

# Read datasets

data\_train = pd.read\_csv(data\_train\_file)

data\_test = pd.read\_csv(data\_test\_file)

# Define feature and target columns

feature\_cols = ['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'GyroZ']

target\_col = 'Class'

timestamp\_col = 'Timestamp'

# Convert feature columns to numeric to handle any inconsistencies

for col in feature\_cols:

data\_train[col] = pd.to\_numeric(data\_train[col], errors='coerce')

data\_test[col] = pd.to\_numeric(data\_test[col], errors='coerce')

# Encode categorical target labels

label\_encoder = LabelEncoder()

data\_train[target\_col] = label\_encoder.fit\_transform(data\_train[target\_col])

data\_test[target\_col] = label\_encoder.transform(data\_test[target\_col])

# Handle missing values by replacing with column means

data\_train.fillna(data\_train.mean(), inplace=True)

data\_test.fillna(data\_test.mean(), inplace=True)

# Normalize feature columns using StandardScaler

scaler = StandardScaler()

data\_train[feature\_cols] = scaler.fit\_transform(data\_train[feature\_cols])

data\_test[feature\_cols] = scaler.transform(data\_test[feature\_cols])

# Convert timestamps to time difference feature

data\_train['time\_diff'] = data\_train[timestamp\_col].diff().fillna(0)

data\_test['time\_diff'] = data\_test[timestamp\_col].diff().fillna(0)

# Function to create sequences for time-series modeling

def create\_sequences(df, sequence\_length=30):

X\_seq, y\_seq = [], []

for i in range(len(df) - sequence\_length):

X\_seq.append(df.iloc[i:i+sequence\_length][feature\_cols].values)

y\_seq.append(df.iloc[i+sequence\_length][target\_col])

return np.array(X\_seq), np.array(y\_seq)

# Define sequence length

seq\_length = 30

# Create sequences for training and testing

X\_train\_seq, y\_train\_seq = create\_sequences(data\_train, seq\_length)

X\_test\_seq, y\_test\_seq = create\_sequences(data\_test, seq\_length)

# Handle class imbalance using SMOTE

smote = SMOTE(random\_state=42)

X\_train\_flat = X\_train\_seq.reshape(X\_train\_seq.shape[0], -1)

X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train\_flat, y\_train\_seq)

X\_train\_resampled = X\_train\_resampled.reshape(-1, seq\_length, len(feature\_cols))

# Split training data into training and validation sets

X\_train\_final, X\_val, y\_train\_final, y\_val = train\_test\_split(

X\_train\_resampled, y\_train\_resampled, test\_size=0.2, random\_state=42)

# One-hot encode labels

y\_train\_final = to\_categorical(y\_train\_final)

y\_val = to\_categorical(y\_val)

y\_test\_one\_hot = to\_categorical(y\_test\_seq)

# Function to build and optimize LSTM model using Keras Tuner

def build\_lstm\_model(hp):

model = Sequential()

model.add(LSTM(hp.Int('units', min\_value=32, max\_value=256, step=32), return\_sequences=True, input\_shape=(seq\_length, len(feature\_cols))))

model.add(Dropout(hp.Float('dropout\_rate', min\_value=0.2, max\_value=0.5, step=0.1)))

model.add(LSTM(hp.Int('lstm\_units', min\_value=32, max\_value=256, step=32)))

model.add(Dense(hp.Int('dense\_units', min\_value=32, max\_value=128, step=32), activation='relu'))

model.add(Dense(len(np.unique(y\_train\_resampled)), activation='softmax'))

model.compile(

optimizer=Adam(learning\_rate=hp.Choice('learning\_rate', values=[1e-2, 1e-3, 1e-4])),

loss='categorical\_crossentropy',

metrics=['accuracy']

)

return model

# Hyperparameter tuning using Keras Tuner

tuner = kt.Hyperband(

build\_lstm\_model,

objective='val\_accuracy',

max\_epochs=50,

factor=3,

directory='lstm\_tuner\_dir',

project\_name='lstm\_optimization')

tuner.search(X\_train\_final, y\_train\_final, epochs=50, validation\_data=(X\_val, y\_val), callbacks=[EarlyStopping(monitor='val\_loss', patience=5)])

# Retrieve best LSTM model

best\_lstm\_model = tuner.get\_best\_models(num\_models=1)[0]

# Train the best LSTM model

best\_lstm\_model.fit(X\_train\_final, y\_train\_final, validation\_data=(X\_val, y\_val), epochs=50, batch\_size=32, callbacks=[EarlyStopping(monitor='val\_loss', patience=5)])

# Evaluate the model on test data

test\_loss, test\_acc = best\_lstm\_model.evaluate(X\_test\_seq, y\_test\_one\_hot)

print(f"Test Accuracy: {test\_acc:.4f}")

# Predictions using the trained model

y\_pred = best\_lstm\_model.predict(X\_test\_seq)

actual\_y\_test = np.argmax(y\_test\_one\_hot, axis=1)

predicted\_classes = np.argmax(y\_pred, axis=1)

# Generate classification report and confusion matrix

print("Classification Report:\n", classification\_report(actual\_y\_test, predicted\_classes))

print("Confusion Matrix:\n", confusion\_matrix(actual\_y\_test, predicted\_classes))

# Save trained LSTM model

best\_lstm\_model.save("optimized\_lstm\_model.keras")”

Output:  
A screenshot of a computer

AI-generated content may be incorrect.

“RNN

# Import necessary libraries

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense, Dropout, Bidirectional

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE

from tensorflow.keras.utils import to\_categorical

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, matthews\_corrcoef, cohen\_kappa\_score, jaccard\_score

# Load dataset filenames

data\_train\_file = "train\_motion\_data.csv"

data\_test\_file = "test\_motion\_data.csv"

# Read datasets

data\_train = pd.read\_csv(data\_train\_file)

data\_test = pd.read\_csv(data\_test\_file)

# Define feature and target columns

feature\_cols = ['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'GyroZ']

target\_col = 'Class'

timestamp\_col = 'Timestamp'

# Convert feature columns to numeric to handle any inconsistencies

for col in feature\_cols:

data\_train[col] = pd.to\_numeric(data\_train[col], errors='coerce')

data\_test[col] = pd.to\_numeric(data\_test[col], errors='coerce')

# Encode categorical target labels

label\_encoder = LabelEncoder()

data\_train[target\_col] = label\_encoder.fit\_transform(data\_train[target\_col])

data\_test[target\_col] = label\_encoder.transform(data\_test[target\_col])

# Handle missing values by replacing with column means

data\_train.fillna(data\_train.mean(), inplace=True)

data\_test.fillna(data\_test.mean(), inplace=True)

# Normalize feature columns using StandardScaler

scaler = StandardScaler()

data\_train[feature\_cols] = scaler.fit\_transform(data\_train[feature\_cols])

data\_test[feature\_cols] = scaler.transform(data\_test[feature\_cols])

# Convert timestamps to time difference feature

data\_train['time\_diff'] = data\_train[timestamp\_col].diff().fillna(0)

data\_test['time\_diff'] = data\_test[timestamp\_col].diff().fillna(0)

# Function to create sequences for time-series modeling

def create\_sequences(df, sequence\_length=30):

X\_seq, y\_seq = [], []

for i in range(len(df) - sequence\_length):

X\_seq.append(df.iloc[i:i+sequence\_length][feature\_cols].values)

y\_seq.append(df.iloc[i+sequence\_length][target\_col])

return np.array(X\_seq), np.array(y\_seq)

# Define sequence length

seq\_length = 30

# Create sequences for training and testing

X\_train\_seq, y\_train\_seq = create\_sequences(data\_train, seq\_length)

X\_test\_seq, y\_test\_seq = create\_sequences(data\_test, seq\_length)

# Handle class imbalance using SMOTE

smote = SMOTE(random\_state=42)

X\_train\_flat = X\_train\_seq.reshape(X\_train\_seq.shape[0], -1)

X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train\_flat, y\_train\_seq)

X\_train\_resampled = X\_train\_resampled.reshape(-1, seq\_length, len(feature\_cols))

# Split training data into training and validation sets

X\_train\_final, X\_val, y\_train\_final, y\_val = train\_test\_split(

X\_train\_resampled, y\_train\_resampled, test\_size=0.2, random\_state=42)

# One-hot encode labels

y\_train\_final = to\_categorical(y\_train\_final)

y\_val = to\_categorical(y\_val)

y\_test\_one\_hot = to\_categorical(y\_test\_seq)

# Build RNN Model

rnn\_model = Sequential([

Bidirectional(SimpleRNN(64, return\_sequences=True, activation='relu')),

Dropout(0.5), # Increased dropout for regularization

Bidirectional(SimpleRNN(32, activation='relu')),

Dropout(0.5),

Dense(32, activation='relu'),

Dense(len(np.unique(y\_train\_resampled)), activation='softmax') # Multi-class classification output

])

# Compile RNN Model

rnn\_model.compile(optimizer=Adam(learning\_rate=0.001), loss='categorical\_crossentropy', metrics=['accuracy'])

# Callbacks for RNN training

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3)

# Train RNN Model

rnn\_model.fit(

X\_train\_final, y\_train\_final,

validation\_data=(X\_val, y\_val),

epochs=50,

batch\_size=32,

callbacks=[early\_stopping, reduce\_lr]

)

# Evaluate RNN Model

rnn\_test\_loss, rnn\_test\_acc = rnn\_model.evaluate(X\_test\_seq, y\_test\_one\_hot)

print(f"RNN Test Accuracy: {rnn\_test\_acc:.4f}")

# Predict with RNN Model

y\_pred\_rnn = rnn\_model.predict(X\_test\_seq)

actual\_y\_test\_rnn = np.argmax(y\_test\_one\_hot, axis=1)

predicted\_classes\_rnn = np.argmax(y\_pred\_rnn, axis=1)

# Generate classification report and confusion matrix for RNN

print("RNN Classification Report:\n", classification\_report(actual\_y\_test\_rnn, predicted\_classes\_rnn))

print("RNN Confusion Matrix:\n", confusion\_matrix(actual\_y\_test\_rnn, predicted\_classes\_rnn))

# Calculate evaluation metrics for RNN

print(f"RNN Balanced Accuracy: {balanced\_accuracy\_score(actual\_y\_test\_rnn, predicted\_classes\_rnn):.4f}")

print(f"RNN Matthews Correlation Coefficient (MCC): {matthews\_corrcoef(actual\_y\_test\_rnn, predicted\_classes\_rnn):.4f}")

print(f"RNN Cohen’s Kappa: {cohen\_kappa\_score(actual\_y\_test\_rnn, predicted\_classes\_rnn):.4f}")

print(f"RNN Jaccard Index: {jaccard\_score(actual\_y\_test\_rnn, predicted\_classes\_rnn, average='macro'):.4f}")

# Save trained RNN Model

rnn\_model.save("rnn\_model.keras")”

Output:  
A screenshot of a computer

AI-generated content may be incorrect.