The Pennsylvania State University
The Graduate School

PREDICTING DRIVER’S TAKEOVER DECISIONS IN A CONDITIONALLY AUTOMATED VEHICLE WITH GAZE BASED DEEP LEARNING ARCHITECTURES

A Thesis in
Industrial Engineering and Operations Research

© 2023 Shantanu Gupta

Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science

May 2023
The thesis of Shantanu Gupta was reviewed and approved by the following:

Yiqi Zhang  
Assistant Professor of Industrial Engineering  
Thesis Co-Advisor

Vittalas V Prabhu  
Professor of Industrial Engineering  
Thesis Co-Advisor

Steven Landry  
Professor of Industrial Engineering  
Head of the Department of Industrial and Manufacturing Engineering
ABSTRACT

This study accounts for the modeling of three different deep learning (DL) network architectures to predict drivers’ takeover behavior in conditionally automated vehicles (AV) exiting a freeway, based on the vehicle’s surroundings and the takeover requests. Previous research has modeled a driver’s takeover time in emergency scenarios that require a brisk response. However, these models may not be useful for scheduled, non-time-critical takeovers as drivers may take longer with varying strategies to take over the vehicle when there is no time pressure. A machine learning (ML) based model which can predict these takeover times, is missing at the current time. Hence, three DL architectures, convolutional neural network (CNN), echo state network (ESN), and inception time network (ITN) were used to train several models on two types of data i.e., the driving data and the driver’s gaze data, combined. These models consider drivers’ responses to take-over requests (ToR) and their situational awareness to predict when they will take over the vehicle. The best model to predict a driver’s takeover behavior employed the CNN architecture and had an F1 score of 0.993. This model can successfully predict drivers’ response time for scheduled, non-time-critical freeway exiting takeovers in conditionally AVs and find its application in human-machine interface design with respect to ToR lead time for enhancing safe freeway exiting takeovers in conditionally AVs. These can also be integrated with driving models of the vehicle to improve freeway exiting takeover performance by optimizing the ToR lead times according to the behavior of individual drivers.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................ vi
LIST OF TABLES .......................................................................................................... vii
LIST OF EQUATIONS ................................................................................................... viii
ACKNOWLEDGEMENTS .............................................................................................. ix

Chapter 1 Introduction ............................................................................................... 1
  Literature Review ...................................................................................................... 2
  Empirical Studies ..................................................................................................... 2
  Use of Physiological Features ............................................................................... 3
  Existing Computational Models of Driver Takeover Performance .................. 4
  Research Objectives ............................................................................................... 7

Chapter 2 Datasets ...................................................................................................... 8
  Participants ............................................................................................................... 8
  Simulator's Driving Data ....................................................................................... 8
  Eye-Tracking Data ............................................................................................... 9
  Data Pre-Processing ............................................................................................ 10

Chapter 3 Neural Networks and Deep Learning ...................................................... 13
  Introduction to Neural Networks ......................................................................... 13
  Perceptron (Neuron) ............................................................................................. 14
  Multi-Layer Perceptron ....................................................................................... 16
  Classification with Multi-Layer Perceptron ....................................................... 17
  Deep Learning for Time Series Classification ............................................... 19
  Model Evaluation .................................................................................................. 21

Chapter 4 Convolutional Neural Network (CNN) .................................................... 23
  Implementation ....................................................................................................... 25
  Type-CNN Models ................................................................................................ 26
  Results .................................................................................................................... 27
  DDACCZ-CNN Models ........................................................................................ 29
  Results .................................................................................................................... 29
  K-Fold-CNN Model .............................................................................................. 31
  Results .................................................................................................................... 32

Chapter 5 Echo State Network (ESN) ....................................................................... 33
  Recurrent Neural Network (RNN) ....................................................................... 33
  Echo State Network Architecture ....................................................................... 34
Reservoir ................................................................................................................... 35
Dimension Reduction ............................................................................................... 36
Implementation ......................................................................................................... 37
Type CNN Models .................................................................................................... 37
Results ....................................................................................................................... 37
DDGZ ESN Models .................................................................................................... 39
Results ....................................................................................................................... 39
K-Fold ESN Model .................................................................................................... 40
Results ....................................................................................................................... 40

Chapter 6 Inception Time Networks (ITN)............................................................... 42
Inception Module ...................................................................................................... 42
Receptive Field ......................................................................................................... 44
Inception Time Architecture ..................................................................................... 45
Implementation ......................................................................................................... 45
Type ITN Models ...................................................................................................... 46
Results ....................................................................................................................... 46
DDACCGYGZ ESN Models ...................................................................................... 47
Results ....................................................................................................................... 47

Chapter 7 Conclusion ............................................................................................. 49
Importance of Window Size ..................................................................................... 50
Limitation and Future Work .................................................................................... 50

References ............................................................................................................... 51
LIST OF FIGURES

Figure 3-1: Architecture of a Perceptron. ................................................................. 14
Figure 3-2: Activation Functions. ................................................................................. 15
Figure 3-3: Architecture of Multi-Layer Perceptron......................................................... 16
Figure 3-4: One-hot labeled vector. ................................................................................ 17
Figure 3-5: Backpropagation Procedure. ......................................................................... 18
Figure 3-6: Deep Learning Framework. .......................................................................... 20
Figure 3-7: Confusion Matrix. ......................................................................................... 21
Figure 4-1: Convolutional Neural Network architecture. ................................................. 23
Figure 4-2: Employed CNN architecture. ........................................................................ 26
Figure 4-3: Training’s and Validation’s accuracy and loss plots. .................................... 28
Figure 4-4: Type-CNN model metrics graph. ................................................................. 29
Figure 4-5: DDACCGZ-CNN model metrics graph. ......................................................... 30
Figure 4-6: K-fold Cross Validation example. ................................................................. 31
Figure 5-1: RNN Architecture. ....................................................................................... 33
Figure 5-2: ESN Architecture. ......................................................................................... 34
Figure 5-3: Reservoir. .................................................................................................... 35
Figure 5-4: Type-ESN models’ metrics plot. ................................................................. 38
Figure 5-5: DDGZ-ESN models’ metrics plot. ............................................................... 40
Figure 6-1: A typical Inception Time Architecture. ......................................................... 42
Figure 6-2: Example of an Inception Module. ............................................................... 43
Figure 6-3: Receptive Field. ........................................................................................ 44
Figure 6-4: DDACCGYGZ-ITN models’ metrics plot. .................................................... 29
LIST OF TABLES

Table 2-1: Features of the simulator's driving data (DD). .......................................................... 9
Table 2-2: Features of the eye-tracking data. ................................................................................. 10
Table 2-3: Unchanging features in each time series. ................................................................. 11
Table 4-1: Dataset permutations. .................................................................................................. 26
Table 4-2: Type-CNN model evaluation metrics. ......................................................................... 28
Table 4-3: DDACCGZ-CNN model evaluation metrics. ............................................................... 30
Table 5-1: Type ESN models’ evaluation metrics. ................................................................. 38
Table 5-2: DDGZ ESN models’ evaluation metrics. ................................................................. 39
Table 6-1: Type ITN model’s evaluation metrics. ................................................................. 47
Table 6-2: DDACCGYGZ ESN models’ evaluation metrics. ................................................ 48
LIST OF EQUATIONS

Equation 3-1: Mathematical operations in a perceptron. ................................................. 15
Equation 3-2: Formulas for model evaluation metrics. ....................................................... 21
Equation 4-1: Convolution function for two matrices. ..................................................... 25
Equation 5-1: Activation by the reservoir. ....................................................................... 35
Equation 6-1: Total Receptive Field. .............................................................................. 45
ACKNOWLEDGEMENTS

I would like to acknowledge the all-around support and guidance of Dr. Yiqi Zhang throughout the duration of this research. I would also like to mention Dr. Fenglong Ma of the Department of Information Sciences and Technology for his keen guidance and knowledge about the different deep learning algorithms that helped me complete this research project and the Ph.D. candidate under Dr. Zhang, Tan Xiaomei who helped me understand the experimental setup and its data that has been used in this study.

The whole research was supported by the National Science Foundation (NSF) under Grant #1850002

Disclaimer: Any opinions, findings, and conclusions or recommendations expressed in this literature are those of the author and do not necessarily reflect the views of the NSF.
Chapter 1

Introduction

As the development of automated vehicles becomes an inevitable trend, researchers have contributed much effort to investigate the way to establish a framework that provides a reliable and comfortable experience of driving on automated vehicles. The Society of Automotive Engineers (SAE) defines the six levels of automated vehicles (SAE International, 2018). Level 3 of automation in driving, known as conditionally automated vehicles (AVs) is under development with the expectation of realizing attention-free driving within limited areas. One challenge to tackle while maintaining attention-free driving in conditionally AVs is takeover transition (Ayoub et al., 2019, Zhou et al., 2020b). Although in conditionally AVs, the driver’s attention towards the vehicle’s surroundings is not required and they can engage in non-driving-related tasks (NDRTs) (SAE International, 2018), they will have to take over the vehicle whenever the driving algorithm reaches its operational limit.

Operational Design Domain (ODD) is a common method for defining the operational requirements and capability limits of automated driving systems (ADSs). It is a description of the circumstances (such as road characteristics, speed ranges, and weather) under which an ADS is designed to function effectively (NHTSA, 2017). For instance, the Highway Pilot feature allows an automobile to drive itself on specifically approved highways, but it necessitates that the driver regains control of the vehicle before exiting (Hungar et al., 2017). When the ADSs initiate a takeover request (ToR), drivers should be given the necessary information, such as the reason for the takeover, the amount of time or distance left before exiting, and any nearby vehicles that require attention, to help them make a successful exit and maintain roadway safety. The information provided by human-machine interfaces (HMIs) helps drivers prepare for driving maneuvers,
complete a takeover task safely, and understand and anticipate the situation quickly (situation awareness (SA); Endsley, 1988).

**Literature Review**

**Empirical Studies**

Previous research has demonstrated that automated driving systems allow drivers to engage in non-driving-related tasks (NDRTs). But conditionally AV drivers are required to gain situational awareness (SA) upon receiving a ToR to resume control of the vehicle when the system deems itself incompetent to handle the driving situation. Endsley (1988) defined situation awareness as a person's perception of the elements of their surroundings (Level 1 SA), understanding of their meaning (Level 2 SA), and projection of their state in the future (Level 3 SA). Gold et al. (2016), Eriksson and Stanton (2017), and Petersen et al. (2019) also found that SA encouraged and improved takeover performance while engaging in NDRTs. Wan and Wu (2018) investigated how takeovers were affected by lead time and a variety of realistic NDRTs. Studies by Gold et al. (2016) and Li et al. (2018) manifested that the vehicle’s surrounding environment also plays a crucial role in determining the driver’s takeover performance.

It has been discovered in studies like Eriksson and Stanton (2017), Gold et al. (2016), and Wan and Wu (2018), that the types of NDRTs affect takeover performance. According to earlier research, those who engaged in NDRTs exhibited longer takeover reaction times, more crashes in heavy traffic, and shorter minimum time to collision (TTC) than those who did not. Additionally, the impact of the NDRT modality on takeover effectiveness was investigated. According to Radlmayr et al. (2014) and Wandtner et al. (2018), using handheld devices for a visual task reduced takeover performance and increased the collision rate, whereas using them for an auditory job
resulted in performance that was comparable to a baseline without any task. In their studies of the effects of manual and cognitive task loads, Zeeb et al. (2016) and Zeeb et al. (2017) discovered that a high level of manual task load slowed reaction times and reduced takeover quality, whereas the impact of cognitive task load on takeover ability varied depending on the type of driver intervention. While braking maneuvers were unaffected, a high degree of cognitive load increased reaction time and reduced takeover quality in steering maneuvers.

These studies, which mostly concentrated on the influence of variables on takeover performance, offered insightful but primarily relational information. For example, a high traffic density increased the takeover time. However, because there are numerous relevant elements that could interact with one another, understanding the correlations between specific parameters and takeover performance is insufficient to effectively forecast a driver's takeover success in the actual world. There is a need for computational models that can forecast in real-time the drivers’ takeover performance under various takeover scenarios, hence the need to develop machine learning models.

Use of Physiological Features

With the development of wearable technology, it is now possible to gather time-series signals from drivers, such as their gaze patterns and eye movements, to get a trustworthy representation of their SA when they are driving conditionally automated vehicles. Roesener et al. (2016), gave a scenario-based assessment approach for classifying human-driving behavior using time-series data. Gold et al. (2016) discovered horizontal gaze dispersion was the most sensitive indicator of drivers' cognitive load.

Solovey et al. (2014) reported classification accuracies of up to 90% for identifying high levels of cognitive strain, and they demonstrate that the addition of physiological data improves classification accuracy compared to vehicle sensor data assessed solely. Wang et al. (2014)
demonstrate how challenging eye-tracking data sets from practical, ecologically sound contexts, such as on-road driving, may be analyzed successfully with maximum sensitivity and little computational load to yield a reliable indicator of a driver's overall attentional allocation. Gold et al. (2016) discovered that in NDRTs during conditionally automated driving, horizontal gaze dispersion was the most sensitive indicator of drivers' cognitive load. Zeeb et al. (2016) analyzed driver’s gaze data to measure the driver’s behavioral response in case of a takeover. Luo et al. (2019) proposed a Bayesian inference model that leverages the hidden Markov model for analyzing gaze trajectory and support vector machine for pupil size for predicting takeover performance when a secondary visual search job was carried out while the human operator teleoperated a simulated High Mobility Multipurpose Wheeled Vehicle (HMMWV). Young et al. (2013) performed a network analysis by calculating the eyes-on-the-road percentage and a similar analysis conducted by Molnar (2017) found that the gaze percentage was connected to drivers' situational awareness and focus on the road environment.

By using machine learning models to accumulate physiological data, it is possible to assess drivers' cognitive and emotional states. Models that predict drivers' states and their interactions with the driving environment can be created using the information gathered by non-intrusive sensors. The combination of environmental elements and physiological information from drivers offers potential indicators for real-time takeover performance prediction in conditionally AVs.

Existing Computational Models of Driver Takeover Performance

The techniques used to create the current computational models for forecasting drivers' takeover performance fall into two categories: data-based modeling and cognitive architecture-based modeling. To find patterns in the data and ultimately uncover the function that links predictors to outcomes, the data-based models were created and tested using data from driving
simulator experiments. With a few notable exceptions, little research has been done on the creation of computational models for forecasting drivers' takeover performance, even though a lot of studies have been done on factors that affect drivers' performance.

To estimate drivers' takeover preparedness, or "takeover quality," Braunagel et al. (2017) employed machine learning (ML) methods. The study used driving metrics including lane deviations to categorize takeover quality into low and high levels. They predicted takeover quality with an accuracy of 79% and an F1-score of 77% using ML methods like k-nearest neighbors (KNN), support vector machine (SVM) with radial basis function (RBF), and linear kernel, Naive Bayes, and linear discriminant. In Gold et al. (2018) takeover performance metrics were modeled as functions of time constraints, traffic volume, non-driving task repetition, the current lane, and driver age. Examples of these variables are takeover time, minimal TTC, braking application, and crash likelihood. The models accurately predicted takeover time, time to collision, and crash probability. Du et al. (2020) trained a Random Forest model on a multi-modal dataset comprising drivers’ galvanic skin responses, heart rate activities, and gaze behaviors in relation to the environmental factors, to predict drivers’ takeover performance with an accuracy of 84.3% and an F1-score of 64.0% using a 3 second time window. In a recent study by Zhou et al. (2021), they built a tree ensemble ML model, LightGBM to predict SA during the takeover transition period in conditionally automated driving using eye-tracking and self-reported data. They also calculated the SHAP (Shapley Additive Explanations) values of individual predictor variables in the LightGBM model to explain the prediction model by identifying the most important factors and their effects on SA. Using only eye-tracking data their model outperformed other selected ML models, having a root-mean-squared error (RMSE) of 0.121, a mean absolute error (MAE) of 0.096, and a 0.719 correlation coefficient between the predicted SA and the ground truth.

The other group of top-down, theory-driven computer modeling techniques for forecasting drivers' takeover performance is based on cognitive architectures. Adaptive Control of Thought-
Rational (ACT-R) manipulates declarative knowledge and environment by activating condition-action production rules. It is built on pieces of declarative information and procedural knowledge (Salvucci, 2006). Measurements of the effects of NDRT complexity and traffic complexity on takeover reaction time have been made using the ACT-R architecture (Scharfe & Russwinkel, 2019a). It has also been used to forecast the timing of individual takeover behaviors, such as looking at the road, stopping NDRT, and placing hands on the wheel (Scharfe & Russwinkel, 2019b).

Queuing network (QN) mathematical foundations and the ACT-R cognitive architecture were combined to create the QN-ACTR architecture, according to Cao and Liu (2013). A Micro Saint Sharp-based computer software is coupled to a driving simulator program to apply the QN-ACTR modeling and generate predictions of driver performance, including the trail of simulated mental activity, behavioral reactions, reaction times, accurate rates, and mental workload (Cao & Liu, 2013). Only Deng et al. (2019) have attempted to model drivers' emergency takeover response times while performing visual or auditory concurrent tasks in conditionally AVs to yet. In the study by Tan and Zhang (2022), they studied the effect of ToR lead times on a driver's SA in a conditionally AV. They employed a likelihood ratio test to report that the effects of greater ToR lead times on driver SA for recovering control to leave freeways were good and they peaked at lead times of 16 to 30 seconds.

These models, however, were created and put to the test when drivers were performing various NDRTs (such as monitoring vs. reading), where it appeared that contextual signals might be used to distinguish between drivers' states. Drivers' states in daily life can vary significantly based on the importance of a certain sort of NDRT, such as writing an email. Additionally, several variables that were purposefully modified in the experiment settings, including emotions, are difficult to access in the actual world. Hence, there is a present need for a model which can take raw unprocessed data as input to predict drivers’ takeover performance.
Research Objectives

This literature contributes to the already done research in three aspects. As it has been observed that visual tasks performed on handheld devices adversely affect the SA of a driver, this literature aims to numerically model drivers’ takeover performance when they engage in a particular type of hand-held NDRT with distinct levels of ToR. Then in addition to the vehicle’s sensory data, this study employs gaze signals to relate drivers’ interaction with the surroundings, which are proven to improve prediction accuracies. To date, only a few models have been developed to predict drivers’ takeover performance in response to the ToR initiated by AVs. Most of them have been developed using ML algorithms which require a lot of pre-processing of the data. Hence, the third aspect is that it makes a novel attempt in predicting drivers’ takeover performance using deep learning architectures. In this research, three different DL architectures were developed to create models which were able to accurately predict drivers’ takeover performance. Furthermore, these models can be used as a design reference for adaptive in-vehicle alert systems to enhance takeover capability in partially autonomous driving. These models are also useful for improving safe freeway exiting takeovers in conditionally autonomous vehicles (AVs) when designing human-machine interfaces in relation to optimizing ToR lead time. To enhance the performance of freeway exiting takeover, they can also be integrated to the vehicle's driving models.

Chapter 2 provides details about the datasets used in this study. Chapter 3 introduces the building blocks of a deep learning network and the metrics used to evaluate their performance. Chapter 4 discusses the CNN architecture-based models. Similarly, Chapter 5 and Chapter 6 discuss the ESN architecture-based and ITN architecture-based models respectively. Chapter 7 concludes this literature with a discussion of all the results and scope for future work.
Chapter 2

Datasets

Participants

All the data was collected by Tan Xiaomei, a Ph.D. candidate under Dr. Zhang. Thirty-two people (15 males, 17 females) aged 18 to 55 (Mean = 24.5, SD = 7.1) participated in the experiment. All the participants were licensed drivers for at least two years, with an average driving experience of 7.5 years (SD = 7.2) and an average annual mileage of 8,140 miles per year (SD = 5,756). Each participant was exposed to four multi-stage ToR designs. The four designs were tested in the experiment following a Latin square design. The data used to train the DL algorithms was sourced from a driving simulator and a wearable eye-tracking pair of glasses.

Simulator’s Driving Data

A fixed-based driving simulator (STISIM Drive M300WS-Console system) that is installed on a Dell™ Workstation was used. It has two cutting-edge foot pedals, a STISIM Drive® ADS high-fidelity, full-size steering wheel with active force feedback, and three driving displays that provide a 135° field of view. Open Module, which enables automated driving programming and control transitions from and to the human driver, is expendable and programmable for the STISIM Drive® Software. The STISIM Drive® Software exported raw data into DAT files for each scenario for each participant thus giving a total of three hundred and sixty files, which were cleaned and converted to CSV (Comma Separated Values) format for easier use in the model development phase. Table 2-1 gives all the twenty-eight features of the driving data’s (DD) time series, stamped every 0.1 seconds.
Eye-Tracking Data

The Tobii Pro Glasses 2, a wearable/head-mounted eye-tracking device was utilized to live-track participant gaze patterns. To record broad-angle images and assure natural viewing behavior, including peripheral viewing, the Glasses are outfitted with four eye cameras (each runs continuously at 100 Hz), an HD (1980 * 1080) scene camera with a 90° field of view, and thin side parts. The dual image sensor technology allows for precise pupil size measurement compensations as well as head movement compensations. This data was a collation of data from three different sources, namely accelerometer (ACC) data, gaze (GZ) data, and gyroscope (GY) data. Each data point for the three time-series-based datasets was stamped every 0.01 second. Tobii Pro Glasses
connects with its proprietary software to export this data which was segregated into the three constituent datasets later using a python script. Both the accelerometer data and the gyroscope data are comprised of three features each whereas the gaze data is comprised of twenty-one features. A comprehensive list of these features is presented in Table 2-2.

Table 2-2: Features of the eye-tracking data.

<table>
<thead>
<tr>
<th>Dataset Type</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer (ACC)</td>
<td>Accelerometer X [m/s²]</td>
</tr>
</tbody>
</table>

**Data Pre-processing**

A famous study by Famili et al. (1997) states two main reasons for performing data pre-processing, firstly it resolves any problem within the datasets and second it formats the datasets for further analysis or as an input to the ML or DL algorithms. As the first step, the features which did not manifest any change over the time series were removed from the data as they will increase the redundancy and correlation in the models. Table 2-3 lists these ten features. As the collected datasets were unlabeled the DD datasets were labeled for the time when the driver takes over the vehicle, serving as the ground truth for model training. This was achieved by looking at the time-
series values of a particular feature i.e., ‘Longitudinal Velocity’ as it was known that takeover occurs as soon as the velocity changes from a constant value of eighty-eight. Matching these takeover timestamps for each participant’s each scenario to the timestamps in the data from the Tobii glasses, the remaining three datasets namely, ACC data, GZ data, and GY data were also labeled. The DD datasets were collected for one record every 0.1 seconds while the rest of the datasets were collected for one record every 0.01 seconds. Hence, it was required to aggregate ten records of the ACC, GZ, and GY datasets by taking the average of them to match the data collection frequency in all four datasets even though they have different modalities. The average values were used as they register the overall movement made by the drivers in that period of 10 milliseconds.

Table 2-3: Unchanging features in each time series.

<table>
<thead>
<tr>
<th>Features</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longitudinal Acceleration due to Braking</td>
<td>Brake Pedal Input Counts</td>
</tr>
<tr>
<td>Running Compilation of Driver Crashes</td>
<td>Vehicle#1s Forward Speed</td>
</tr>
<tr>
<td>Drivers Response Time</td>
<td>Minimum Range (with all vehicles opposing the</td>
</tr>
<tr>
<td></td>
<td>driver’s direction)</td>
</tr>
<tr>
<td>Vehicle#2s Forward Speed</td>
<td>Lateral Position of Vehicle#1 with Respect to the</td>
</tr>
<tr>
<td></td>
<td>Roadways Dividing Line</td>
</tr>
<tr>
<td>Lateral Position of Vehicle#3 with Respect to the</td>
<td>Minimum Time to Collison (with all vehicles</td>
</tr>
<tr>
<td>Roadways Dividing Line</td>
<td>opposing the driver’s direction)</td>
</tr>
</tbody>
</table>

The duration of the experiments varied because they were carried out on various people. Yet, because the data was always gathered during set time frames, there were varying numbers of rows for each experiment. Hence, following the study by Gupta et al. (2022), the data was taken from a window spanning a predetermined time frame also called window size to standardize the number of rows. For example, if an experiment took 5 seconds and the time step was 0.1 seconds, the experiment would include 50 observations (rows). For this experiment, a window size of 1 second (10 rows per window) would result in 41 data points, each lasting one second (10 rows). As a result, the total number of data points that might be utilized to train the model increased which helps in better training a DL model. A binary variable indicating whether the driver seized control
of the conditionally AV within that time interval would be the data point's ground truth. The window size, which governs how many data points are generated, was a crucial component. A bigger window size resulted in fewer data points being produced. The amount of data the algorithm had access to make its forecast was similarly governed by the window's length. Finally, the information produced when the driver takes control of the AVs can only represent a small piece of the window corresponding to the period when the vehicle is being taken over, which necessitates the use of different window sizes to increase the number of data points for the takeover class.
Chapter 3

Neural Networks and Deep Learning

Introduction to Neural Networks

Time-series classification has evolved over the past few years into one of data science's most difficult challenges. This occurred because every classification issue that employs data while considering a sorting concept can be handled as a time series classification issue. Time series are used in a wide variety of real-world applications, including automated illness identification, anomaly detection, finance, marketing, human activity recognition, healthcare, and cyber-security. As the amount of temporal data has grown dramatically over the past few years, many fields have developed a keen interest in time series-based applications, leading to the development of numerous novel algorithms.

All these methods, except for those based on deep learning, demand feature engineering as a distinct task prior to classification as done by Du et al. (2020), which may result in some information being lost and an extension of the development time. On the contrary, deep learning models already include this form of internal feature engineering, optimizing it and removing the need for human feature engineering. As a result, they can extract data from the time series in a quicker, more efficient, and more thorough manner.

A formal definition of a time-series classification problem is where a definite number of different classes and a set of objects all have the same structure (for instance, real numbers, vectors, and matrices all having the same size, etc.). We refer to a dataset as a collection of pairs (object, class), meaning that each object has a certain class associated with it. Building a model to assign a new item, with the same structure as the others, the probability of belonging to the available classes in accordance with the features of the objects associated with each class is called a classification
issue is given a dataset. An ordered set of real values is referred to as a univariate time series, whereas an M-dimensional multivariate time series is made up of M distinct univariate time series of the same length. A time series classification problem is a classification problem where the dataset's objects are time series, either univariate or multivariate.

**Perceptron (Neuron)**

The perceptron is the fundamental building block of many ML or DL algorithms. It is also known as a neuron since its design was influenced by the operation of biological brain networks. Figure 3-1 displays the architecture of a perceptron. From the architecture its similarity with the neuron of a biological brain becomes evident.

![Architecture of a Perceptron](image)

Figure 3-1: Architecture of a Perceptron.

A perceptron's objective is to calculate the weighted sum of the input values and then take that value and apply an activation function to it. Each path that is used to connect the input nodes
(\(x_i\) where \(i \in (1,2,3,\ldots,n)\)) to the weighted sum node has a random weight (\(w_i\) where \(i \in (1,2,3,\ldots,n)\)) assigned to it. These weights represent the strength that is assigned to each input node.

The next node is used to map the weighted sum between a pair of required values like (0,1) or (-1,1) and hence is called the activation node. The sigmoid, hyperbolic tangent, and rectifier activation functions are most frequently used in literature and real-world applications. Figure 3-2 displays them. The activation of the perceptron is the outcome of the activation function and is the output value of the perceptron.

\[
\text{Sigmoid} \quad \frac{1}{1 + e^{-x}} \quad \text{Tanh} \quad \tanh(x) \quad \text{ReLu} \quad \max(0, x)
\]

Figure 3-2: Activation Functions.

The underlying mathematics for a perceptron is based on basic operations and applying a function to get the output. Equation 3-1 states these operations for the three activation functions mentioned above.

Equation 3-1: Mathematical operations in a perceptron.

\[
\text{Weighted Sum (WS)} = (x_1 \cdot w_1) + (x_2 \cdot w_2) + (x_3 \cdot w_3) + \cdots + (x_n \cdot w_n)
\]

\[
\text{Output}_{\text{Sigmoid}} = \frac{1}{1 + e^{-WS}}
\]

\[
\text{Output}_{\text{Tanh}} = \tanh(WS)
\]

\[
\text{Output}_{\text{ReLU}} = \max(0, WS)
\]
Multi-Layer Perceptron

Several DL architectures for time-series classification use the multi-layer perceptron (MLP) as a building block. One input layer, one or more hidden layers, and one output layer are the layers that make up this kind of feed-forward neural network. Each node has connections to every other node in its layer, the layer above it, and the layer below it. We refer to the MLP as completely connected for this reason. A perceptron makes up each node of the hidden layers and the output layer. The function that connects the input and the output layers depend on the weight values of each path connecting the nodes. The output of the MLP is computed by sequentially activating its Perceptrons. Figure 3-3 shows a completely connected MLP.

![Architecture of Multi-Layer Perceptron](image-url)

Figure 3-3: Architecture of Multi-Layer Perceptron.
Classification with Multi-Layer Perceptron (MLP)

MLP is commonly used in DL for classification problems. Given a dataset (that we recall being a collection of pairs (object – input values, class – output labels, or target)), it can be fitted to compute the probability of any new object belonging to each possible class. To do this, first, we need to represent the pairs (object, class) in the dataset in a more suitable way. To start with every object must be flattened and then represented with a vector, which will be the input vector for training the algorithm. Then every class in the dataset must be represented with its one-hot label vector. A one-hot label vector is a vector with a size equal to the number of different classes in the dataset. Each element of the array corresponds to a possible class, and every value is 0 apart from that related to the represented class, which is 1. The one-hot label vector will be the target for training. Figure 3-4 contains an example of a one-hot labeled vector.

<table>
<thead>
<tr>
<th>id</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>red</td>
</tr>
<tr>
<td>2</td>
<td>blue</td>
</tr>
<tr>
<td>3</td>
<td>green</td>
</tr>
<tr>
<td>4</td>
<td>blue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>color_red</th>
<th>color_blue</th>
<th>color_green</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3-4: One-hot labeled vector.

A new dataset of pairs (input vector, target) is generated following the method described above, and the dataset is ready for training purposes. Backpropagation, a supervised learning method that iterates on the input vectors of the dataset, is employed by MLP for this procedure. The procedure of the backpropagation method is listed below and is also displayed in Figure 3-5:

- At every iteration, the output of the MLP is computed using the current input vector.
• The output is a vector whose components are the estimated probabilities of belonging to each output class.
• The model’s prediction error is computed using a defined cost function. The cost function is used to quantify the difference (loss) between the predicted and the actual values during the training phase.
• Then, using the gradient descent method, the weights are updated in a backward pass to propagate the error. Gradient descent is an optimization algorithm used to find the best parameters (coefficients) of a function that minimizes a cost function (loss).

Thus, by iteratively taking a forward pass followed by backpropagation, the model’s weights are updated in a way that minimizes the loss of the training data. After the training, when the input of the MLP is the vector corresponding to an object with the given structure and not already present in the dataset, the output is a vector whose components are the estimated probabilities of belonging to each output class.
One question arises naturally, why not utilize the MLP to classify time series using input from the entire multivariate time series? The answer to this is in that the length of a time series substantially decreases the computational performance, which is why MLP, and many other Machine Learning algorithms don't perform well for Time Series Classification. As a result, to achieve good results for time-series classification, it is required to extract the pertinent features from the input time-series and utilize them as input to a classification algorithm to achieve better results with much less processing. The solution to this problem lies in DL algorithms.

The main benefit of DL algorithms over other methods is that the pertinent features mentioned earlier are learned during the training phase rather than being created manually. The precision of the results is greatly enhanced, and the time required for data preparation reduces significantly. Then, almost all the DL architectures make use of algorithms like MLP to achieve the necessary classification after several layers were used to extract the pertinent features.

**Deep Learning for Time Series Classification**

A simple DL architecture is a composition of several layers that implement non-linear functions on the input layer to extract the pertinent features. The input is a multivariate time series. Every layer takes as input the output of the previous layer and applies its non-linear transformation to compute its own output. The behavior of these non-linear transformations is controlled by a set of defined parameters for each layer. These parameters link the input of the layer to its output and are trainable (like the weights of the MLP). Often, the last layer is a Multi-Layer Perceptron or a Ridge regressor to estimate the probability of belonging to a class from the calculated features. Figure 3-6 shows a general Deep Learning framework for time-series classification.
In this literature, 3 different DL architectures for time-series classifications are employed to develop different types of classification models. These architectures are listed below.

- Convolutional Neural Network (CNN) is the most classical and used architecture for time-series classification problems.
- Echo State Network (ESN), is another recent architecture, based on recurrent neural networks (RNN).
- Inception Time Network (ITN) is a new architecture based on convolutional neural networks.

ESNs differ from the other two architectures in its ability to process temporal information, data that occurs in a sequence like a time-series. Another feature of the ESN is the autonomous operation in prediction i.e., if the ESN is trained with an input that is a back shifted version of the output, then it can be used for signal generation/prediction by using the previous output as input. A CNN module is the building block for an ITN comprising of networks within networks, which attempt to find information embedded deep within the input data. This architecture while maintaining the computational budget, increases the dept and width of the network, resulting in an improved deep learning network. CNNs are the simplest of the three and hence the starting point of this study.
Model Evaluation

There are four possible results in a binary classification problem: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). These four results together form the confusion matrix of a binary classification problem as shown in Figure 3-7. The number of positive samples anticipated as belonging to the positive class is denoted by the letters TP, the number of negative samples predicted as belonging to the positive class is denoted by the letters FP, the number of negative samples predicted as belonging to the negative class is denoted by the letters TN, and the number of positive samples predicted as belonging to the negative class is denoted by the letters FN. To evaluate the trained models five metrics, namely, accuracy, specificity, sensitivity, precision, and F1 score will be calculated as per the relations shown in Equation 3-2.

![Confusion Matrix](image)

**Figure 3-7: Confusion Matrix.**

**Equation 3-2:** Formulas for model evaluation metrics.

\[
\text{Accuracy} = \frac{TN + TP}{TN + FN + FP + TP}
\]

\[
\text{Specificity} = \frac{TN}{TN + FP}
\]

\[
\text{Sensitivity OR Recall} = \frac{TP}{FN + TP}
\]
Accuracy represents the percentage of data that is classified correctly. This metric fails to prove a good model evaluation metric when the data is not balanced. Specificity represents the percentage of negatives that are classified correctly. Similarly, sensitivity also known as recall is the measure of the percentage of positives classified correctly or the measure of completeness. Precision measures the percent of true positives among all the predicted positives or it’s a measure of exactness. Ideally, for a good classifier, the value of recall and precision should be as close to one as possible. F1 score is a metric that takes both these metrics into account and is the harmonic mean of them. As stated by Du et al. (2020), this is the most realistic metric to measure the performance of a classification model.

\[
\text{Precision} = \frac{TP}{FP + TP}
\]

\[
F1 \text{ Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]
Chapter 4

Convolutional Neural Network

A convolutional neural network is a DL algorithm that accepts an input of a multivariate time series, correctly detects spatial and temporal patterns using application-trainable filters, and then prioritizes these patterns using trainable weights. In comparison to basic ML classification techniques, a CNN requires substantially less pre-processing. While filters are frequently hand-engineered in approaches, CNN learns these filters on its own. A CNN comprises three layers, convolutional layers, pooling layers, and a fully connected MLP layer. A typical CNN architecture is shown in Figure 4-1.

![Convolutional Neural Network architecture](image)

Figure 4-1: Convolutional Neural Network architecture.

CNN gets its name from the convolution operation because it serves as the main structural component of these networks. To extract the high-level features, it convolutes a series of input feature maps with a filter matrix to produce another series of output feature maps. A collection of filters, which are matrices with predetermined sizes, define the convolution. When a filter is applied to a submatrix of the input feature map that has the same size as the submatrix, the outcome is determined by adding the products of each filter element that is at the same position in the
submatrix. Equation 4-1 represents the formula used in the discrete convolution of matrices. The ordered feature map that is produced by applying the filter across the width and height of the input feature map is the result of the convolution between one input feature map and one filter. Every filter and each input feature map are convoluted by a convolutional layer, producing several feature maps as the output. The filter values are trainable weights and are learned during the training phase.

After almost every convolution is the pooling operation. Its purpose is to reduce the dimension of the feature maps while retaining as much data as possible. It can also be used to extract dominant, rotational, and positional invariant features. A series of feature maps serve as their input, while a distinct series of feature maps with a smaller dimension serves as their output. Each input feature map is subjected to pooling using sliding windows that are a set size across its width and height. Max Pooling and Average Pooling are the two different types of pooling based on which the outcome of the pooling operation is either the maximum value or the average value of each sliding window. Max Pooling also functions as a noise suppressor, eliminating any noisy activations. As a result, it typically outperforms Average Pooling. The benefit of the pooling operation is that it reduces variability in the hidden activations by downsampling the convolutional output bands.

After learning all the pertinent features of the time series using convolution and pooling, a fully connected layer implementing MLP is employed to learn non-linear combinations of the high-level features. The original time series is represented by a set of feature maps following numerous convolution and pooling processes. The final representation of the original input multivariate time series is made up of all these feature maps that have been flattened into a column vector. The MLP, whose output has as many neurons as there are potential classes or output labels of time series, is connected to the flattened column. Every iteration of training uses backpropagation. The prominent high-level properties of the input time series enable the model to identify and classify them, using a defined number of epochs.
Equation 4-1: Convolution function for two matrices.

\[(f * g)[n] = \sum_{m=-M}^{M} f[n - m] g[m]\]

\[(f * g)[n] = \text{Convolution function}\]

\[f[n - m] = \text{Input Time Series Matrix}\]

\[g[m] = \text{Kernel Matrix}\]

**Implementation**

The CNN architecture employed for training models is made up of two convolution layers combined with two pooling layers. Each convolution layer employs a kernel of 3*3 matrix size, and the first layer convolutes the input to eight filters which after pooling are convoluted by the second layer to sixteen filters and pooled again. Both the pooling layers use a kernel of 2*2 matrix size. The pooled sixteen filters are flattened and further connected to the fully connected MLP layer with two hidden layers of sixteen and eight nodes each and the output layer with just one node representing the takeover class. The convolution layers use the relu activation function, the hidden layers use the tanh activation function, and the output layer uses the sigmoid activation function. Relu activation function has proved to extract the pertinent features efficiently when compared to other activation functions. The tanh and sigmoid activation functions are used within the fully connected layer as they prove more efficient to tackle binary classification problems. Figure 4-2 displays this CNN architecture. The built architecture was compiled together using the ‘adam’ optimizer and ‘binary cross entropy as the loss function. Both gave better results when compared to their alternatives. A training strategy was devised where the pre-processed data was divided into three parts. Twenty-Five percent part was used for testing purposes. Twenty-five percent of the
remaining was used for validation purposes and the final remaining part was used for training the model. The models were trained for twenty epochs. The number of epochs was optimized, making sure that the models neither underfit data nor overfit the data.

Figure 4-2: Employed CNN architecture.

**Types-CNN Models**

**Table 4-1**: Dataset permutations.

<table>
<thead>
<tr>
<th>Type of Dataset</th>
<th>Dataset Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDACCGYGZ</td>
<td>Driving; Accelerometer; Gyroscope; Gaze</td>
</tr>
<tr>
<td>DD</td>
<td>Driving Data</td>
</tr>
<tr>
<td>DDACC</td>
<td>Driving; Accelerometer</td>
</tr>
<tr>
<td>DDACCGY</td>
<td>Driving; Accelerometer; Gyroscope</td>
</tr>
<tr>
<td>DDACCGZ</td>
<td>Driving; Accelerometer; Gaze</td>
</tr>
<tr>
<td>DDGY</td>
<td>Driving; Gyroscope</td>
</tr>
<tr>
<td>DDGYGZ</td>
<td>Driving; Gyroscope; Gaze</td>
</tr>
<tr>
<td>DDGZ</td>
<td>Driving; Gaze</td>
</tr>
</tbody>
</table>
First, the architecture is trained on different permutations of the four types of data discussed in chapter 2 and hence the name, Types-CNN Models. To permute, the driving dataset (DD) was combined with different combinations of the accelerometer (ACC) data, gaze (GZ) data, and gyroscope (GY) data. The DD data was kept constant in all these combinations because it contains the most vital information related to the takeover by the drivers. Table 4-1 presents all eight permutations. To generate training datasets, a one-second window size (ten data points per training record) was applied to the eight datasets and eight models were trained.

**Results**

An important model evaluation characteristic to start with is to validate if the model is trained to optimality. This is achieved by plotting the training/validation accuracy and loss values versus each epoch iteration for all eight models. From Figure 4-3, it can be observed that the training and validation accuracies and losses for one of the eight models display near zero variance as the number of epochs approaches twenty. The remaining seven models follow a similar trend. The performance of the eight models was then measured by calculating the five metrics mentioned in the Model Evaluation section. These metrics were calculated for the test dataset which the model never sees during the training phase. Table 4-2 tabulates the five metrics plus the training time for each model. The model built using the combination of driving data (DD), accelerometer (ACC) data, and gaze (GZ) data outperformed all the other models, having the best F1 Score of 0.965 and the lowest training time of seventeen seconds. This model also performed consistently across all the calculated metrics as is evident from Figure 4-4.

The second-best model was built using the combination of DD data and ACC data. This model achieved an F1 Score of 0.965 with an increase in training time by three seconds and was slightly more consistent across the metrics. The rest of the models even though achieved similar F1
Scores but were not consistent and exhibited huge variances across all the metrics. This is shown in Figure 4-4.

Figure 4-3: Training’s and Validation’s accuracy and loss plots.

Table 4-2: Type-CNN model evaluation metrics.

<table>
<thead>
<tr>
<th>Dataset Type</th>
<th>Accuracy</th>
<th>Specificity</th>
<th>Sensitivity</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Training Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDACCGYGZ</td>
<td>0.945</td>
<td>0.91</td>
<td>0.981</td>
<td>0.912</td>
<td>0.945</td>
<td>17</td>
</tr>
<tr>
<td>DD</td>
<td>0.927</td>
<td>0.866</td>
<td>0.988</td>
<td>0.878</td>
<td>0.93</td>
<td>20</td>
</tr>
<tr>
<td>DDACC</td>
<td>0.965</td>
<td>0.955</td>
<td>0.974</td>
<td>0.957</td>
<td>0.965</td>
<td>20</td>
</tr>
<tr>
<td>DDGY</td>
<td>0.96</td>
<td>0.963</td>
<td>0.957</td>
<td>0.964</td>
<td>0.96</td>
<td>20</td>
</tr>
<tr>
<td>DDGZ</td>
<td>0.941</td>
<td>0.916</td>
<td>0.967</td>
<td>0.916</td>
<td>0.941</td>
<td>17</td>
</tr>
<tr>
<td>DDACCGGY</td>
<td>0.934</td>
<td>0.884</td>
<td>0.985</td>
<td>0.894</td>
<td>0.937</td>
<td>21</td>
</tr>
<tr>
<td>DDACCGZ</td>
<td>0.965</td>
<td>0.956</td>
<td>0.974</td>
<td>0.956</td>
<td>0.965</td>
<td>17</td>
</tr>
<tr>
<td>DDGYGZ</td>
<td>0.927</td>
<td>0.865</td>
<td>0.984</td>
<td>0.886</td>
<td>0.933</td>
<td>17</td>
</tr>
</tbody>
</table>
From the eight permutations in the Types-CNN model the DD data, ACC data, and GZ data combination proved to be the best. To analyze this further, the CNN architecture shown in Figure 4-2 was employed but for varying time window sizes. The window sizes varied from one second to ten with an increment of one second.

**Results**

For evaluating the model, the training/validation accuracies and losses were plotted to test the model for optimality. Like earlier models, these models too reached optimality around twenty epochs. Table 4-3 presents the model evaluation metrics, and it can be observed that the five and
six-second window sizes performed consistently across the metrics (Figure 4-5) with an F1 Score of 0.98 and 0.982 respectively. But the five-second window size was selected as the best model because of the significant fourteen-second difference in the training times of the two models. It is also observed that the training time increased with the increase in window size.

Table 4-3: DDACCGZ-CNN model evaluation metrics.

<table>
<thead>
<tr>
<th>Window Size (Seconds)</th>
<th>Accuracy</th>
<th>Specificity</th>
<th>Sensitivity</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Training Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.954</td>
<td>0.962</td>
<td>0.947</td>
<td>0.964</td>
<td>0.955</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>0.972</td>
<td>0.962</td>
<td>0.981</td>
<td>0.963</td>
<td>0.972</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>0.975</td>
<td>0.972</td>
<td>0.979</td>
<td>0.972</td>
<td>0.975</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>0.97</td>
<td>0.984</td>
<td>0.956</td>
<td>0.985</td>
<td>0.97</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>0.981</td>
<td>0.98</td>
<td>0.981</td>
<td>0.98</td>
<td>0.98</td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td>0.981</td>
<td>0.982</td>
<td>0.98</td>
<td>0.983</td>
<td>0.982</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>0.979</td>
<td>0.966</td>
<td>0.993</td>
<td>0.966</td>
<td>0.98</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>0.966</td>
<td>0.992</td>
<td>0.94</td>
<td>0.992</td>
<td>0.965</td>
<td>70</td>
</tr>
<tr>
<td>9</td>
<td>0.986</td>
<td>0.991</td>
<td>0.98</td>
<td>0.99</td>
<td>0.985</td>
<td>78</td>
</tr>
<tr>
<td>10</td>
<td>0.987</td>
<td>0.978</td>
<td>0.996</td>
<td>0.979</td>
<td>0.987</td>
<td>83</td>
</tr>
</tbody>
</table>

Figure 4-5: DDACCGZ-CNN model metrics graph.
**K-Fold-CNN Model**

To further improve the five-second window size model, a k-fold cross-validation technique was employed. Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. In the k-fold resampling technique the data is randomly divided into training, validation, and testing datasets k times. The model is then trained over these k datasets and the error rate is calculated while saving the best training weights. The mean of errors from all the iterations is calculated. Figure 4-6 depicts an example of a ten-fold cross-validation technique. This technique has proven to result in low variance as well as low bias in the model. Typically, k-fold cross-validation is performed using k as 5 or 10 as these values have been empirically shown to yield test error estimates that neither have high bias nor high variance. Hence a ten-fold cross-validation model was built for the five-second window size CNN model. The model follows the proposed CNN architecture and differs only in the use of the k-fold cross-validation technique.

![K-fold Cross Validation example](image)

**Figure 4-6**: K-fold Cross Validation example.
Results

After applying the k-fold cross-validation procedure, the performance metrics of the five-second window size CNN model improved further with an F1 Score of 0.993, precision of 0.991, recall of 0.996, specificity of 0.991, and an accuracy of 0.993. From these metrics, it is evident that the model is consistent across the metrics with low variance and has a low average training time of 46 seconds. This results in a robust, high-performance model which neither underfits nor overfits the data and maintains low variance and bias.
Chapter 5

Echo State Network (ESN)

Recurrent Neural Network (RNN)

As Echo State Networks are a subset of Recurrent Neural Networks (RNNs), a brief overview of them may be helpful. RNNs have an architecture that is like that of normal neural networks and are made up of networks of nodes that resemble neurons and are arranged in successive layers. In fact, neurons are separated into an input layer, hidden layer, and output layer, just like in conventional neural networks. Each neuronal connection has an associated trainable weight. The distinction is that each neuron in this instance is given a set timestep. The neurons in the hidden layer are also forwarded in a time-dependent direction, which means that each of them has a one-way connection to each neuron allocated to the timestep after which they are fully connected only with other neurons in the hidden layer with the same given timestep. Just the hidden layers with the set timestep are connected to the input and output neurons. Since the output of the hidden layer of one timestep is part of the input of the next timestep, the activation of the neurons is computed in time order i.e., at any given timestep, only the neurons assigned to that timestep compute their activation. Figure 5-1 displays this architecture.

![Figure 5-1: RNN Architecture.](image-url)
RNNs are rarely applied for time series classification mainly due to three factors. This architecture's primary purpose is to forecast output for each time series component. RNNs frequently experience the vanishing gradient problem when trained on lengthy time series, which means that the hidden layer parameters either don't change significantly or cause numerical instability and unpredictable behavior. RNN training is costly computationally and difficult to parallelize. By eliminating the requirement to compute the gradient for the hidden layers, cutting training time, and avoiding the vanishing gradient problem, ESNs were created to address the issues with RNNs. In fact, a lot of findings demonstrate how effective ESNs are at handling erratic time series.

**Echo State Network Architecture**

Figure 5-2: ESN Architecture.

The input layer, dimension reduction layer, a fully linked layer called readout, and output layer make up the architecture of an ESN, as depicted in Figure 5-2. The main component of this architecture, the reservoir, is set up like a sparsely linked random RNN. Principal component analysis (PCA) is typically used to implement the dimension reduction technique. Typically, MLP
or a ridge regressor is used to accomplish the readout. The weights in the reservoir and those between the input layer and reservoir are chosen at random and cannot be trained. The readout's weights are trainable, allowing the network to learn and produce specific patterns. The activation by the reservoir \((h(t))\) to feed into the dimension reduction layer is represented in Equation 5-1.

**Equation 5-1: Activation by the reservoir.**

\[
h(t) = f(W_{in}x(t) + W_{res}h(t-1))
\]

\(f = Activation\ Function\ | \ W_{in} \& W_{res} = Weights\)

The reservoir is set up like a sparsely connected random RNN as shown in Figure 5-3. It includes its own output neurons in addition to a group of internal, sparsely linked neurons that are connected to the input layer. There are four different weight categories in the reservoir. Input
weights, which link the input layer to the internal neurons; internal weights, which link the internal neurons to one another; output weights, which link the internal neurons to the output; and backpropagation weights, which link the output to the internal neurons. These weights are equal for each time step, initialized at random, and are not trainable. Since the outcome of one timestep is included in the input of the following timestep, the reservoir computes its output separately for each timestep, much like RNNs do. For every timestep, the output for the current timestep is calculated by computing the activation of every internal and output neuron. The big advantage of ESNs is that the Reservoir creates a recurrent nonlinear embedding of the input into a higher dimension representation, but since only the weights in the Readout are trainable, the training computation time remains low.

**Dimension Reduction**

Many studies demonstrate that by selecting the appropriate dimension reduction, it is possible to shorten execution times without compromising accuracy. Moreover, dimensional reduction offers a regularization that raises the models' overall generalizability and resilience. The accuracy grows quickly if the subspace dimension is below a specific threshold, after which it slows down significantly and stays essentially constant. In most circumstances, the training time increases almost linearly with the subspace dimension. This threshold is therefore the optimal option for the subspace dimension because a larger number would result in a longer execution time without appreciable accuracy improvements.
Implementation

The ESN architecture employed for training models is made up of a reservoir with 400 neurons with a spectral radius of 0.59, which serves as the largest eigenvalue of the reservoir matrix of connection weights, connectivity between these neurons was kept at twenty percent and was decided to have no leakage between the neurons of the reservoir. PCA was employed next to reduce the dimension of the output of the reservoir to 60 components. These components were fed into the readout layer employing a ridge regressor with an alpha value of five to predict the output class. The training dataset was divided into two parts, twenty-five percent testing data, and seventy-five percent training data. Validation data was not required for training this architecture. This implementation is an adaption of the architecture proposed in the study by Bianchi et al. (2020). The used hyperparameters were optimized by hit and trial and monitoring the F1 scores in each trial.

Type ESN Models

For ESN architecture, the first eight models were trained on the eight permuted datasets like the methodology followed in the CNN architecture section and tested for the best combination of data types for this architecture.

Results

The five model evaluation metrics were calculated and tabulated in Table 5-1 with the training time of each model. Clearly, there are two models which outperform the other models in all areas. The first one is trained on data which is a combination of DD and GZ datasets and has an F1 score of 0.94 with a training time of 3 seconds. The second model is trained on the data which
is a combination of DD, ACC, and GY data and has the same F1 score of 0.94 as the first model, but a higher training time of 4 seconds, which is because it is generating the reservoir from a larger dataset as compared to the first model. Hence the combination of DD and GZ datasets is taken as the best combination and varying time window size models were built for further analysis. Another observation was that all these eight models had a similar trend across all five metrics which is shown in Figure 5-4.

Table 5-1: Type ESN models’ evaluation metrics.

<table>
<thead>
<tr>
<th>Dataset Type</th>
<th>Accuracy</th>
<th>Specificity</th>
<th>Sensitivity</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Training Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDACCGYGZ</td>
<td>0.929</td>
<td>0.908</td>
<td>0.951</td>
<td>0.911</td>
<td>0.931</td>
<td>3</td>
</tr>
<tr>
<td>DD</td>
<td>0.913</td>
<td>0.889</td>
<td>0.936</td>
<td>0.897</td>
<td>0.916</td>
<td>4</td>
</tr>
<tr>
<td>DDACC</td>
<td>0.929</td>
<td>0.906</td>
<td>0.954</td>
<td>0.904</td>
<td>0.929</td>
<td>4</td>
</tr>
<tr>
<td>DDGZ</td>
<td>0.931</td>
<td>0.913</td>
<td>0.949</td>
<td>0.914</td>
<td>0.931</td>
<td>4</td>
</tr>
<tr>
<td>DDACCGY</td>
<td>0.938</td>
<td>0.916</td>
<td>0.96</td>
<td>0.922</td>
<td>0.94</td>
<td>3</td>
</tr>
<tr>
<td>DDACCGZ</td>
<td>0.937</td>
<td>0.916</td>
<td>0.957</td>
<td>0.923</td>
<td>0.94</td>
<td>4</td>
</tr>
<tr>
<td>DDGYGZ</td>
<td>0.932</td>
<td>0.907</td>
<td>0.958</td>
<td>0.91</td>
<td>0.933</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 5-4: Type-ESN models’ metrics plot.
DDGZ ESN Models

Again, following the same methodology as followed for the CNN architecture, for analyzing the impact of window sizes on the performance of the model, ten models were trained using the same ESN architecture, but for the time window sizes varying from one to ten seconds with an increment of one second each.

Results

It can be observed from Table 5-2 that the nine-second window size model has the highest F1 score of 0.982 but shows a lack of consistency among the remaining metrics. It also had a high training time of 146 seconds. A more robust model is the 6-second window size model, which has a somewhat lower F1 score of 0.979, but has a significantly better training time of 54 seconds and performs consistently across all five metrics (Figure 5-5). The insignificant decrease in F1 score is sufficiently overpowered by the increase in computational efficiency of this model when compared to the nine seconds window size model.

<table>
<thead>
<tr>
<th>Window Size (Seconds)</th>
<th>Accuracy</th>
<th>Specificity</th>
<th>Sensitivity</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Training Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.935</td>
<td>0.917</td>
<td>0.953</td>
<td>0.918</td>
<td>0.935</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0.964</td>
<td>0.956</td>
<td>0.972</td>
<td>0.958</td>
<td>0.965</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0.963</td>
<td>0.958</td>
<td>0.969</td>
<td>0.961</td>
<td>0.965</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>0.971</td>
<td>0.966</td>
<td>0.976</td>
<td>0.966</td>
<td>0.971</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>0.975</td>
<td>0.969</td>
<td>0.982</td>
<td>0.969</td>
<td>0.975</td>
<td>42</td>
</tr>
<tr>
<td>6</td>
<td>0.979</td>
<td>0.977</td>
<td>0.98</td>
<td>0.978</td>
<td>0.979</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>0.976</td>
<td>0.968</td>
<td>0.984</td>
<td>0.969</td>
<td>0.976</td>
<td>76</td>
</tr>
<tr>
<td>8</td>
<td>0.979</td>
<td>0.969</td>
<td>0.99</td>
<td>0.969</td>
<td>0.98</td>
<td>95</td>
</tr>
<tr>
<td>9</td>
<td>0.982</td>
<td>0.977</td>
<td>0.986</td>
<td>0.978</td>
<td>0.982</td>
<td>146</td>
</tr>
<tr>
<td>10</td>
<td>0.981</td>
<td>0.977</td>
<td>0.985</td>
<td>0.978</td>
<td>0.981</td>
<td>176</td>
</tr>
</tbody>
</table>
K-Fold ESN Model

To further improve the six seconds window size ESN model, a ten-fold cross-validation sampling technique is used employing the proposed ESN architecture. This increases the robustness of the model and improves its performance on the data it has not seen before i.e., the test dataset.

Results

After applying the k-fold cross-validation procedure, the performance metrics of the six-second window size ESN model improved further with an F1 Score of 0.979, precision of 0.973, recall of 0.984, specificity of 0.973, and an accuracy of 0.979. From these metrics, it is evident that the model is consistent across the metrics with low variance and has a low average training time of
58 seconds. This results in a robust, high-performance model which neither under fits nor overfits the data and maintains low variance and bias.
Chapter 6

Inception Time Networks (ITN)

An inception network's architecture is comparable to that of a convolutional neural network, with the exception that inception modules are used in place of the convolutional layers and pooling layers.

Figure 6-1: A typical Inception Time Architecture.

The Inception Network, as depicted in Figure 6-1, is made up of several inception modules, a global average pooling layer, and a fully connected layer (usually an MLP). Also, every third inception module adds a residual connection. By enabling a direct flow of the gradient, the input from each residual block is transferred via a short-cut linear link and added to the input from the following block, alleviating the vanishing gradient issue.

Inception Module

The inception module, seen in Figure 6-2, is a key component of an inception network. It consists of four layers. The inputs' dimensionality is decreased by the first layer, which serves as a
bottleneck layer. This decreases both the number of parameters and the cost of computation, accelerating training and enhancing generalization. A group of parallel convolutional layers functioning on the same input feature map in varying sizes makes up the second main part of the inception module. For instance, three distinct convolutions with kernel sizes of 10, 20, and 40 are shown in Figure 6-2. Max pooling, which is introduced in the third layer, gives models the capacity to be invariant to tiny perturbations. The output of each separate parallel convolution and of the max pooling are concatenated to create the output multivariate time series of the current inception module in the fourth and final layer, known as the depth concatenation layer. The network can extract latent hierarchical features of several resolutions by stacking many inception modules and training the values of the filters via backpropagation. This is the main benefit of the inception module since it enables the internal layers to select the filter size that is appropriate for learning the necessary information. Finding a high-level feature that can appear in varied sizes on multiple input feature maps using this method is highly beneficial.

Figure 6-2: Example of an Inception Module.
Receptive Field

The receptive field of an inception network is the most important metric to comprehend. As shown in Figure 6-3, a neuron in an inception network, in contrast to fully connected networks, only depends on a portion of the input characteristics map. The receptive field of the neuron is where this area is located. It is obvious that neurons in the bottom layer rely on smaller regions than those in the top layer. Then, it is anticipated that bottom-layer neurons will record the local structure of a time series while top-layer neurons will recognize more intricate patterns. For time series data, the total receptive field of an inception network is defined from Equation 6-1, which depends only on the length of the filters $k_i$ and on the depth of the network $d$ (that is the number of Inception Modules). It's fascinating to investigate how an inception network's accuracy changes as the receptive field changes. We can alter the depth of the network or the filter lengths to alter the receptive field. Typically, a longer filter is needed to provide more precise results. This can be explained by the fact that longer filters have a better possibility of catching longer patterns than shorter ones. On the other hand, for datasets with a short training set, expanding the Receptive Field by adding additional layers does not always result in an enhancement of the network's performance. So, rather than adding more layers to increase accuracy, it is usually preferable to lengthen the filters.

![Figure 6-3: Receptive Field.](image-url)
Inception Time Architecture

Several studies have revealed that a single inception network can occasionally display considerable accuracy variance. This is most likely caused by the variability that the initialization of the random weights provides. Inception time is typically implemented as an ensemble of several inception networks, each prediction having the same weight, to combat this instability. In doing so, the algorithm displays the previously indicated great accuracy and excellent scalability, as well as improving its stability. This is an excellent finding given that many other algorithms expand quadratically about the same magnitudes. Various investigations have demonstrated that its time complexity develops linearly with both the training set size and the time series length.

Implementation

The implementation of this architecture for this study is an adaptation of the architecture created by Fawaz et al. (2020). The architecture closely follows Figure 6-1, with six inception modules, two residual connection layers, and one average pooling layer connected to the fully connected layer which predicts the label or class of the input in the output layer. The inception module is constructed as shown in Figure 6-2. The input multivariate time series is connected to a bottleneck layer which convolutes the input to thirty-two filters using a kernel of size one and linear activation function. The outputs of the bottleneck layer are convoluted by three convolutional layers.
of ten, twenty, and forty kernel sizes using thirty-two filters and a linear activation function. The outputs are max pooled and then the output of each parallel convolution is concatenated, normalized, and activated using the ReLu activation function to form the output multivariate time series of the current inception module. Six such modules are connected in series with the output of one module serving as the input to the other module. Every third inception module has a residual connection layer that convolutes using a filter size of one and ReLu activation function. Finally, all the convolutions are average pooled and connected to a dense layer having a softmax activation function. To compile the model categorical cross entropy loss function and adam optimizer were employed. The training phase employs test and train datasets only with test data being twenty-five percent and train data being the remaining percent of the dataset.

**Type ITN Models**

For ITN architecture, the first eight models were trained on the eight permuted datasets like the methodology followed in the CNN and ESN architecture sections and tested for the best combination of data types for this architecture.

**Results**

The five model evaluation metrics were calculated and are tabulated in Table 6-1 with the training time of each model. Clearly, there are two models which outperform the other models in all areas. The first one is trained on data which is a combination of all datasets i.e., DD, ACC, GY, and GZ datasets, and has an F1 score of 0.981 with a training time of 52 seconds. The second model is trained on the data which is a combination of DD and GY datasets and has the same F1 score of 0.983, but a higher training time of 75 seconds. Because of the significant difference in the training
times of the models, the combination of DD, ACC, GY, and GZ datasets is taken as the best combination, and varying time window size models were built for further analysis.

Table 6-1: Type ITN models’ evaluation metrics.

<table>
<thead>
<tr>
<th>Dataset Type</th>
<th>Accuracy</th>
<th>Specificity</th>
<th>Sensitivity</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Training Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDACC GYGZ</td>
<td>0.981</td>
<td>0.97</td>
<td>0.991</td>
<td>0.971</td>
<td>0.981</td>
<td>52</td>
</tr>
<tr>
<td>DD</td>
<td>0.965</td>
<td>0.978</td>
<td>0.952</td>
<td>0.977</td>
<td>0.964</td>
<td>71</td>
</tr>
<tr>
<td>DDACC</td>
<td>0.978</td>
<td>0.983</td>
<td>0.974</td>
<td>0.983</td>
<td>0.978</td>
<td>74</td>
</tr>
<tr>
<td>DDGY</td>
<td>0.983</td>
<td>0.98</td>
<td>0.985</td>
<td>0.98</td>
<td>0.983</td>
<td>75</td>
</tr>
<tr>
<td>DDGZ</td>
<td>0.971</td>
<td>0.963</td>
<td>0.979</td>
<td>0.963</td>
<td>0.971</td>
<td>55</td>
</tr>
<tr>
<td>DDACC GY</td>
<td>0.973</td>
<td>0.975</td>
<td>0.971</td>
<td>0.974</td>
<td>0.973</td>
<td>75</td>
</tr>
<tr>
<td>DDACC GZ</td>
<td>0.977</td>
<td>0.97</td>
<td>0.985</td>
<td>0.969</td>
<td>0.977</td>
<td>56</td>
</tr>
<tr>
<td>DDGY GZ</td>
<td>0.978</td>
<td>0.975</td>
<td>0.982</td>
<td>0.976</td>
<td>0.979</td>
<td>58</td>
</tr>
</tbody>
</table>

DDACC GGYGZ ITN Models

Again, following the same methodology as followed for the CNN and ESN architecture, for analyzing the impact of window sizes on the performance of the model, ten models were trained using the same ITN architecture, but for the time window sizes varying from one to ten seconds with an increment of one second each.

Results

It can be observed from Table 6-2 that the six-second window size model has the highest F1 score of 0.991 but shows a lack of consistency among the metrics as well as a high training time of 161 seconds. A better model is the four-second window size model, which has a somewhat lower F1 score of 0.99, but has a significantly better training time of 136 seconds and performs somewhat consistently across all five metrics (Figure 6-4). The insignificant decrease in the F1 score is
sufficiently overpowered by the increase in computational efficiency of this model when compared to the six seconds window size model.

Table 6-1: DDACCGYGZ ITN models’ evaluation metrics.

<table>
<thead>
<tr>
<th>Window Size (Seconds)</th>
<th>Accuracy</th>
<th>Specificity</th>
<th>Sensitivity</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Training Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.966</td>
<td>0.945</td>
<td>0.987</td>
<td>0.949</td>
<td>0.967</td>
<td>51</td>
</tr>
<tr>
<td>2</td>
<td>0.974</td>
<td>0.964</td>
<td>0.985</td>
<td>0.962</td>
<td>0.973</td>
<td>91</td>
</tr>
<tr>
<td>3</td>
<td>0.986</td>
<td>0.985</td>
<td>0.987</td>
<td>0.985</td>
<td>0.986</td>
<td>115</td>
</tr>
<tr>
<td>4</td>
<td>0.989</td>
<td>0.986</td>
<td>0.993</td>
<td>0.986</td>
<td>0.99</td>
<td>136</td>
</tr>
<tr>
<td>5</td>
<td>0.986</td>
<td>0.984</td>
<td>0.988</td>
<td>0.984</td>
<td>0.986</td>
<td>150</td>
</tr>
<tr>
<td>6</td>
<td>0.991</td>
<td>0.985</td>
<td>0.998</td>
<td>0.985</td>
<td>0.991</td>
<td>161</td>
</tr>
<tr>
<td>7</td>
<td>0.989</td>
<td>0.984</td>
<td>0.994</td>
<td>0.984</td>
<td>0.989</td>
<td>171</td>
</tr>
<tr>
<td>8</td>
<td>0.983</td>
<td>0.968</td>
<td>0.997</td>
<td>0.97</td>
<td>0.983</td>
<td>189</td>
</tr>
<tr>
<td>9</td>
<td>0.99</td>
<td>0.986</td>
<td>0.993</td>
<td>0.986</td>
<td>0.989</td>
<td>224</td>
</tr>
<tr>
<td>10</td>
<td>0.985</td>
<td>0.983</td>
<td>0.988</td>
<td>0.982</td>
<td>0.985</td>
<td>236</td>
</tr>
</tbody>
</table>

Figure 6-4: DDACCGYGZ ITN models’ metrics plot.
This literature talks about the process of developing a robust framework with a multiple-input Convolutional Neural Network (CNN) architecture, Echo State Network (ESN) architecture, and Inception Time Network (ITN) architecture to predict a driver’s takeover behavior in conditionally AVs with drivers’ driving and eye-tracking data. It is one of the first DL architectures that could predict takeover for scheduled non-time-critical freeway exiting by drivers in conditionally AVs. Since the larger time windows can anticipate a longer future in the time series scenario, multiple window sizes were used to ensure the appropriate configuration of the three architectures. The model for the combination of DD, ACC, and GZ data built using a 10-fold cross-validation technique with a window size of 5 seconds was identified as the best CNN model with an F1-Score of 0.993 and a training time of 46 seconds. The model for the combination of DD and GZ data built using a 10-fold cross-validation technique with a window size of 6 seconds was identified as the best ESN model with an F1-Score of 0.979 with a training time of 58 seconds. And the model for the combination of DD, ACC, GY, and GZ data built using an ITN architecture with a window size of 4 seconds was identified as the best ITN model with an F1 score of 0.99 and a training time of 136 seconds. The CNN architecture proved to be the best out of the three for the collected data, which means that a simple CNN had the ability to extract valuable information for predicting the driver’s takeover behavior in comparison to an ITN which even though had comparable evaluation metrics but had a significantly higher training time because of its higher complexity. Both the convolution based networks performed better than the RNN based ESN, which derives that temporal information of the time series fails to provide the necessary features for accurately predicting a driver’s takeover, whereas convolutions of the time series were able to extract such features.
Furthermore, these models can be used as a design reference for adaptive in-vehicle alert systems to enhance takeover capability in partially autonomous driving. These models are also useful for improving safe freeway exiting takeovers in conditionally autonomous vehicles (AVs) when designing human-machine interfaces regarding ToR lead time. To enhance the performance of freeway exiting takeover, they can also relate to the vehicle's driving models.

**Importance of Window Size**

The window size controlled the size of each data point which in turn affected how much information was given to the model to make an individual prediction. Moreover, processing data from a longer window would also require additional time thereby limiting the usefulness of the prediction. Therefore, the most effective window size would be a balance between model performance and the time required to make the prediction. In the current study, models obtained from window sizes of 4 or 5, or 6 seconds had very similar performance. This suggests that the driver’s takeover behavior is not dependent on the information far out in the past in the time series.

**Limitation and Future Work**

Although the current work highlights the importance of a DL architecture to precisely predict takeover behaviors while exiting the freeway in a conditionally AV, there are certain limitations that need to be addressed. The data used for the study is derived from a driving simulator study. The models need to be validated in the future with naturalistic driver data in AV. Secondly, an experiment with a mere thirty participants cannot provide data that accurately simulates the whole human population. DL architectures perform well for huge datasets which were engineered for this study and hence there is a need for a large experiment that solves this problem.
References


