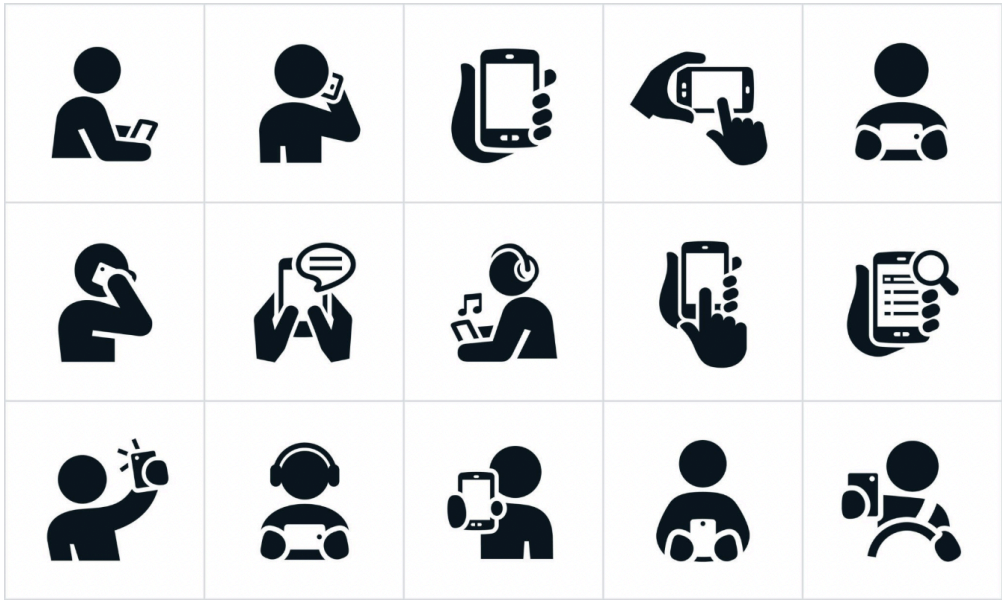
Distracted Driver Detection - With Transformer, ResNet, and LSTM Architectures



Springboard Capstone Project

Sook hyun Lee

November 2025

1. Introduction

1.1 Problem statement

Driving while distracted, fatigued, or drowsy is a significant cause of road accidents and traffic fatalities. Distractions that divert a driver's attention from the road, such as engaging in conversations with passengers, making or receiving phone calls, sending or reading text messages, eating while driving, or reacting to external events, can critically impair driving performance. Likewise, fatigue and drowsiness, often resulting from prolonged driving hours or insufficient sleep, substantially reduce a driver's alertness and reaction time.

In this project, we aim to develop deep learning–based classifiers capable of detecting distracted driving behaviors. The proposed system integrates **Transformer encoders**, **Convolutional Neural Networks (CNNs) / ResNet architectures**, and **Recurrent Neural Networks (RNNs) / Long Short-Term Memory (LSTM) networks** within the **PyTorch** framework. By leveraging these architectures, the project seeks to accurately identify and classify various forms of driver distraction, thereby contributing to improved road safety and accident prevention.

1.2 Value to client

Correctly predicting the state of drivers can bring values to the following customers:

Automotive companies:
Improve autonomous or semi-autonomous driving transitions

Fleet management companies & Commercial operators:
Prevent fatigue-related crashes in logistics and delivery services.

Tech companies (sensors, software):
Apps or platforms that alert drivers to stay focused.

Insurance companies:
Incentivize safe driving. Determine liability in case of crashes.

Healthcare and Research institutions:
Inform public health policies. Develop assistive technologies for impaired drivers.

Consumers / Drivers:
Personal driving analytics and feedback.

2. Data acquisition and overview

2.1 Data source

The FordChallenge dataset is obtained from Kaggle and consists of data from 600 real-time driving sessions, each lasting approximately 2 minutes and sampled at 100ms intervals [1] (i.e., a sampling rate of 10 Hz). These sessions include trials from 100 drivers of varying ages and genders, with specific details such as names and units undisclosed by the challenge organizers. Each data point is labeled with a binary outcome: 0 for **"distracted"** and 1 for **"alert"**. The dataset contains **physiological**, **environmental**, and **vehicular** measurements. The data files include:

- FordChallenge_X.csv (Multi-modal Features), FordChallenge_y.csv (Labels)
- FordChallenge_subject_id.csv

The final file containing *subject_id* information is excluded from this work for the sake of simplicity, as the primary focus is on classifying time-series data. However, this information can be incorporated in future extensions of the model, where subject-specific and other non-temporal features are utilized to enhance classification performance and personalization.

While the feature labels are removed, one can infer potential feature meanings by examining correlations, distributions, and their temporal patterns. Furthermore, one can still train and evaluate models to assess their predictive performance without explicit feature names. The table below lists some possible feature labels.

Table 1

Modality	**Data Type / Examples**	**Purpose / What It Tells Us**
Physiological data	Heart rate (HR), heart rate variability (HRV), EEG, EDA (skin conductance), eye movement, facial EMG	Reflects the **driver's internal state** - stress, fatigue, attention, emotional arousal.
Environmental data	Road type, weather, lighting, traffic density, noise level, GPS location	Provides **context** - external factors influencing driving behavior or risk.
Vehicular data	Speed, acceleration, steering angle, brake pressure, lane position, throttle input	Captures how the **vehicle behaves** and how the **driver interacts** with it - can reveal distraction, fatigue, or aggressive driving.

2.2 Data summary

The dataset consists of 36257 multivariate time series. Each time series is of length 40, representing 4 seconds of data per time series at 10 Hz. At each timestamp, the data includes 30 (8 physiological, 11 environmental, and 11 vehicular) measurements.

2.3 Class imbalance

The dataset is imbalanced, with only 36.5% of instances labeled as "distracted".

Figure 1. Class imbalance in the data.

3. Data exploration

In this section, we examine time series features from three data categories, separately for distracted and alert drivers. Through visual inspection, we can obtain insights into which features may be most relevant.

3.1 Physiological data

In Figure 1, each panel shows 10 time series samples of a given feature for each category.

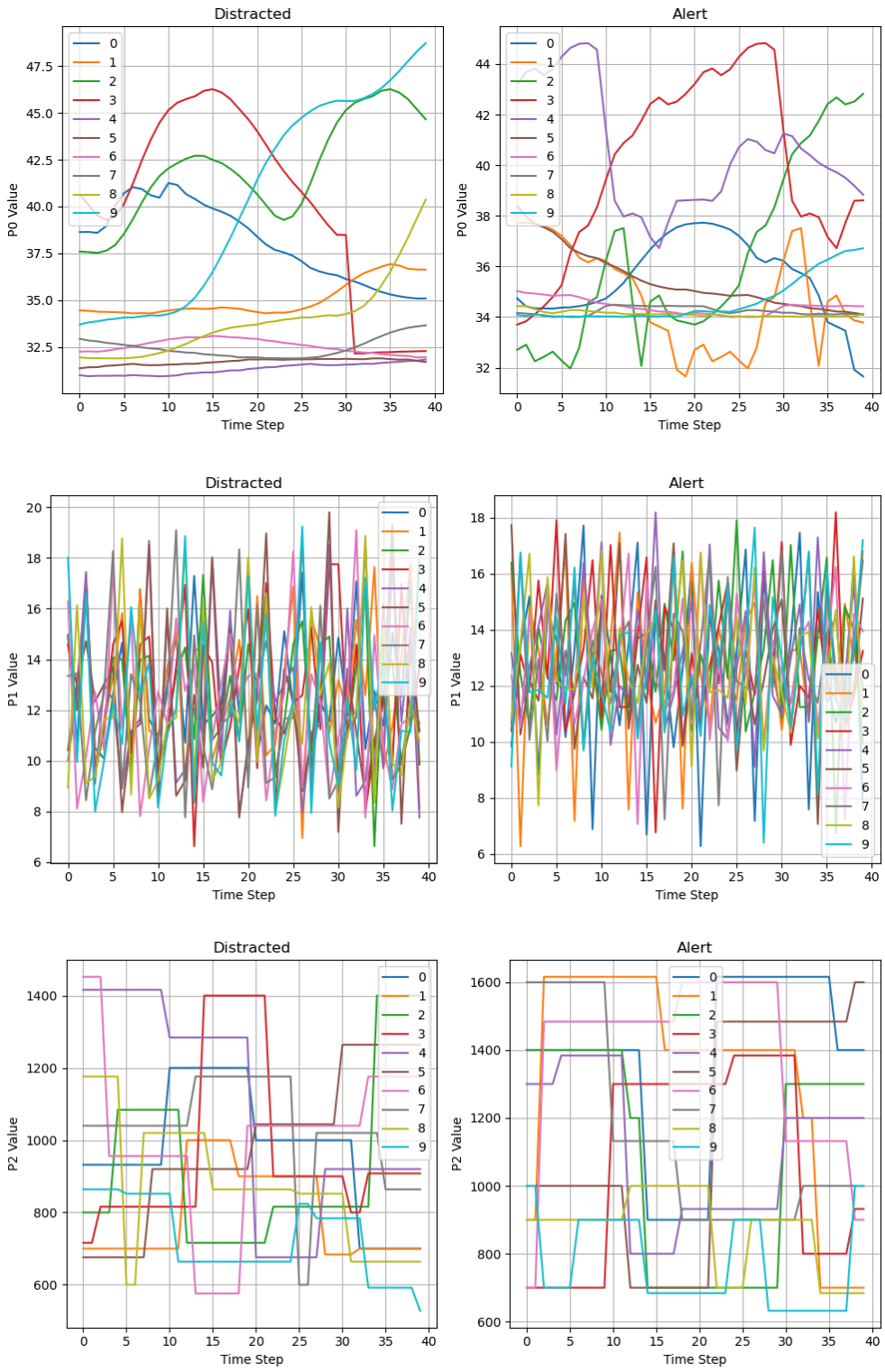


Figure 1. Time series samples of physiological features for distracted and alert Drivers. Legend entries correspond to the numbered time series. (cont.)

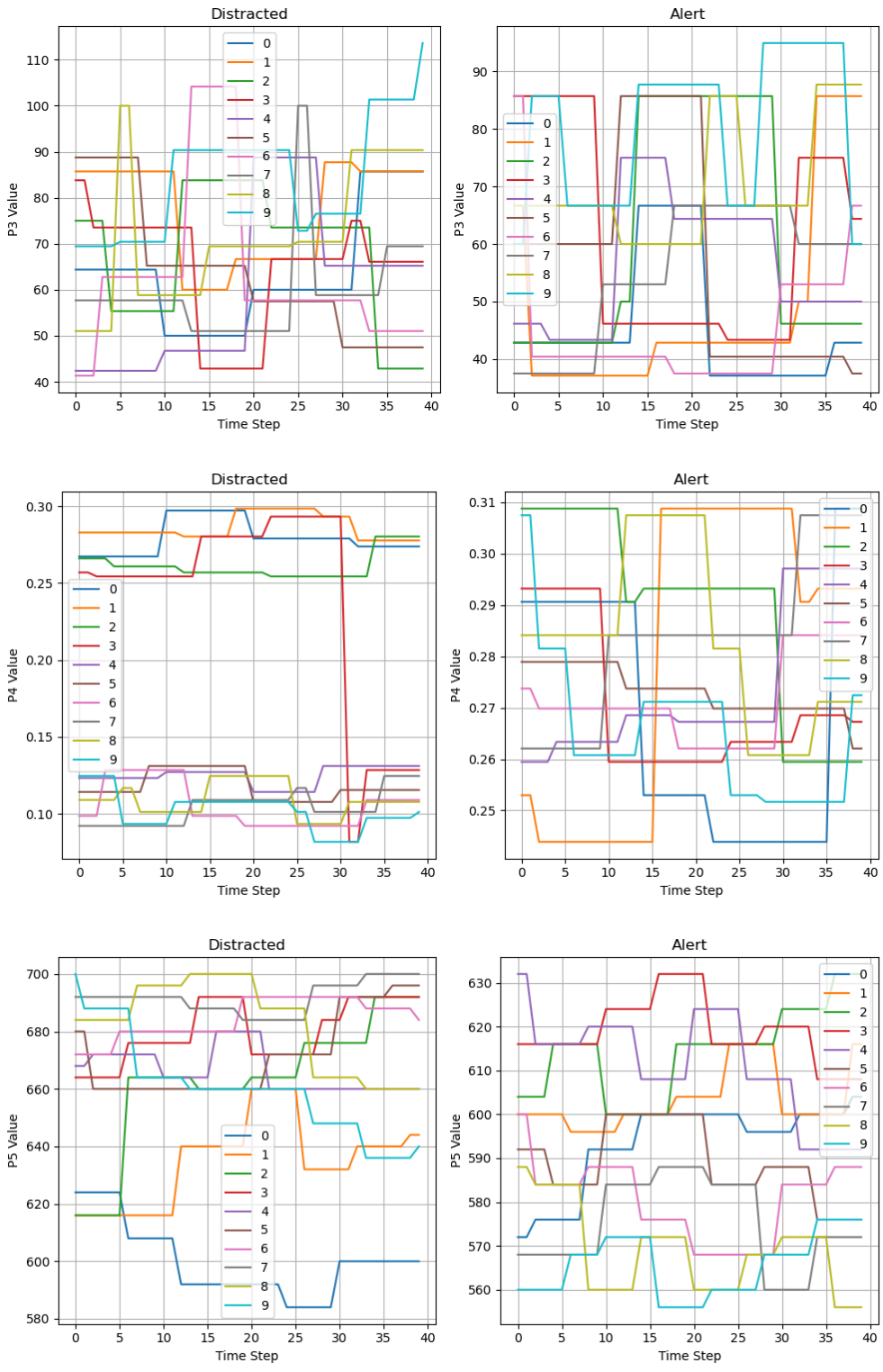


Figure 1. Time series samples of physiological features for distracted and alert Drivers. Legend entries correspond to the numbered time series. (cont.)

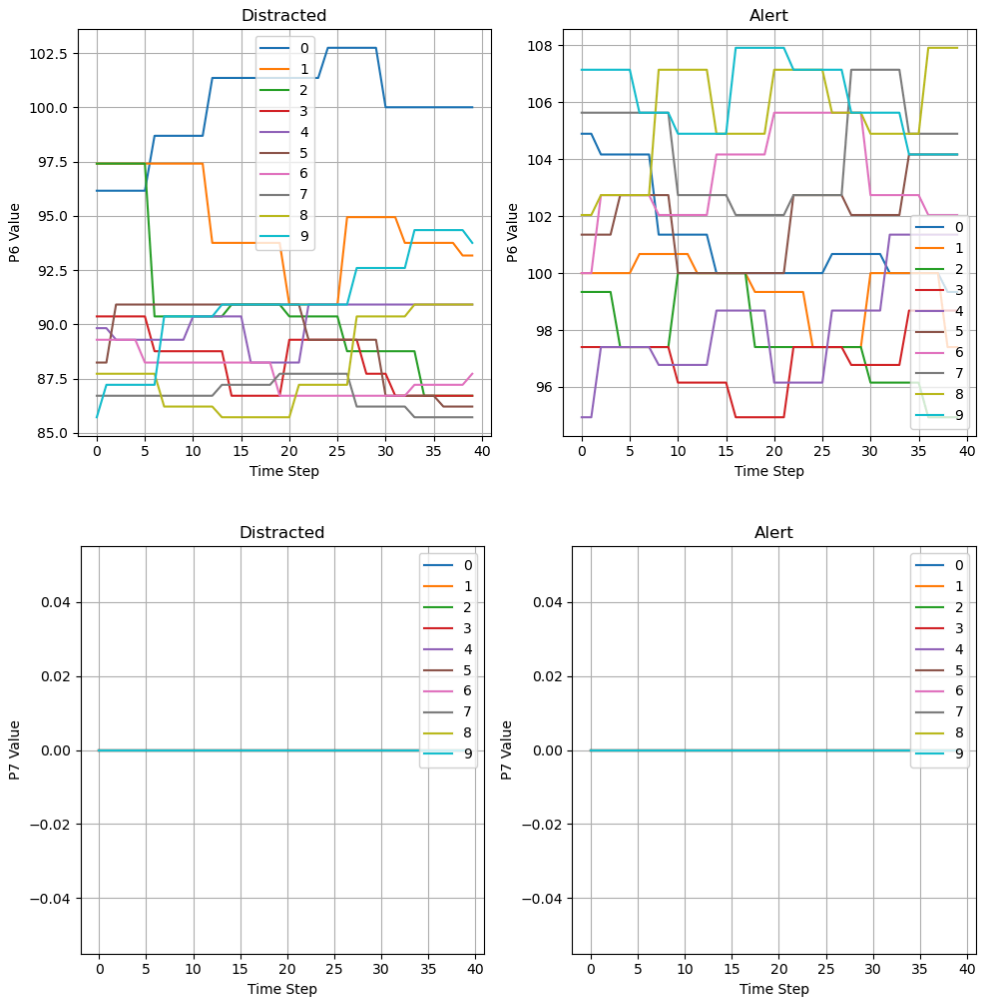


Figure 1. Time series samples of physiological features for distracted and alert Drivers. Legend entries correspond to the numbered time series.

The data includes various types, such as continuous, step-like, and binary signals, and some features exhibit visibly distinct patterns between distracted and alert drivers.

3.1.1 ECG data and machine learning

ECG signals are typically 1D waveforms recorded over time that represent the heart's electrical activity. While deep learning algorithms are capable of learning complex feature interactions directly from raw data, converting these signals into *clean, normalized*, and meaningful *numerical* representations for model training can potentially enhance model performance.

- Signal cleaning and filtering
    - Baseline wander (slow drift) 0.5 Hz
    - High-frequency noise (muscle, power line) 45-60 Hz
    - Artifacts (movement, poor electrodes)
- Normalization / Standardization for stability across people and devices
- Use filtered + normalized signals
  Optionally compute derived representations:
    - Spectrograms (Short-Time Fourier Transform)
    - Wavelet transforms
    - Heartbeats centered around R-peaks (for CNN classification)

In our data:

- Sampling frequency (fs) = **10 Hz**
- Therefore, **Nyquist frequency** = fs / 2 = **5** Hz → This means you cannot represent or filter frequencies above 5 Hz (they will alias).

3.1.2 Feature P1

The P1 data appears to consist of high-frequency signals and it is challenging to distinguish between the two states. First, Fourier analysis will be applied to examine the frequency components. Next, a Butterworth bandpass filter will be applied to remove low-frequency components, and the resulting filtered time series distributions for the two classes will be compared.

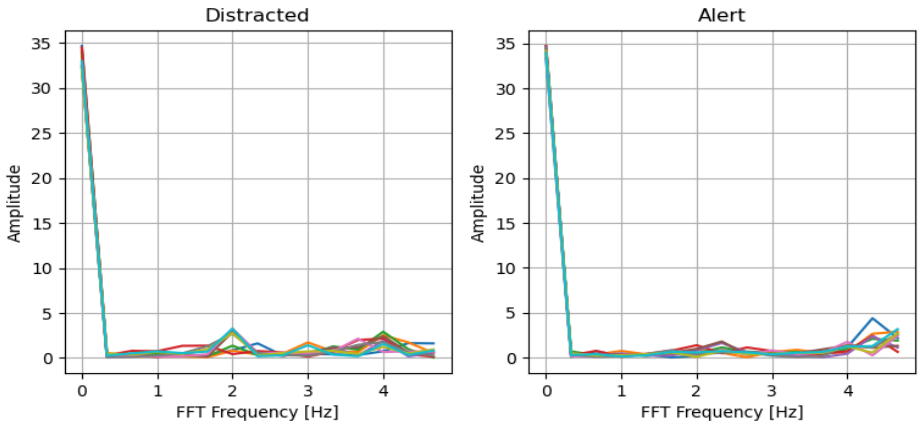


Figure 2. Fourier-transformed signals of feature P1.

Figure 2 shows the Fourier-transformed signals of feature P1. The peak frequencies differ between the two classes, indicating that the frequency distribution may serve as a valuable feature. Notably, frequencies below 0.3 Hz are identical across both classes and can therefore be removed.

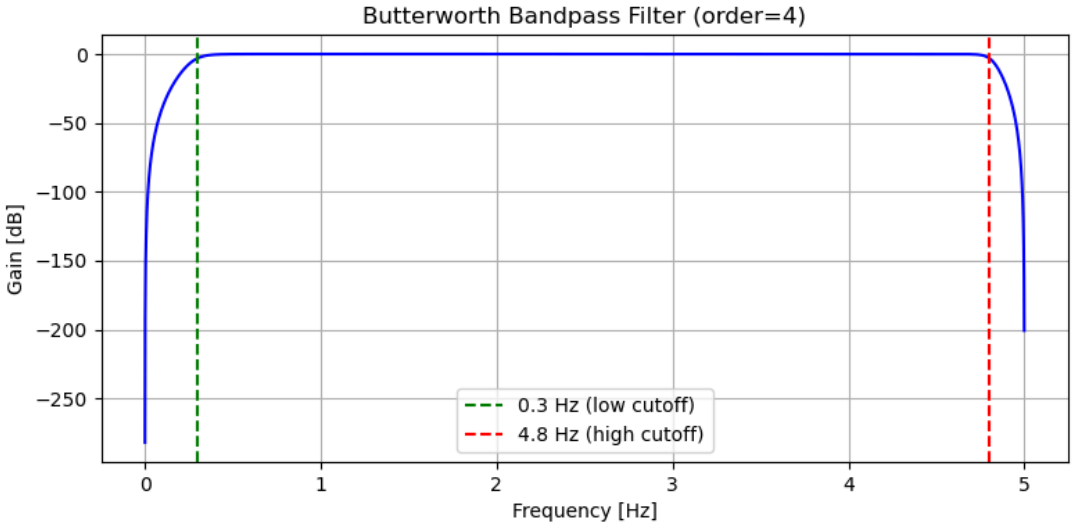


Figure 3. Gain and cutoff frequencies from Butterworth bandpass filter.

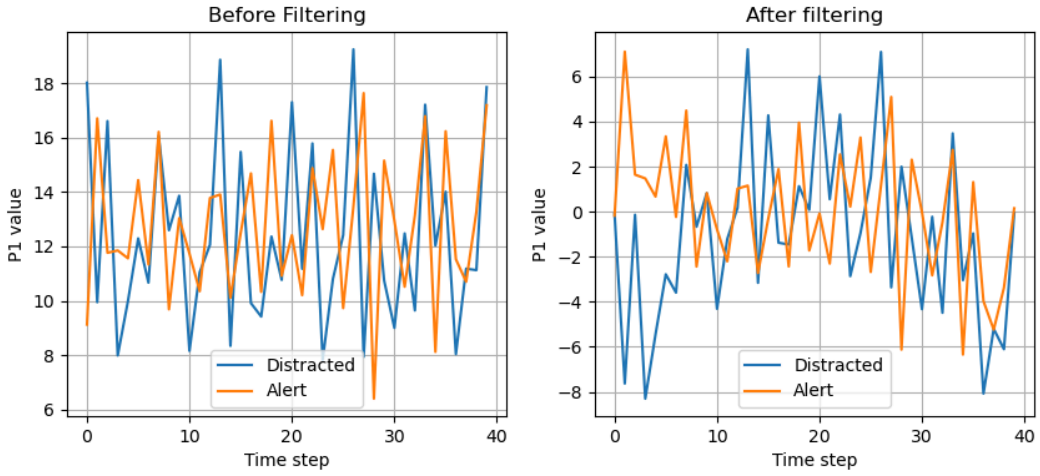


Figure 4. Feature P1 – Time series before and after Butterworth bandpass filtering.

Figure 3 shows the gain and cutoff frequencies obtained by applying a Butterworth bandpass filter. The frequency range passing through the filter exhibits a uniform gain. Figure 4 compares the time series of P1 before and after applying the Butterworth bandpass filter.

3.1.3 Feature P7 - Constant zero

The eighth feature consistently has a value of zero and will therefore be excluded from our analysis. Features with constant zero values can contribute noise to the dataset; hence, a thorough examination of all features is essential.

3.2 Environmental data

In Figure 3, similar to the physiological data, each panel shows 10 time series samples of a given environmental feature for each category.

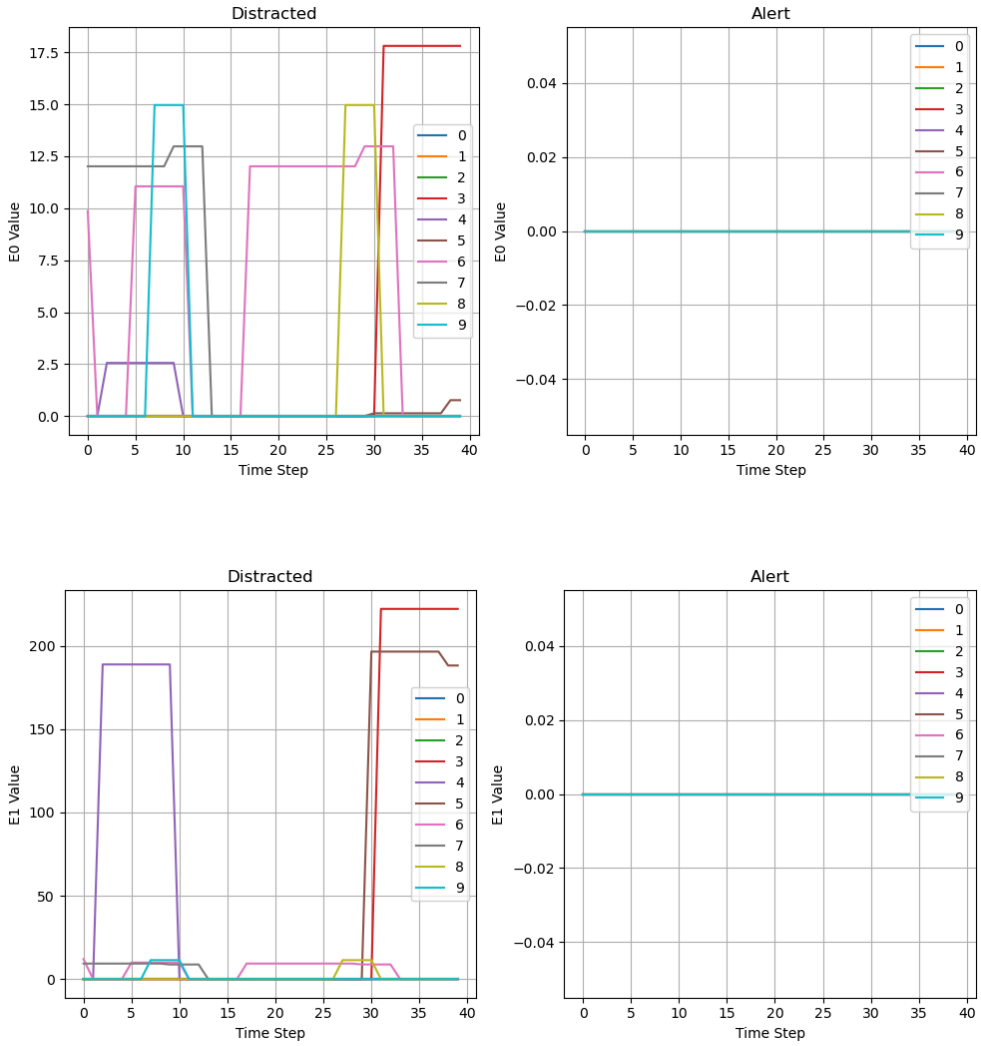


Figure 5. Time series samples of environmental features for distracted and alert Drivers. Legend entries correspond to the numbered time series. (cont.)

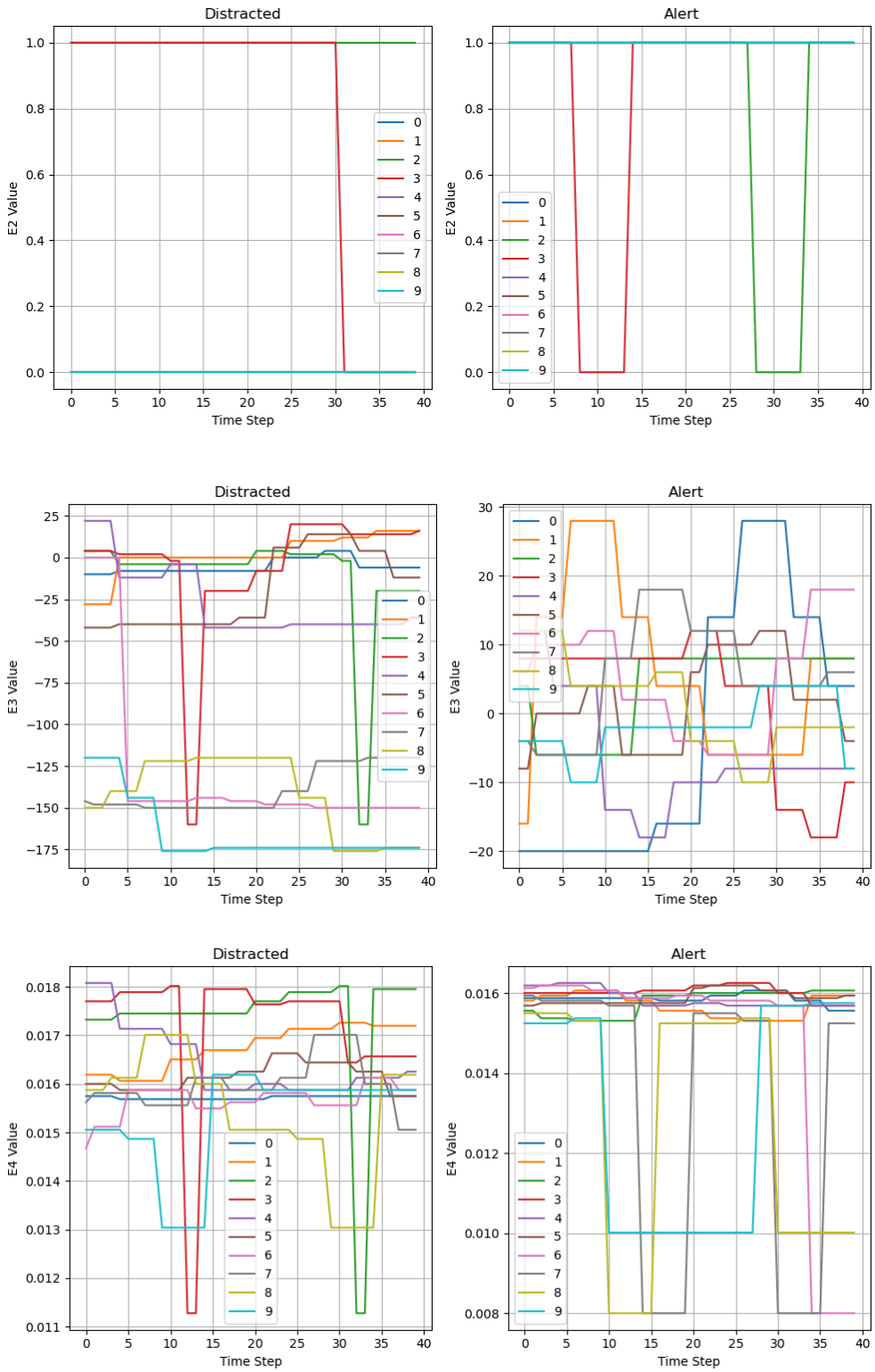


Figure 5. Time series samples of environmental features for distracted and alert Drivers. Legend entries correspond to the numbered time series. (cont.)

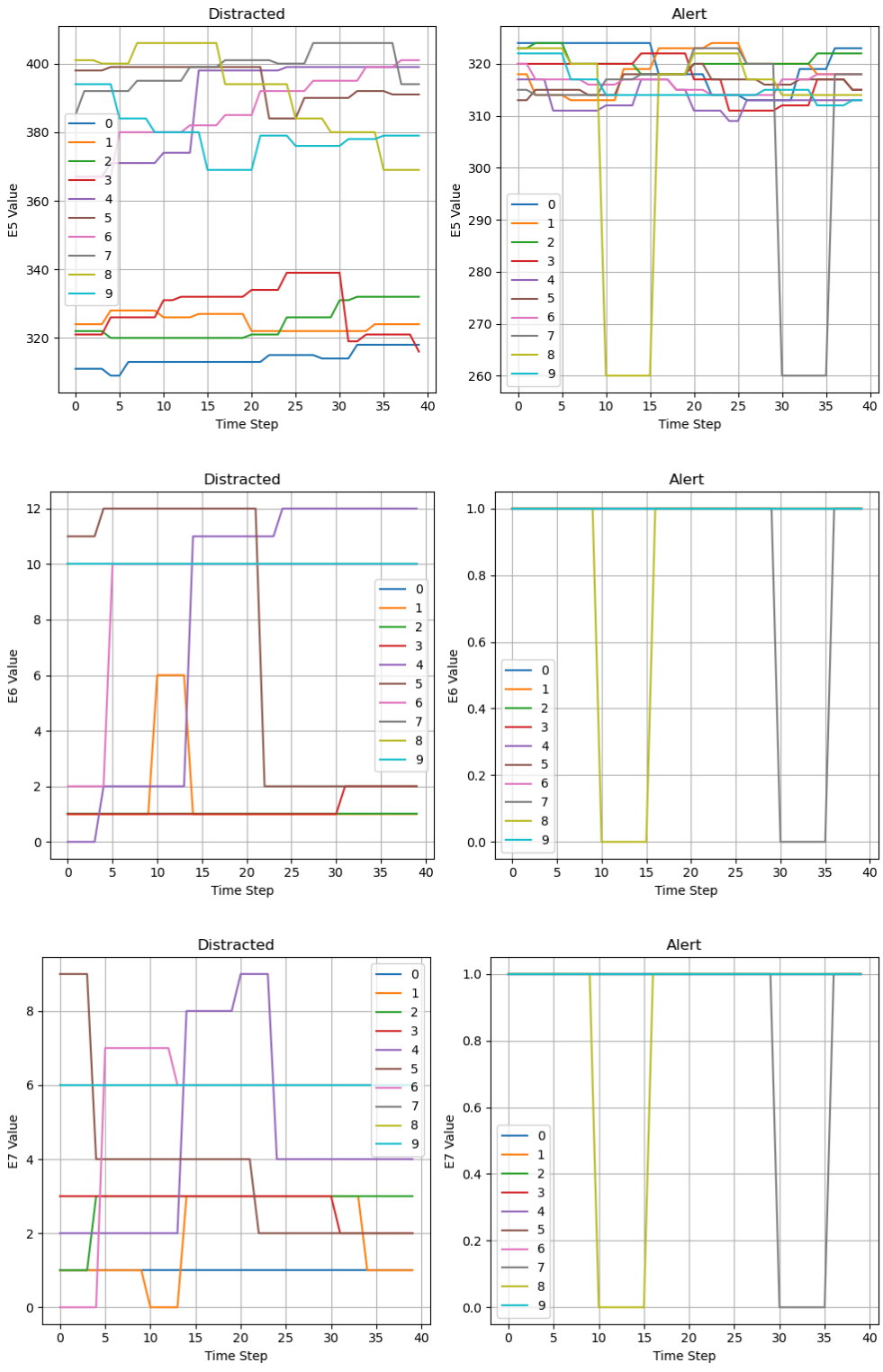


Figure 5. Time series samples of environmental features for distracted and alert Drivers. Legend entries correspond to the numbered time series. (cont.)

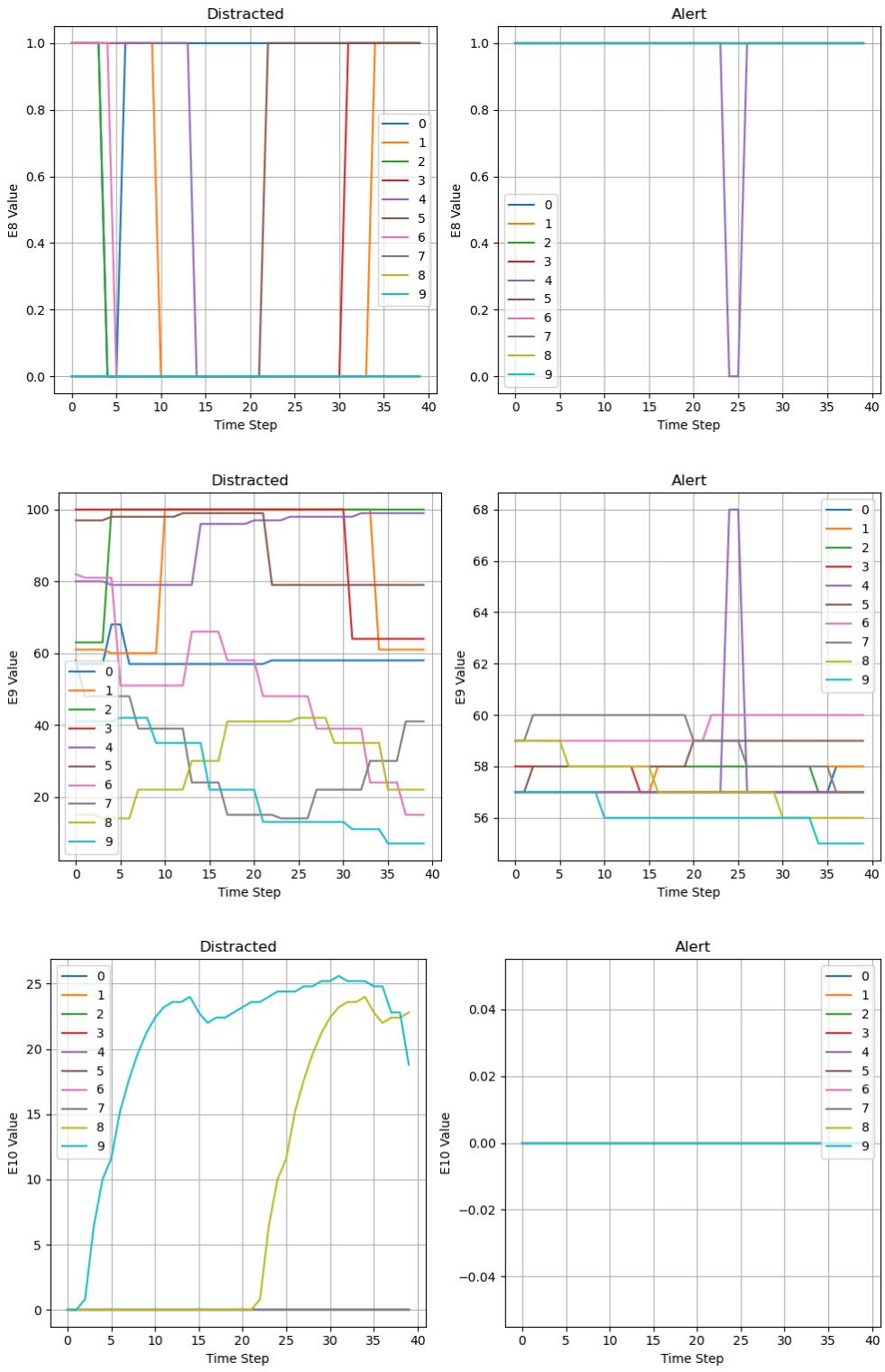


Figure 5. Time series samples of environmental features for distracted and alert Drivers. Legend entries correspond to the numbered time series.

Most of the environmental data exhibit step-like values, and some signals differ distinctly between the two classes. E2 and E8 are binary categorical variables.

3.3 Vehicular data

In Figure 6, similar to previous subsections, each panel shows 10 time series samples of a given vehicular feature for each category.

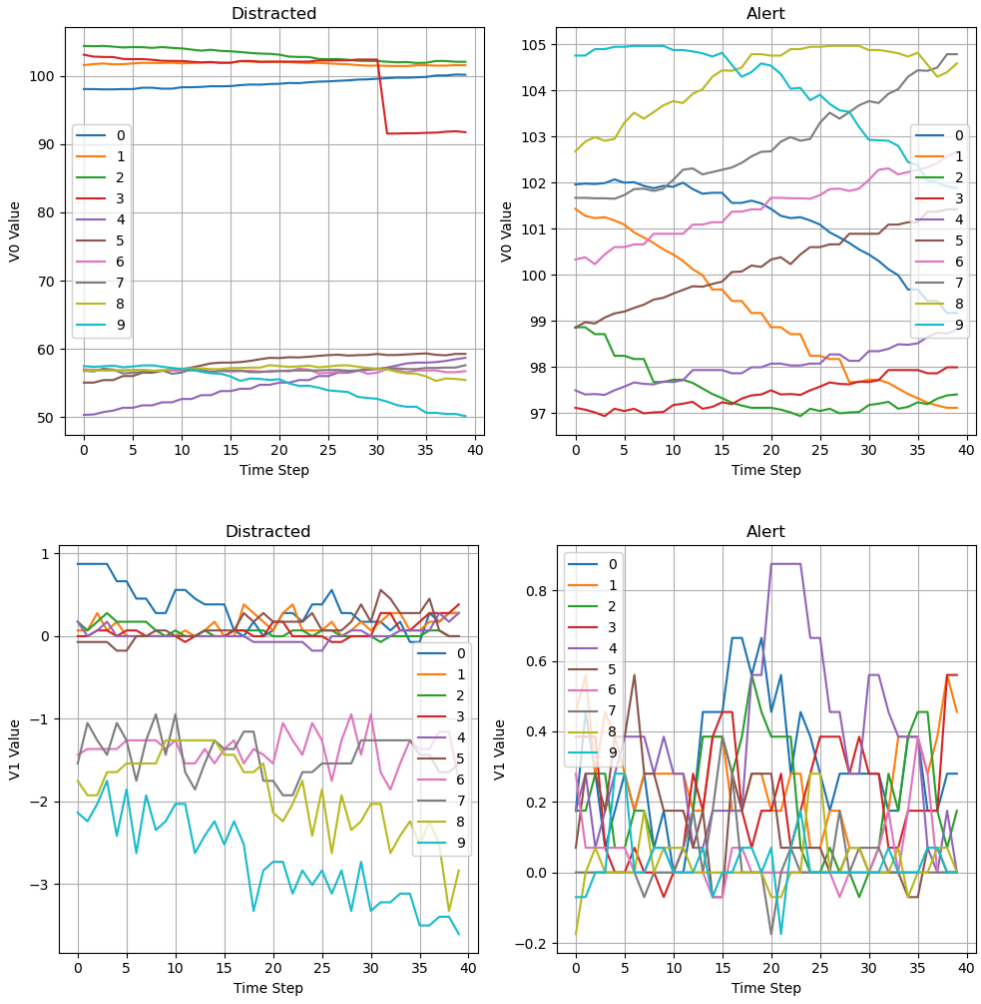


Figure 6. Time series samples of vehicular features for distracted and alert Drivers. Legend entries correspond to the numbered time series. (cont.)

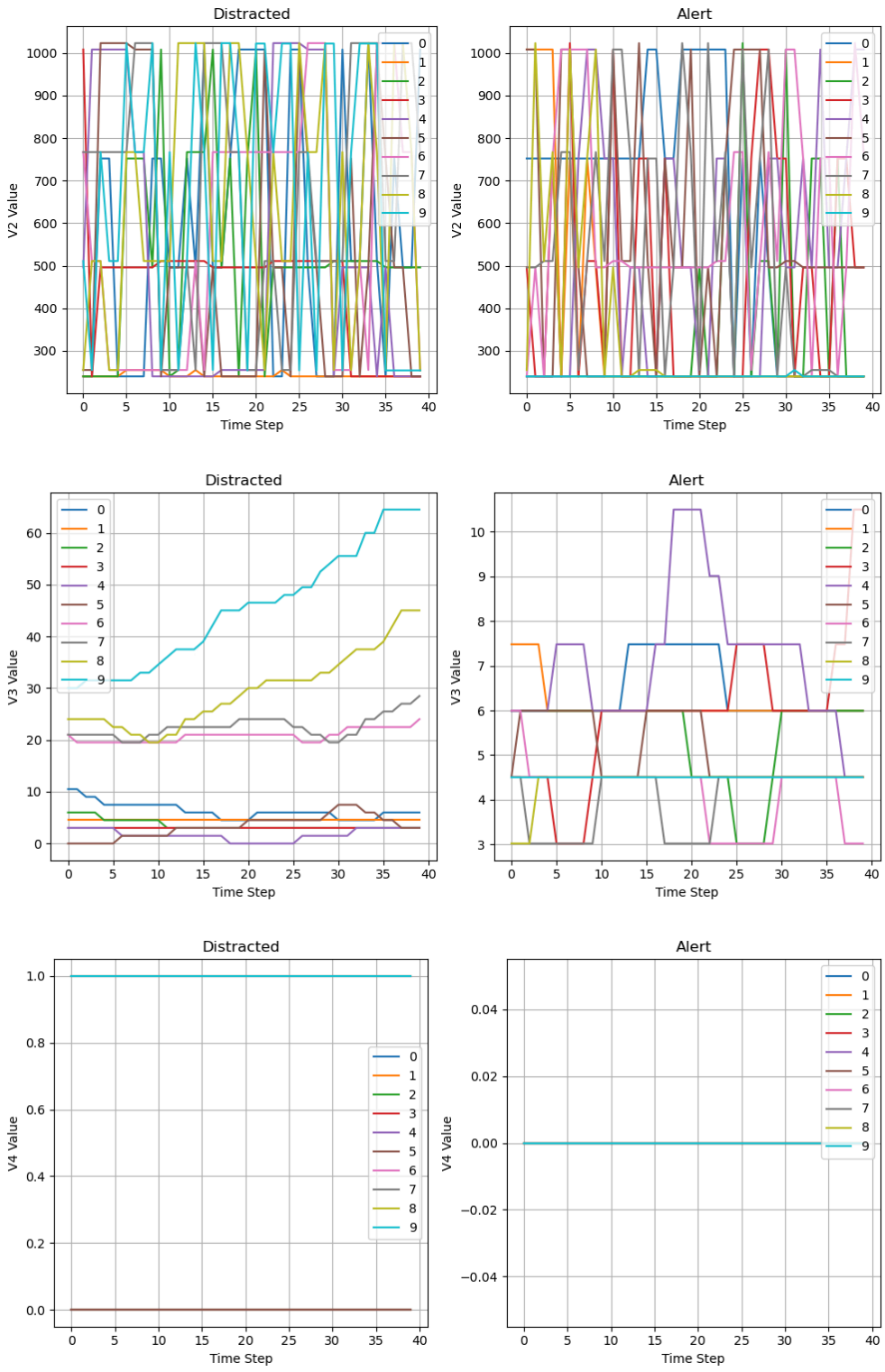


Figure 6. Time series samples of vehicular features for distracted and alert Drivers. Legend entries correspond to the numbered time series. (cont.)

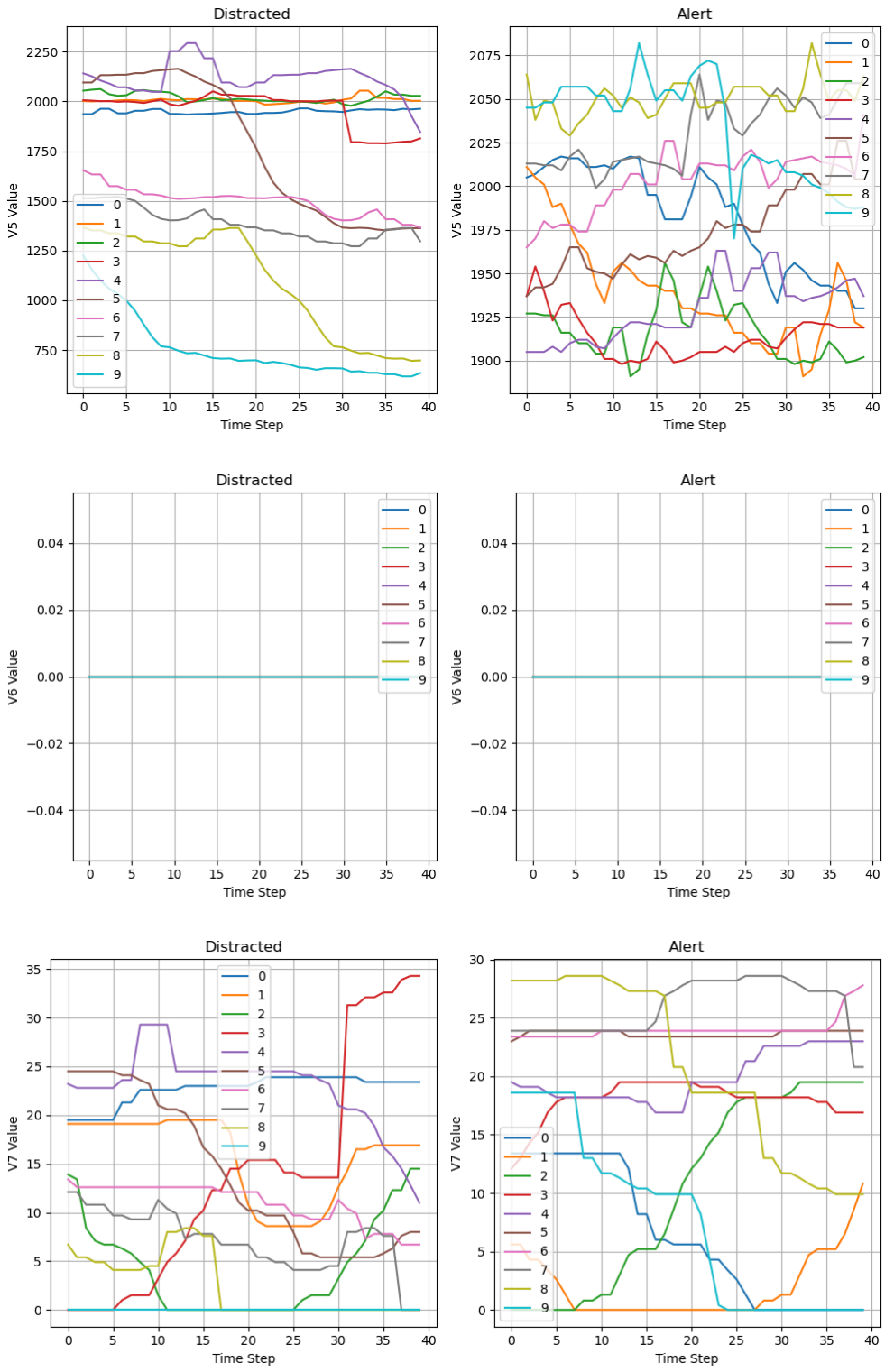


Figure 6. Time series samples of vehicular features for distracted and alert Drivers. Legend entries correspond to the numbered time series. (cont.)

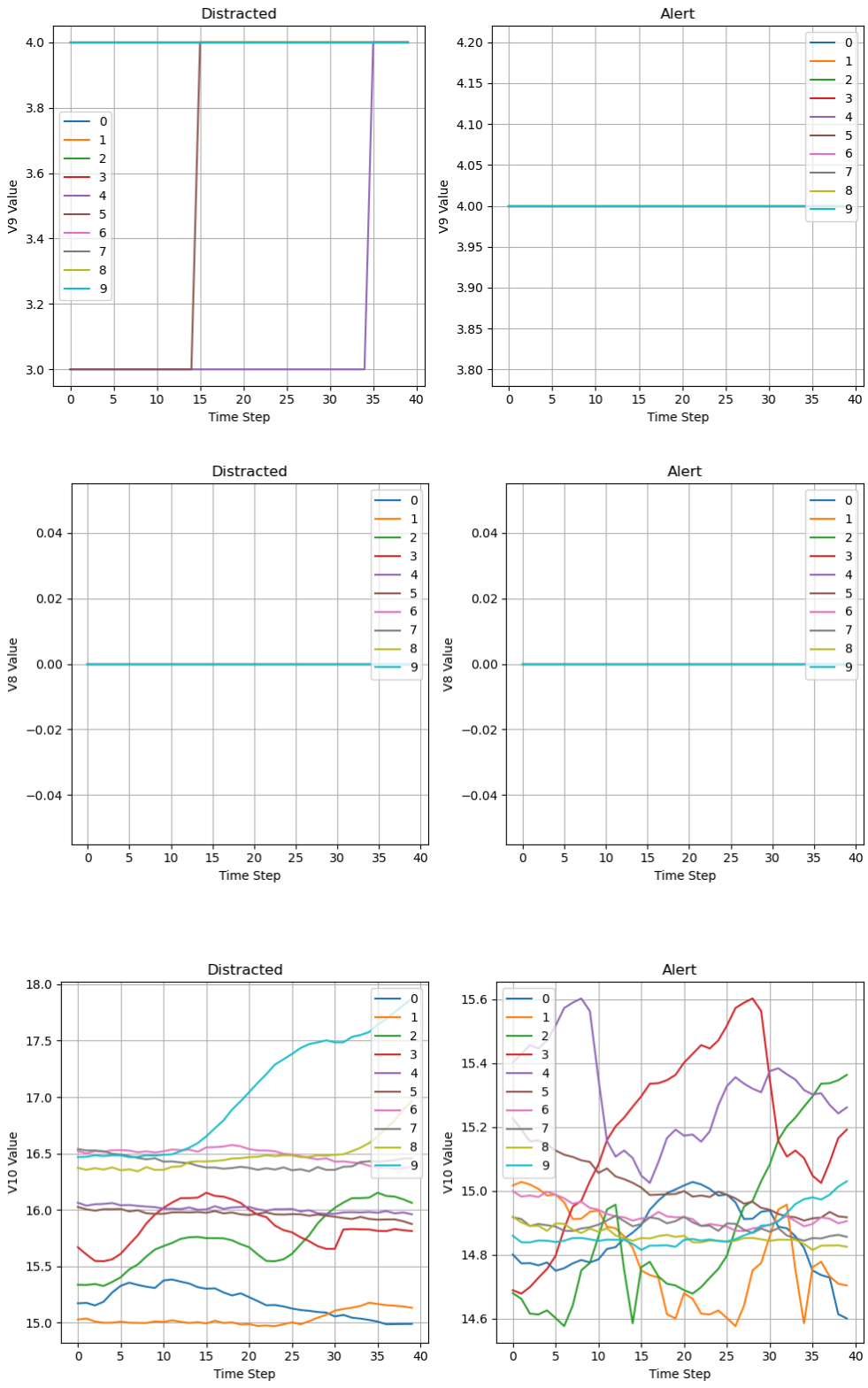


Figure 6. Time series samples of vehicular features for distracted and alert Drivers. Legend entries correspond to the numbered time series.

3.4 Data composition summary

- In the physiological dataset, five variables exhibit *step-wise* patterns, one (P0) shows *continuous* pattern, one (P1) represents a high-frequency electrical signal, and *one* (P7) remains constant at zero.
- The environmental dataset consists of eight *step-wise* variables, two (E2, E8) *binary categorical* variables, and one *continuous* (E10) variable.
- The vehicular dataset contains five *continuous* variables, two (V3, V9) *step-wise* variables, one (V2) *discrete* variable, one *binary* (V4) variable, and two (V6, V8) are always zero.

4. Feature engineering and train data development

4.1 Data reshaping and and removal of zero-only features

The data are reshaped into the format `[num_samples, num_features, num_timesteps],` where num_samples is the number of time series samples, num_features is the number of features and num_timesteps is the number of time steps in each time series. Additionally, features containing only zeros are removed from the data.

4.2 Normalization of data

We first examine the distribution of features to assess their relative scales and identify any outliers. As shown in Figure 7, the feature scales vary widely, ranging from hundredths to thousands. Training data that differ by orders of magnitude can introduce biases; normalizing them to the same scale improves training efficiency and model performance. All features are normalized to the range **[0, 1]** before being used for model training.

4.3 Train and test data

For our analysis, we use a mid-sized dataset of 20K samples for efficient training. This selection automatically excludes data where feature P5 contains outliers. The data are then split 50-50 for training and testing data.

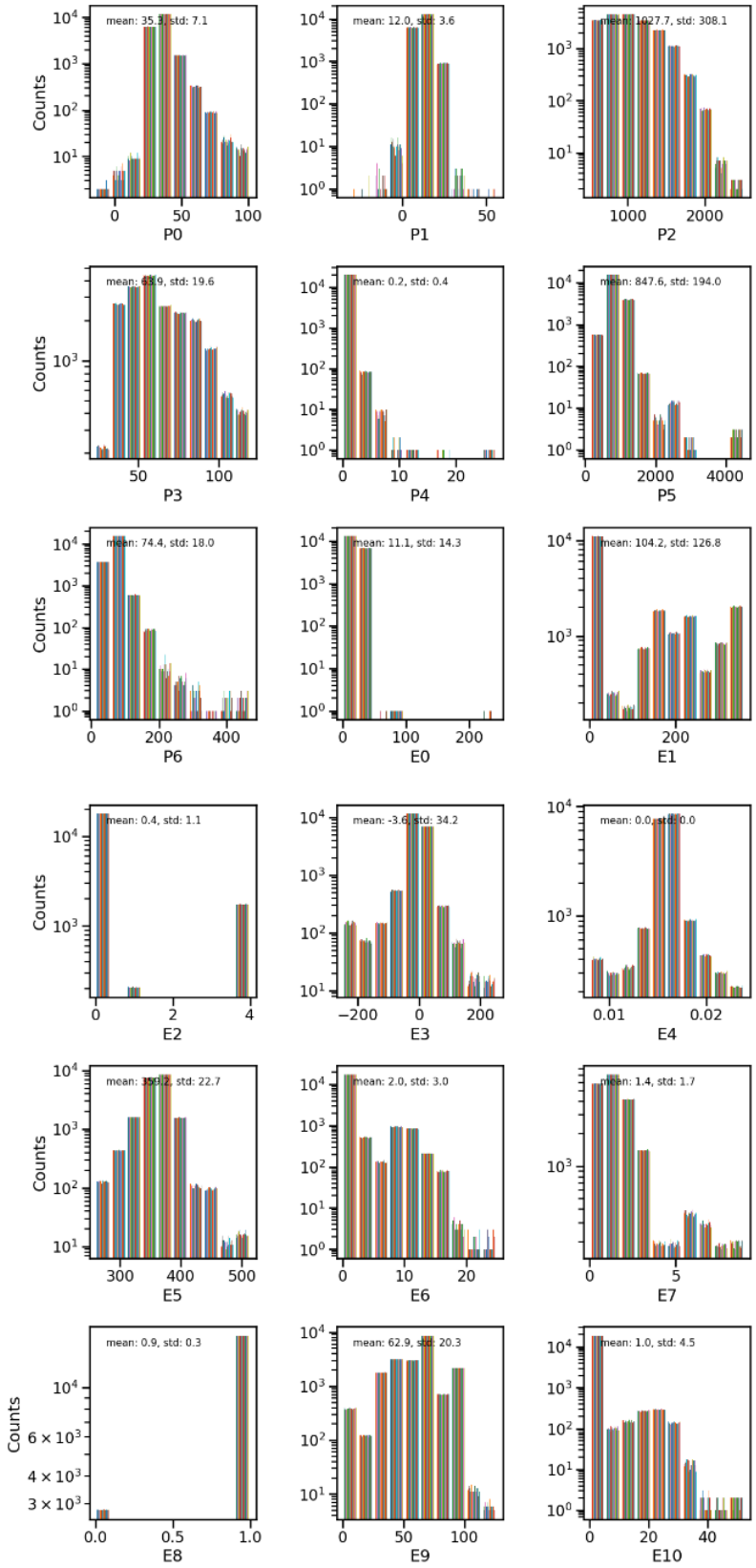


Figure 7. Distribution of features. (cont.)

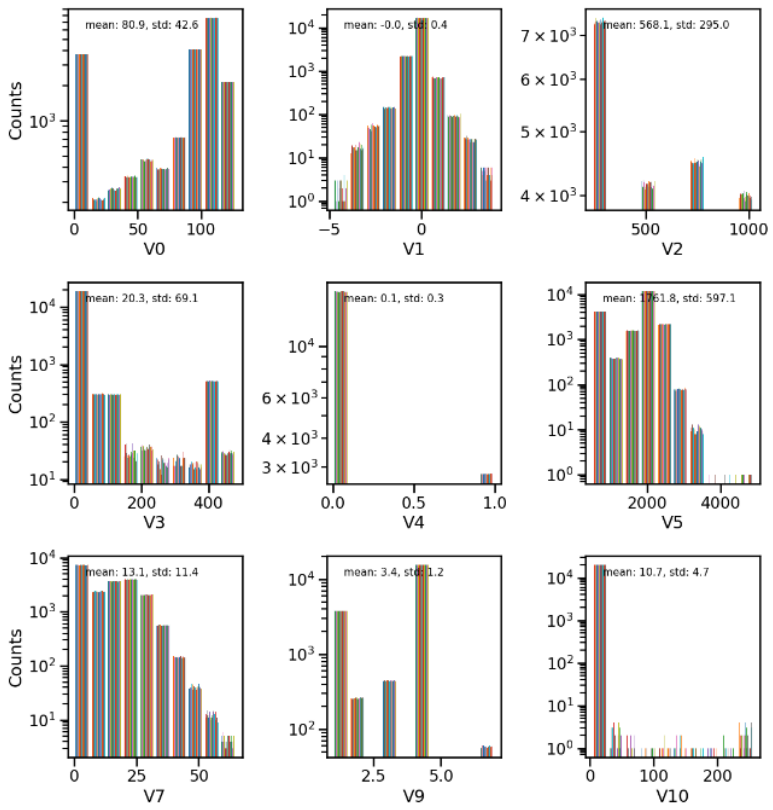


Figure 7. Distribution of features.

5. Time Series Classification

Time series classification (TSC) is a central problem in machine learning, with applications spanning a wide range of domains such as healthcare (e.g., electrocardiogram signal analysis), finance (e.g., stock price prediction), environmental monitoring (e.g., weather and climate forecasting), and industrial systems (e.g., fault detection and predictive maintenance). The goal of TSC is to assign a categorical label to a temporal sequence, where the ordering and temporal dependencies of data points carry critical information about the underlying process.

5.1 History of Time Series Classification

Historically, early research on time series classification was dominated by **distance-based** and **feature-based** methods. One of the most influential approaches is **Dynamic Time Warping (DTW)**, which aligns sequences of varying lengths by minimizing the temporal distortion required to match them. Variants such as 1-Nearest

Neighbor with DTW and Derivative DTW became strong baselines for decades due to their simplicity and robustness. In parallel, **feature-based** techniques emerged, which transformed time series into fixed-length feature vectors using statistical summaries, frequency-domain representations, e.g., Fourier or wavelet transforms, or manually engineered descriptors. These features were then used with traditional classifiers such as **Support Vector Machines (SVMs), Random Forests**, or **k-Nearest Neighbors (k-NN)**. While these classical approaches achieved competitive results on small and well-structured datasets, they heavily relied on domain expertise and handcrafted features. Moreover, they struggled to generalize across datasets with complex temporal dynamics, non-linear dependencies, and high-dimensional multivariate signals. As data volumes and complexity increased, the limitations of manual feature design and distance metrics became increasingly apparent.

5.2 Recent Deep Learning-based Approaches

The advent of *deep learning* marked a paradigm shift in time series analysis. Neural networks offered the ability to *learn hierarchical representations directly from raw data,* eliminating the need for handcrafted feature extraction. Among the most prominent architectures are **Recurrent Neural Networks** (RNNs), **Convolutional Neural Networks** (CNNs), and more recently, **Transformer-based encoders**. Each family of models offers distinct advantages for capturing temporal patterns and dependencies.

Early applications employed **RNNs**, particularly **Long Short-Term Memory** (LSTM) and Gated Recurrent Unit (GRU) networks, to capture sequential dependencies over time. These architectures introduced gating mechanisms to mitigate the vanishing gradient problem and model long-term temporal relationships effectively. However, RNN-based models often suffered from slow training, limited parallelization, and difficulties in capturing very long-range dependencies.

In parallel, **CNNs**, originally designed for computer vision, were adapted for one-dimensional time series data. CNN-based models exploit convolutional filters to detect local temporal patterns and compositional features. Architectures such as **ResNet** introduced residual connections, enabling deeper networks to be trained efficiently and improving representational capacity. These models proved particularly effective in capturing local structures and frequency-related features in both univariate and multivariate time series.

More recently, the **Transformer** architecture, first proposed for natural language processing, has gained traction in time series modeling. Unlike recurrent or convolutional models, Transformers rely on **self-attention mechanisms** to capture *global dependencies*

between all time steps simultaneously. This allows them to model long-term temporal relationships more efficiently and with greater interpretability. The **Transformer encoder**, in particular, has demonstrated strong performance across various time series applications, benefiting from its scalability, parallelism, and ability to dynamically attend to relevant parts of the sequence.

6. Modeling

In this study, we explore and compare several deep learning architectures—LSTM, CNN/ResNet, and Transformer encoders—for multivariate time series classification. Our goal is to evaluate their capacity to model temporal dynamics, handle variable-length sequences, and generalize across diverse time series datasets. By analyzing their strengths and limitations, we aim to provide practical insights into the selection and design of deep architectures for time series classification tasks.

6.1 LSTM time series classifier

LSTM networks were among the first deep architectures to demonstrate strong performance on sequential data.We use a single LSTM layer implemented in PyTorch and vary the number of hidden units to optimize performance.

Figure 8 compares training and validation accuracies across different hidden layer dimensions. No strong dependence on the hidden dimension was observed. We select a hidden dimension of 128 and drop-out rate of 0.3.

6.2 CNN/ResNet time series classifier

ResNet models extract local temporal features and hierarchical representations through one-dimensional convolutions. The inclusion of residual connections improves gradient flow and facilitates the training of deep networks, allowing the model to learn both low- and high-level temporal patterns efficiently.

6.2.1 Network size

There are three ResNet layers in the model architecture, where the output size of each layer doubles that of the previous one. This design increases the model's capacity to compensate for the loss of spatial or temporal resolution. The output size of the layer is set to f_{mult} * num_features. Three values are used to determine the optimal factor.

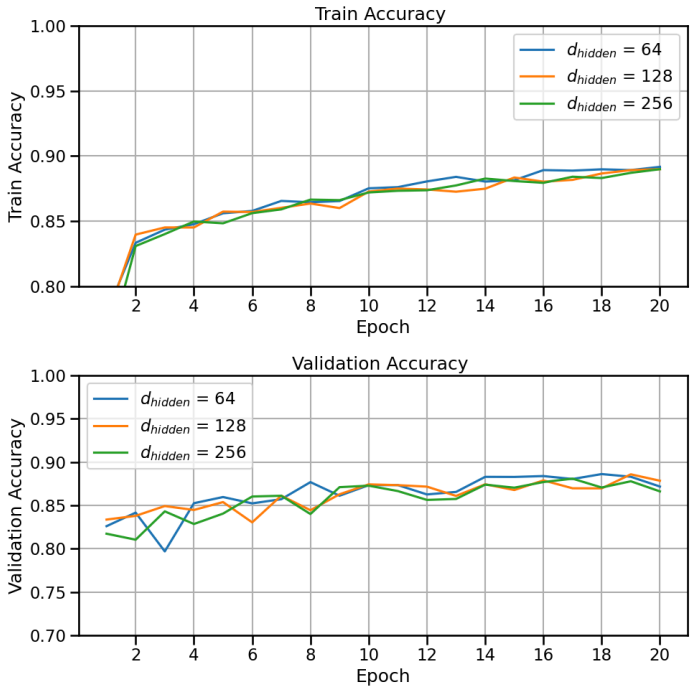


Figure 8. Training and validation accuracies vs. hidden layer dimension for LSTM models.

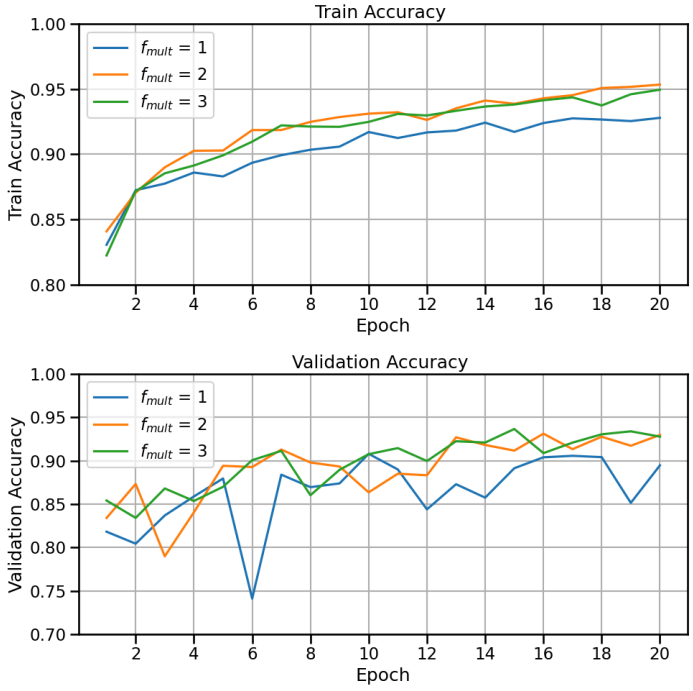


Figure 9. Training and validation accuracies vs. network size factor f_{mult} for ResNet models.

Figure 9 compares training and validation accuracies across different values of multiplication factors. Since the factors of 2 and 3 exhibit comparable learning efficiency, we select 2 for simplicity.

6.2.2 Training hyperparameter tuning - Learning rate and batch size

Learning rate and batch size are critical hyperparameters that significantly influence model convergence and overall performance. A smaller learning rate may lead to more stable convergence but slower training, while a larger batch size can improve training efficiency but may reduce generalization. Figure 10 shows train and validation accuracies across different learning rates and batch sizes. In conclusion, a batch size of 64 and a learning rate of 0.001- 0.01, combined with an f_{mult} value of 2, were found to yield optimal performance.

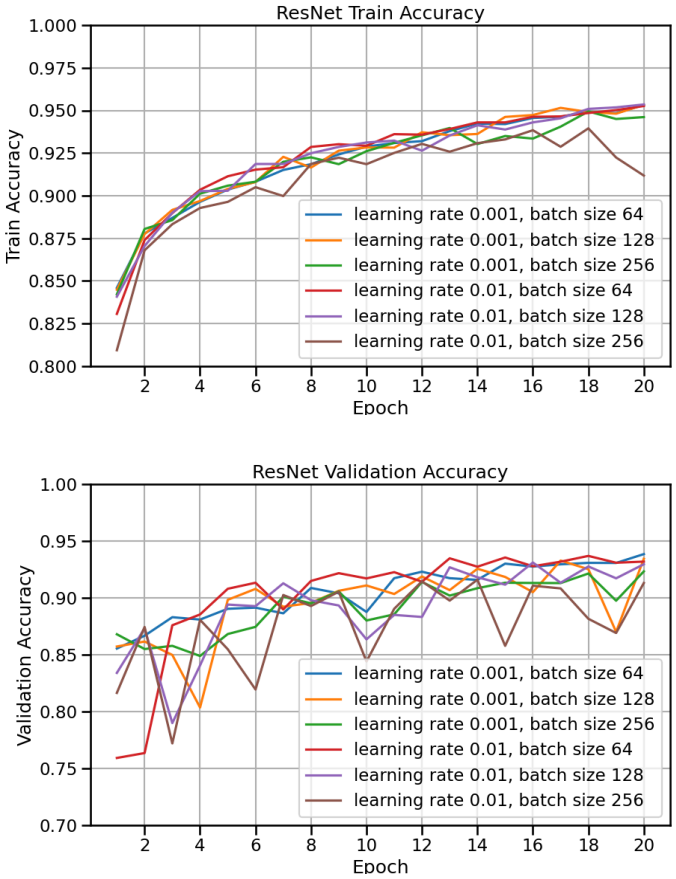


Figure 10 Comparison of training and validation accuracies obtained with varying learning rates and batch sizes for ResNet models.

6.3 Transformer time series classifier

A Transformer encoder classifier leverages self-attention mechanisms to capture complex temporal and contextual relationships within input sequences. It encodes the data into a latent representation through stacked encoder layers, followed by a classification head that predicts the target class based on the aggregated embeddings. In this work, a single Transformer encoder layer is utilized, consisting of embedding, multi-head self-attention, and position-wise feed-forward sublayers, integrated via residual connections and layer normalization.

6.3.1 Training hyperparameter tuning

The following batch sizes and learning rates are explored.

- batch_sizes = [16,32], learning_rates = [0.0001, 0.001]

Figure 11 shows train and validation accuracies across different learning rates and batch sizes for the Transformer encoder model. We choose a learning rate of 0.001 and a batch size of 32.

6.3.2 Embedding

Two alternative embedding strategies were investigated. In the first approach, both the full feature embeddings and temporal embeddings were added to the linearly projected feature representations. While this method provided a comprehensive joint embedding space, it was computationally expensive and inefficient for the given dataset. In contrast, a more effective approach involved applying a linear projection to the feature vectors and subsequently incorporating only the temporal embedding components, resulting in a more efficient and scalable representation. Several embedding dimensions from 64 to 256 were explored to identify the most effective configuration. We select 128 for optimal performance based on results shown in Figure 12.

6.3.3 Multi-headed self-attention mechanism

The self-attention framework enables the model to weigh the relative importance of different time steps dynamically, making it well suited for complex multivariate time series with intricate interdependencies between features. The number of attention heads determines how the model distributes its representational capacity across different subspaces of the embedding. Consequently, the embedding dimension must be a multiple of the number of heads to allow an even partitioning of the embedding vector across all attention heads. We select N_{head} of 4 based on the results shown in Figure 13.

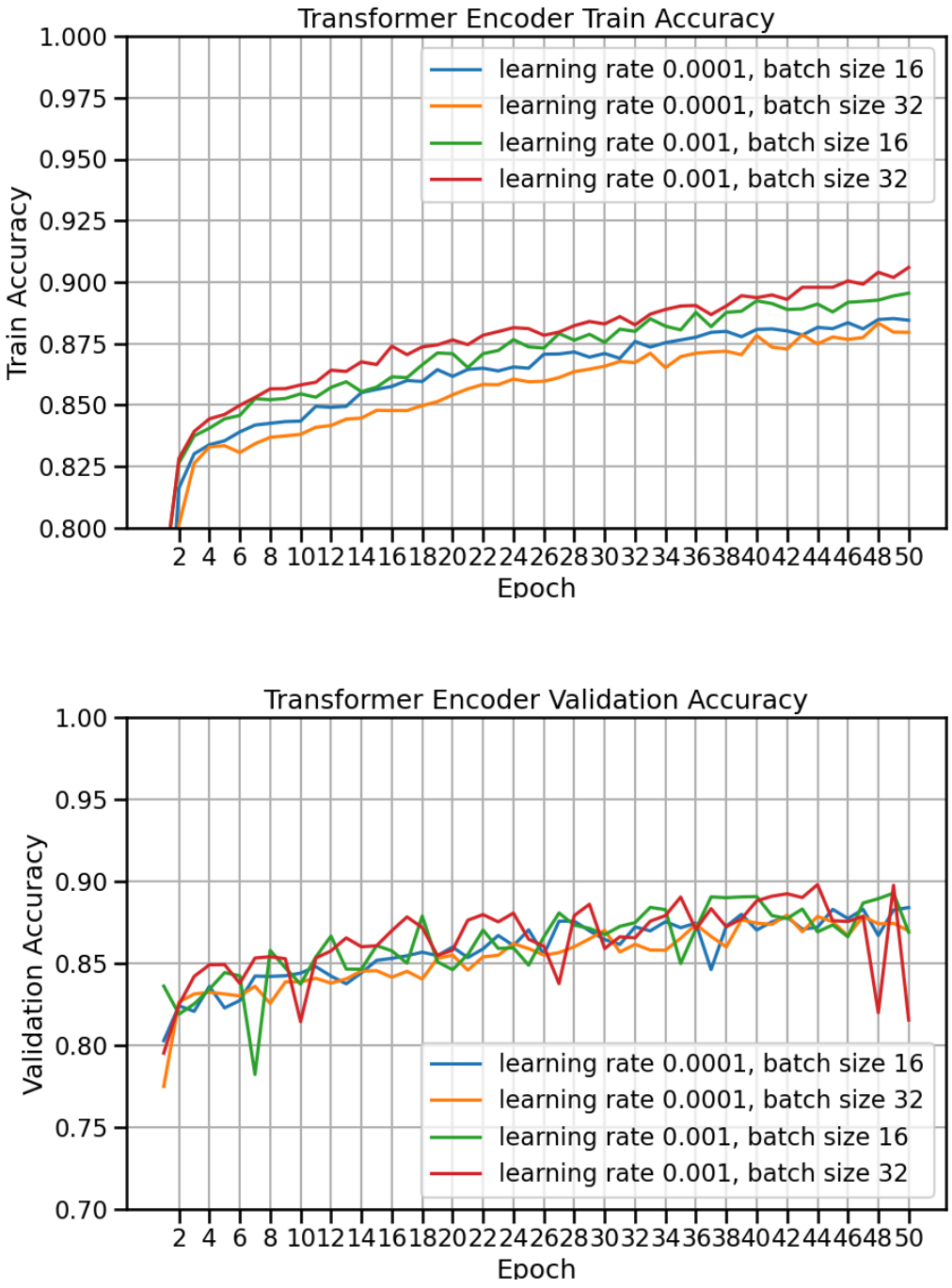


Figure 11 Comparison of training and validation accuracies obtained with varying learning rates and batch sizes for Transformer encoder models.

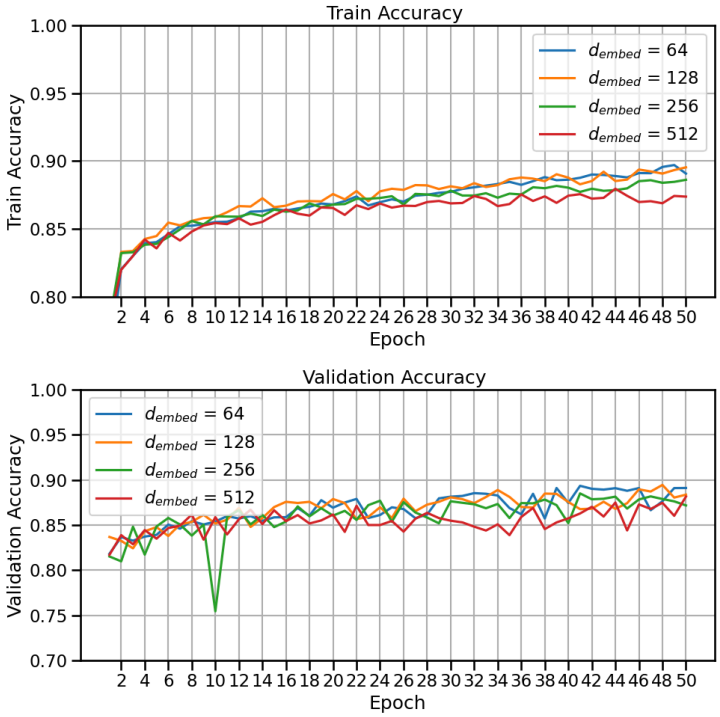


Figure 12 Training and validation accuracies vs. d_{embed} in Transformer models.

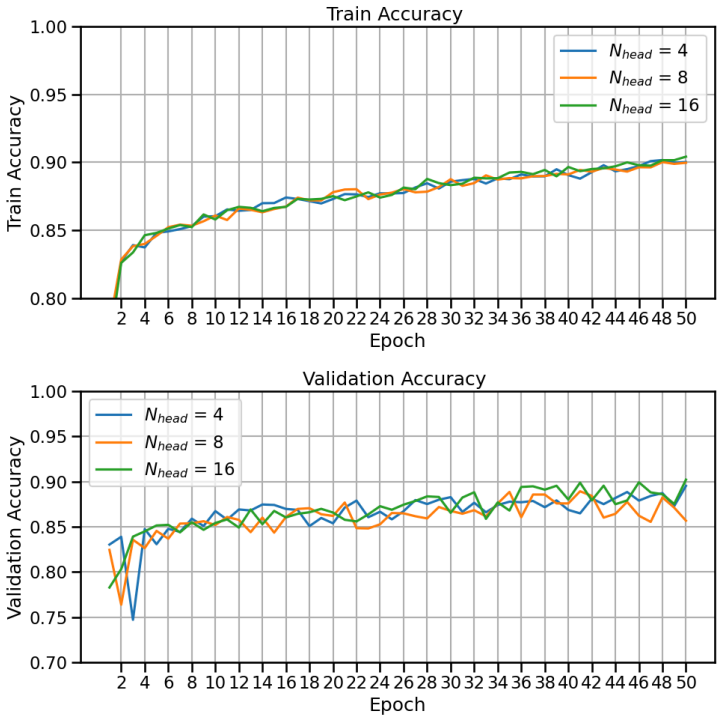


Figure 13 Training and validation accuracies vs. N_{head} in Transformer models.

6.3.4 Fee-forward and residual connections

While Transformer encoders can consist of multiple stacked layers, our implementation utilizes a single encoder layer for efficiency and simplicity. In each Transformer encoder block, after the multi-head self-attention, there's a position-wise feed-forward network (FFN).

In almost all Transformer architectures $d_{ff} = 4 * d_{embed}$. The self-attention layer mixes information across time steps. The feed-forward network expands this information within each time step, i.e., it gives the model non-linear capacity to represent more complex transformations. A larger d_{ff} increases model capacity, but also computation and memory. We evaluate the model's performance using the following parameter values {4,6,8}. We select 6 based on the results shown in Figure 14.

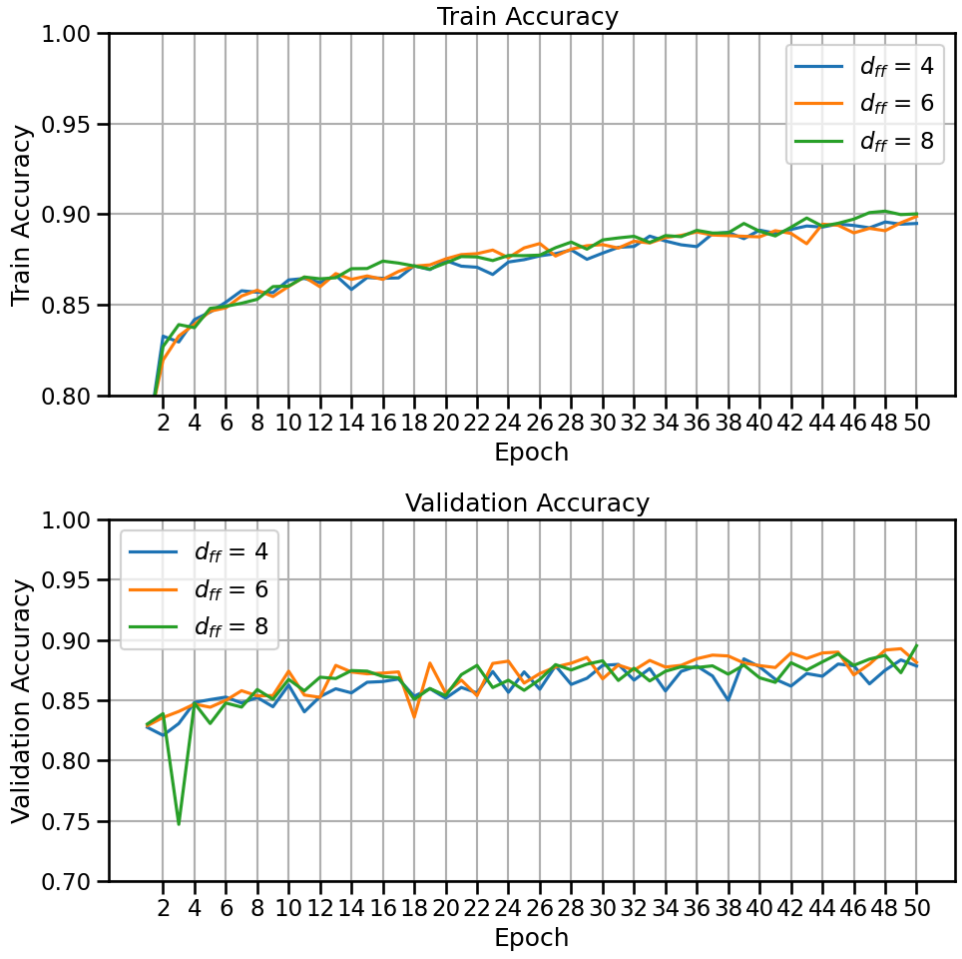


Figure 14 Training and validation accuracies vs. d_{ff} in Transformer encoder models.

6.3.5 Drop-out rate

Dropout is applied to prevent overfitting by randomly deactivating a subset of neurons during training, which encourages the model to learn more robust and generalizable features. The dropout rate appears to be the most influential hyperparameter affecting training quality. We also observe in Figure 15 that the encoder model continues to improve in performance up to 50 epochs, suggesting that it requires a larger number of training epochs to reach optimal performance.

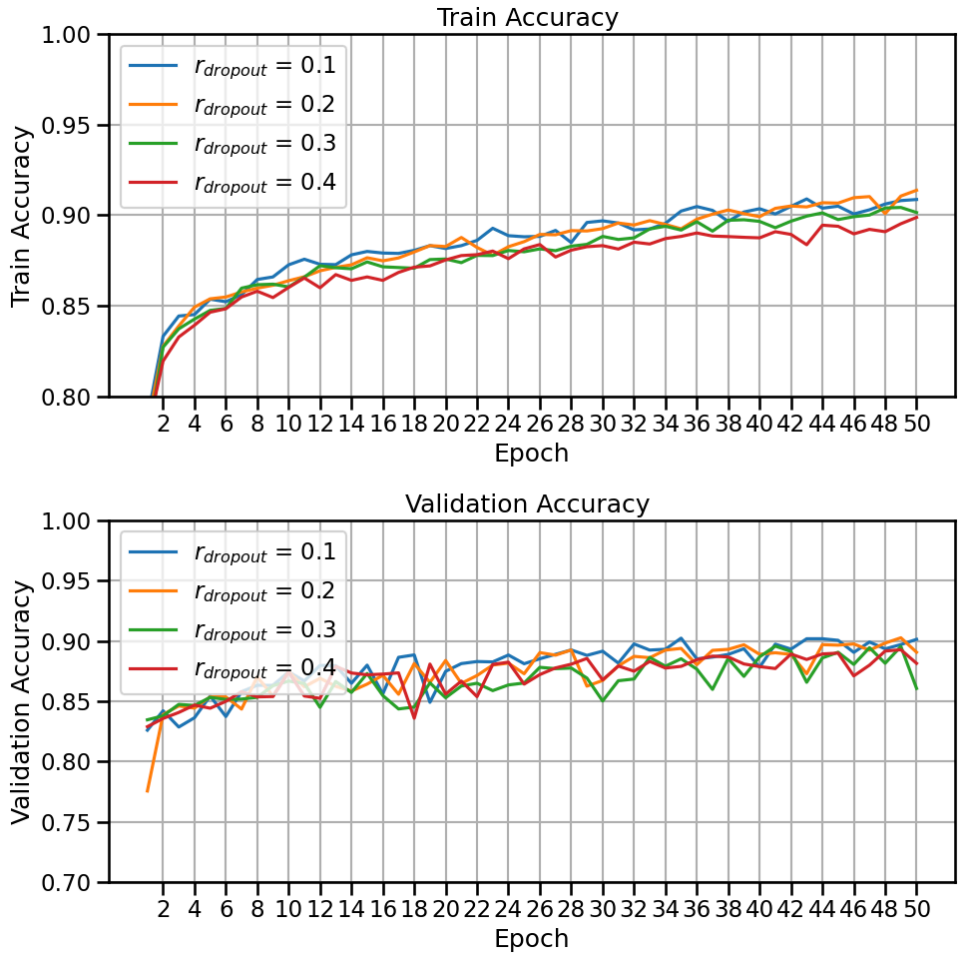


Figure 15 Training and validation accuracies vs. $r_{dropout}$ in Transformer encoder models.

6.3.6 Feature importance

We compute feature importance scores using probability-based gradients, averaged over all time steps. The resulting saliency scores are then normalized relative to the maximum value. The normalized feature importance is presented in Figure 16.
The most important feature is **V10**, which showed clear differences between distracted and alert drivers in Figure 6. The next few most important features are physiological, **P5** and **P6**, which is reasonable since distraction reflects both mental and physical states.

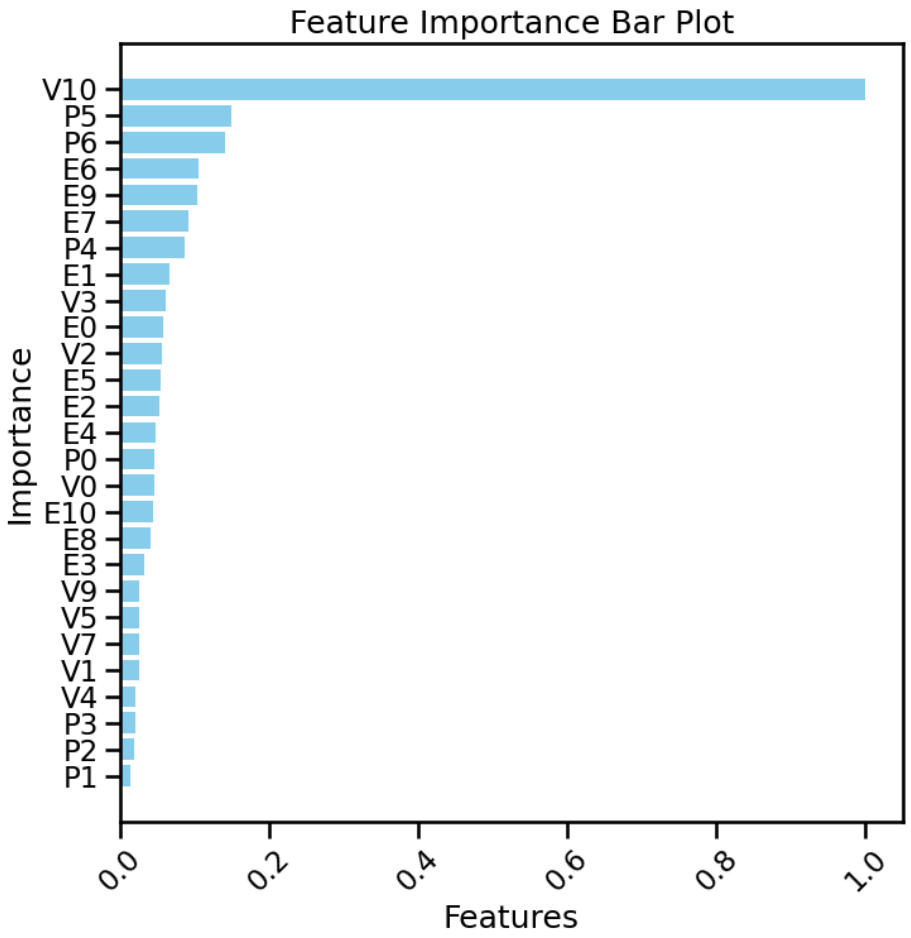


Figure 16 Feature importance scores, measured using saliency and normalized with respect to the maximum value.

7. Results

7.1 Evaluation metrics

We evaluate the models using Recall, Precision, and the AUC–ROC curve, which provide insights into model performance on our imbalanced dataset. We also save the validation data associated with the saved models for later result visualization.

7.2 Performance comparison of Models

Figures 17–19 present the Precision, Recall, and AUC metrics for the three models, along with their corresponding AUC-ROC curves. Figure 20 provides a comparison of the AUC performance across all three models in a single panel.

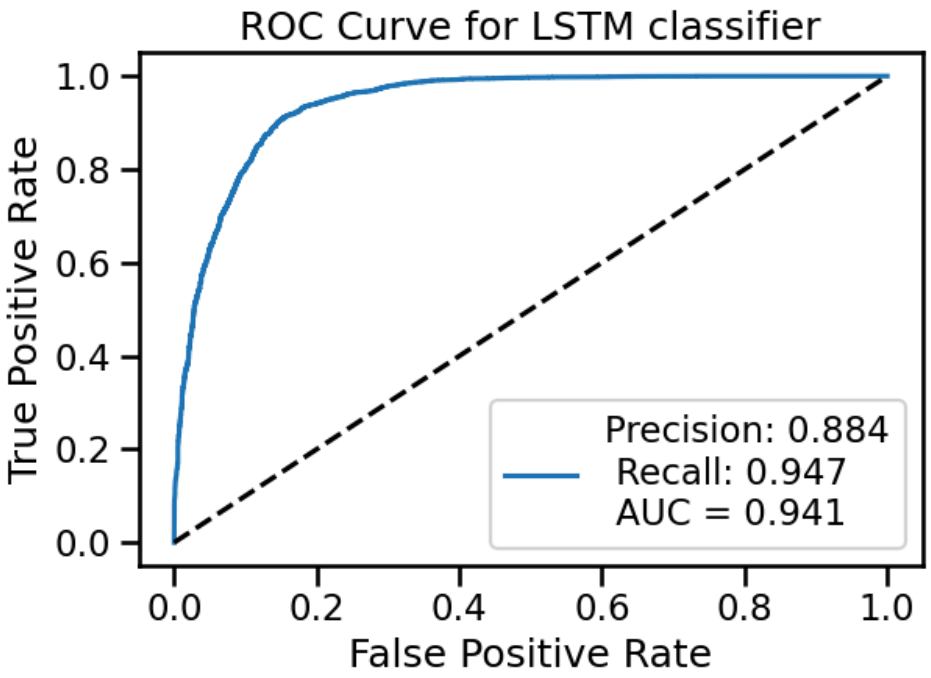


Figure 17 Performance metrics of the RNN/LSTM model.

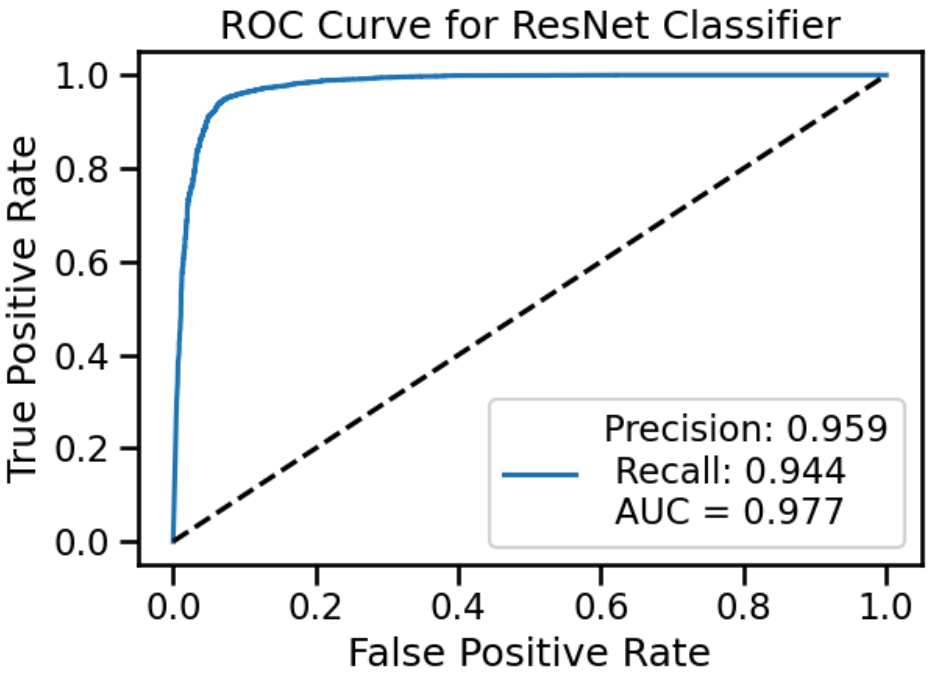


Figure 18 Performance metrics of the CNN/ResNet model.

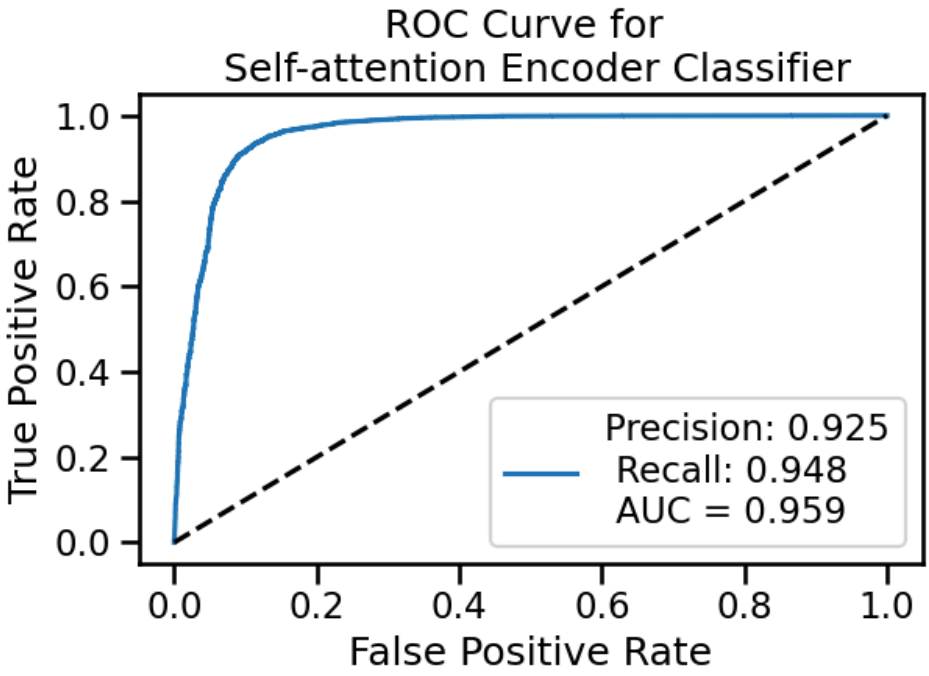


Figure 19 Performance metrics of the Transformer encoder model.

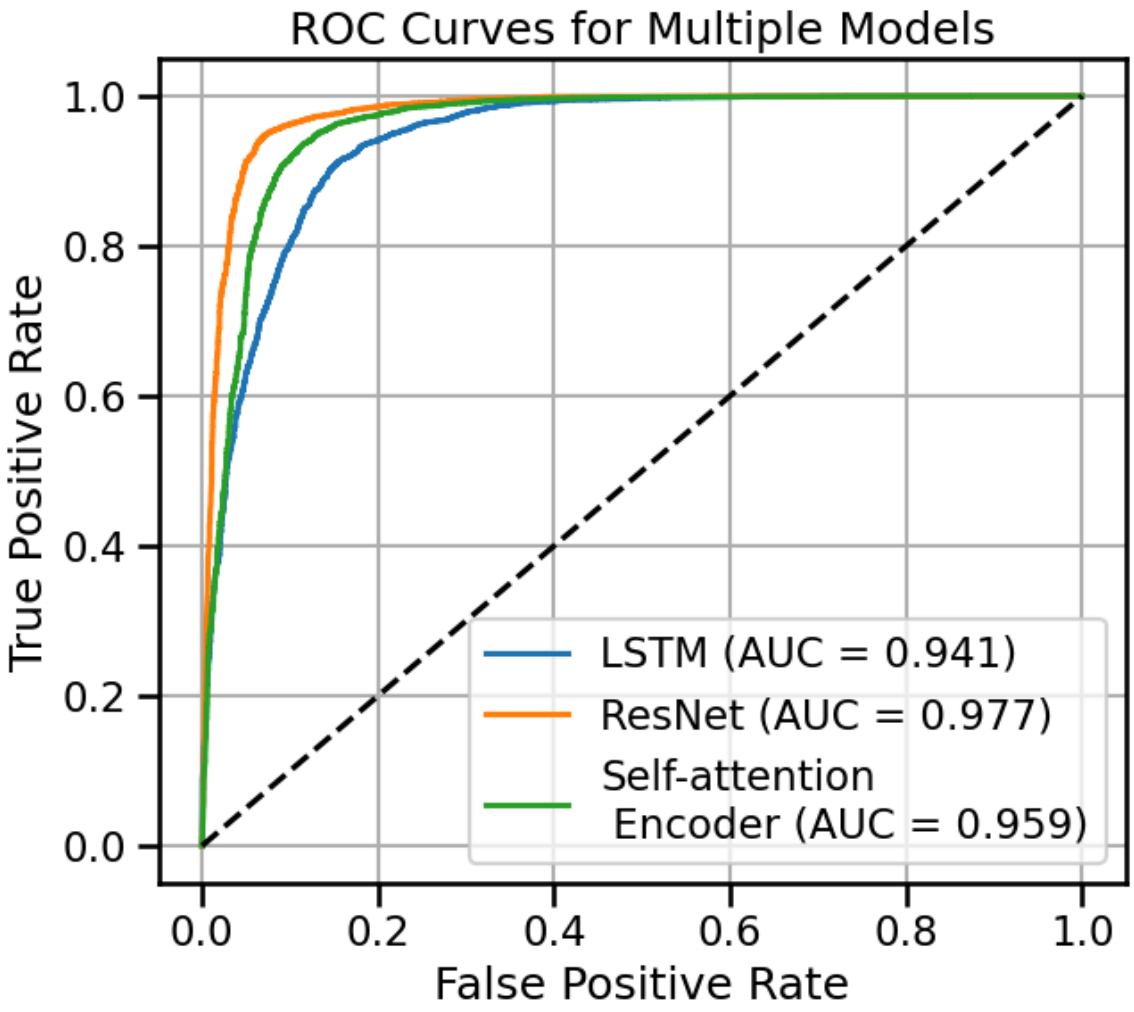


Figure 20 A comparison of the AUC metric across the three models, presented in a single panel for direct performance evaluation.

By ***Recall***, a safety-critical metric since missing a distracted driver is costly, the **self-attention encoder** performs best, while other models trail closely. In contrast, the **ResNe**t model achieves the highest ***Precision***, helping to reduce over-alerting, and shows the best overall performance by ***AUC***. Both models' strengths can be explained by their architectures: **ResNe**t excels at hierarchical spatial feature learning, enabling it to capture detailed spatial patterns, whereas the **self-attention encoder** effectively captures

long-range dependencies and contextual relationships. Together, these characteristics account for their superior performance in detecting distracted versus alert drivers.

8. Summary and outlook

This study investigates distracted driver detection using multimodal sensor data, including physiological, environmental, and vehicular signals, leveraging three deep learning architectures: LSTM, ResNet, and Transformer encoders. Physiological data (e.g., heart rate, eye movement), environmental data (e.g., road conditions, lighting), and vehicular data (e.g., steering, pedal usage) were integrated to capture comprehensive driver behavior. The LSTM model exploits temporal dependencies in the sequential data, ResNet captures hierarchical feature representations, and the Transformer encoder leverages self-attention mechanisms to model long-range dependencies effectively. Performance was evaluated using Precision, Recall, and AUC metrics, with AUC-ROC curves illustrating comparative model effectiveness. Results demonstrate that multimodal deep learning can significantly enhance real-time distracted driver detection by capturing complementary behavioral cues from multiple data sources.

Although this study focuses exclusively on time series data, future research could incorporate non-temporal features, such as demographic information, driver history, or contextual environmental factors, to further improve detection accuracy and robustness in real-world scenarios. Overall, this work demonstrates the potential of deep learning for real-time driver monitoring using time series sensor data.

Reference

[1] Mahmoud Abou-Nasr. (2011). Stay Alert! The Ford Challenge. https://kaggle.com/competitions/stayalert. Kaggle.
[2] RNN/LSTM
[3] CNN/ResNet
[4] Transformer